STRUCTURAL ANALYSIS OF DAEs

Mikael Zebbelin Poulsen

IMM-PHD-2001-91

IMM

Informatics and Mathematical Modelling
Technical University of Denmark

Contents

Abstract

Differential algebraic equations (DAEs) constitute a fundamental model class for many modelling purposes in engineering and other sciences, especially for dynamical simulation of component based systems.

This thesis describes a practical methodology and approach for analysing general DAE. The methodology is mainly based on *strutural index analysis* which is not limited by the index of the DAE as other methodologies. As a result of structural index analysis one can perform *index reduction* of the DAE and obtain the so-called *augmented underlying ODE*. It is also described, how to use the augmented underlying ODE for finding *consistent initial values* and solve the *initial value problem* for the original DAE.

As a methodology for integrating the augmented underlying ODE, the dummy derivative method is investigated. The methodology avoids the traditional stability and drift-off problems of using the underlying ODE. The investigations concern the identification of quantities that can trigger the automatic choice of new dummy derivatives during integration. This is a practical problem that needs to be solved before implementations of the method are possible.

The general methodology is tested in practice, by the implementation of the Simpy tool box. This is an object oriented system implemented in the Python language. It can be used for analysis of DAEs, ODEs and non-linear equation and uses e.g. symbolic representations of expressions and equations.

The presentations of theory and algorithms for structural index analysis of DAE is original in the sense that it is based on a new matrix representation of the structural information of a general DAE system instead of a graph oriented representation. Also the presentation of the theory is found to be more complete compared to other presentations, since it e.g. proves the uniqueness of the structural index reduction process. Also included, is a discussion of criticism and defence of structural analysis.

Resumè

Differential algebraiske ligninger (DAE'er) er grundlæggende for mange modelleringsopgaver indenfor tekniske- og andre videnskaber, især når det gælder dynamisk simulering af komponentbaserede systemer.

Denne afhandling beskriver en praktisk anvendelig metode til analyse af DAE'er. Metoden er hovedsageligt baseret på *strukturel indeks-analyse* og er ikke begrænset af indeks for DAE systemet. Som resultat af strukturel indeks-analyse, kan man udføre *indeks-reduktion* af DAE systemet og derved finde den såkaldte *udvidede underliggende sædvanlige differentialligning (AUODE)*. Det er også beskrevet hvordan man kan bruge AUODE'en til at finde konsistente begyndelsesværdier og løse begyndelsesværdiproblemet for det oprindelige DAE system.

Som metode for integration af AUODE'er er *"dummy derivative"-metoden* undersøgt. Metoden undgår de sædvanlige problemer med stabilitet og "drift-off" som kendetegner brugen af den underliggende sædvanlige differentialligning alene. Undersøgelserne omhandler identifikation af mål for hvornår der automatisk skal skiftes "dummy derivatives" under integrationen. Dette er et praktisk problem der skal løses før man kan implementere metoden.

Metoden i sin helhed, er demonstret i praksis ved implementeringen af værktøjet *Simpy*. Dette er et objekt-orienteret softwaresystem implementeret i programmeringssproget Python. Det kan anvendes til analyse og løsning af DAE'er, sædvanlige differentialligninger og ikke-lineære ligningssystemer og bruger f.eks. symbolsk repræsentation af udtryk og ligninger.

Præsentationen af teorien og algoritmerne for strukturel indeks-analyse er original i den forstand at den er baseret på en nyudviklet repræsentation af den strukturelle information for et generelt DAE system, som er baseret på en matrix i stedet for en matematisk graf. Præsentationen er også mere komplet sammenlignet med andre fremstillinger, idet den f.eks. beviser entydigheden af den strukturelle index-reduktionsprocess. Den inkluderer også en diskussion af kritik og forsvar af strukturel indeks-analyse.

Preface

This thesis is intended to meet the partial requirements for obtaining the Ph.D. degree from the Technical University of Denmark (DTU). The studies were carried out according to the Ph.D. programme in Mathematics, at the former Department of Mathematical Modelling, now Informatics and Mathematical Modelling (IMM), under the supervision of Per Grove Thomsen.

In August 1997, I finished a master project at the Department of Energy Systems (ET) at DTU in collaboration with Falko J. Wagner. It had the (translated) title *Object-oriented modelling and dynamical simulation* . See [PW97].

This project concerned the development of a concept and software system for modelling of component based dynamical systems, with the intention of modelling energy systems.

As the project went on we discovered problems in analysing and solving the mathematical models. The mathematical models for these systems, often showed to be of the kind *differential algebraic equations (DAEs)*.

During spring 1997 I applied for a Ph.D. position under the Ph.D. programme in Mathematics, to be able to continue the study of these problems.

My Ph.D. project started October 1997, with Per Grove Thomsen (IMM) as supervisor, and Niels Houbak (ET) as associated supervisor. The topic was component based modelling, DAEs, and continued research related to the GODESS project, in applying object-oriented techniques to the design and implementation of numerical algorithms and mathematical software.

Description of the thesis

The thesis is intended to describe the theory and methodology for analysing DAE system, which have been partly identified and partly developed during the Ph.D. project. Readers are supposed to have some background in technical mathematics and numerical analysis.

The first three chapters basically contains motivation and background material for the thesis. *Chapter 1 - Introduction* contains a motivation for treating DAE systems. *Chapter 2 - Background material* describes the background material of relevance for modelling using DAEs and for understanding the theory behind the presented methodology. This especially includes theory for ODEs, non-linear equations and implicit functions, and structural analysis. *Chapter 3 - Differential algebraic equations* contains an introduction to DAEs, a discussion of DAE characteristics e.g. *indices*, and finally discusses direct methods and index reduction for DAEs.

The following three chapters presents the approach, theory and methodology for analysis of DAE system identified and developed during the Ph.D. project. This is the core material of the thesis. *Chapter 4 - DAEs on integral form* discusses the overlooked topic of assigning model variables to the general implicit DAE form. It introduces the unambigous *integral form* for representing DAEs, based on the integral notation of the DAE and defines the *general ODE*. *Chapter 5 - Structural index analysis* describes the partly original theory for structural index analysis of DAEs, based on the new index matrix representation. It also presents corresponding algorithms. *Chapter 6 - Augmented underlying ODE* describes how one can use the augmented underlying ODE for solving initial value problems for DAEs. This includes finding consistent initial conditions. A special method called the *dummy derivative method* is investigated, along with some issues for practical implementation of the method.

The following chapters presents general discussions and demonstration of the methodology. *Chapter 7 - Discussion of methodology* presents the overall methodology proposed in the thesis. Hereafter follows a discussion of this methodology and some general aspects of DAEs. *Chapter 8 - Simpy* describes the "proof of principle" and experimental tool box Simpy developed in this Ph.D. project. It can be used for analysing DAEs.

Finally *Chapter 9 - Conclusion* presents the conclusions and summary of results and methodology. Two appendices follows hereafter: *Appendix A - Some theory and proofs on structural analysis* and *Appendix B - List of symbols*.

Acknowledgements

I wish to thank my supervisor Per Grove Thomsen for giving me the opportunity to study and investigate an interesting subject. Thank you for patience and encouragement to stay focused even when results in periods seemed absent. I also want to thank my associated supervisor Niels Houbak for motivation and guidance.

I want to thank for the hospitality of Gustaf Söderlind and Claus Führer, showed to me under my stay at Lund University. During my stay I made acquaintance with a number of interesting people at the department, who all made my stay even more pleasant. Thank you Lina, Anders and José.

As a Ph.D. student at the Numerical Analysis section at IMM, I have received help and input from all personnel, and I send thanks to my fellow Ph.D. students at the department for encouragements during a long process.

Finally, I sincerely thank my dear family and friends for constant support.

Mikael Zebbelin Poulsen, IMM
April 2002.

Chapter 1

Introduction

The central task in engineering and applied technical sciences is development and analysis of mathematical models of real world systems.

A very important class of these models have the intention of describing the dynamics of such systems. For this purpose we use various types of mathematical models: Ordinary differential equations (ODEs) are well-known for this purpose, but a more general model type, called *differential algebraic equations (DAEs)*, have become increasingly important. Especially, this is the case when large mathematical models are build from a so called *component based description*.

This thesis focuses on use and practical analysis of *DAEs*. Some general topics and problems concerning DAEs are discussed, and a general methodology, based on structural analysis and subsequent transformation to an ODE model type, the *augmented underlying ODE*, is proposed.

The theory and use of ODEs and DAEs will be presented in the context of dynamical systems modelling, although it can be presented in a more general context[1].

For demonstrating the methodology for analysing DAEs, a software tool box Simpy is implemented. The purpose of this tool box is also to *contribute to development and design* of mathematical software tools that scientist and engineers can use when building and analysing models of dynamical systems.

[1]Here the independent variable of the ODEs and DAEs represents time, but in other contexts it can represent other quantities as e.g. a spatial coordinate.

1.1 Ordinary differential equations

Ordinary differential equations (ODEs) are a fundamental and classical approach to modelling of dynamical systems.

The use of ODEs have a long history, and can be traced back to at least Newton, the year 1671, in the youth of differential calculus. See e.g. [HWN93]. As a consequence ODEs are well-understood and from a modelling perspective, scientists are generally confident with their use.

We will take a short look at the ODE model: If quantities modelling the system state are represented by a vector function $y(t) : \mathbb{R} \mapsto \mathbb{R}^n$, where t is the time, it is well-known that the (first-order) ODE has the general form:

$$\dot{y} = f(y, t). \tag{1.1}$$

Here $f : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$ and for short notation we use $y = y(t)$, and $\dot{y} = \dot{y}(t)$ (pronounced "y-dot") denotes the time-derivative dy/dt.[2]

We note the characteristics of the ODE model: *We have an explicit expression for each time-derivative $\dot{y}$ as a function of y and t.* This means that for a given state of the modelled system, the ODE model expresses the *rate of change* for each quantity dependent on the state of the system. That this allows one to obtain a continuous description of the modelled quantities back and forth in time is a fundamental theorem from the theory of ODEs. Obtaining this continuous description is called *solving the initial value problem.*

A thorough understanding of the ODE model type is important for understanding DAEs, and therefore some fundamental theory and aspects of using ODEs will be introduced as background material in section 2.2.

1.2 Differential algebraic equations

Differential algebraic equations (DAEs) can describe the dynamics of a system, at a very similar way as ODEs: The system state is represented by a vector $y(t) : \mathbb{R} \mapsto \mathbb{R}^n$ dependent on time t, and the model uses expressions based on $y(t)$, the derivatives of these $\dot{y}(t)$ and time t.

A DAE system is, in contrast, characterised by not explicitly expressing the time derivatives $\dot{y}(t)$. Instead, the system dynamics are specified by implicit equations:

The *general implicit DAE system* has the (first-order) form:

$$f(\dot{y}, y, t) = 0. \tag{1.2}$$

[2]Another commonly used notation style is $y' = y'(t)$ (pronounced y-prime).

Here $f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$ represents the residual of a general non-linear equation system. Again for shorter notations $y = y(t)$ and $\dot{y} = \dot{y}(t)$.

We notice that the explicit ODE (1.1) is a subclass of the general implicit DAE (1.2), when rewriting the explicit ODE to residual form. This illustrates that the general implicit DAE is a is a very general model type: It can contain any model, also ODEs, using general non-linear equation based a set of model quantities y and their time-derivatives $\dot{y}$.

The fundamental difference between ODEs and *proper*[3] DAEs, is that for proper DAEs, *the derivatives $\dot{y}$ are not given as an explicit expression of y and t, nor as an implicit function of y and t.* Trying to solve for the derivatives $\dot{y}$ directly from a proper DAE system as (1.2), we experience that the equation system is *under-determined.*

Despite this, the DAE model also allows one to determine a continuous description of the model quantities y as a function of time t, just as would be expected from the ODE model. In this way the *initial value problem is also relevant for DAEs.* But, instead of expressing all the derivatives, the DAE system specify additional constraints on the model quantities, implying that the state variable and consequently the derivatives are limited to some manifold.

This and the near relation to ODEs, is nicely illustrated by special DAEs which consist of a combination of ODE like equations and algebraic constraints on the model quantities:

$$\begin{aligned} \dot{y} &= f(y, x, t) \\ 0 &= g(y, x, t). \end{aligned} \tag{1.3}$$

Sometimes these are referred to as *semi-explicit DAEs.*

The following example will show how a model of a simple system, though looking very similar to an ODE, is an example of a DAE of the form (1.3):

■ **Example 1.1 (2D-Pendulum system)** *The pendulum system is a classical example of a system which can be modelled as a DAE.*

The pendulum system, see Figure 1.1, consist a of mechanical mass-particle element (no moment of inertia) with mass M, connected to a point in a 2-D space by a pole of length L. Furthermore, the particle is moving under influence of gravity of size g. By choosing appropriate coordinates, see

[3] Proper, denotes that the DAE is not just an *implicit ODE*. This is the simplest subclass of the general implicit DAE (1.2), and it has properties almost identical to ODEs. More on this follows in Chapter 3.

Figure 1.1, the model equations will be

$$M\ddot{x} = -F\frac{x}{L} \tag{1.4a}$$

$$M\ddot{y} = -F\frac{y}{L} - Mg \tag{1.4b}$$

$$x^2 + y^2 = L^2 \ . \tag{1.4c}$$

Here $\ddot{x}$ and $\ddot{y}$ denotes second-order time-derivatives of x and y, i.e. horizontal and vertical acceleration.

We can try to interpret the equations (1.4) *as an ODE* (1.1). *To do this we rewrite to first-order form, by introducing the velocity variables v_x and v_y. We also try to isolate the derivatives of the model quantities on the left side of the equations to reveal if the derivatives are explicit functions of the other variables:*

$$\dot{v_x} = -F\frac{x}{ML} \tag{1.5a}$$

$$\dot{v_y} = -F\frac{y}{ML} - g \tag{1.5b}$$

$$\dot{x} = v_x \tag{1.5c}$$

$$\dot{y} = v_y \tag{1.5d}$$

$$0 = x^2 + y^2 - L^2 \tag{1.5e}$$

We notice that we can not succeed in isolating all derivatives: The derivative of F does not appear anywhere. Nor can F, itself, be determined implicitly from the model equations. Therefore we conclude that the model is not an ODE.

We can alternatively confirm that (1.5) can *be considered a DAE of the form* (1.3), *where* (1.5e) *constitutes the algebraic constraints, or more generally* (1.5) *can be considered of the form* (1.2). ■

1.2.1 Problems concerning DAEs

To study and develop methodologies for analysing the general form (1.2) has been one of the main objectives of this Ph.D. project, and it is the central topic of this thesis.

Properties of DAE systems are not as well understood as the properties of traditional ODE systems, and scientist are not as confident in their use. Along with this, the theories for numerical methods and their ability to solve the important *initial values problem (IVP)* for DAEs are not developed to the same extent. As we will see later, one can not trivially adapt the methods for ODEs to DAEs.

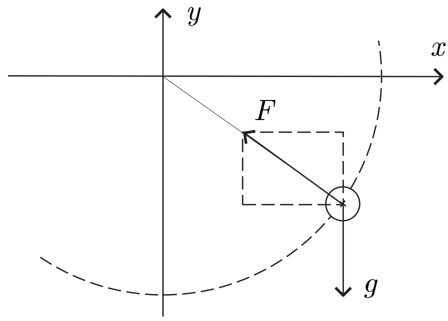


FIGURE 1.1: Illustration of the 2-D pendulum system. Using a natural choice of variables and setting up the classical equations for the movement, including the constraint from the length of the pendulum, this system results in a DAE model.

Yet another important problem concerning DAEs is to identify *consistent initial values* for initializing an IVP. Finding initial values is not "trivial" in the same way as for ODE systems. Methods for identifying these are not generally available.

Therefore the objective of this Ph.D. study has been to concentrate on the general DAE, and investigate

- methodologies for finding consistent initial values.
- methodologies for solving the IVP for the general implicit DAEs.
- investigate their practical implementation.

1.2.2 Motivation for considering DAEs

We have already seen, in Example 1.1, that DAEs arise as dynamical models, consisting of a combination of ODEs and algebraic constraints. While this example comes from mechanics, similar DAE models arise for energy systems, electrical circuits and chemical reactions, etc.

Yet another motivation is the general form that the DAE (1.2) has. Assume a software tool where scientist can specify and analyse dynamical models of their systems. In this modelling tool they prefer not to care if the system looks like the classical ODE (1.1), but they want to freely use general non-linear equation, implicit equation and algebraic constraints using the model quantities and their time-derivatives among each other. The general implicit DAE will cover any model type arising from such a flexible differential equation modelling tool.

Another motivation is analysis of *partial differential equations*. These describe the system state, by continuous functions over a domain, more than by a finite number of quantities as is the case for ODEs. In practical (numerical) computations the PDEs are discretized into a finite dimensional representation of the state, and at this point the resulting discretized mathematical model might form a DAE system. See e.g. [BCP96, p. 177 ff.].

The last, but not the least motivation for treating DAE, is that DAEs arise from *component based modelling*. This is described in the following section.

1.3 Component based modelling

As mentioned, a very important motivation for doing research in DAEs, is the modelling principle called *component based modelling*. This modelling principle can be applied for modelling large complex systems from different technical fields in engineering but also other sciences.

Examples in engineering could be energy systems as power plants and refrigeration systems, mechanical systems as robots and cars, electrical systems and circuits or chemical systems as chemical process plants. This list could be much longer.

1.3.1 Divide and conquer

The component modelling principle is based on the well-known problem solving strategy - "to divide an conquer". Component based modelling is about considering large and complex systems as composed of smaller parts called *components*.

Subsequently, one can concentrate on developing models for these components which are hopefully simpler to describe. In addition, for a each major field of application, one will often discover, that the large complex systems, can be composed from a relatively small set of *fundamental components*, using the same component models in several "places" of the model.

As example a power plant consist of several pumps, pipes, turbines and condensers which constitutes some of the fundamental components for energy systems modelling.

1.3.2 Modelling languages and libraries

Component based modelling motivates the development of large component model libraries for each field of application. This has been an increasing research activity in many fields. For energy systems, see e.g. [EHL97] or [Wag00].

The development of these libraries is dependent on specification methods for the component models. Therefore research has also be put into defining well-suited general specification or modelling languages, with general application to various fields.

To learn more about this and component based modelling in general it is recommended to read about "Modelica" (see [Mod99] or [MOE99]) or study some of the ancestors "Omola" or "SMILE" (see e.g. [And94] or [TKE97]). A related effort is the construction of the "MOSES" which is described in [MG98].

One can read about using C++ as a component modelling language in [PW98]. In [PWTH99] one can find an example on creating a small library for simulation of simple mechanical systems using the C++ modelling language.

1.3.3 Component based modelling results in DAEs

Component based modelling often result in mathematical models being DAEs. This will be illustrated in this section.

First assume that we have two abstract components A and B. For the simplest case assume that these components can individually be modelled by ODE-like equations:

$$\begin{aligned} \dot{y}_A &= f_A(y_A, x_A, t) \\ \dot{y}_B &= f_B(y_B, x_B, t) \ . \end{aligned} \tag{1.6}$$

The individual component models and also the combined equation system are all under-determined. But when the components are joined, by adding constraints which couple the variables of the components, then the combined system can suddenly be determined:

$$\begin{aligned} \dot{y}_A &= f_A(y_A, x_A, t) \\ \dot{y}_B &= f_B(y_B, x_B, t) \\ 0 &= g(y_A, x_A, y_B, x_B, t) \ . \end{aligned} \tag{1.7}$$

We realise that this system is exactly of the kind (1.3), and we consequently accept that component based modelling can result in DAEs.

Component based model can also result in DAE models for which we need the fully implicit DAE form (1.2). The following example is intended to illustrate this.

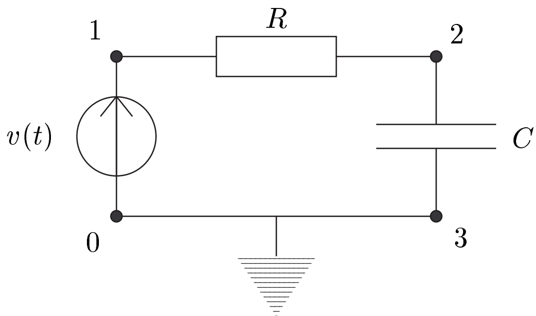


FIGURE 1.2: Example of simple electrical circuit build from components. This system would typically result in a DAE system, when modelled using the component principle.

■ **EXAMPLE 1.2 (SIMPLE ELECTRICAL CIRCUIT)** *We will create a model for the system illustrated in Figure 1.2. This system has four basic components: A resistor, a capacitor, a voltage-source, and a ground connection.*

The resistor is characterized by the voltage drop $\Delta u = RI$. The capacitor by the time-derivative of the voltage drop $C\Delta\dot{u} = I$. The voltage-source defines a voltage drop $\Delta u = v(t)$ as a function of time. The ground node simply defines a level for the voltage at both ends $u = 0$.

As seen on Figure 1.2 we define four nodes. We name the corresponding voltage levels: u_0, u_1, u_2 and u_3. We quickly realise that the currency in all the components should be the same. Therefore we choose to model the currency using one variable I.

$$\begin{aligned} u_1 - u_2 &= v(t) \\ u_1 - u_2 &= RI \\ C(\dot{u}_3 - \dot{u}_2) &= I \\ u_3 &= 0 \\ u_0 &= 0 \end{aligned} \tag{1.8}$$

Now we try to identify what kind of mathematical model corresponds to (1.8)*. We see that the equations use derivatives and therefore we have some sort of differential equation. We quickly realise that* (1.8) *is* not an ODE *since it is not possible to directly isolate the derivatives on one side.*

Instead we see that it can fit into the form of the general implicit DAE (1.2)*, and we acknowledge that modelling this component system resulted in a DAE.* ■

1.4 Software for mathematical modelling

Today, tools for modelling and analysing dynamical systems using e.g. component based models are in great demand. Therefore also implementation of software tools and numerical algorithms for analysing these models, and especially DAE systems are of great importance.

The relevance of using modern software techniques for implementing ODE software and other numerical software can be motivated by a number of reasons. Fundamentally, one can just observe that the character of large mathematical and numerical software project has great similarity with general software projects.

An objective in this Ph.D. project has also been to investigate modern design of such modelling software and numerical software. Especially, general patterns for object-oriented design has been investigated.

Hence, for demonstrating the results regarding the methodology for analysing DAEs, but also for demonstrating and experimenting with design principles of numerical software, the tool-box Simpy was created. This is described in Chapter 8.

Chapter 2

Background material

This chapter presents some of the background material which is important when dealing with differential algebraic equations, and especially material which is important for the approach taken in this thesis.

The first topic treated in this chapter is mathematical modelling of dynamical systems. Here some fundamentals of dynamical systems modelling are introduced.

Next, analysis and theory of differential algebraic systems, is hardly realised without a fundamental understanding of some theorems and results from the topic of ordinary differential equations. Therefore some basic theory for ODEs and some basic material on numerical methods for ODEs are included.

The complexity of analysing differential algebraic equations, comes partly from the fact that we allow the implicit definition of the variables in the model. To understand the possibilities and problems of using implicit relations, a section about implicit functions and non-linear equations is included.

The last section is intended to present an introduction and the necessary background knowledge on structural analysis of non-linear equation systems, to be able to understand the theory for structural analysis of DAEs which is central for this thesis.

2.1 Mathematical models for dynamical systems

Throughout the thesis, differential equations, algebraic or ordinary, are mainly considered as mathematical models for modelling of dynamical systems, though other uses are possible. This depend on the independent variable, representing either time or some other quantity.

The section introduces some basic definitions from modelling of dynamical systems, including the viewpoint, terminology and notation which the author holds and uses. The introduction will concentrate on the terminology relevant for modelling, using ordinary differential equations and differential algebraic equations.

Dynamical systems modelling is applied in various technical fields and quite natural terminology and notation varies between fields. It is therefore found relevant to present the terminology used by the author, a terminology which has developed from discussions with other researchers.

2.1.1 Fundamentals of a dynamical model

The fundamental purpose of a dynamical model, is to describe the behaviour of a system over time. We choose the symbol t for representing the time, and we model the time as a real-valued variable

$$t \in \mathbb{R}. \tag{2.1}$$

In the modelling process a number of mathematical quantities are chosen to represent the *system state*, and the purpose of the dynamical model is to define how these quantities depend on time. We can choose to represent the state of a system by a variable y and the set of all possible states, called the *state space*, by $\mathbb{Y}$. We write

$$y \in \mathbb{Y}. \tag{2.2}$$

2.1.2 Continuous time model

A description of the system state at any point in time over a time-interval $D = [t_a, t_b]$, can be represented by a function

$$y(t) : D \mapsto \mathbb{Y}. \tag{2.3}$$

If the purpose of a mathematical model is to define such functions $y(t)$, as in (2.3), we call the model a *continuous time model.*

If we consider $y = y(t)$ as a variable in $\mathbb{Y}$, we can say that the model defines y as a *dependent* variable with t as the *independent* variable.

This thesis focuses on continuous time models, but not all models of dynamical systems are continuous time models. Alternatively, we might describe the system state only at discrete time points $t_0, t_1, t_2, \ldots$.

2.1.3 Finite dimensional real-valued model

The kind of models, central to this thesis, will be characterized by the state being represented by a finite-dimensional and real-valued vector, i.e.

$$y \in \mathbb{Y} = \mathbb{R}^n \qquad n \geq 1. \tag{2.4}$$

Alternatively, the dependent variable y can be other mathematical quantities: y can be complex-valued in contrast to real-valued, or functions e.g. over spatial coordinates, in contrast to finite-dimensional vectors as for partial differential equations. These are not directly concerned in this thesis.

When we consider the model type defined by (2.4), $y(t)$ is a vector function,

$$y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{bmatrix} : \mathbb{R} \mapsto \mathbb{R}^n \tag{2.5}$$

and we can consider $y(t)$ as a collection of individual functions or variables $y_i = y_i(t), i = 1, \ldots, n$. We sometimes say that the model has a number of dependent variables or *dynamical variables*.

2.1.4 Continuity, differentiation and integration

If $\mathbb{Y}$ is a metric space, the function $y = y(t)$ can be characterised by general properties of functions between metric spaces: *continuity*, *differentiability* etc. See e.g. [Rud76]. This further implies that differentiation and integration are defined.

Using differentiation and integration, we will for $y = y(t)$, subject to certain criteria of differentiability and integrability respectively, be able to define two new dependent variables $\dot{y}$ and Y:

$$\dot{y} = \dot{y}(t) = \frac{dy}{dt} \quad \text{and} \quad Y = Y(t) = \int_{t_0}^{t} y(\tau)d\tau + Y_0 \tag{2.6}$$

This is essential for the ODE/DAE model types which we are going to treat in this thesis. Definitions of differentiation and integration can be found in e.g. [Rud76].

The finite-dimensional real-valued model, just presented in (2.4), has state space $\mathbb{Y} = \mathbb{R}^n$, which can trivially be considered a metric space. Therefore integration and differentiation is defined for this model type.

Note the "dot" notation for differentiation, which is often used in dynamical modelling as a way to denote *time-derivatives*.

2.1.5 Deterministic model

Given the state of the system at one point in time, a *deterministic model* will determine all other states in a surrounding time interval. When modelling dynamical systems using ordinary or differential algebraic equations, these are examples of deterministic models.

As alternatives to deterministic models, one often mentions stochastic models, where the state is described by a stochastic function, with a distribution possibly dependent on knowledge about the state at one or more given points in time.

Some models are only deterministic in one direction of time, i.e. the model cannot be used for determining the state backwards in time. This is the case if several states in the past can lead to a single state at a later point in time. This is specially the case for some models, using a combination of differential equations and *state events*.

2.1.6 Initial value problem

Considering the deterministic models, the classical use of these models is to assume a state at some point in time and then use the model for determining the states over a time interval or at some discrete points in time. This popular use of a deterministic model is called solving an *initial value problem (IVP)*.

When discussing simulation of a dynamical system, the terms *simulation* often refers to solving an IVP.

For some models finding an initial value which is consistent with the model is in itself a problem. This is called the problem of *finding consistent initial values*. This especially characterises the use of DAE models and therefore this problem is relevant for this thesis.

2.2 Ordinary differential equations

Ordinary differential equations (ODEs) play the central role in simulation of dynamical systems. Referring to the introduced terminology, dynamical models build on ODEs belong to the class of deterministic, finite-dimensional, real-valued, continuous time models.

Definition 2.1 (Ordinary Differential Equation) *An ordinary differential equation (of first-order) is an equation system defining a dependent variable vector function* $y = y(t) : \mathbb{R} \mapsto \mathbb{R}^n$ *by*

$$\dot{y} = f(y, t) \ , \text{ for } t \in D \subseteq \mathbb{R} \tag{2.7}$$

where $f : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$ *and* $\dot{y} = dy/dt$*, and* D *is an interval in* $\mathbb{R}$*.*

The remark in Definition 2.1 concerning the *order* of the ODE, is related to the high-order form of ordinary differential equation: A q'th-order ordinary differential equation uses derivatives up to the q'th order and has the form

$$y^{(q)} = f(y^{(q-1)}, \dots, \dot{y}, y, t) \ , \text{ for } t \in D \tag{2.8}$$

where $f : (\mathbb{R}^n)^q \times \mathbb{R} \mapsto \mathbb{R}^n$.

Because it is always possible to rewrite a high-order differential equations to the first-order form, we normally choose to treat only the first-order ODE without loss of generality. The first-order form can be obtained by proper definition of a new vector

$$Y = (y, \dot{y}, \dots, y^{q-1}) \ .$$

Based on this new vector Y, we can transform the high-order form (2.8) into a first-order form

$$\dot{Y} = F(Y, t) \ .$$

2.2.1 Initial value problem for ODEs

The typical use of a dynamical model consisting of ODEs is to define the state of the system $y_0 \in \mathbb{R}^n$ at a certain point in time t_0, and then use the model for finding the behaviour of the system in a time interval around the time t_0. As we have learned, this is called solving an *initial value problem*.

We formally state the initial value problem for ODEs by the following: Find a function $y(t)$ (and an interval $D \subseteq \mathbb{R}$) so that

$$\begin{aligned} \dot{y} &= f(y,t), \qquad \text{for } t \in D \\ y(t_0) &= y_0 \end{aligned} \tag{2.9}$$

We do not normally consider the choice of initial values y_0 as a problem when working with ODEs. Mathematically, the ODE only require that we choose an initial values such that $f(y,t)$ is defined. This will be obvious from the existence and uniqueness theorem in the next section.

Recall, that in technical discussions, using the term *simulation* or *dynamical simulation* often refers to the process of solving initial value problems like (2.9).

2.2.2 Existence and uniqueness

The question one can ask is if the IVP (2.9) has a solution and furthermore a unique solution for $y(t)$. This question can be partly answered by various versions of a theorem stating *sufficient conditions* for existence and uniqueness of a solution.

The existence and uniqueness theorem presented here is a combination of the theorems in [Lam91, p. 7] and [HWN93, p. 37-39]. This combination is done to include both systems of equations and a definition based on a possibly limited subset of $\mathbb{R}^n \times \mathbb{R}$.

Theorem 2.2 (Existence and uniqueness) *Assume an ordinary differential equation*

$$\dot{y} = f(y,t) \tag{2.10}$$

with $f(y,t) : D \mapsto \mathbb{R}^n$, where D is an open subset of $\mathbb{R}^n \times \mathbb{R}$, f is continuous for all (y,t) in D. Furthermore there exist a constant L, such that f satisfies a Lipschitz condition

$$\|f(y_1,t) - f(y_2,t)\| \le L\|y_1 - y_2\|$$

for every $(y_1,t),(y_2,t) \in D$.

Then for every $(y_0,t_0) \in D$ there exist a unique solution $y(t)$ of (2.10)*, with $y(t_0) = y_0$, where $y(t)$ is a continuous and differentiable function of $t \in U$, where U is an interval such that $(y(t),t) \in D$ for $t \in U$.*

The apparent thing to learn from this theorem, is that if f is a "nice" function, then every initial value (t_0, y_0), will have exactly one unique solution $y(t)$.

This definition of "nice" refers to the Lipschitz criterion: As [Lam91] explains it, the Lipschitz criterion is a harder requirement than just continuity of f, but less than demanding differentiability of f.

2.2.3 Continuity and smoothness

From a modelling perspective it can be interesting to realise the smoothness properties of the solutions to the IVP. The ODE model is based on the relation

$$y(t) = \int_{t_0}^{t} \dot{y}(\tau) d\tau + y_0$$

This essentially requires that $\dot{y}(t)$ is integrable. This means that it must be bounded and must be continuous except at a countable number of points. See e.g. [Rud76].

It follows the that $y(t)$ is continuous, since it is an integral. Furthermore, $y(t)$ will be differentiable at every point where $\dot{y}(t)$ is continuous.

In Theorem 2.2 it is assumed that f is at least continuous. We shall loosely motivate that some kind of discontinuities of f can be allowed and still keep the existence and uniqueness property of the ODE.

We first notice that, assuming a solution $y(t)$, then $f(y(t), t)$ should be an integrable function. This means that if we are sure that $f(y(t), t)$ is bounded and will only have a finite number of discontinuities, then we can split the time interval into small intervals where f fulfills the conditions of the Existence and uniqueness theorem.

Handling discontinuities is highly relevant for practical modelling situations, but is not central for this thesis.

2.2.4 Sensitivity to perturbations

The consequences of perturbing an ODE system will now be described: Assume that we want to solve an initial value problem defined by

$$\dot{y} = f(y, t), \qquad y(t_0) = y_0 \; . \tag{2.11}$$

Now we define a perturbed initial value problem by adding a perturbation of the derivative expression and the initial values:

$$\dot{\hat{y}} = f(\hat{y}, t) + \delta(t), \qquad \hat{y}(t_0) = y_0 + \Delta_0 \; . \tag{2.12}$$

Then an upper bound on the difference between the two solutions y and $\hat{y}$ is given by the following theorem:

DEFINITION 2.3 (PERTURBATION OF ODES) *Assume that $y(t)$ and $\hat{y}(t)$ are solutions of* (2.11) *and* (2.12) *respectively, then for $t > t_0$ the difference can be bounded by*

$$||\hat{y}(t) - y(t)|| \leq e^{L(t-t_0)} \left(||\Delta_0|| + \max_{t_0 \leq \tilde{t} \leq t} || \int_{t_0}^{\tilde{t}} \delta(\tau) d\tau || \right) ,$$

where L is the Lipschitz constant for f.

This theorem is presented in e.g. [ESF98, p. 144].

The important thing to notice from this theorem is that, loosely speaking, the consequence of perturbing an ODE, has effect on the solution $y(t)$, only depending on the initial value perturbation, the stability of the system and then the *integral* (and not the size) of the perturbations of the derivatives. This will be important later when characterizing the difference between ODEs and DAEs.

2.3 Numerical methods for ODEs

As mentioned, dynamical models of ODEs are often investigated by solving one or several initial value problems (IVPs). Although some ODEs can be solved analytically, it cannot be done in general. Therefore the general approach, and the approach in this thesis, is to solve the IVP using numerical methods.

This introduction to numerical methods for ODEs is very brief. It only serves to state the basic ideas. A textbook giving both introductory and general treatment is [Lam91]. A more theoretical overview can e.g. be found in [HWN93] and [HW96].

2.3.1 Discretization

Numerical solution of IVPs for ODEs use the principle of *discretization*. This means that one considers only discrete values of the independent variable, e.g. $t = t_0, t_1, t_2, \ldots, t_n$, often with $t_0 < t_1 < ... < t_n$.

Given an initial value $y_0 = y(t_0)$, then the numerical methods result in approximations to the solution at the discrete time points:

$$y_i \approx y(t_i) \qquad i = 1, 2, \ldots, n.$$

These approximations are successively computed using a *discretization formula*. These methods are divided into: The *one-step* formulas

$$y_i = \Phi_1(f)(t_i, (t_{i-1}, y_{i-1})) \tag{2.13a}$$

and the *multi-step* formulas (k-steps)

$$\begin{aligned} y_i = \Phi_k(f)(t_i,&(t_{i-1}, y_{i-1}), \\ &(t_{i-2}, y_{i-2}), \\ &\quad\vdots \\ &(t_{i-k}, y_{i-k})) \end{aligned} \tag{2.13b}$$

Here f denotes that the methods use the ODE. This notation assumes that the discretization points t_i are given. As can be seen, the formulas can advance the initial value y_0 step by step, generating solutions y_1, y_2, etc. at the following time points. Multi-step methods might need special start-up procedures.

2.3.2 The Euler methods

Two of the most elementary methods are the *explicit Euler method* and the *implicit Euler method*. These methods are presented since they are ideal for describing the problems of trying to adapt traditional methods for ODEs to DAEs.

We state the stepping formulas for both methods. A step using the explicit Euler method is done by the formula

$$y_i = y_{i-1} + hf(y_{i-1}, t_{i-1}) \ . \tag{2.14}$$

Here h denotes the step size $h = t_i - t_{i-1}$. We see that it is easy to compute the next solution point (t_i, y_i) from the previous point.

Alternatively, the *implicit* Euler method uses a stepping formula

$$y_i = y_{i-1} + hf(y_i, t_i) \tag{2.15}$$

Again the formula looks simple, but we realise that if the function f is the least complicated this formula does not state a direct explicit formula for the next solution point. The next solution point is *implicitly* given by the formula.

2.3.3 Accuracy

The general idea of using discretization, is that the smaller steps one takes, the more accurate the computed solution points will be, compared to the exact solution. The following concerns the consequences of using a discretization scheme instead of solving the ODE analytically.

For the purpose of analyzing the accuracy we define the *local error*. Local error describes the error committed in a single step. For step n this could look as

$$\begin{aligned} y_{n-1}(t_{n-1}) &= y_{n-1} \\ LE_n &= y_{n-1}(t_n) - y_n, \end{aligned} \tag{2.16}$$

where $y_{n-1}(t)$ denotes the exact solution of the ODE, going "through" the previous solution point y_{n-1}. One can say that we for a moment assume that the previous solution point is exact. This means that the error we measure by LE_n is only the error committed in the last step. This definition of local error "works" best for one-step methods. For multi-step methods the definition needs some comments. See e.g. [HWN93, p. 368].

The local error can often be separated in a part originating from the *local truncation error* (LTE) (a term related to the Taylor expansion), which is the error caused by using the dicretisation scheme assuming exact computations, and a part called *round-off errors* (LRE) caused by finite precision computations: $LE_n = LTE_n + LRE_n$.

Methods are typically characterised by the limiting behaviour of the local truncation error as the step size decreases. The local truncation error is of order $p+1$, if

$$||LTE_n|| = O(h_n^{p+1}) \tag{2.17}$$

where $h_n = t_n - t_{n-1}$ is the *step-length* of the n'th step, and $||\cdot||$ is some norm. Formulas for the LTE are obtainable for most methods and from this the order of the local truncation error can be determined. As example, the order of the local truncation error for both Euler methods is 2.

If the IVP is solved over an interval $[t_0, t_n]$ then it becomes interesting to examine the accuracy of the numerical solution obtained at the point t_n, now only accepting the initial value (t_0, y_0) to be exact. For this we define the *global error*:

$$\begin{aligned} y(t_0) &= y_0 \\ GE_n &= y(t_n) - y_n, \end{aligned} \tag{2.18}$$

It is possible to obtain some bounds on the global error: The global error is basically the accumulation of the local errors, but to this comes the important fact, that the stability of the system can cause "amplification" of the local errors committed.

If we consider only exact computations, then the global error is called the *global truncation error* (GTE) since only truncation errors, and no round-off errors are done locally. A necessary criterion for a numerical discretization formula to be useful, is that this global truncation error can converge to zero.

We say that a method is *convergent of order p* if the *global (truncation) error* behaves like

$$||GTE_n|| = O(h^p) \quad \text{for } h \to 0, \tag{2.19}$$

where we assume a fixed step size $h = (t_n - t_0)/n$.

From [HWN93, p. 160] we get bounds on the global *truncation* error assuming we have local truncation error $p+1$, i.e. $||LTE|| = Ch^{p+1}$:

$$||GTE_n|| \leq h^p \frac{C}{L} \left(\exp(L|t_n - t_0|) - 1\right). \tag{2.20}$$

Here $L \geq ||\partial f/\partial y||$. We notice that the order of the global truncation error is p, one order lower than the order of the local truncation error. As example, the Euler methods are both convergent of order 1.

Generally, the obtainable bounds on the global error are much too pessimistic, compared to practical experiences.

2.3.4 Linear multi-step methods

A general class of methods are the *linear multi-step methods.* A k-step method (with fixed step-size) has the general form[1]

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(y_{n+j}, t_{n+j}). \tag{2.21}$$

A specific method is obtained by choosing the vectors α and β. If $\beta_k = 0$ then the next step y_{n+k} can be computed explicitly, i.e. the method is *explicit*, otherwise the method is *implicit.* The Euler methods are possibly the simplest examples of linear multi-step methods.

The convergence order of a linear multi-step method can be determined from the parameters α and β. A description of this can be found in e.g. [Lam91, p. 48].

[1] Note that the use of indices is slightly different compared to the general multi-step formula (2.13b)

A special subclass of the linear multi-step methods are the *Backward Difference Formulae (BDF)* methods. Especially the *implicit* versions have had success in solving special problems, including DAE systems as will be discussed later. The general implicit BDF method is defined by

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\beta_k f(y_{n+k}, t_{n+k}). \tag{2.22}$$

Note that the implicit Euler method is an implicit BDF method.

For more details on linear multi-step methods see e.g. [Lam91]. A treatment of the general linear multi-step methods with variable step-size, can be found in [Sjö99].

2.3.5 Runge-Kutta methods

Runge-Kutta methods are an important class of one-step methods. The basic principle is to introduce *internal stages* $Y_i, i = 1, 2, \ldots, s$ for each step. These stages are defined by

$$Y_i = y_n + h\sum_{j=1}^{s} a_{ij} f(Y_j, t_n + hc_j) \qquad , i = 1, 2, \ldots, s \tag{2.23}$$

and the next step y_{n+1} is defined by

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{j=1}^{s} b_j f(Y_j, t_n + hc_j) \\ t_{n+1} &= t_n + h \end{aligned} \tag{2.24}$$

This form denotes an s-stage Runge-Kutta method. The parameters of the method are the coefficient matrix A, containing the a_{ij}'s, and the vectors b and c. The final stage y_{n+1} of the method, does not have to be treated separately by the b vector, but can for simplicity of the representation be included as a row of A.

Runge-Kutta methods can form both explicit and implicit systems of equations, dependent on the choice of A. This defines two classes of Runge-Kutta methods, the *explicit* and *implicit* methods.

The Runge-Kutta methods are also characterized by their *order*, denoting the order of the local error. The order can be determined by the parameters. A detailed description can be found in e.g. [Lam91]. Here several sets of parameters are described for specific Runge-Kutta methods as e.g. the DOPRI54 method, an example of a very popular explicit Runge-Kutta method.

The Runge-Kutta methods are quite popular. Especially they constitute high-order one-step methods, which are appropriate for e.g. ODE models with state-events since there is no need for a special restart procedure. The Euler methods are as well as being the examples of linear multi-step methods also the simplest examples of Runge-Kutta methods.

Later it will be commented that Runge-Kutta methods have not been very popular for solving DAE problems, because they are not necessarily convergent for high-index DAEs. The methodology for treating DAE systems described in this thesis, reintroduce Runge-Kutta methods for DAEs (using their augmented underlying ODE).

2.3.6 Error and step-size control

As a part of a practical implementation of a numerical method, an important feature is the *automatic* or *adaptive* step-size control. Solving the IVP numerically, it is normally convenient that the user does not have to choose the discretization points, i.e. the step-sizes.

In practice the step-size control is based on controlling the local error. Since we do not want to compute the exact local error, an estimate is often used. As a general principle for error estimation one could use e.g. Richardson extrapolation, but often discretization formulas have accompanying formulas for error estimates, which can be computed for little extra cost.

As example, the Runge-Kutta methods can be extended with a so-called *embedded* error estimate. This estimate is in principle built on an extra formula for the end-stage

$$\tilde{y}_{n+1} = y_n + h \sum_{j=1}^{s} \tilde{b}_j f(Y_j, t_n + hc_j), \tag{2.25}$$

and the error estimate is obtained by

$$LEST = y_{n+1} - \tilde{y}_{n+1}. \tag{2.26}$$

If the objective is to keep the local error lower than some error tolerance E_{TOL}, one has to change step-size when advancing the computed steps, only accepting the next step if the error estimate is below the tolerance.

Studying the optimal choice of step-size, is a central research topic. Sometimes one regulates the step-size by halving or doubling the step-size, but the classical approach, uses knowledge of the order of the local truncation error:

$$h_{new} = h \sqrt[p+1]{\frac{0.8E_{TOL}}{LEST}}$$

The factor 0.8 is intended to ensure that the new step-size will actually result in a local error below the tolerance.

A popular modern approach to choosing step-size is primarily due to the work in [Gus92]. Here the step-size choice is viewed as a control problem, and it is found that a step-size choice, can be calculated using a strategy inspired from a PI-controller. A more recent publication based on this is [GS97].

2.4 Implicit functions and non-linear equations

The topic of general implicit DAEs is dependent on some notions and results from the general treatment of non-linear equations and implicit functions.

In the following we introduce some terminology and discuss some of the basic problems of using non-linear equations and implicit functions. We also discuss the concept of *regularity*, as a property of certain well-behaving models.

2.4.1 Equations on residual form

An equation system based on real value expressions and variables can be transformed and represented on the *residual form*:

$$f(x) = 0 \qquad , \tag{2.27}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ with $n, m \geq 1$. At this point we make no assumption of an equal number of equations m and variables n. x can be limited to a domain $x \in D \subseteq \mathbb{R}^n$.

We generally assume that f is continuously differentiable. This property has special importance for some of the definitions and theorems in the following sections. In this context we define the *Jacobian matrix* as the matrix:

$$J(x) = \frac{\partial f(x)}{\partial x}$$

This is an $m \times n$-matrix being a function of x.

2.4.2 Consistency and determinedness

To such a general non-linear system, there might exist no, one, several or infinitely many solutions (manifolds of solutions), both for $n = m$ and $n \neq m$.

Generally, we expect every sensible model and corresponding equation system to have at least one solution:

Definition 2.4 (consistency) *We say that an equations system (or model), represented by* (2.27) *is* consistent *if there exists a solution $x \in D$ to the equation system.*

In this way consistency, so to speak, denotes a sensible model.

We do not have a set of sufficient conditions for determining consistency of a general non-linear equation system. Often one must work with the assumption that a solution exists, which is "later" proven by example, when and if a solution is found.

The next question could be, how many solutions exists? For a large class of modelling problems we expect the model to have exactly one solution:

Definition 2.5 (determinedness) *We say that an equations system (or model), represented by* (2.27) *is* determined *if there exists a unique solution $x \in D$ to the equation system.*

Quite natural, it is only sensible to speak of a determined equation system if a solution exists, i.e. the system is consistent.

An equation system which does not specify a unique solution, is often referred to as being *under-determined*. This is especially the cases when an infinity of solutions exist, represented by a manifold in $\mathbb{R}^n$. The notion is typically not used for the situation where an equations system have a number of isolated solutions x_i, $i = 1, \dots$.

The notion of *over-determinedness* can be used to explain the modelling situation that some of the equations are either superfluous in specifying the solution set, or that the equations are collectively *in-consistent*.

The terms consistency and determinedness can often be used interchangeably with the terms *existence* and *uniqueness* of a solution, respectively. It seems, the latter notions are often used in more mathematical discussions, whereas consistency and determinedness are used more in modelling discussions.

2.4.3 Implicit functions

Often, the intention of an equation system is to define $x \in R^n$ as an implicit functions of $y \in \mathbb{R}^m$, using

$$f(x, y) = 0 \ , \tag{2.28}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$. We can also say that x represent the *dependent* and y represent the *independent* variables of the equation system.

The problem is whether a concrete equation system really defines an implicit function for x, i.e. that there exist *one unique x for every y*?

The *implicit function theorem* gives sufficient conditions for such a function to exist (locally), and further more claims some smoothness properties of the implicit function. See e.g. [Rud76].

Theorem 2.6 (Implicit function theorem) *Let $f(x, y) \in C^1$ be a continuously differentiable mapping of an open set $E \subset \mathbb{R}^n \times \mathbb{R}^m$ into $\mathbb{R}^n$. Assume that for some point $(x^*, y^*) \in E$,*

$$f(x^*, y^*) = 0 \ .$$

Let the Jacobian matrix $J_x(x, y) = \frac{\partial f(x,y)}{\partial x}$ and $J_y(x, y) = \frac{\partial f(x,y)}{\partial y}$ and further assume that

$$J_x(x^*, y^*) \text{ is invertible.}$$

Then there exist neighbourhoods U of x^* and W of y^* having the following property:

To every $y \in W$ corresponds a unique $x \in U$ such that $f(x, y) = 0$. If this x is defined to be $g(y)$, i.e.

$$f(g(y), y) = 0, \qquad \text{for } y \in W$$

then $g(y) \in C^1 : W \mapsto U$ is a continuously differentiable function with derivative (Jacobian matrix)

$$\frac{\partial g(y^*)}{\partial y} = J_x(x^*, y^*)^{-1} J_y(x^*, y^*) \ .$$

From a modelling perspective, the theorem explains that we can apply a regularity condition for $J_x(x, y)$, if we want to be sure that, at least locally, the model defines a variable x as a function of y. This assuming that the model is consistent.

Note, that at this point we assume the number of equations to match the dimension of x, since regularity is only defined for a quadratic matrix $J_x(x)$.

We end the discussion of implicit functions, by mentioning a simpler class of implicit functions. By a *linearly implicit function* we understand an implicit equation system

$$J(y)x - f(y) = 0 \; . \tag{2.29}$$

For this class of implicit functions, in contrast to the general non-linear case, the conditions for both existence and uniqueness of a solution x as a function of y becomes quite simple: If there exist a set D such that, $J(y)$ is regular for all $y \in D$, then for given $y \in D$, there exist a solution x given uniquely by

$$x = J(y)^{-1} f(y) \; .$$

where $J(y)^{-1}$ denotes the inverse of $J(y)$. Obviously, determining x is a linear algebra problem, more than a non-linear equation problem.

2.4.4 Newton method for non-linear equations and implicit functions

It is well-known that solving non-linear equations, i.e. evaluating implicit functions is in practice done by numerical methods. We will shortly state the Newton method which is the important basis for many of the numerical methods. For generality it is sufficient to consider the case of the implicit function.

We assume that we have an implicit function as (2.28) and that y^* is given, and we have an initial guess x_0 for the exact value x^* of the implicit function, i.e.

$$f(x^*, y^*) = 0 \; .$$

Newton iterations are defines as the iteration

$$x_i = x_{i-1} - J_x^{-1}(x_{i-1}, y^*) f(x_{i-1}, y^*) \; , \quad i = 1, 2, \ldots \tag{2.30}$$

where $J_X^{-1}(x_{i-1}, y^*)$ denotes the inverse of the Jacobian matrix and $f(x_{i-1}, y^*)$ the residual evaluated in the point (x_{i-1}, y^*).

What is important is, that if the Jacobian matrix $J(x, y)$ is regular at the solution point (just as the implicit function theorem requires) then there exist a neighbourhood U around x^*, such that if the initial guess x_0 is chosen from this neighbourhood, then Newton iterations $x_1, x_2, \ldots$ *will* converge towards the solution x^*, in addition, with at least quadratic convergence. Note also that the linearly implicit function will be solved in one step.

2.4.5 Regularity - a model quality

The Theorem 2.6 is only a sufficient condition for the implicit function to exist, therefore implicit functions exist which does not fulfill the regularity condition. The following section is intended to motivate that regularity of an equation system is still a reasonable requirement, since it also implies other relevant qualities of the model.

We have already seen a practical and very important quality of regularity: That we can use Newton iterations and that these will give convergence if we start close enough to the solution.

Next, the implicit function $x = g(y)$ is stable to small perturbations δ_y of y and small perturbations δ_f of the equations

$$f(x, y + \delta_y) + \delta_f = 0 ,$$

since according to the implicit function theorem, we have learned that the implicit function is continuously differentiable, and we can consider δ_y and δ_f as input to the implicit function analogous to y, and therefore we can locally obtain bounds on the implication of these perturbations.

We noted that some fundamental qualities characterizes the regular implicit function:

- Uniqueness of solution. (locally)
- Convergence of Newton iterations.
- Solution is stable to perturbations of equations.
- Solution is stable to perturbation of input.

We remember that by regularity we do not discuss consistency / existence of a solution.

As seen, these are all important qualities of a model used in practical computations. Especially when considering finite precision computations these properties are important, since round-off errors might be considered as perturbations of the input/equations.

When moving on to discussing implicit ODEs and especially DAEs, it seems appropriate to find similar conditions on these models, and use the notion of *regularity* to denote a model which has properties similar to the fundamental properties mentioned above.

2.4.6 Under-determined equation system

We will have a short look at methods for the under-determined equation system, since these problems often arise in practical model situations. This is the case e.g. when finding special initial values for ODE systems, but as will be seen later in the thesis, it is the case when finding the so-called *consistent initial values* for DAE systems.

The under-determined equation system have the form

$$f(X) = 0 \tag{2.31}$$

Here $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, $X \in \mathbb{R}^n$ where $n > m$. Generally, we will also assume that the rank of this system is full, i.e. that for general values of X, then

$$\text{rank } J_X(X) = n \ .$$

We will here shortly describe methods for handling the under-determined system and define a solution. The methods are briefly the following:

- Choosing dependent/independent (output/input) variables.
- Adding equations, i.e. further constraints on initial value.
- Introduce objective function (preferences) for initial value.

These are all relevant in practical modelling situations, and they are described here since it is found reasonable to try designing modelling and simulation tools so that these methods can be used. Some of them can actually be used in the experimental tool Simpy developed in this project and described in chapter 8.

Choosing independent variables. One of the basic approaches to solving an under-determined system of equations is to separate the variables of X in *dependent variables* x and *independent variables* y

$$X = (x, y)$$

as would be the case of an implicit function. Therefore, with reference to the implicit function theorem, the two variable sets are sometimes referred to as the *output variables* and *input variables* respectively.

The corresponding requirement loosely states that apart form being a consistent equation system, we should pick the dependent variables x so that the Jacobian matrix

$$J(X) = \frac{\partial F}{\partial x} \tag{2.32}$$

is regular, at least around the solution. It is quite obvious that the number of output and input variables are n and $m - n$ respectively. Sometimes we refer to $m - n$ as the *degrees of freedom*.

When we use some kind of iteration to obtain the solution, we know that the regularity requirement also guarantees a neighbourhood of convergence towards the solution/output of the implicit function.

The problem of choosing dependent variables will be discussed later in the theses in the theory for the so called *dummy derivative method*. Here we have to define a principle for automatically determining the dependent variables.

Adding equations. Another possibility is to add more equations to the under-determined equation system, and thereby making the combined system determined:

$$f(X) = 0$$
$$h(X) = 0$$

It is obvious that these additional equations or constraints should not already be implied by the existing equations, and therefore one should concentrate on adding equations which further constrain the possible initial value. This means that each of the added equations should increase the rank of the Jacobian matrix, so that the Jacobian matrix of the final system is regular.

It is also obvious that we can choose only to add a few additional equations, and then if the system is still under-determined, we can combine this principle with e.g. the principle of choosing dependent/independent variables.

Introducing objective function. Yet another possibility is to add an objective function, or preference to the values. Generally this will result in a *constrained optimization problem* which is another general topic in mathematical modelling and numerical analysis.

The problem can mathematically formulated as:

$$\max_{X} \quad h(X) \tag{2.33}$$

subject to

$$f(X) = 0 \tag{2.34}$$

This means that one should find a value of X which maximizes the objective function $h(X)$ under the conditions that $f(X) = 0$.

We will not describe the numerical treatment of this problem further, but the reader can look in general textbooks on numerical analysis for further details.

2.5 Structural analysis of equation systems

Structural analysis in general concerns a high-level analysis of equation systems, essentially based on information of which variables are used in which equations. It is used for analysis of both linear and non-linear equation systems. Much work have been done in this field and results are used in practical modelling tools. A general reference on theory and methods is e.g. [DER86].

2.5.1 Representation of structural information

The general equation systems considered in structural analysis can be represented in the residual form

$$f(x) = 0, \qquad f : \mathbb{R}^n \mapsto \mathbb{R}^m \tag{2.35}$$

The vector function $f(x)$ represents m equations/residuals based on n variables represented by x. We shall mostly use structural analysis for systems where $m = n$, and we typically assume that $f(x)$ is continuous and differentiable w.r.t x.

We want information on which equations depend on which variables. For this purpose the Jacobian matrix of the system can be used:

$$J(x) = \frac{\partial f}{\partial x}$$

Considering equation i and variable j, if we look at the corresponding element of the Jacobian, we note that if and only if the element is *not* identically zero, i.e.

$$J(x)_{i,j} = \frac{\partial f_i}{\partial x_j} \neq 0, \quad \text{for any } x,$$

then equation i depends on variable j. Remember that we assumed continuity and differentiability.

We can represent this dependency information in an $m \times n$-matrix S containing only the values 1 or 0 for structural non-zero and zero elements respectively:

$$S_{i,j} = \begin{cases} 1 & \text{if } J(x)_{i,j} \neq 0 \\ 0 & \text{if } J(x)_{i,j} = 0 \end{cases} \tag{2.36}$$

Often, we do not need to compute the elements of the Jacobian matrix symbolically to derive the elements of S. They can e.g. be constructed by simply "looking" at the equations.

As graphical representation [DER86] uses a matrix-like representation, where 1's are represented as $\times$'s, and the 0 are left out as blanks.

The following example illustrates how the structural information can be obtained for an equation system:

■ **Example 2.1 (Pendulum sys. - structural information)**
A system of equations already presented is the pendulum system from Example 1.1. Reordered to the residual form (2.35) the equations are:

$$\begin{aligned} M\ddot{x} + F\frac{x}{L} &= 0 \\ M\ddot{y} + F\frac{y}{L} + Mg &= 0 \\ x^2 + y^2 - L^2 &= 0 \end{aligned} \tag{2.37}$$

We will see how the structural information is obtained for this system.

We first compute the Jacobian matrix w.r.t. all the variables used in the model.

$$\begin{array}{c}
\begin{array}{ccccccc} \ddot{x} & \ddot{y} & \dot{x} & \dot{y} & x & y & F \end{array}\\
\left[\begin{array}{cc|cc|ccc} M & & & & F/L & & x/L \\ & M & & & & F/L & y/L \\ & & & & 2x & 2y & \end{array}\right]
\end{array} \tag{2.38}$$

The Jacobian matrix is ordered such that the ordering of the rows correspond to the ordering of the rows in (2.37). Blanks correspond to zeros.

The corresponding structural information can be graphically represented by the matrix

$$\begin{array}{c}
\begin{array}{ccccccc} \ddot{x} & \ddot{y} & \dot{x} & \dot{y} & x & y & F \end{array}\\
\left[\begin{array}{cc|cc|ccc} \times & & & & \times & & \times \\ & \times & & & & \times & \times \\ & & & & \times & \times & \end{array}\right]
\end{array} \tag{2.39}$$

Note that in general we do not need to compute the Jacobian matrix to obtain the structural information. It can fairly easy be determined if an equation depends on a variable just by looking at the equation.

We note that there are more variables than equations. It could seem that the equation system is under-determined. As we shall discuss later, what is really interesting for a DAE/ODE system like (2.37) is, if one can solve for $\ddot{x}, \ddot{y}, F$ as an implicit function of $\dot{x}, \dot{y}, x, y$ and t.

For this purpose the interesting parts of the Jacobian matrix (2.38) and therefore the structural representation (2.39) are the $\ddot{x}$-, $\ddot{y}$- and F-columns:

$$\begin{array}{c}
\begin{array}{ccc} \ddot{x} & \ddot{y} & F \end{array}\\
\left[\begin{array}{ccc} \times & & \times \\ & \times & \times \\ & & \end{array}\right]
\end{array} \tag{2.40}$$

It is simple to see, that no Jacobian matrix with this structure can be regular, and therefore the system does not define $\ddot{x}, \ddot{y}, F$ as an implicit function of $\dot{x}, \dot{y}, x, y$ and t. This makes the system (2.37) a DAE system in contrast to a traditional (implicit) ODE. ■

For computations, numerous representation schemes can be used. The choice depends on what information is needed, and trying to optimize queries which need efficiency. The trivial choice could be an $m \times n$ array of integers.

2.5.2 Short application overview

Structural analysis is primarily used for so-called *sparse systems*. This refers to the sparsity of non-zero elements in the Jacobian matrix.

Structural analysis can be applied to both linear systems and non-linear systems. Since the analysis depends only the zero/non-zero property of the Jacobian elements, structural analysis is often done in an *initial analysis* of the system, and the results can thereafter be reused for particular values of the Jacobian matrix.

For *linear equations systems*, structural analysis can e.g. result in a non-trivial reordering of the rows and columns of the matrix such that a factorization of the matrix, to some extend, preserves or utilises the sparse structure of the problem. Advantages can regard both size of representation and efficiency of computations.

For *non-linear equation systems*, structural analysis can also be applied in order to obtain an ordering of the equations, which can also be utilised in the numerical solution process. Often the analysis will result in a *blocking* of equations and variables, representing a certain coupling in the equation system or causality of the underlying model. This blocking can have the *bordered block diagonal form* and the *block triangular form*. See [DER86].

Structural analysis of equations is used in a number of practical modelling tools, see e.g. [MOE99] or [And94], which discusses use of structural analysis both for general performance reasons when solving non-linear equations, but also for analysis of DAEs as we will treat later.

2.5.3 Structural regularity and singularity

Based on the structural information we can define several properties of the equation system. For DAEs we are specially interested in detecting the properties *structural regularity* and *singularity*.

From the implicit function theorem, see section 2.4, we acknowledge the important criterion of regularity of the Jacobian matrix

$$J(x) = \frac{\partial f}{\partial x} \ .$$

Naturally, $J(x)$ should be an $n \times n$-matrix.

In structural analysis we are interested in defining a property similar to regularity/singularity of a matrix-function $J(x)$ based on the structural representation S of $J(x)$. We define structural regularity and singularity:

Definition 2.7 (Structurally regular/singular) *An $n \times n$ structural matrix S is called* structurally regular, *if the rows/columns can be permuted such that there are only* non-zero elements on the diagonal. *Otherwise, we call it* structurally singular.

These terms are also used for the corresponding matrix-function $J(x)$ and equation system $f(x) = 0$, if the corresponding structural matrix S is structurally regular/singular.

To motivate the definition, assume that we want to perform a Gauss elimination on the Jacobian $J(x)$. During this process we successively need non-zero elements to appear or be permuted onto the diagonal.

Structural singularity of S is a sufficient condition for this process to *fail*, and therefore a *sufficient condition for $J(x)$ to be singular* independent on specific values of x. This is captured in the following theorem:

Theorem 2.8 *Assume that S is the structural representation of a matrix $J(x)$. If S is structurally singular, then $J(x)$ is (algebraically) singular for all values of x.*

The proof can be found in Appendix A.

We note that structural singularity is a *sufficient condition* for (algebraic) singularity of $J(x)$, while structural regularity is a *necessary condition* for (algebraic) regularity of $J(x)$.

Structural regularity/singularity can also be related to the *assignment problem*: The problem is to uniquely assign variables to equations. The assignment of a variable j to an equation i requires that the variable *appears* in the equation, i.e. $S_{i,j} = 1$. If all variables can be assigned an individual equation, i.e. one can make a *full assignment*, then the system is structurally regular. Note that a full non-zero diagonal actually represents a full assignment and vice versa!

■ **Example 2.2 (Pendulum sys. - structural singularity)**
We can now continue example 2.1. In the end of the example it was discussed whether the Jacobian can be regular, when the structural representation (2.40) *is as it is.*

Now we understand that this system is exactly structurally singular, since we cannot permute non-zeros onto the entire diagonal:

$$\begin{array}{c} \begin{array}{ccc} \ddot{x} & \ddot{y} & F \end{array} \\ \begin{bmatrix} (\times) & & \times \\ & (\times) & \times \\ & & (?) \end{bmatrix} \end{array} \tag{2.41}$$

Although this example might be somewhat trivial, we can now use Theorem 2.8 to conclude that every Jacobian of the system will be singular.

■

2.5.4 Properties of subsets of equations

For structural analysis of DAEs, presented later, we are interested in identifying special subsets of the equation systems, with special structural properties, which are relevant in the process of *index reduction*.

In the following, note the terminology used: When a subset of equations *has variables*, then it simply means *all* the variables on which the equations depend.

We define the first property:

Definition 2.9 (Structurally singular subset) *A subset of equations is* structurally singular *if it has more equations than variables.*

Normally, we consider such an equation system *over-determined*, since it has more equations than variables. Accordingly we could have chosen the name *structurally over-determined*, but we choose the same terminology as is used in e.g. [Pan88]. We should not confuse this term concerning a *subset* of equations, with the same term concerning the *entire* system from Definition 2.7.

Now we move on to defining a "minimal-size" property of a structurally singular subset of equations.

DEFINITION 2.10 (MINIMALLY STRUCTURALLY SINGULAR) *A subset of equations is* minimally structurally singular (MSS) *if it is structurally singular, and none of its proper subsets are structurally singular.*

As described in Appendix A, every structurally singular subset, contains an MSS subset. Understanding the minimal-size property is also helped by noticing the result in Lemma A.8 saying, that there is exactly one equation more than variables in an MSS subset.

Importantly, the MSS subset is the smallest subset of equations which implies structural singularity of the complete system. This is emphasized by the following theorem:

THEOREM 2.11 *An equation system is structurally singular, if and only if there exist a subset of equations which is minimally structurally singular.*

A proof can be found in the Appendix A.

Detecting MSS sets becomes important for DAE systems. Structural singularity is characteristic for many DAE systems and the MSS subsets are important for performing *index reduction* to obtain the *underlying ODE*.

Notice that in the Pendulum system, Example 2.2, there exist an MSS subset: The set containing only the last equation of the system, corresponding to the last (empty) row of the structural representation (2.41). This equation has *no* variables, and therefore more equations than variables, and is therefore structurally singular. Furthermore it has no proper subsets, and it is therefore *minimally structurally singular*.

2.5.5 Algorithm for detecting structural singularity

There are a number of algorithms for detecting structural regularity/singularity or, which is the same, *full assignments*. One of the classical algorithms is presented in [DER86] as the "depth-first search for transversal extension".

The version presented here is a slightly extended version of the general algorithm, since we also need it for detecting MSS subsets. Essentially this just means that we have to recall which variables are visited under the assignment search process. The algorithm presented in the following, is in this respect inspired from the algorithm presented in [Pan88].

The setup is as follows: We can assume that two sets E and V holds identifiers for the equations and variables respectively. These identifiers could simply be the integers $1, 2, \ldots, n$. The $n \times n$-matrix S holds the structural representation, such that $S(i,j) = 1$, if equation i depends on variable j, and $S(i,j) = 0$ otherwise.

Algorithm 2.1 For detecting structural regularity.

```
StructurallyRegular():
    Assign(:):=None
    for e ∈ E:
        InSearch(:):=False
        Result:=FindAssignment(e)
        if Result = False:
            return False

    return True
```

The top-level algorithm, Algorithm 2.1, for detecting structural regularity works by finding a variable assignment for the equations, one at a time, by the call of *FindAssignment*(e), Algorithm 2.2, each time without making previously assigned equations unassigned.

The n-array *Assign* continuously holds the information about current assignments: *Assign*(v) returns the equation to which variable v is assigned. If v is not assigned, then *Assign*(v) = *None*. (The *None*-value could be represented by a zero.) Naturally, the array should be *initialised* with *Assign*$(:) :=$ *None* before the first call of the *FindAssignment*-algorithm, when no assignments are done yet.

If the call returns *True*, an assignment has been found, and the assignment information has been stored in the *Assign*-array. If all equations can be assigned a variable by successive calls of *FindAssignment*(e), then the system is *structurally regular*, and there are no MSS subsets.

If the *FindAssignment* for some equation returns *False*, then no assignments could be found. Instead the equation will be part of an MSS subset which can be computed from information stored in the array *InSearch*, described later. It is important that the *InSearch* array is initialised to $InSearch(:) := False$ before every call of the *FindAssignment*-algorithm.

For efficiency, the *FindAssignment*-algorithm makes use of a "compressed" representation of S: We create an array *AssignCandidates* for which each element $AssignCandidates(e)$ is a set/list of the variables in equation e, i.e. the set of all variables which can legally be assigned to the equations e:

$$AssignCandidates(e) := \{v \in V \,|S(e,v) = 1\}$$

Algorithm 2.2 For successively finding assignments.

```
1:  FindAssignment(e):
2:      for v ∈ AssignCandidates(e):
3:          if Assign(v) = None:
4:              InSearch(v):=True
5:              Assign(v):=e
6:              return True
7:
8:      for v ∈ AssignCandidates(e):
9:          if not InSearch(v):
10:             InSearch(v):=True
11:             e2:=Assign(v)
12:             Result := FindAssignment(e2)
13:             if Result = True:
14:                 Assign(v):=e
15:                 return True
16:
17:     return False
```

Here follows a description of the *FindAssignment*-algorithm, Algorithm 2.2: Line 2-6 are sometimes referred to as the "look-ahead part": Here, the algorithm initially searches for a "cheap" variable assignment for e, i.e. if by chance a variable from $AssignCandidates(e)$ should be currently *unassigned*. If an unassigned variable v is found, then it is assigned to e, in line 5, and the algorithm returns with information that an assignment is found.

Otherwise the algorithm continues to line 8-15, the "depth-search part": Again the algorithm looks through all the potentially assignable variables from $AssignCandidates(e)$. This time it knows that all variables are already assigned. Therefore, for each variable v, the algorithm tries to find an alternative assignment for the equations e_2 which is currently assigned to variable v, so that variable v becomes available for assignment to the equation e. This search for alternative assignment is a *recursive* call $FindAssignment(e_2)$. The array $InSearch$ keeps track of which variables are already or currently being tried out as alternative assignment variables. If the recursive call returns and has found an alternative assignment for e_2, the algorithm can assign variable v to e instead, line 15, and return $True$. If no alternative assignments can be found for any v, no assignment can be found at all for equation e. (An assignment may be found for this equation, but not without doing previously assigned equations unassigned.) For further discussion of the algorithm see [DER86] or general texts on e.g. the assignment problem.

This special version of the *FindAssignment*-algorithm can be used for detecting an MSS set: If no assignment can be found for an equation e, i.e. the call $FindAssignment(e)$ returns $False$, then the MSS set can be found by

$$\begin{aligned} V_{MSS} &= \{v \in V | InSearch(v)\} \\ E_{MSS} &= \{e\} \cup Assign(V_{MSS}) \end{aligned} \tag{2.42}$$

where V_{MSS} and E_{MSS} is the set of variables and equations of the MSS subset respectively. For proof see [Pan88].

It is possible to derive (pessimistic) bounds on the computational cost of the algorithm:

Each call or recursive call of the algorithm searches for an assignment or alternative assignment for a specific equation e. The computational work in each call, when ignoring the recursive calls, is due to the two (interruptible) loops over $AssignCandidates(e)$, and therefore an upper bound on the work is proportional to the number of variables in the specific equation. Next, the algorithm can only be called once, recursively or not, for each equation. This is because of the $InSearch$-check on variables.

This means that a bound on the computational work must be the sum of the work bounds for each equation-call. In total this is proportional to the total number of entries τ in the structural representation S. If we search for a full assignment, we have to call the algorithm once for each equation, and therefore the bound is proportional to

$$\tau n \ .$$

Often the algorithm performs much better than this bound indicates. See [DER86].

Chapter 3

Differential algebraic equations

This chapter introduces the characteristics of differential algebraic equations (DAEs), and introduces the problems concerned using and analysing these.

To understand the characteristics of DAEs, it important to have a basic understanding of ordinary differential equations, and numerical methods for these. It is also relevant to have an understanding of basic theory for non-linear equations and implicit function, and how to solve/evaluate these. These topics are treated as background material in Chapter 2.

3.1 General implicit DAE

The introduction offsets by defining the *general implicit DAE systems*

$$f(\dot{y}, y, t) = 0 \tag{3.1}$$

where $f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$, and y and $\dot{y}$ denotes functions of t. This is traditionally considered the most general form used for describing any DAE system.

Note that (3.1) is the general *first-order* form. The general *high-order form* looks as

$$f(y^{(n)}, y^{(n-1)}, \dots, \dot{y}, y, t) = 0. \tag{3.2}$$

As for ODEs, it is simple to rewrite all high-order systems to a corresponding first-order system. In this thesis, it is not generally recommended that the modeller should reformulate the problem to the first-order form, since we might use information from the high-order form. A discussion follows in section 5.6.6, but in much of the theory we will focus on the first-order form without loss of generality.

The differential equation system (3.1) can in the simplest case be much like an ODE presented in section 2.2 - in this case we call it an *implicit ODE*. Clearly, this is not always the case.

The theory for DAEs explains that even though (3.1) *is not* an (implicit) ODE, see section 3.3, there might very well be a unique solution for the initial value problem defined by (3.1) and

$$y(t_0) = y_0 \ ,$$

as would be expected from an ODE. We refer to such a system, as a *proper* DAE. Methods for finding these solution are the central topic of this thesis.

A seldomly discussed topic is how a modeller should interpret the general forms and assign model variables to the vectors y, $\dot{y}$, etc. An approach recommended and emphasized in this thesis is described in chapter 4.

3.2 Introductory example

As an introduction we treat a simple but illustrative example of a DAE system. It also illustrates how DAEs in practise arise from a component oriented modelling of a system.

3.2.1 Example: 1-D mechanical two-mass system

We consider a mechanical system consisting of two masses coupled by a rod of fixed length, put on a frictionless table. Furthermore, one of the masses is influenced by an external force possibly varying over time. See figure 3.1.

We describe the position of the masses in only one dimension along an x-axis, and the positions represented by the variables x_1 and x_2 respectively. From Newton second law we can write up the equations of motion for each of the masses, i.e. for *each of the components*:

$$M_1\ddot{x}_1 = F \tag{3.3a}$$

$$M_2\ddot{x}_2 = -F + f(t). \tag{3.3b}$$

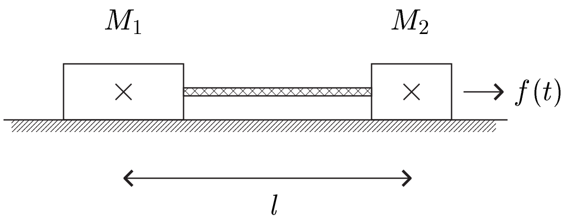


FIGURE 3.1: Illustration of a mechanical two-mass system coupled by a fixed-length rod.

Here M_1 and M_2 represent the respective masses, F represents the force acting between the masses, because of the connecting rod, and $f(t)$ represents the external force working on mass M_2.

We can change this into a first-order description, by introducing the velocity variables v_1 and v_2. We get

$$M_1\dot{v}_1 = F \tag{3.4a}$$
$$M_2\dot{v}_2 = -F + f(t) \tag{3.4b}$$
$$\dot{x}_1 = v_1 \tag{3.4c}$$
$$\dot{x}_2 = v_2. \tag{3.4d}$$

The system (3.4) looks almost as a classical ODE system, but we realise that the additional variable F is also an unknown, and as such the system can not be a classical ODE.

If we could just solve system (3.4) for the derivatives

$$\dot{v}_1, \dot{v}_2, \dot{x}_1, \dot{x}_2 \text{ and } F \tag{3.5}$$

as an implicit function of the base variables v_1, v_2, x_1, x_2 and t, the system would be an implicit ODE system. But we observe that the system, having five unknowns and only four equations, is under-determined.

3.2.2 Including constraints in the model

The clue lies in modelling the constraint on the system, i.e. that the masses should keep a fixed distance l corresponding to the fixed length of the rod. This constraint can be included in the model as

$$x_1 - x_2 = l. \tag{3.6}$$

By first sight, this seem like little help. It is not at all obvious that the combined system of (3.4) and (3.6) determines the desired variables: The new constraint do not include the variables which we were interested in solving for, i.e. the determinedness of the combined equation system, with respect to the variables (3.5), is not changed.

Actually, the model we get from combining (3.4) and (3.6) *is* well-determined, i.e. it *will* determine a unique solution over time given some initial conditions. The combined system of equations, (3.4) and (3.6), poses a solvable, well-determined differential algebraic equation system.

3.2.3 Consistent initial conditions

We continue the example by analysing how to choose the initial conditions.

From ODEs we can almost freely choose initial conditions, by defining initial values for the positions and velocities. But for this model we note that the equation (3.6) will impose some constraints on the variables x_1 and x_2.

The constraint on x_1 and x_2 is obvious. The constraint on v_1 and v_2 is less obvious. The "hidden" constraint can be found by differentiation of (3.6) with respect to time.

$$\begin{aligned} x_1 - x_2 = l \quad &\Rightarrow \quad \dot{x_1} - \dot{x_2} = 0 \\ &\Leftrightarrow \quad v_1 - v_2 = 0 \ . \end{aligned} \tag{3.7}$$

We substitute $\dot{x_1}$ and $\dot{x_2}$, using the equations (3.4c) and (3.4d). We now see the constraint on v_1 and v_2.

Next, we are to realise that there are also constraints on the choice of an initial value for F. This can be seen by continuing the differentiation of (3.7) with substitutions from (3.4a) and (3.4b):

$$\begin{aligned} v_1 - v_2 = 0 \quad &\Rightarrow \quad \dot{v_1} - \dot{v_2} = 0 \\ &\Leftrightarrow \quad (F/M_1) - (-F + f(t))/M_2 = 0 \\ &\Leftrightarrow \quad F = \tfrac{M_1}{M_1+M_2} f(t) \ . \end{aligned} \tag{3.8}$$

The conclusion is that the choice of initial values is non-trivial as opposed to normal ODE systems where initial conditions can be chosen almost freely. Therefore, the notion of *consistent initial values* or *consistent initial conditions* is important in the theory for DAEs.

Comment: The manipulations we have done in this example, are of a kind which most engineering and science students would have to do, in order to solve problems from elementary mechanics. It seems though that when we are presented with the system (3.4)*, out of context, the existence of hidden constraints and their importance is not so obvious.*

3.2.4 Underlying ODE

One of the simplest ways to recognise the well-determinedness of the model, is to use differentiation to reveal the so called *underlying ordinary differential equation.*

The differentiations we have performed in the previous section, has actually given us this underlying ODE. We gather the equations in

$$M_1\dot{v}_1 = F \tag{3.9a}$$

$$M_2\dot{v}_2 = -F + f(t) \tag{3.9b}$$

$$\dot{x}_1 = v_1 \tag{3.9c}$$

$$\dot{x}_2 = v_2 \tag{3.9d}$$

$$F = \tfrac{M_1}{M_1+M_2} f(t)\ . \tag{3.9e}$$

As seen, this system is solvable for all the variables (3.5) which had our interest. In fact, the system poses an explicit ODE, and as such we can analyse this system accordingly. We just have to remember the differentiated constraints on the initial values.

The above approach for analysing and solving DAEs seems very simple, but in general it is not as simple to find out how to perform the differentiations. Neither is the underlying ODE always the right system to solve, if one expects to obtain certain properties of the solution.

The thesis is concerned finding a general approaches for analysing and solving DAE systems, and define properties of the systems.

Comment: In some texts, system (3.9) *is not considered the underlying ODE. In these texts equation* (3.8) *is differentiated once more before it is combined with* (3.4).

$$\dot{F}_{rod} = \tfrac{M_1}{M_1+M_2} \dot{f}(t)\ . \tag{3.10}$$

We clearly see that this system is also an ODE, but in this thesis it is not recommended to generally perform this extra differentiation. This relate to a discussion which follows, regarding ambiguity of interpreting the general implicit DAE system.

3.3 Implicit ordinary differential equations

We will start the more theoretical discussion on the general implicit DAE. We consider the simplest case of

$$f(\dot{y}, y, t) = 0\ . \tag{3.11}$$

From the description of ODEs in section 2.2 we have learned that an ODE is characterised by having explicit expressions for the derivatives $\dot{y}$. At this point we will introduce the notion of an *implicit ODE*, for the case where the derivatives $\dot{y}$ are implicitly defined as a function of y and t by (3.11).

The potential problems, when introducing the implicit ODE, are consistency and determinedness of (3.11) just as we had from non-linear equations and implicit function described in section 2.4. Therefore we will generally assume that the system is consistent and determined

In the following, we will also assume that the implicit function theorem applies. This means that the Jacobian matrix $f_{\dot{y}}$ is regular in a neighbourhood of the solution. We call an implicit ODE which full-fills the implicit function theorem, *a regular implicit ODE* or just *regular ODE*[1].

This regularity of implicit ODEs has similar qualities as regularity had for non-linear equations since it also allows to bound the consequences of perturbations of the equations (3.11) corresponding to Theorem 2.3.

Naturally these notions of *implicit ODE* and regularity also applies to the high-order implicit forms as (3.2).

We will return to discuss how to discover that a specific implicit DAE model can in-fact be interpreted as an implicit ODE in chapter 4. This chapter describes the problem of assigning variables of a specific model to the representation vectors of (3.11).

3.3.1 Methods for implicit ODEs

How can we solve an initial value problem for the implicit ODE? We briefly sketch why the implicit ODE can be treated much like the explicit ODE, and be solved using traditional numerical methods for ODEs. We simply have to rewrite the step methods of section 2.3, to match the implicit form, and then check that the implicit form is still solvable. It is for a great deal possible to generalise the theory of explicit ODEs to the implicit ODE case.

We can e.g. rewrite the methods for explicit ODEs, such that every time $f(Y,T)$ appears[2], we replace the appearance with the dot-variable $\dot{Y}$, and introduce the implicit equations $f(\dot{Y},Y,T)=0$ for implicitly determining $\dot{Y}$. We will show some specific methods to verify this.

[1]Note that also explicit ODEs can be considered as regular ODEs

[2]Note that the f is not the same function. Just note the form.

Runge-Kutta methods. For the Runge-Kutta methods this could look as

$$\begin{aligned} Y_i &= y_n + h\sum_{j=1}^{s} a_{ij}\dot{Y}_j \\ 0 &= f(\dot{Y}_i, Y_i, t_n + hc_i) \end{aligned} \qquad i = 1,2,\ldots,s \tag{3.12}$$

We note that if $Y_i \in \mathbb{R}^n$, this generally constitute a non-linear equation system of $2sn$ simultaneous equations with $(\dot{Y}_i, Y_i)$, $i = 1,2,\ldots,s$ as the unknowns. We can choose to substitute Y_i in the equation system, halving the number of equations/unknowns:

$$0 = f(\dot{Y}_i, y_n + h\sum_{j=1}^{s} a_{ij}\dot{Y}_j, t_n + hc_i) \qquad i = 1,2,\ldots,s \tag{3.13}$$

Assuming that we can apply the implicit function theorem and use Newton iterations to solve the equation system, we should check the regularity of the corresponding Jacobian matrix w.r.t $\dot{Y}_i$, $i = 1,2,\ldots,s$:

$$\begin{aligned} J = &\begin{bmatrix} f_{\dot{Y}}(\dot{Y}_1) & & & \\ & f_{\dot{Y}}(\dot{Y}_2) & & \\ & & \ddots & \\ & & & f_{\dot{Y}}(\dot{Y}_s) \end{bmatrix} \\ +h &\begin{bmatrix} f_{Y}(\dot{Y}_1) & & & \\ & f_{Y}(\dot{Y}_2) & & \\ & & \ddots & \\ & & & f_{Y}(\dot{Y}_s) \end{bmatrix} A \otimes I \; . \end{aligned} \tag{3.14}$$

Here $f_{\dot{Y}}(\dot{Y}_1)$ and $f_Y(\dot{Y}_1)$ denotes the Jacobian matrix of f w.r.t $\dot{Y}$ and Y respectively, both evaluated at the stage $(\dot{Y}_1, Y_1)$.

We see, that if the system is an implicit ODE, then $f_{\dot{Y}}$ is regular, at least locally around the solution, and therefore if h is small enough, then regularity of J is guaranteed, since all the $f_{\dot{Y}}(Y_i)$'s will dominate the (block) diagonal of J. Since we also assume the existence of a solution, we conclude that Runge-Kutta methods in general can be used for the implicit ODE.

Note, that if one uses explicit Runge-Kutta methods, the equations decouple into s-systems of n equations. This has computational advantages, but we also note that it lacks some of the classical advantages of using explicit methods, since we can no longer avoid solving non-linear equations.

BDF methods. As representative for the linear multi-step methods we look at the BDF method. We rewrite this to the implicit form:

$$\begin{aligned} f(\dot{y}_{n+k}, y_{n+k}, t_{n+k}) &= 0 \\ \sum_{j=0}^{k} \alpha_j y_{n+j} &= h\beta_k \dot{y}_{n+k}. \end{aligned} \tag{3.15}$$

This time we substitute $\dot{y}_{n+k}$. If $y_{n+k} \in \mathbb{R}^n$ we get n non-linear equations:

$$f\Big(\frac{1}{h\beta_k}\sum_{j=0}^{k} \alpha_j y_{n+j}, y_{n+k}, t_{n+k}\Big) = 0\ . \tag{3.16}$$

Using the implicit function theorem and possibly Newton iterations, we check for regularity of the Jacobian matrix of this system:

$$J = f_{\dot{y}}(y_{n+k})\frac{\alpha_k}{h\beta_k} + f_y(y_{n+k}) \tag{3.17}$$

Again we see, that since $f_{\dot{y}}$ is regular, we can guarantee that J is regular by choosing h small enough. We conclude that also the BDF methods can be used for the implicit ODE.

3.4 Properties of DAEs

Having described the implicit ODE, the almost trivial case of (3.1), we will move on to describing the non-trivial case, the proper DAE system, which is the central topic of this thesis.

We will, in the following sections, step by step, introduce the basic characteristics of a DAE system.

3.4.1 Singular Jacobian matrix

If we again start by looking at the general implicit DAE of first-order

$$f(\dot{y}, y, t) = 0\ . \tag{3.18}$$

then the fundamental property which separates the DAE from the implicit ODE is that some components of $\dot{y}$ are *not* given as implicit functions of y and t by (3.18).

Usually we expect implicit functions to full-fill the requirement of the implicit function theorem, that the Jacobian matrix $f_{\dot{y}}$ should be regular. Therefore a general property of a DAE system is that $f_{\dot{y}}$ *is singular*. Since $f_{\dot{y}}$ is a functional matrix, this means singular for all the relevant values of $\dot{y}$, y. The relevant values might be more accurately specified, but often the singularity is detectable by the overall structure of the equation system.

3.4.2 Underlying ODE and differentiation index

We can state a simple example of DAE system as

$$\begin{aligned} f(\dot{y}, y, t) &= 0 \\ g(y, t) &= 0 \ , \end{aligned} \tag{3.19}$$

where $f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^{m_1}$ and $g : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^{m_2}$ with $m_1 + m_2 = n$. This means that the Jacobian matrix w.r.t $\dot{y}$ of the combined equations system

$$J = \begin{bmatrix} \frac{\partial f}{\partial \dot{y}} \\ \frac{\partial g}{\partial \dot{y}} \end{bmatrix} = \begin{bmatrix} f_{\dot{y}} \\ 0 \end{bmatrix} \tag{3.20}$$

is singular as expected. We will now see that appropriate differentiation of the equations $g(y, t) = 0$ can turn this system into an ODE. We differentiate g with respect to time t and get:

$$g^{(1)} = \frac{dg}{dt} = g_y(y, t)\dot{y} + g_t(y, t) \tag{3.21}$$

Substituting the g-equations with their differentiated versions we get the system

$$\begin{aligned} f(\dot{y}, y, t) &= 0 \\ g_y(y, t)\dot{y} + g_t(y, t) &= 0 \ , \end{aligned} \tag{3.22}$$

and the Jacobian matrix, corresponding to (3.20), is now

$$J = \begin{bmatrix} f_{\dot{y}}(\dot{y}, y, t) \\ g_y(y, t) \end{bmatrix} . \tag{3.23}$$

This Jacobian matrix might be regular, and if it is, then we can assume that (3.22) is an implicit ODE. (This still requires consistency of the equations.)

We have now seen that appropriate differentiations of some of the equations of a DAE w.r.t time, can lead to an (possibly implicit) ODE system. We call this the *underlying ODE* of the original DAE system. The underlying ODE can be used for analysing the DAE as will be obvious later.

For some DAE systems we might have to differentiate some of the equations two times or more to reveal the underlying ODE. This motivates the definition of an integer-valued quantity called the *differentiation index* for DAE systems.

Definition 3.1 (Differentiation index) *The minimum number of times that all or a part of* (3.18) *has to be differentiates w.r.t t to reveal the underlying ODE, is called the* differentiation index *of the DAE.*

Generally, the differentiation index describes how complex the DAE is. Problems with index 0, has simplest properties corresponding to the (implicit) ODE, and increasing index describes increasing complexity. The example (3.19) is assumed to have differentiation index of 1.

The process of differentiating equations to obtain the underlying ODE is called *index reduction* and is a central part of the methodology described in this thesis.

3.4.3 Consistent initial values

Next, a characteristic property of the DAE system is that it is usually harder to find initial values for the initial values problem. This is already described in section 3.2.3.

The initial value problem is defined as, finding a function $y(t)$ (and an interval $D \subseteq \mathbb{R}$) such that

$$\begin{aligned} f(\dot{y}, y, t) &= 0, \qquad \text{for } t \in D \\ y(t_0) &= y_0 \end{aligned} \tag{3.24}$$

Reconsider the abstract example of a DAE system (3.19), assuming that it has a differentiation index of 1. It is obvious that we cannot choose initial values for $y(t_0) = y_0$, as freely as we are used to for ODE systems. Every initial value has to satisfy the conditions given by

$$g(y_0, t_0) = 0 \; .$$

For DAEs of higher differentiation index, we need to perform more differentiations to obtain the underlying ODE. In these cases the initial values will have to satisfy all the original equations along with the intermediate equations obtained in the index reduction process.

3.4.4 Sensitivity to perturbations and the perturbation index

One of the characteristics of a DAE is that the solution will depend differently on perturbation of the residual equations, than what is normally expected from ODEs, including regular implicit ODEs.

We bring a definition inspired from [ESF98] and [HW96]:

Definition 3.2 (Perturbation index) *The DAE system*

$$f(\dot{y}, y, t) = 0$$

has index k along a solution $y(t)$, if all functions $\hat{y}(t)$ having a defect

$$f(\dot{\hat{y}}, \hat{y}, t) = \delta(t)$$

satisfies the inequality

$$\begin{aligned} ||\hat{y}(t) - y(t)|| \ \leq \ C \ (\ & ||\hat{y}(t_0) - y(t_0)|| \\ & + \max_{t_0 \leq \tilde{t} \leq t} ||\delta^{(-1)}(\tilde{t})|| \\ & + \max_{t_0 \leq \tilde{t} \leq t} ||\delta(\tilde{t})|| \\ & + \max_{t_0 \leq \tilde{t} \leq t} ||\delta^{(1)}(\tilde{t})|| \\ & + \ldots \\ & + \max_{t_0 \leq \tilde{t} \leq t} ||\delta^{(k-1)}(\tilde{t})|| \ ) \end{aligned}$$

as long as $\delta(t)$ is sufficiently small. C is a constant independent of the defect $\delta(t)$ and depends only on f and the length of the time interval $[t_0, t]$.

Here $\delta^{(-1)}(\tilde{t}) = \int_{t_0}^{\tilde{t}} \delta(\tau) d\tau$. This notation is suggested in [ESF98] and will be used much more later.

Note that the definition does not give bounds on an *arbitrary* perturbation, since we cannot guarantee that such a solution would exist.

This is due to the fact that a perturbation might make the perturbed DAE system inconsistent. Either by narrowing/removing the set of consistent initial values, or not being sufficiently smooth for a solution to exist.

We will recall for later that, when we have earlier used the term *regular* to describe the quality of a non-linear equation system or (implicit) ODEs, it has meant that the effects of small arbitrary perturbation should be bounded. If we should try to define the term regularity for DAEs we might look into this as a requirement.

3.4.5 Nearly singular ODEs

We have now seen that if (3.1) is consistent and $f_{\dot{y}}$ is regular, then the system can be considered an implicit ODE, and if $f_{\dot{y}}$ is singular then the system is a DAE.

Now what characterizes the system where $f_{\dot{y}}$ is nearly singular? To investigate this, we can consider what happens to an implicit ODE system, as the system changes towards being singular and finally becoming a DAE at the singular point.

If $f_{\dot{y}}$ is regular and the system (3.1) specifies $\dot{y}$ as an implicit function of y, then we can use the implicit function theorem to get the derivative of the implicit function:

$$\frac{\partial \dot{y}}{\partial y} = f_{\dot{y}}^{-1} f_y$$

If we for simplicity use the limited linear stability theory, then the stability of the system is determined by the eigenvalues of $\frac{\partial \dot{y}}{\partial y}$. Since $f_{\dot{y}}^{-1}$ will generally make the eigenvalues of $f_{\dot{y}}^{-1} f_y$ increase towards infinity as $f_{\dot{y}}$ becomes nearly singular, then it is obvious that we can *not generally expect the solutions of a nearly singular implicit ODE to converge towards the solution of a corresponding DAE system*, since they are much more likely to become unstable and unbounded.

Special studies of particular cases where this is not the case, concerns the classical *singular perturbation problems* described in many text on DAEs:

$$\begin{aligned} \dot{y} &= f(y, z) \\ \epsilon \dot{z} &= g(y, z) \end{aligned} \tag{3.25}$$

But even in these cases, special stability requirements on g are needed to guarantee a convergence of the solution as $\epsilon \to 0^+$. See e.g. [BCP96, p. 106].

Another interesting aspect relating to this discussion is presented in [Söd92]. Here it is described how infinitely small perturbations of DAE can have un-bounded consequences measured on the solutions.

3.5 Direct methods for DAEs

In this section we describe some results from applying the classical discretization schemes, used in section 3.3.1, directly on the (proper) first-order implicit DAE systems:

$$f(\dot{y}, y, t) = 0 \ . \tag{3.26}$$

These methods assume that the function f is given numerically and not symbolically, and this is appealing to the user of the methods.

Generally, the direct method has problems for high-index problems. Convergence for high-index problems is only showed for a limited class of problems having a special structure. It also seems problematic to construct methods for finding consistent initial values based directly on (3.26).

Therefore the problems for the direct methods are:

- Solving the consistent IVP.
- Finding consistent initial values.

3.5.1 BDF methods

As representative for the linear multi-step methods the BDF methods are by far the most popular for DAEs. We recall the stepping formula (3.16) described for use with implicit ODEs:

$$f(\frac{1}{h\beta_k}\sum_{j=0}^{k}\alpha_j y_{n+j}, y_{n+k}, t_{n+k}) = 0 \; . \tag{3.27}$$

This formula is also applied in the DAE case.

The BDF methods have shown convergence properties for many DAE problems. One of the important results for BDF methods, regards their convergence when applied to constant linear coefficient DAEs:

$$A\dot{y} + By = f(t) \tag{3.28}$$

For these systems, the k-step BDF methods will converge as $O(h^k)$ after some initial steps are taken.

For general non-linear implicit DAE systems, it is hard to establish convergence results. There only exist convergence results for lower index system, and they assume that the initial values are consistent to some order. This is a practical problem, since as we will shortly discuss, there exist no general codes which computes consistent initial values directly from a numerical representation of (3.26).

Yet, one of the practical problems of using the BDF methods is the ill-conditioning of the Jacobian matrix used for Newton iterations on (3.27). Again, we examine the Jacobian matrix of the corresponding equation system:

$$J = f_{\dot{y}}(y_{n+k})\frac{\alpha_k}{h\beta_k} + f_y(y_{n+k}) \tag{3.29}$$

Now, since $f_{\dot{y}}(y_{n+k})$ is now singular, it is quite obvious that the matrix J will become increasingly singular as $h \to 0$. This is specially the case for high-index problems. In-fact, if the index is ν then the condition number will behave as $O(h^{\nu})$. See [BCP96, p. 144].

In practice the BDF methods have had success solving many DAE problems. In this connection it is important to mention the popular code DASSL implementing the BDF methods for DAEs. A thorough description of this code can be found in [BCP96].

3.5.2 Runge-Kutta methods

Whereas the BDF methods have been the typical choice of method for solving DAEs, the Runge-Kutta methods have been less used for solving DAEs.

The traditional principle for applying the Runge-Kutta methods for DAE is the same as for implicit ODEs:

$$0 = f(\dot{Y}_i, y_n + h\sum_{j=1}^{s} a_{ij}\dot{Y}_j, t_n + hc_i) \qquad i = 1, 2, \ldots, s \tag{3.30}$$

Generally the methods which can be used for DAEs are methods that are also applied to highly stiff problems. This means methods which are *stiffly accurate*, are L-stable, and have high stage-order. We shall not go into details about the design of Runge-Kutta methods, see instead [HW96], [AP99] or [Kvæ90].

Generally, it is for Runge-Kutta methods also difficult to establish convergence results. These results exist only for lower index systems when considering the general non-linear DAE or for systems with a special structure. In the references mentioned above one can find tables for the obtainable order when applying Runge-Kutta methods to various DAEs.

3.5.3 Consistent initial values

Most of the results regarding convergence of these methods when applied directly to the DAE, assumes consistent initial values are given. Therefore, these methods can only work if other methods can be used to obtain consistent initial values. Some methods are proposed for lower index system e.g. [BHP98].

Methods for performing this task working directly on the implicit DAE system, are not generally available. See e.g. the discussion [BCP96, p. 138].

3.5.4 General discussion of direct methods

We end this section on direct methods to give a short argument for the lack of general convergence results for the direct methods.

If we choose to represent the DAE only by its numerical representation (3.26), then for any method attacking the direct form, it will be impossible to detect whether the system is singular or just nearly singular.

Since we have seen that a nearly singular problem, does not generally converge towards the solutions of the limiting DAE, then we surely cannot expect to generally see convergence when taking smaller steps, since the method should converge towards the possibly unstable solution to the nearly singular ODE, which the numerical representation actually corresponds to.

The reason for the success of the BDF methods and some Runge-Kutta methods should rather be seen as the results of regularization of the problem. In this respect the step-size, is a kind of regularization parameter.

The problems presented in this section, Section 3.5, are the reasons that the studies of this project have been into methods using index reduction, and subsequently a variant of the underlying ODE for solving the initial value problems for DAEs and finding consistent initial values.

3.6 Index reduction

As an alternative principle for the analysis of DAEs, one can try to simplify the DAE by lowering the index. As explained this can be done by appropriate differentiation of some of the equations in the original DAE system.

One can choose to reduce all the way to the underlying ODE, having differentiation index 0. The motivation behind this is, that DAEs can be transformed into ODEs, or possibly implicit ODEs, and thereby solved by traditional ODE solving methods.

Alternatively, we do not always need to reduce all the way to the underlying ODE. We can choose to just lower the index, so that we might get better performance from e.g. the direct methods discussed in section 3.5.

Finally, index reduction can give some in-sight about the system, regarding the differentiability of expressions, and very important, what are the conditions on the initial values.

3.6.1 Index analysis

One of the problems of using index reduction is to find out which equations to differentiate. For some problems it is easy to conclude, but in the general case care needs to be taken.

There are many approaches to performing the index reduction. There are some which are based on differentiating all equations. This generates a large system of equations which can the be uniquely solved for the initial derivative $\dot{y}$. See [CM95] or [HW96, p. 478] for a survey. These methods necessitate the differentiability of the entire equation system and this can be a problem.

The approach presented later in this thesis, is to use structural analysis of the DAE. Altogether, this is called *structural index analysis*. The results of such an analysis will describe which equations to differentiate in order to reduce the index and obtain the underlying ODE. This method also has shortcomings, some of which are typical for structural analysis. See e.g. section 2.5. A discussion, in later section 7.2.1, of these shortcomings are intended to anyway motivate the use of structural index analysis.

3.6.2 Implications of index reduction

In this section we emphasize the relation between the original DAE and the index reduced system and especially the underlying ODE. We will study the implications of index reduction, i.e. differentiation of parts of the DAE.

For this purpose we notice the following: Let $g(y,t) = 0$ denote a set of non-linear equations. If $y(t)$ is a differentiable function of the independent variable t, and g is continuously differentiable in its arguments, then

$$g(y(t),t) = 0 \tag{3.31}$$

$$\Updownarrow$$

$$\frac{d}{dt}\, g(y(t),t) = 0 \quad \wedge \quad g(y(t_0),t_0) = 0 \tag{3.32}$$

This means that it is equivalent to consider each of the two systems (3.31) and (3.32), and hence the set of solutions are the same. This tells us that every time we differentiate an equations we imply a constraint on the initial value, and if this constraint is meet, then the solutions to the reduced system will be identical to those of the original system.

We conclude the following: (1) The underlying ODE can be used for solving the DAE, given consistent initial conditions. (2) Consistent initial conditions, can be determined from the equations which are reduced by differentiation in the index reduction process.

Drift-off. The above arguments suggests that the underlying ODE is sufficient for solving the IVP, if consistent initial values are given. This is also true, and using a numerical integration scheme for solving an IVP, we will generally observe the classical convergence results.

But since the reduced equations are only considered for the consistent initial value, the integration scheme cannot correct the solution according to these as it advances. At every step the numerical method will only full-fill the differentiated constraints and therefore small defects compared to the reduced constraints will not be discovered, and will build up. This is called the *drift off*-phenomenon.

This phenomenon is described several places. See e.g. [BT99] or [HW96].

In-stability. Another special consequence of index reduction is that the underlying ODE can potentially be severely unstable. This is described in e.g. [FL91].

This means that while consistent initial conditions should guarantee that the solution of the underlying ODE is the same as for the original DAE, then this in-stability will amplify the effect of drift-off and making the numerical solution difficult.

3.6.3 Augmented underlying ODE

Drift-off and the possible introduction of instability by differentiation motivates the inclusion of the reduced constraint when solving the IVP in some way.

Therefore we consider using the underlying ODE of a DAE system combined with the reduced equations.

$$\begin{aligned} f_0(\dot{y}, y, t) &= 0 \\ g(y, t) &= 0 \end{aligned} \tag{3.33}$$

Here f_0 represents the underlying ODE, and g the reduced constraints. We call this the *augmented underlying ODE (AUODE)*. This name appears in [MS93].

Chapter 6 will go into describing how the augmented underlying ODE can be used for both finding consistent initial values and solving the IVP for a DAE system.

3.7 Summary

In this chapter we introduce the general implicit DAE, being the general class for both the implicit ODE and the proper high-index DAEs.

An introductory example (section 3.2), shows a dynamical model of the general implicit DAE form (3.1). We observe how a component oriented model of a dynamical system, becomes a DAE, when combining the model equations, from each of the components, by introducing the constraint equations which couple the model variables. During this introductory example some of the characteristic topics of DAEs were introduced: *index reduction*, *underlying ODE* and *consistent initial values*.

The simplest case of the general implicit DAE is the *implicit ODE*. The implicit ODE has much of the same characteristic properties as the explicit ODE. Even the methods for explicit ODEs can easily be generalised to the implicit case.

Next, we defined characteristics of the *proper* DAE system: singularity of the Jacobian matrix w.r.t the derivatives, the differentiation index and the perturbation index. We noted that we cannot generally expect a nearly singular ODE to have properties similar to the corresponding DAE. Working with such (proper) differential algebraic equation system, we discovered some special characteristic phenomena, like consistency of initial values, the underlying ODEs, the differentiation index.

Direct methods for DAEs use the general implicit DAE directly. We discussed the BDF methods and Runge-Kutta methods, which are used in practice for lower index systems, but which are not suitable for treating the general high-index DAE.

Index reduction represents an alternative to treating the DAE directly. We mentioned *structural index analysis*, which will be described in Chapter 5 as a method for analysing how to perform index reduction. Index reduction can have drawbacks as well. Two important examples are *drift-off* and *in-stability* of the underlying ODE. Therefore, we will focus on using the *augmented underlying ODE* as a basis for analysing the DAE in this thesis.

Chapter 4

DAEs on integral form

Literature on DAEs often starts by introducing a general implicit DAE system on first-order form

$$f(\dot{y}, y, t) = 0 \; . \tag{4.1}$$

What is often omitted is a discussion of how a concrete model is interpreted in terms of (4.1), and more precisely, how the variables are assigned to the representation vectors y, $\dot{y}$.

This question might appear trivial, since there seems to be an almost natural way of doing this, given a model with an number of variables and derivatives of these. One problem is that this natural way of doing the assignment is often inappropriate.

Another problem is that this freedom in assigning variables to the general form (4.1), propagates as a vagueness of definitions based on this form. As example, depending on the chosen assignment, one might classify and turn an implicit ODE into an index-1 DAE. This means the differentiation index has an inherent vagueness.

The scientific and educating contribution of this chapter is to show how appropriate assignment is chosen, how one can consider using alternative notation of the general DAE form (4.1), and remove the ambiguity and vagueness of the general form by introducing the *integral form*. This integral form is considered a ideal basis for discussing properties of DAEs, and is used frequently during the rest of this thesis.

4.1 Appropriate assignment of variables

This section illustrates the freedom that the general form (4.1) leaves for assigning the model variables to the vectors y and $\dot{y}$, and gives an initial answer to the question of how to choose *appropriate assignment*.

The freedom of assigning variables to either y or $\dot{y}$ does not concern the *normal* variables in an ODE model. By normal variable, we consider a variables which appear both "as it self" and "as derivative" in the model. The freedom comes in when considering variables which appear only "as them selves". These can in principle be assigned to either y or $\dot{y}$. We will soon see that the natural choice, which is to assign such variables to y is often inappropriate.

How should we then choose assignment? Generally we prefer to treat ODE systems in contrast to DAEs. According to section 3.3, the (implicit) ODE is characterised by defining $\dot{y}$ as a (possibly implicit) function of y and t. Therefore we should prefer a variable assignment so that (4.1) defines $\dot{y}$ as an implicit function of y and t.

We exemplify and examine this using the two-mass system from section 3.2:

■ **Example 4.1 (Interpreting the two-mass system)** *If we recall the two-mass mechanical system, from the example in section 3.2.1. We state the model equations after some index reduction, i.e. differentiations have been performed:*

$$\begin{aligned} M_1\dot{v}_1 &= F \\ M_2\dot{v}_2 &= -F + f(t) \\ \dot{x}_1 &= v_1 \\ \dot{x}_2 &= v_2 \\ (1/M_1 + 1/M_2)F &= f(t)/M_2 \quad . \end{aligned} \tag{4.2}$$

The last equation comes from differentiating the original constraint $x_1 - x_2 = l$ twice, and substituting $\dot{v}_1$ and $\dot{v}_2$. We restate all the model variables, including derivatives:

$$x_1,\ x_2,\ v_1,\ v_2,\ F,\ \dot{x}_1,\ \dot{x}_2,\ \dot{v}_1,\ \dot{v}_2\,.$$

Now assume that we want to use a numerical integration routine, which requires that we use the form

$$f(\dot{Y}, Y, t) = 0\ .$$

How are we going to assign these model variables to the vectors $\dot{Y}$ and Y. We will look into two different ways of doing this:

(1) The first way and perhaps the most natural way of assigning the variables, concerning the notation we have used:

$$\dot{Y} = (\dot{x_1}, \dot{x_2}, \dot{v_1}, \dot{v_2}, \dot{F})$$
$$Y = (x_1, x_2, v_1, v_2, F)$$

We note the assignment of F. Using this assignment, it implies the introduction of the variable $\dot{F}$, and (4.2) is not an explicit nor implicit ODE, since we cannot compute or solve for all variables in $\dot{Y}$ as a function of Y and t.

(2) Alternatively, let us use the assignment defined by:

$$\dot{Y} = (\dot{x_1}, \dot{x_2}, \dot{v_1}, \dot{v_2}, F)$$
$$Y = (x_1, x_2, v_1, v_2, \textstyle\int F)$$

Note the assignment of F. Using this assignment we have suddenly introduced the variable $\int F$, the integral of F over time. Here $\int F$ is a short notation for $\int F dt$.

Now using this assignment, we realise that the system (4.2) *is an implicit ODE, since we can solve for all the variables in vector $\dot{Y}$ as a function of t and the variables in Y. Furthermore, applying the numerical routine, it should not handle a DAE but only an implicit ODE, which is far less restrictive on the choice of methods.* ■

From this example we can see that choosing an appropriate variable assignment is important. It can imply whether the model interpreted as an ODE or a DAE.

What seems to be the problem can be illustrated by the rank of the Jacobian matrix

$$\frac{\partial f}{\partial \dot{y}}$$

If we have full rank, the we can assume that the system is a (regular) implicit ODE. Consequently, to discover that the system is essentially an ODE, we have to look for an assignment which maximizes the rank of this Jacobian matrix. To do this we should generally try to assign variables to $\dot{y}$ if there is any choice.

4.1.1 Consequences of ambiguity

We can list some of the consequences of this ambiguity since it propagates to classification of DAEs, index definitions and even the use of numerical integration routines.

It is easy to see that some of the classification of DAEs inherit this ambiguity: Assume a classical *semi-explicit index-1 DAE*, see e.g. [BCP96, p. 34],

$$\begin{aligned} \dot{y} &= f(y, z, t) \\ 0 &= g(y, z) \end{aligned} \tag{4.3}$$

where y and z are vectors and $\partial g / \partial z$ is nonsingular.

Assume that we want to assign the variables to the general DAE

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

Then the system (4.3) is an DAE of index-1, if we assign the z-vector to Y. If we assign the z-vector to $\dot{Y}$, then the system is an implicit ODE, i.e. an index-0 system. In this way we *reduce the differentiation index* of the semi-explicit index-1 DAE.

Considering this, we realise that many systems and models, considered to be of a given index, can actually be reduced one index, if the system is carefully interpreted. We also realise, that this has importance not only for the definition of the differentiation index, but also for the definition of the perturbation index.

Furthermore, it is obvious that choosing the wrong style of assignment, this will also have consequences for the use of the numerical integration schemes: Assume that we want to take an explicit Runge-Kutta step. As described in section 3.5 an explicit step can only be used for ODEs and not DAEs. This illustrates that choosing the right assignment have importance for the application of the numerical integration routines.

4.2 Integral notation and form

The previous section showed that, unfortunately, what seem to be the standard and most natural way of assigning model variables to the $\dot{y}$ and y-vectors of the general implicit DAE, form is actually inappropriate.

One of the reasons for this can be related to the fact that we use derivatives in our notation of models. What happens if we use an *integral notation* instead? Does this make the natural way of assigning variables more appropriate? This is examined in this section.

4.2.1 Integral notation

First we define a notation for a variable which is an integral of another variable, as we have a notation for a derivative - the “dot” notation. The classical notation of the integral, $\int ydt$, would be too long and awkward, and therefore we propose writing

$$\smallint y = \int ydt$$

instead. This can be though of as some kind of operator notation. We have already used this notation in Example 4.1.

Using integral notation on the general implicit first-order form, as (4.1), becomes

$$f(y, \smallint y, t) = 0 \ . \tag{4.4}$$

We continue the example of the previous section:

■ **EXAMPLE 4.2 (TWO-MASS SYSTEM - INTEGRAL NOTATION)** *We continue Example 4.1. We want to reformulate the model, such that it uses integral notation instead of the traditional derivative notation.*

We simply have to re-think *all the variables of the model, stating their mutual relations in terms of integrals. Therefore we introduce accelerations a_1 and a_2, re-think the role of the velocity variables v_1 and v_2, keep the force variable F, but we drop the position variables x_1 and x_2, since these are now substituted by the integrals $\smallint v_1$ and $\smallint v_2$.*

Then we restate the model equations using the new variables and integral notation:

$$\begin{aligned} M_1 a_1 &= F \\ M_2 a_2 &= -F + f(t) \\ v_1 &= \smallint a_1 \\ v_2 &= \smallint a_2 \\ (1/M_1 + 1/M_2)F &= f(t)/M_2 \quad . \end{aligned} \tag{4.5}$$

Also recall the original constraint $x_1 - x_2 = l$ which we reduced by differentiation to get the last equations. This constraint will be expressed as $\smallint v_1 - \smallint v_2 = l$ when using integral notation.

Once more, assume that we want to use a numerical integration routine. This time the routine requires that we use the form

$$f(Y, \smallint Y, t) = 0 \ .$$

How should we assign the variables? The natural way of doing this is now

$$\begin{aligned} Y &= (v_1, v_2, a_1, a_2, F) \\ \smallint Y &= (\smallint v_1, \smallint v_2, \smallint a_1, \smallint a_2, \smallint F) \ . \end{aligned}$$

It can be seen that using this assignment, then Y is given as an implicit function of $\smallint Y$ and t, which makes the system an implicit ODE. *And therefor the natural assignment to the integral form is appropriate.* ■

From this example we conclude that integral notation helps choosing the appropriate assignment.

Since we some times use high-order formulation of DAEs and ODEs, we present the definition of integral notation using higher order derivatives and integrals.

Definition 4.1 (Integral notation) *If a general implicit high-order DAE, normally represented by the form*

$$f(y^{(n)}, y^{(n-1)}, \dots, \dot{y}, y, t) = 0 \tag{4.6}$$

is alternatively represented by the form

$$f(y, \textstyle\int y, \iint y, \dots, \int^{(n)} y, t) = 0 \tag{4.7}$$

we say, that we use integral notation.

In [ESF98] they introduce an alternative notation for the first-order integral: They propose writing $y^{(-1)} = \int y$. We extend this to higher order integrals, e.g. $y^{(-2)} = \iint y$. Using this notation-style, (4.7) can be written as

$$f(y, y^{(-1)}, y^{(-2)}, \dots, y^{(-n)}, t) = 0. \tag{4.8}$$

We refer to this as the *Eich-Führer notation style.*

It is not the intension that we should generally skip using derivatives and only use integral notation in future work with ODEs and DAEs, but it is found important that one in certain cases tries to re-think the model in terms of pure integral notation.

4.2.2 Base variables and mixed notation

For a model using derivative or integral notation we define the *base variables*: Base variables are variables which appear directly "as themselves" in the model. All other variables are either represented as derivatives or integrals of these base variables. Considering Example 4.1, the base variables are x_1, x_2, v_1, v_2 and F, since all other variables appear as derivatives. Considering the re-interpretation to integral notation in Example 4.2, then suddenly the base variables are a_1, a_2, v_1, v_2 and F instead.

Since the basic aim is to develop methodologies for analysing the general implicit DAE, and give as much freedom in formulating these, we emphasize the use of *mixed notation* using both derivatives and integrals. Dependent on this we introduce the mixed notation of the general implicit DAE:

$$f(\ldots, \dot{y}, y, \textstyle\int y, \ldots, t) = 0, \tag{4.9}$$

where $f = (f_1, f_2, \ldots, f_n)$ are the general non-linear residuals and $y = (y_1, y_2, \ldots, y_n)$ are the *base variables*. Note, that we do not consider exactly how high-order derivatives and integrals are used. Using Eich-Führer-notation (4.9) can be written as

$$f(\ldots, y^{(1)}, y^{(0)}, y^{(-1)}, \ldots, t) = 0 \ . \tag{4.10}$$

This mixed form will be the basis of the methodology invented in Chapter 5 on structural index analysis. It is proposed that modelling and software tools should generally support this high-order mixed notation. In this way modellers can choose between specifying derivatives or integrals of any order.

4.2.3 Integral form

Returning to the question on how to choose the appropriate assignment of variables to the representation vectors, and remove the freedom previously characterizing the general DAE form, we define the *integral form*:

Definition 4.2 (Integral form) *Assume a DAE system represented using pure integral notation*

$$f(y, \textstyle\int y, \iint y, \ldots, \int^{(n)} y, t) = 0 \ ,$$

so that y contains all the base variables $y_1, y_2, \ldots, y_n$.

If the assignment of variables is done, such that every non-integral variable is assigned to y, then the DAE system is represented on integral form.

By non-integral variables[1] we understand a variable which by notation is not an integral of another variable.

The integral form should guarantee that we chose an assignment, that maximizes the rank of the Jacobian matrix $J = \partial f / \partial y$. Therefor it is appropriate for revealing if a system is essentially an (implicit) ODE.

[1]In other contexts they are sometimes referred to as the *highest derivatives*.

Notice that every DAE system using either derivative, integral or mixed notation, see (4.6), (4.7) or (4.9) respectively, can be reassigned to obtain the integral form. It is only a matter of identifying all the non-integral variables and assigning these to y. From this follows the assignment of the remaining variables, the *integral variables*, to the vectors $\int y$, $\iint y$, etc.

Notice that this way of assigning the variables has removed the freedom in assigning variables, since there is no doubt whether a variable should be assigned to y, $\int y$ or $\iint y$ etc.

■ **Example 4.3 (Two mass system - Integral form)** *We consider the DAE system* (4.2) *from Example 4.1. We will try to put it on (first-order) integral form*

$$f(Y, \textstyle\int Y, t) = 0 \ .$$

Therefore we identify the non-integral variables in the model: These are $\dot{x}_1, \dot{x}_2, \dot{v}_1, \dot{v}_2$ *and* F.

One could ask if $\dot{x}_1$ *and* $\dot{x}_2$ *are really non-integrals, since we can deduce from the equations that they are actually integrals of* $\dot{v}_1$ *and* $\dot{v}_2$ *respectively. But since we only consider what follows from notation, and not what is deduced from the equations, we still classify* $\dot{x}_1$ *and* $\dot{x}_2$ *as non-integral variables.*

We can now assign the variables to the vectors Y *and* $\int Y$:

$$\begin{aligned} Y &= (\dot{x}_1, \dot{x}_2, \dot{v}_1, \dot{v}_2, F) \\ \textstyle\int Y &= (x_1, x_2, v_1, v_2, \textstyle\int F) \ . \end{aligned}$$

We can see that this way of assigning the variables is appropriate since it makes the system (4.2) *appear as an implicit ODE. Notice that the assignment also correspond to the appropriate assignments found in Example 4.1 and Example 4.2.* ■

4.3 A general ODE class

To simplify the analysis of general implicit DAE systems, we would like to ensure that any system which by proper variable assignment can be interpreted as an ODE, is actually treated as an ODE.

Therefore we define the term *general ODE* for any system which in this way can be interpreted as an ODE:

Definition 4.3 (General ODE) *Assume a general DAE represented on integral form*

$$f(y, \int y, \iint y, \ldots, \int^{(n)} y, t) = 0 \ . \tag{4.11}$$

If by (4.11), *all* non-integral *variables y are given as a (possibly implicit) function of the* integral variables $\int y$, $\iint y$, $\ldots$, $\int^{(n)} y$ *and the independent variable t, then the system is a* general ODE.

From this point the term general ODE will be used for any system which by proper assignment of the variables will be an ODE according to this definition. For example, this means that we will no longer consider the semi-explicit index-1 DAE as DAE but as a general ODE. A discussion of which model types are essentially general ODEs is given in section 4.3.2.

4.3.1 Regular ODE

As mentioned in section 3.3, on implicit ODEs, sufficient conditions for a system (4.11) to be an implicit ODE, can be found in the implicit function theorem 2.6:

If we assume that the system (4.11) is consistent and has a solution $y(t)$, then if the Jacobian matrix

$$\frac{\partial f}{\partial y}$$

of (4.11) is regular in the neighbourhood of the solution, then (4.11) will be an implicit function for y, and hence (4.11) will be a general ODE, according to Definition 4.3. We call such a system a regular ODE, as described in section 3.3.

Since we generally assume that every well-behaved general ODE is a regular ODE, this Jacobian matrix becomes very important. It will in the following be referred to as the *Leading Jacobian.*

We can now argue that by using the integral form, we maximise the possible rank of the Leading Jacobian matrix, and in this respect we have the best chances for detecting whether the system is a general (regular) ODE.

4.3.2 Extension of the general ODE class

By introducing the definition of the general ODE, we can subclass many traditional model types used in science and engineering, which are usually not considered as ODEs. We do this to underline the extensiveness of the general ODE, here represented in the first-order form

$$0 = f(Y, \textstyle\int Y, t) \ . \tag{4.12}$$

First, we trivially note that the classical ODE (2.7) and the systems previously reckoned as implicit ODEs (3.11) are also general ODEs. We simply reorder to residual form and change notation from derivative notation to integral notation.

Continuing with a very basic model type for a dynamical system(!), we take simple algebraic equations specifying a number of dependent variables $y \in \mathbb{R}^n$ as a function of time t. No derivatives of variables are included in this type of model. It can be represented on the fully implicit form

$$0 = f(y, t) \ . \tag{4.13}$$

We can interpret (4.13) as a general implicit DAE on integral form. If (4.13) implicitly expresses y as a function of t, i.e. then (4.13) is essentially a general ODE.

It should be mentioned that this model type can have several names depending on the field of application. Some examples are

- algebraic equations,
- static equations,
- quasi static equations.

Next, we take a short step to a model type which is also used much in practise but seldom considered an ODE model. We have already considered this mode type, namely the *semi-explicit index-1 DAE*. The model often defines two vectors of dependent variables $y \in \mathbb{R}^n$ and $x \in \mathbb{R}^m$. The model equations have the following form

$$\dot{y} = f(y, x, t) \tag{4.14a}$$
$$0 = g(y, x, t) \tag{4.14b}$$

and typically $\partial g/\partial x$ is regular and therefore (4.14b) implicitly expresses x as a function of y and t.

If we re-interpret in terms of the integral form, then the non-integral variables are $Y = (\dot{y}, x)$ and the integral variables are $\int Y = (y, \int x)$. Since (4.14a) and (4.14b) collectively expresses the variables in Y as a function of the variables in $\int Y$ and t this system is also a general ODE.

We can generalise this system to a fully implicit version where the ODE part cannot be separated from the algebraic equations. The system is represented by

$$0 = f(\dot{y}, y, x, t)$$

but still, if $\dot{Y} = (\dot{y}, x)$ can be expressed as a function of $Y = (y, \int x)$ and t, then this system is a general ODE also.

We briefly mention that in practical applications other terms are used for this model:

- semi-explicit index-1 DAE.
- ODE with algebraic variables.

The important thing to learn from this section is that a vast number of classical model types for dynamical system belong to this general ODE class. And therefore software designed to treat the general ODE, will do for all these model types.

4.3.3 Numerical routines for general ODE

The extensiveness of the general ODE makes it an obvious basis for a software library for numerical integration. Most of the important libraries also have methods for treating the implicit ODE and therefore the general ODE. There are though a number of comments to this.

Often the libraries assumes a first-order form and assumes derivative notation. The user is expected to define and implement a *residual function*, which in e.g. C++ could have the declaration:

```
double* CalcResidual(double* ydot, double* y, double t)
```

where `double*` denotes an array of some size.

For one, this means that the user should be careful to assign variables to the appropriate vector, since when using derivative notation, we have seen the natural way of doing this, is not necessarily appropriate.

Next, the library methods for the general implicit ODE do not always save the derivatives as output. Since the appropriate assignment of variables sometimes requires that variables, that should be saved, are assigned to the derivative vector, then it should be possible to save variables in the derivatives as well.

As numerical routines treating the implicit ODE and therefore the general ODE, we mention DASSL, see e.g. [BCP96], or GODESS, see e.g. [Ols98].

The tool Simpy developed in this project, see Chapter 8, also treats the general ODE, although the implemented integration routines are not as advanced and efficient as the before mentioned routines.

At this moment a tools as Matlab do not have a method for the general implicit ODE. It has a variant, where a so-called mass matrix $M(y,t)$ is multiplied on the derivative side of a classical first-order ODE:

$$M(y,t)\dot{y} = f(y,t)$$

This is not enough for treating the general ODE.

4.4 Summary

Traditional texts introduce the general implicit DAE form, and leaves out a thorough discussion on how to interpret and use it.

We show that this have special consequences for much of the theory and definitions concerning DAEs. There are e.g. freedom in assigning model variables to the vectors of the general implicit DAE form. This means that some of the central definitions in traditional texts are actually ambiguous or at least vague, since this freedom affects the definitions.

Motivated by this we introduce the *integral notation* and especially the *integral form*, which removes the freedom in assigning model variables to the vectors of the general DAE. The idea is that we interpret model variables, as dependent variables, where some variables are *integrals* or *high-order integrals* of the others, instead of derivatives.

Among all model variables, we can now easily determine the so-called *non-integral variables* which are not integrals of other variables. We acknowledge that these are exactly the variables of interest if we are going to determine if a general implicit DAE/ODE, is really an ODE.

We introduce the definition of the *general ODE*. A general ODE is a general implicit DAE system, where all non-integral variables are given as an (implicit) function of the integral variable and the independent variable.

We shortly discuss that a large class of traditional models for modelling of e.g dynamical systems, can be categorized as a general (implicit) ODE.

Introducing the integral form and the general ODE might seem like just a matter of form, but it is in-fact important when using numerical routines for integrating the initial value problem. If the variables are assigned inappropriately to the vectors of the general DAE/ODE form, then seen from a numerical point of view one possibly raises the index and this have consequences for the numerical computations.

Finally it is discussed how libraries for numerical integration of ODE, should include a code for solving the general (implicit) ODE.

Overall the approach for analysing DAE systems presented in this chapter are considered an interesting contribution of this Ph.D. project.

Chapter 5

Structural index analysis

This chapter presents theory and algorithms for structural index analysis of DAEs. Structural index analysis represents an approach for analysing DAEs based on high-level information on the equation systems. This is *structural information*, which is basically is information about which variables are included in which equations.

In this chapter we will present the theory and algorithms of structural index analysis based on a simplified structural representation of DAEs developed under this Ph.D. project. The presentation is believed to be a unique contribution to research in DAEs and especially structural analysis of DAEs, because the representation, corresponding theory and algorithms will hopefully be accessible to a larger audience that before, being based on a matrix representation compared to a graph theoretical representation. The presentation of the theory is also believed to be more complete than existing presentations of structural index analysis.

5.1 Introduction and motivation

The are a number of reasons for performing a structural analysis of a DAE system.

The prime motivation is that structural analysis can give the answer on how to perform index reduction of a DAE system, by differentiating equations and thereby revealing the underlying ODE. At the same time this uncover the underlying constraints of the DAE, which in combination gives the augmented underlying ODE. This can subsequently be used for determination of consistent initial values.

Another reason is that for higher-index problems, the direct numerical methods often fail to converge or they pose very ill-conditioned problems. Structural analysis is not generally limited by the index.

Structural analysis can also explain important smoothness demands on functions and tables in the model, which might prevent the modeller from building potentially inconsistent models.

Criticism of structural analysis is mainly that the analysis will not in all cases reveal the underlying ODE. This is discussed in later Chapter 7.

5.1.1 Basic index reduction

Structural index analysis is concerned about obtaining information on which equations to differentiate to obtain the *underlying structural ODE* defined shortly.

Importantly, structural analysis limits itself to considering what is called *basic index reduction*. This means that index reduction is done solely by differentiations of subsets of equations. An important property of basic index reduction is that we can structurally represent any consequences of such differentiations, as will be obvious later. Hence the complete index analysis can be done using only structural information of the original DAE system.

5.1.2 Differentiating MSS subsets

One of the initial, important contributions to structural analysis is [Pan88]. In this article the fundamental principle of structural index analysis is described.

The motivation is very simple: The DAE is singular w.r.t. the non-integral variables. By differentiation of a set of equations we introduce new derivative variables. But if it is possible to identify a set of equations which by differentiation will introduce less new variables than new equations produced, then there is a chance that we can decrease the rank-deficiency, hopefully ending with full rank..

These sets of equations are the so-called MSS subsets, defined in the background material on structural analysis, section 2.5. It will later be theoretically shown that MSS subsets are exactly what to look for when performing the structural index analysis.

5.1.3 Review

An early references on structural analysis is [DG86]. Here they devise an algorithm which can determine if the structural index is 0,1 or 2.

One of the first algorithms for the general structural analysis of arbitrarily high index DAE is described in previously mentioned [Pan88]. In this article the emphasis is on detecting the equations governing consistent initial values. This article already mentions that the analysis can potentially be used for obtaining the underlying ODE.

Following this, a number of articles discussing the algorithms and the *structural index* defined by the methods, have bee published. See e.g. [CG95] or [UKM95].

One of the obvious shortcomings of structural analysis is that structural analysis might consider a system (structurally) regular, when it is algebraically singular. This means that structural analysis will stop index reduction at the structurally regular system, while algebraic or numerical investigation could reveal that the system is in-fact a higher index DAE.

This is probably one of the main reasons why structural analysis is not included in classical texts on DAEs as e.g. [BCP96] and [HW96]. Despite this there is a number of documented practical uses of structural analysis e.g. [And94], [Mat95] or [UKM95].

In Chapter 7 we discuss the impact of these shortcomings and elaborate on the problems regarding the structural index compared to other indices. In Chapter 8 the software package Simpy is described. Simpy is developed during the Ph.D. project to demonstrate that applicability of structural analysis as the essential part of a general methodology for DAEs.

5.2 Structural properties of ODEs

The purpose of index reduction is to perform appropriate differentiations on the general implicit DAE and ultimately reveal the underlying ODE. In the ideal index reduction process, we continuously want to check when the current reduced system is a *general ODE*, consequently the underlying ODE, and then stop the reduction process. The conditions for a system to be a general ODE are found in Definition 4.3. Recall that the term general ODE cover both explicit and implicit ODEs.

Structural analysis can not determine and guarantee if a general implicit DAE system is essentially a general ODE, and therefore we need another principle for determining the endpoint of index reduction process.

We define another endpoint, the *structurally regular ODE* or just the *structural ODE*, consequently the *underlying structural ODE*, which *can* be determined by structural analysis, and which is a close parallel and indication of a system being of a general ODE.

5.2.1 Structural ODE

The definition of the *structural ODE* is motivated by the sufficient condition for a general ODE. These were presented in section 4.3.1 on regular ODEs:

Assume that we have first brought a general implicit DAE to integral form, as explained in chapter 4:

$$f(y, \int y, \iint y, \ldots, t) = 0 \ . \tag{5.1}$$

We also assume that the system is consistent and has a solution $y(t)$. Then if the so-called *leading Jacobian matrix*, i.e.

$$J = \frac{\partial f}{\partial y}$$

is regular, then (5.1) is a regular (implicit) ODE, consequently a general ODE. As mentioned earlier we generally assume that general ODEs are regular.

Subsequently we use that structural analysis can give a *necessary condition* for J to be regular, namely that J is structurally regular. See section 2.5. We define the following:

Definition 5.1 (Structural ODE) *Assume a general implicit DAE system* (5.1) *on integral form: If the Jacobian matrix*

$$J = \frac{\partial f}{\partial y} \tag{5.2}$$

is structurally regular, then the system is called a structural ODE.

We can choose to emphasize the meaning of the term structural ODE, by giving using the term *structurally regular ODE*. In structural analysis of ODEs/ DAEs we take this as the fundamental indication that the system is probably a general ODE.

We can use Algorithm 2.2 to determine if a system is a structural ODE, if we compute the structural representation S of the leading Jacobian matrix.

5.2.2 The underlying structural ODE

Subsequently we use the structural ODE as the endpoint of the structural index reduction process. We call the endpoint the *underlying structural ODE*.

We will later, in Chapter 7, discuss the consequences and weaknesses of using the structural ODE as an endpoint of the index reduction process. This has actually been one of the central points of criticism considering structural analysis for DAEs:

Using the structural ODE as endpoint of index reduction, we must assume that higher index DAEs are not generally structural ODEs. There are in-fact many examples in literature, that higher index DAEs structurally has the properties of ODEs. Though, there is a number of ways to oppose to this criticism and defend using the structural ODE as an endpoint of index reduction.

In the following, lets us for simplicity assume when we identify the underlying *structural ODE*, this will actually be a *consistent and regular ODE* - i.e. the "real" underlying ODE of the DAE.

5.3 Structural representation of DAEs

To perform structural index analysis on a DAE system, a special representation of the structural information of a DAE system has been developed during this Ph.D. . The representation is called the *index matrix representation*.

This representation is more compact than representations used in literature, e.g. [Pan88], since it contains less superfluous information. Furthermore the use of a matrix representation instead of a bipartite graph[1] representation is believed to introduce less complexity in the implementation of the algorithms.

5.3.1 The general implicit DAE

Assume a general implicit (high-order) DAE system, consisting of n equations, using n base variables, and (high-order) derivatives and integrals of these: We represent this using mixed derivative/integral notation[2]:

$$f(\ldots, \dot{y}, y, \textstyle\int y, \ldots, t) = 0, \tag{5.3}$$

where $f = (f_1, f_2, \ldots, f_n)$ are the residuals and $y = (y_1, y_2, \ldots, y_n)$ are the base variables. Using the Eich-Führer notation, equation (5.3) can be written

[1] See e.g. [Har69] for more on bipartite graphs.

[2] See section 4.2.2.

as

$$f(\dots, y^{(1)}, y^{(0)}, y^{(-1)}, \dots, t) = 0. \tag{5.4}$$

In the following we present two examples of DAE systems of the form (5.3). The first example is a classical technical example, while the second example is more contrived. At this point the DAE systems will just be presented without further analysis:

■ **Example 5.1 (Pendulum system)** *The classical pendulum system is characterised by the (residual) equations:*

$$\begin{aligned} \mathrm{M}\ddot{x} + \frac{Fx}{L} &= 0 \\ \mathrm{M}\ddot{y} + \frac{Fy}{L} + Mg &= 0 \\ x^2 + y^2 - L^2 &= 0 \end{aligned} \tag{5.5}$$

The equations (5.5) *suggests using* $Y = (x, y, F)$ *as base variables.* ■

■ **Example 5.2 (Contrived DAE)** *The following system is an example of a general implicit DAE system, using highly non-linear expressions, and using both high-order derivatives and integrals of a set of base variables* $Y = (x, y, z)$.

$$\begin{aligned} sin(\textstyle\int x + x\dddot{y}) - \int z &= 0 \\ x + \textstyle\iint z - 2 &= 0 \\ x^2 + y^2 - 4 &= 0 \end{aligned}$$

One could imagine this DAE system being build using a flexible DAE modelling system, where it is allowed to freely specify the DAE using implicit non-linear equations, and high-order integrals and derivatives of the base variables. Such a system would utilize the freedom of a general form as (5.3). ■

The intension of the examples is to show the generality of the general implicit DAE (5.3) and of structural analysis. These two examples will follow the rest of the presentation.

5.3.2 Leading derivative index

In this section we define an integer number called the *leading derivative index*. This is a quantity that describes the relation between each single equation and each single base variable of the system.

Let us first concentrate on a single equation of the DAE system (5.4) denoted by the index i:

$$f_i(\ldots, y^{(1)}, y^{(0)}, y^{(-1)}, \ldots, t) = 0 \; . \tag{5.6}$$

In principle, this equation can depend on every one of the base variables in $y = (y_1, y_2, \ldots, y_n)$ and integrals and derivatives of these variables.

We will only concentrate on dependency on a *specific* base variable y_j, denoted by the index j, and its derivatives and integrals: We organize a chain of these derivatives/integrals of base variable y_j in the following order:

$$\ldots, y_j^{(2)}, y_j^{(1)}, y_j^{(0)}, y_j^{(-1)}, y_j^{(-2)}, \ldots \tag{5.7}$$

For structural index analysis we need to identify the *highest derivative* of y_j, i.e. the *first* variables in this chain, on which equation (5.6) depends. Often this highest-derivative can be identified just by looking at the equations, but a little more formalism is introduced:

We compute all the *partial derivatives* of f_i, in the following order according to (5.7):

$$\ldots, \quad \frac{\partial f_i}{\partial y_j^{(2)}}, \quad \frac{\partial f_i}{\partial y_j^{(1)}}, \quad \frac{\partial f_i}{\partial y_j^{(0)}}, \quad \frac{\partial f_i}{\partial y_j^{(-1)}}, \quad \frac{\partial f_i}{\partial y_j^{(-2)}}, \quad \ldots \; . \tag{5.8}$$

The first derivative *not identically zero*, i.e.

$$\frac{\partial f_i}{\partial y_j^{(m)}} \neq 0 \; ,$$

is called the *leading derivative* of equation $f_i = 0$ with respect to variable y_j.

Note, that sometimes all derivatives in (5.8) might be zero, if there is no dependency on y_j or any derivatives and integrals of this variable. Then we say that there is *no* leading derivative.

One of the benefits of structural analysis is that we do not need to know the leading derivative $\partial f_i / \partial y_j^{(m)}$ exactly/symbolically. All we need to know is *if* and *when* the leading derivative appears, i.e. the *leading derivative index*:

Definition 5.2 (Leading derivative index) *Assume a general implicit DAE system* (5.4). *The* leading derivative index *for a base variable* y_j

considering equation $f_i = 0$, is defined as the integer m, such that

$$\begin{aligned} \frac{\partial f_i}{\partial y_j^{(m)}} &\neq 0 \\ \frac{\partial f_i}{\partial y_j^{(m+k)}} &= 0 \quad , \text{for } k = 1, 2, \ldots \end{aligned} \tag{5.9}$$

Note that $\partial f_i / \partial y_j^{(m)}$ is the leading derivative*.*

We specially define that the leading derivative index is $m = -\infty$, if there is no leading derivative, i.e.

$$\frac{\partial f_i}{\partial y_j^{(k)}} = 0 \quad , \text{for } k = \ldots, -1, 0, 1, \ldots .$$

In this way we have formalised a specific equation's dependency on the highest derivative of a specific variable.

5.3.3 Index matrix representation

We will now continue with the definition of the *index matrix representation*, which corresponds to the structural representation of the complete DAE system.

The index matrix representation basically consist of a matrix, where the elements have integer values, corresponding to the *leading derivative index*:

Definition 5.3 (Index matrix representation) *Assume a DAE system* (5.4) *with base variables y.*

An $n \times n$-matrix M consisting of leading derivative indices, where each element $M_{i,j}$ corresponds to the leading derivative index of variable y_j in equation $f_i = 0$, is called the index matrix *of the DAE system.*

The following sections will show that this representation of a DAE system is ideal for performing basic index reduction because fundamental operations like differentiation and re-ordering to integral form becomes trivial. They are merely a question of addition or subtraction of constants to rows or columns.

As graphical notation for the index matrix we can use a matrix-like notation with elements

$$M_{i,j} \text{ represented by } \begin{cases} M_{i,j} & \text{if } M_{i,j} \neq \infty \\ \text{(blank)} & \text{if } M_{i,j} = -\infty \end{cases}$$

The following are examples on computing the index matrix for DAE systems. These examples continues the examples presented earlier.

■ **Example 5.3 (Pendulum system - index matrix)** *We continue Example 5.1 by computing the index matrix of this system. We recall the model equations:*

$$\begin{aligned} \mathrm{M}\ddot{x} + \frac{Fx}{L} &= 0 \\ \mathrm{M}\ddot{y} + \frac{Fy}{L} + Mg &= 0 \\ x^2 + y^2 - L^2 &= 0 \end{aligned}$$

$Y = (x, y, F)$ *are the base variables. Note that* M *is the mass (not to be confused with the index matrix* M*).*

We start by concentrating on the first equation $f_1 = \mathrm{M}\ddot{x} + \frac{Fx}{L} = 0$*. For this particular equation, we successively concentrate on each variables in* Y*:*

Considering x*, we see that the* leading derivative index *must be 2, since* $\ddot{x} = x^{(2)}$ *is the highest derivative appearing in the equation. The leading derivative is therefore*

$$\frac{\partial f_1}{\partial x^{(2)}} = \mathrm{M} \, .$$

Considering y*, neither* y *or any of its derivatives or integrals appear in the equation, and therefore there is no leading derivative and the index is* $-\infty$*.*

Finally considering F*, we see that the leading derivative index is 0, since the base variables itself,* $F = F^{(0)}$*, is the highest derivative appearing in the equation. The leading derivative is*

$$\frac{\partial f_1}{\partial F^{(0)}} = \frac{x}{L} \, .$$

We have now computed the first row of the index matrix:

$$[2, -\infty, 0]$$

We can do the same for the last two equations/rows, and all together this gives the index matrix:

$$M = \begin{bmatrix} 2 & & 0 \\ & 2 & 0 \\ 0 & 0 & \end{bmatrix} \tag{5.10}$$

Recall that a blank element denotes index $-\infty$, i.e. no dependency of the variable or any of its derivatives/integrals! ■

■ **EXAMPLE 5.4 (CONTRIVED DAE - INDEX MATRIX)** *We will also compute the index matrix of the contrived DAE in Example 5.2.*

$$\begin{aligned} sin(\textstyle\int x + x\ddot{y}) - \int z = 0 \\ \dot{x} + \textstyle\iint z - 2 = 0 \\ x^2 + y^2 - 4 = 0 \end{aligned}$$

We identify the base variables $Y = (x, y, z)$.

We concentrate on the first equation $f_1 = sin(\int x + x\ddot{y}) - \int z = 0$:

Considering x, the leading derivative index must be 0 since there is no dependency on higher derivatives of x than $x = x^{(0)}$ itself, and the leading derivative is

$$\frac{\partial f_1}{\partial x^{(0)}} = cos(\textstyle\int x + x\ddot{y})\ddot{y} \ .$$

Considering y, the leading derivative index is 2, since there is no dependency on higher derivatives of y that $\ddot{y} = y^{(2)}$. Note that the dependency can be concluded without always computing the leading derivative symbolically!

Considering z, the leading derivative index is -1, since there is no dependency on higher derivatives of z than $\int z = z^{(-1)}$.

This gives us the first row of the index matrix:

$$[0, 2, -1]$$

We can compute the indices in the remaining rows, and we get:

$$M = \begin{bmatrix} 0 & 2 & -1 \\ 1 & & -2 \\ 0 & 0 & \end{bmatrix} \tag{5.11}$$

■

We should now be able to put up the index matrix for any DAE system using derivative, integral or mixed notation.

5.4 Properties of the index matrix

The index matrix is ideal for representing the operations needed for basic index reduction. Three fundamental operations are needed:

- Re-ordering to integral form.
- Detecting structural ODE.
- Differentiation of equations.

The following sections explain how these operations performed on the original DAE can be represented by corresponding operations on the index matrix.

5.4.1 Re-ordering to integral form

As we have seen, the integral form is relevant for determining if a *general implicit DAE system* is essentially an *implicit ODE system*. See Chapter 4.

We shall see how we can use the index matrix representation to bring the system on integral form. As it will be shown, it is only a matter of "scaling" the columns of the index matrix, or more precisely, adding/subtracting a constant from each column, such that the *maximum element in each column are zero.*

Assume the general DAE system as (5.4):

$$f(\ldots, y^{(1)}, y^{(0)}, y^{(-1)}, \ldots, t) = 0. \tag{5.12}$$

Bringing a system on integral form is a matter of choosing base variables[3]. Consequently, we want to choose a new set of base variables $\widetilde{y} = (\widetilde{y}_1, \widetilde{y}_2, \ldots, \widetilde{y}_n)$, such that the system (5.12) can be put on integral form:

$$f(\widetilde{y}, \widetilde{y}^{(-1)}, \widetilde{y}^{(-2)}, \ldots, t) = 0 \ . \tag{5.13}$$

Each new base variables $\widetilde{y}_j$ in $\widetilde{y}$, are chosen from the integrals or derivatives of the original base variable y_j indicated by the integer constant m_j:

$$\widetilde{y}_j = y_j^{(m_j)} \quad , \text{for } j = 1, 2, \ldots, n \ . \tag{5.14}$$

Note that m_j can be both positive, zero or negative. We can deduce the m_j's from considering the (5.12) directly or we can use the index matrix representation:

Assume that the index matrix of (5.12) is M and the index matrix of the integral form (5.13) is $\widetilde{M}$. Note that M will have both positive, zero and negative entries including $-\infty$.

What are the requirements on $\widetilde{M}$ to be on integral form? First note that $\widetilde{M}$ must only have *negative or zero entries.* Second, every base variable $\widetilde{y}_j$ must be used in at least one equation. This, means that each column of M,

[3] See chapter 4.

must have at least one element of index-0. Combining these two requirements, it means that the *maximum element of each column in $\widetilde{M}$ must be zero.*

Assuming that the base variables are chosen according to (5.14), then $\widetilde{M}$ can be constructed from M simply by subtracting each m_j from column j. This can be deduced from Definition 5.2 on the leading derivative index.

Consequently we compute $\widetilde{M}$ from M, by choosing each m_j as the maximum element of column j of M:

$$\boxed{\begin{aligned} m_j &= \max_i \ M_{i,j} &&, \text{for } j = 1, 2, \ldots, n \\ \widetilde{M}_{i,j} &= M_{i,j} - m_j &&, \text{for } i, j = 1, 2, \ldots, n \ . \end{aligned}} \tag{5.15}$$

This ensures that the maximum elements of each column of $\widetilde{M}$ is exactly zero.

The following example shows how a DAE system can be put on integral form using the index matrix representation:

■ **Example 5.5 (Pendulum system - integral form)** *We continue using the index matrix* (5.10) *found in Example 5.3. We recall that the base variables were* $Y = (x, y, F)$.

$$M = \begin{bmatrix} 2 & & 0 \\ & 2 & 0 \\ 0 & 0 & \end{bmatrix}$$

We identify the maximum index of each row:

$$m_1 = 2, \quad m_2 = 2, \quad m_3 = 0$$

The new index matrix can be computed by subtracting the $m_1 = 2$ *from column 1,* $m_2 = 2$ *from column 2, and* $m_3 = 0$ *from column 3. This gives the index matrix*

$$\widetilde{M} = \begin{bmatrix} 0 & & 0 \\ & 0 & 0 \\ -2 & -2 & \end{bmatrix} \tag{5.16}$$

We see that no positive elements remain, and that there is a zero element in each column, as is required for the integral form.

Correspondingly we have chosen a new set of base variables

$$\widetilde{Y} = (Y_1^{(2)}, Y_2^{(2)}, Y_3^{(0)}) = (\ddot{x}, \ddot{y}, F)$$

■

■ **Example 5.6 (Contrived DAE - integral form)** *We found the index matrix* (5.11) *for the contrived DAE in Example 5.4:*

$$M = \begin{bmatrix} 0 & 2 & -1 \\ 1 & & -2 \\ 0 & 0 & \end{bmatrix}$$

The base variables were $Y = (x, y, z)$.

We want to bring this system to integral form. We compute the maximum elements in each column:

$$m_1 = 1, \quad m_2 = 2, \quad m_3 = -1$$

To bring the system to integral form we simply subtract these values from the corresponding columns, and we get:

$$\widetilde{M} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & & -1 \\ -1 & -2 & \end{bmatrix} \tag{5.17}$$

Again, we see that no positive elements remain and that there is a zero element in each column. The new base variables are

$$\widetilde{Y} = (Y_1^{(1)}, Y_2^{(2)}, Y_3^{(-1)}) = (\dot{x}, \ddot{y}, \textstyle\int z) \ .$$

■

5.4.2 Detecting structural ODE

It is almost trivial to use the index matrix to determine if the represented system is a structural ODE.

We assume that M is the index matrix representing a general implicit DAE on integral form (!):

$$f(y, y^{(-1)}, y^{(-2)}, \ldots, t) = 0 \ . \tag{5.18}$$

How to obtain the integral form is described in the previous section. For determining the structural ODE we normally use the leading Jacobian matrix w.r.t the non-integral variables $J = \partial f / \partial y$ or the corresponding structural representation S and examine for structural regularity. See section 5.2.

Now, realise that the structural representation S of the Jacobian matrix $J = \partial f / \partial y$, can trivially be obtained from the index matrix M. Since M is on integral form, the index-0 elements of M correspond to the entries in S:

$$S_{i,j} = \begin{cases} 1 & , \text{if } M_{i,j} = 0 \\ 0 & , \text{otherwise} \end{cases}$$

Hence, by using the index matrix M directly we have an alternative condition for the structural ODE:

Lemma 5.4 (Structural ODE and index matrix) *Assume a general implicit DAE system* (5.18) *on integral form. Let M be the index matrix of the system, then the system is a structural ODE, if and only if the rows or columns of M can be permuted such that there are index-0 elements on the entire diagonal.*

The proof is trivially realised from the above discussion and Definition 5.1.

For detecting the structural ODE we can still choose to obtain S and use Algorithm 2.2, but in the algorithm section a version dedicated to the index matrix representation is presented.

In the later Example 5.10, we will see how the structural ODE is detected, during the index reduction process.

5.4.3 Representing differentiations

This section will show why the index matrix is also ideal for representing the differentiation of equations, which is fundamental for index reduction. We will see that differentiations are just a matter of "scaling" the rows, by adding a constant to each row, corresponding to the number of differentiations.

Assume a general DAE system as (5.4), not necessarily on integral form:

$$f(\dots, y^{(1)}, y^{(0)}, y^{(-1)}, \dots, t) = 0.$$

The index matrix of the system is M. M will have both positive, zero and negative entries including $-\infty$.

We construct a new DAE system

$$\widetilde{f}(\dots, y^{(1)}, y^{(0)}, y^{(-1)}, \dots, t) = 0.$$

by differentiating each equation a number of times, denoted by p_i:

$$\widetilde{f}_i = \frac{d^{p_i} f_i}{dt^{p_i}} = f_i^{(p_i)}$$

The index matrix $\widetilde{M}$ of the new system, can be computed from the original index matrix M, by simply adding the number of differentiations p_i of equation

i to the corresponding row of the index matrix:

$$\boxed{\widetilde{M}_{i,j} = M_{i,j} + p_i \ , \quad \text{for i,j=1,2,\ldots,n} \ .} \tag{5.19}$$

Particularly, for each differentiation we simply increment every element of the corresponding row in the index matrix by one. Also note that, if $M_{i,j} = -\infty$, then $\widetilde{M}_{i,j} = -\infty + p_i = -\infty$.

Leading derivatives under differentiations. To realise (5.19) it is important to understand what happens considering the leading derivative, and especially that *leading derivatives are unchanged during differentiations.*

We concentrate on a single equation

$$f_i(\ldots, y^{(1)}, y^{(0)}, y^{(-1)}, \ldots, t) = 0 \tag{5.20}$$

and a variable y_j. If the leading derivative exist for variable y_j and the index is $m = M_{i,j}$, then the leading derivative is

$$\frac{\partial f_i}{\partial y_j^{(m)}} \ . \tag{5.21}$$

Subsequently (5.20) is *not* dependent on "higher derivatives" of y_j than $y_j^{(m)}$. Therefore we can write (5.20) as

$$f_i(y_j^{(m)}, Y) = 0 \tag{5.22}$$

where Y contains all other variables, except $y_j^{(m)}$ and the "higher derivatives" $y_j^{(m+1)}$, $y_j^{(m+2)}$, $\ldots$.

Now we differentiate (5.22) once with respect to time, using the chain rule:

$$\begin{aligned} \frac{d}{dt} f_i(y_j^{(m)}, Y) &= \frac{\partial f_i}{\partial y_j^{(m)}} \frac{dy_j^{(m)}}{dt} \ + \ \frac{\partial f_i}{\partial Y} \frac{dY}{dt} \\ &= \frac{\partial f_i}{\partial y_j^{(m)}} y_j^{(m+1)} \ + \ F(y_j^{(m)}, Y) = 0 \end{aligned} \tag{5.23}$$

Here F denotes the second term, only dependent variables in Y and possibly $y_j^{(m)}$. The result (5.23) can be written as

$$f_i^{(1)}(y_j^{(m+1)}, y_j^{(m)}, Y) = 0$$

We note, that differentiation introduces dependency on $y_j^{(m+1)}$, but that there is no dependency on "higher derivatives". Therefor, the *leading derivative index* of variable y_j in the new differentiated equation is

$$\widetilde{M}_{i,j} := m + 1 = M_{i,j} + 1 \ , \quad \text{for } j = 1, 2, \ldots, n \ . \tag{5.24}$$

From (5.23) we also see that the leading derivative of the differentiated equation is

$$\frac{\partial f_i^{(1)}}{\partial y_j^{(m+1)}} = \frac{\partial f_i}{\partial y_j^{(m)}} \tag{5.25}$$

Observe that the leading derivative is unchanged from (5.21).

We conclude that when an equation is differentiated once, the leading derivative index is incremented by one for all base variables y_j, i.e. all elements of the corresponding *row* of M are incremented by one, and the *leading derivative is unchanged.*

We shortly note that if $M_{i,j} = -\infty$, then there is no dependency on any derivative/integral of the variable y_j. This will remain after differentiation and therefore $\widetilde{M}_{i,j} := -\infty + 1 = -\infty$.

Generalisation to multiple differentiations. The conclusions are easily generalised to multiple differentiations: Generalisation of (5.24) for several differentiations gives us the result (5.19). The *unchanged leading derivatives* during differentiation, can by induction from (5.25) be generalised to an arbitrary number of differentiations p:

$$\boxed{\frac{\partial f_i^{(p)}}{\partial y_j^{(m+p)}} = \frac{\partial f_i}{\partial y_j^{(m)}} \qquad , \text{for } p = 0, 1, 2, \ldots} \tag{5.26}$$

Here $m = M_{i,j}$ is the leading derivative index before the first differentiation.

The following example shows what happens under differentiation, and how this is represented using the index matrix:

■ **Example 5.7 (Pendulum system - differentiations)** *Recall the equations for the pendulum system in Example 5.3,*

$$\begin{aligned} \mathrm{M}\ddot{x} + \frac{Fx}{L} &= 0 \\ \mathrm{M}\ddot{y} + \frac{Fy}{L} + Mg &= 0 \\ x^2 + y^2 - L^2 &= 0 \end{aligned}$$

and the corresponding index matrix (5.16) *after the system was brought to integral form - see Example 5.5:*

$$M = \begin{bmatrix} 0 & & 0 \\ & 0 & 0 \\ -2 & -2 & \end{bmatrix}$$

Note that this means that the base variables are $Y = (\ddot{x}, \ddot{y}, F)$.

We concentrate on the last of the tree equations. We differentiate this equations w.r.t. time two times:

$$\begin{aligned} & f_3 &&= x^2 + y^2 - L^2 = 0 \\ \Rightarrow \quad & f_3^{(1)} &&= 2x\dot{x} + 2y\dot{y} = 0 \\ \Rightarrow \quad & f_3^{(2)} &&= 2\dot{x}^2 + 2x\ddot{x} + 2\dot{y}^2 + 2y\ddot{y} = 0 \end{aligned}$$

We can now observe that the leading derivative w.r.t. the variables x *and* y, *are unchanged:*

$$\begin{aligned} \frac{\partial f_3}{\partial x} &= \frac{\partial f_3^{(1)}}{\partial \dot{x}} = \frac{\partial f_3^{(2)}}{\partial \ddot{x}} = 2x \\ \frac{\partial f_3}{\partial y} &= \frac{\partial f_3^{(1)}}{\partial \dot{y}} = \frac{\partial f_3^{(2)}}{\partial \ddot{y}} = 2y \end{aligned}$$

We also observe that the leading derivative index is successively incremented by one. One can assert how the last row of the index matrix is changed during differentiations:

$$\begin{aligned} & [-2, -2, -\infty] \\ \Rightarrow \quad & [-1, -1, -\infty] \\ \Rightarrow \quad & [\ \ 0, \ \ 0, -\infty]. \end{aligned}$$

We observe that after these two differentiations the complete index matrix is

$$\widetilde{M} = \begin{bmatrix} 0^* & & 0 \\ & 0 & 0^* \\ 0 & 0^* & \end{bmatrix}$$

*We note, that we can e.g. choose to permute the *-marked elements onto the diagonal, and therefore the new system is a* structural ODE. *By further investigations we will observe that the system is in fact a* regular implicit ODE. *We conclude that these differentiations correspond to the index reduction, which we are interested in.* ■

5.5 Index reduction using index matrix

We have seen that we can obtain the underlying structural ODE by performing appropriate differentiations. Now we want to find a general rule for which equations to differentiate to perform the index reduction and obtain the underlying structural ODE.

5.5.1 Trace of index matrix

First we introduce a characteristic property of a DAE system. The *trace*[4] and especially the *maximum trace* of the index matrix:

Assume that M represents a general implicit DAE system. Now, we try to permute non-infinity elements, i.e. elements $M_{i,j} \neq -\infty$ onto the diagonal. We can now compute the diagonal sum, i.e. the trace of the index matrix:

$$tr\ M = \sum_{i=1,n} M_{i,i}\ .$$

For now, we assume that it is possible to permute non-infinity elements onto the entire diagonal.

Furthermore we assume that the system is on *integral form*. This means that all elements $M_{i,j} \leq 0$. Therefore the trace is generally negative or zero. Furthermore, the maximum value, zero, is obtainable if and only if we can permute index-0 elements onto the entire diagonal, and this again means that the system is a *structural ODE* - the end of the index reduction process.

Therefore the *maximum trace* is interesting:

Definition 5.5 (Maximum trace) *Assume an index matrix M representing a general implicit DAE system on integral form. By the* maximum trace

$$\max\ tr\ M\ ,$$

we understand the maximum value for the trace considering permutations of columns or rows.

Note that the definition of maximum trace assumes that the system is on integral form.

The conclusion from the discussion above can be stated as a lemma:

[4] From linear algebra, meaning the sum of the diagonal.

Lemma 5.6 (Maximum trace and structural ODE) *Assume a general implicit DAE system on integral form. This system is a structural ODE, if and only if the maximum trace is* 0.

The following example should exemplify the use of the maximum trace definition and the Lemma just stated:

■ **Example 5.8 (Contrived DAE - maximum trace)** *We continue Example 5.6. We have brought the system onto integral form and the corresponding index matrix is:*

$$\widetilde{M} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & & -1 \\ -1 & -2 & \end{bmatrix}$$

We can choose between a number of permutations to compute the trace. The elements to put on the diagonal are indicated by the marks:

$$\begin{bmatrix} -1^* & 0 & 0 \\ 0 & & -1^* \\ -1 & -2^* & \end{bmatrix} \quad \begin{bmatrix} -1 & 0^* & 0 \\ 0 & & -1^* \\ -1^* & -2 & \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0^* \\ 0^* & & -1 \\ -1 & -2^* & \end{bmatrix}$$

The maximum trace comes from one of the last two permutations and has the value -2. We can according to Lemma 5.6 conclude that the system is not *a structural ODE.* ■

5.5.2 Choosing equations to differentiate

We shall now see that proper differentiations will increment the maximum trace until we reach the underlying structural ODE.

We keep an eye on the maximum trace: If max $tr\ M < 0$, then the system is not a structural ODE, and we have not reached the underlying structural ODE yet.

As long as max $tr\ M < 0$ the leading Jacobian matrix and the corresponding structural representation S must be structurally singular. Also M will be structurally singular if we consider only the index-0 elements[5].

Since M is structurally singular we can detect an MSS subset using e.g. Algorithm 2.2 considering only the index-0 elements. From the theory of

[5]See section 5.4.2 on obtaining S from M.

structural analysis, we know that an MSS subset containing k rows of M, will have (index-0) entries in *exactly* $k-1$ columns. See Lemma A.8.

We will now perform a combined differentiation and re-ordering to integral form:

(1) We differentiate the k equations in the MSS subset by incrementing all elements of the corresponding k rows of M by one. This will change all index-0 elements in these rows into index-1 elements. As a result M will suddenly end up with *index-1 elements in exactly* $k-1$ *columns.*

(2) Since positive indices are not allowed in the integral form, we need to *re-order* to integral form. This is done by appropriately "scaling" these $k-1$ columns, i.e. decrementing all elements of these columns by one.

Finally, note what happens to an arbitrary trace, i.e. the sum of the diagonal elements, assuming an arbitrary permutation, during these differentiations:

Each time we differentiate an MSS subset, we (1) increment all elements in k rows of M by one, and (2) decrement all elements in $k-1$ columns by on. This will (1) increment the trace by k, and (2) decrement the trace by $k-1$. In total this gives an *increment by* one *each time an MSS subset is differentiated* for an arbitrary trace.

Since this goes for an arbitrary trace independent on permutation, it also goes for a permutation corresponding to the maximum trace, and consequently we state the following lemma:

Lemma 5.7 *Consider an index matrix M on integral form. Differentiating an MSS subset of equations and subsequently re-ordering to integral form, will increment the maximum trace by one.*

Since we can always detect an MSS subset if the maximum trace is not zero, we can continue this process until the maximum trace is zero, meaning that the underlying structural ODE is obtained.

5.5.3 Structurally regular DAE

Previously, we just assumed that the trace is well-defined, i.e. that we can permute non-infinity elements onto the diagonal. As we will see, this assumption is very reasonable and it defines a fundamental property for every well posed DAE system:

Definition 5.8 (Structurally regular DAE) *Assume a general implicit DAE system, and the corresponding index matrix M. If there exist a permutation of the rows and columns such that the entire diagonal contains non-infinity-index elements, i.e.*

$$M(i,i) \neq \infty \text{ , for all } i \text{ ,}$$

then the system is called a structurally regular DAE. *Alternatively, the system is called a* structurally singular DAE.

Note, that there is no assumption on using integral form. This term is also used in [Pan88], and the definitions correspond but are note identical. This definition also includes high-order DAEs and uses the index matrix as a basis.

Structural regularity is an important property of every reasonable DAE model. If a DAE model is structurally singular, then using corollary A.2, there must be at least one subset of equations which relates to fewer base variables than it contains equations. This is discussed further in section 7.2.2.

Existence of underlying structural ODE. Note that it is always possible to compute a finite trace, trM, of a structurally regular DAE. We further notice, that we can always put such a system on integral from by "scaling" the columns and still keep non-infinity elements on the diagonal. Now we can permute and find the maximum trace, and we can use the process described in section 5.5.2, to increase the maximum trace until we have obtained the underlying structural ODE.

This motivates a central theorem of structural index analysis:

Theorem 5.9 (Existence of underlying structural ODE) *Assume a general implicit DAE system. There exist an* underlying structural ODE *obtainable by basic index reduction, i.e. proper differentiations of subsets of equations, if and only if the DAE system is* structurally regular.

Proof. The theorem states a double implication. We have already argued that structural regularity implies existence of underlying structural ODE, using the well-defined trace and differentiation of MSS subsets.

We need to argue that we can never obtain an underlying structural ODE by basic index reduction of a structurally singular DAE: A structurally singular DAE will have at least one $-\infty$-element on the diagonal of the index matrix independent on the permutation. Since basic index reduction only allow differentiations of equations and re-ordering to integral form, and since non of these operations can change $-\infty$-elements to non-infinity elements, then every system obtained from these operations, will still have at least one $-\infty$-elements on the diagonal.

The double implication is proven. □

This theorem concerns essentially the same as Theorem 4.2 in [Pan88], but it is believed that this proof is more complete[6].

Assuming that any reasonable DAE is structurally regular, we can conclude that every reasonable DAE model, must have an underlying structural ODE.

We give an example of determining structural regularity:

■ **Example 5.9 (Contrived DAE - structural regularity)**
We will check if the contrived DAE from Example 5.4 is structurally regular. We have already computed the index matrix:

$$\begin{bmatrix} 0 & 2 & -1^* \\ 1^* & & -2 \\ 0 & 0^* & \end{bmatrix}$$

We have marked some of the non-infinity elements. We note that these can be permuted onto the diagonal, and therefore the system is a structurally regular DAE. ■

5.5.4 Sufficient and necessary differentiations

That one should differentiate MSS subsets to obtain the underlying ODE was already suggested in [Pan88]. This paper states in the conclusion that this gives the minimum number of differentiations required to obtain the underlying structural ODE. This can be hard to read from the paper, and this gives rise to questions regarding the sufficiency an necessity of differentiating these MSS subsets, lately in [RMB00][7]. Consequently, this section is hoped to contribute to the clarify the proof of this conclusion.

We restate that we limit structural index analysis to concern basic index reduction i.e. differentiations of subsets of equations, i.e. no algebraic manip-

[6] Especially the assumption, used in [Pan88], that non-singular subset remain so after differentiating MSS subsets is vague.

[7] This particular issue is discussed in the later Chapter 7.

ulations, and we use the underlying *structural* ODE as the endpoint of the index reduction process.

We state the fundamental theorem:

Theorem 5.10 (Sufficiency and necessity) *Assume a general implicit DAE. We further assume that the system is a structurally regular DAE.*

To obtain the underlying structural ODE by basic index reduction, it is sufficient and necessary to differentiate the MSS subsets.

Proof. Sufficiency of differentiating MSS subsets should be recognizable from the previous sections. One should notice, that this means sufficiency in determining the underlying *structural ODE*. This structural ODE might actually be a DAE of higher index and possibly even inconsistent. See the discussion in section 7.2.1.

Arguing for necessity is a little harder and will be done in two steps. (1) We prove that at least one equations of an MSS subset must be differentiated. (2) We prove that all equations have to be differentiated. We use the index matrix representation in the argumentation:

(1) Assume the we have identified an MSS subset of an index matrix M, containing k equations/rows. This subset will have index-0 elements in exactly $k-1$ columns.

Because the index matrix of the final underlying structural ODE must have index-0 elements on the entire diagonal, then every subset containing k rows must have index-0 elements in *at least* k columns. Therefore, to obtain the underlying structural ODE from M, we have to produce new index-0 elements in at least one more column of the MSS subset, such that the subset obtains index-0 elements in at least k columns.

Since the only way to produce index-0 elements in a new column of the subset is to *differentiate one or more of these equations*, necessity of differentiating at least one equation of the MSS subset i proven.

But why do we need to differentiate all equations in the MSS subset?

(2) Assume that we do not have to differentiate all equations. We assume that we only have to differentiate a (proper) subset of size $l(< k)$.

Such a subset contains index-0 elements in at least l columns! This can be seen from the definition of the MSS subset, since no proper subset of an MSS subset must be structurally singular, i.e. all subsets of size l must have entries in *at least* l columns.

This means that when differentiating the subset of size l by adding one to all the elements in the corresponding rows, then we introduce index-1 elements in at least l columns. These columns must subsequently be “scaled” by subtracting 1 to keep integral form.

By this re-ordering process the remaining $k - l$ equations/rows, which are not incremented by differentiation, will have all the index-0 elements in these l columns decremented. As a result these remaining $k - l$ equations will end up having index-0 elements in maximally $(k-1)-l$ of the remaining columns, consequently be structurally singular and therefore contain at least one *new* MSS subset, from which at least one equation should be differentiated.

Since we assumed we only had to differentiate l equations this proves a contradiction. We have to differentiate all equations of an MSS subset. □

An interpretation of this result is exactly that differentiating all the MSS subset represents a minimum of differentiations, as stated in [Pan88].

Furthermore, this theorem can be used to define *uniqueness* of the underlying structural ODE, if we define the *unique underlying structural ODE* as exactly the underlying structural ODE obtained by differentiation of MSS subsets.

5.5.5 Index reduction process

We end this section by describing the theory of structural index analysis and reduction by giving the layout of an algorithm for performing the index analysis and reduction. An algorithm could be based on the following steps:

1. Compute index matrix M.
2. Test for structural regularity of DAE.
3. Re-order to integral form by updating M.
4. Check for structural ODE. If confirmed, index reduction ends!
5. Detect MSS subset of equations.
6. Differentiate MSS subset by updating M.
7. Goto step 3.

We also give an example on the index reduction process on a specific example. The following example is continued from Example 5.6.

■ **Example 5.10 (Contrived DAE - index reduction)** *We recall the integral form index matrix (5.17) of Example 5.4. We name it M_{0i} denoting that it is the index matrix at the starting point of the index reduction process and that the system is on integral form.*

$$M_{0i} = \begin{bmatrix} -1 & 0^* & 0 \\ 0 & & -1^* \\ -1^* & -2 & \end{bmatrix}$$

We conclude that this system is a structurally regular DAE, *because we can permute all the elements marked with $\cdot^*$ onto the diagonal. This means that either the system is* a structural ODE *or we can perform index reduction by differentiation of MSS subsets until we have obtained the underlying structural ODE. We also note that the maximum trace is -2 which indicates that we should expect to perform 2 steps of the algorithm to identify the underlying structural ODE.*

For determining if the system is a structural ODE, we construct the corresponding structural representation of the Jacobian matrix by using all the 0-index elements in M_{0i}:

$$S_0 = \begin{bmatrix} & \times & \times \\ \times & & \\ & & \end{bmatrix}$$

We notice that this system is truly structurally singular, since it is impossible to permute non-zero elements onto the entire diagonal. Hence, we can identify an MSS subset. We identify the MSS subset containing only the last equation, i.e. the last row. Notice that it matches the MSS definition 2.10.

We differentiate this equation subset, by adding 1 to the last row of the index matrix and we get

$$M_{1i} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & & -1 \\ 0 & -1 & \end{bmatrix}.$$

For determining whether the system is a structural ODE, we have to ensure that the index matrix is on integral form. Since it has a zero in each column and no positive elements it is on integral form.

We continue by examining if the system represents a structural ODE:

$$S_1 = \begin{bmatrix} & \times & \times \\ \times & & \\ \times & & \end{bmatrix}$$

We notice that the system is still structurally singular. Therefore it is not a structural ODE, and we can identify an MSS subset. The last two rows constitute an MSS subset.

Subsequently, we differentiate these two rows by subtracting one from the last two rows, and we get

$$M_2 = \begin{bmatrix} -1 & 0 & 0 \\ 1 & & 0 \\ 1 & 0 & \end{bmatrix}.$$

We observe that the system is no longer in integral form. The first column is the problem: We observe that the maximum element is 1, and therefore we subtract one from this column. We get

$$M_{2i} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & & 0 \\ 0 & 0 & \end{bmatrix}.$$

To check if the system is a structural ODE we obtain the structural representation of the leading Jacobian:

$$S_2 = \begin{bmatrix} & \times^* & \times \\ \times & & \times^* \\ \times^* & \times & \end{bmatrix}$$

*As seen, the structural matrix is now structurally regular, denoted by the *-marking, and therefore the corresponding system is a structural ODE! Note, that we have used 2 steps as predicted by the maximum trace of the initial system.*

Note also, that during the index-reduction process we have not actually differentiated the equations, but we have obtained sufficient information to perform the symbolic index reduction afterwards. The information can be saved in the vectors m and p. Here p tells the number of differentiations for each equation, and m explains how the base variables of the final system are derivatives of the original base variables. The vector have the values:

$$p_1 = 0, p_2 = 1, p_3 = 2$$

$$m_1 = 1, m_2 = 0, m_3 = 0$$

See section 5.4 describing re-ordering to integral form and differentiations respectively for further description of these vectors. ∎

5.6 Algorithms

The above theory can be used for constructing practical algorithms. In this section versions of all the necessary algorithms are presented.

The algorithms are presented as pseudo code. The syntax is inspired by the Python programming language. See e.g. [vRD01]. This especially goes for the indentation style and loops, but also the associative container (dictionary). As example $A = \{1 : a, 2 : b\}$ means that $A(1) = a$ and $A(2) = b$. The intention is though that the algorithms should be easily transferable to implementations in other programming languages.

The algorithms are not optimized for efficiency. Priorities are simplicity and a close relation to the presented theory. In a later section, computation complexity is discussed.

5.6.1 General input

We start by describing what is expected as global input to the algorithms.

The algorithms are based on the following information of the DAE system of size n equations, and n variables:

- E: A set containing identifiers for all the equations. Identifiers could be the integer values $1, 2, \ldots, n$.
- V: A set containing identifiers for all the variables. Each variable includes all its integrals. Identifiers could be the integer values $1, 2, \ldots, n$.
- M: Assoc. array. The index matrix of the DAE system on integral form. $M(e, v)$ returns the element index of equation e w.r.t. variable v. The ∞-value is returned for undefined element indices. M can be represented by a full $n \times n$-matrix of integers. Choose a special value for representing ∞.

For simplicity of the algorithm we assume that M is already on integral form. Implementing a code for initial reordering to integral form is considered a simple task.

5.6.2 Finding assignments using index matrix

We have realised from the theory of structural index analysis, that detecting structural regularity/singularity and detecting MSS subsets is a central task. An algorithm for doing this was already described in section 2.5.5 as Algorithm 2.2. The fundamental principle is to successively find variable-assignments for each equation.

For simplicity we want to implement a version of the algorithm which uses the index matrix directly, and which can be used for both detecting structural regularity of the DAE, detecting structural ODE, and MSS subsets.

The assignment algorithm is presented as Algorithm 5.1. The algorithm is almost identical to Algorithm 2.2, but differs at two points in line 3 and line

10:

Algorithm 5.1 For finding assignment using the index matrix directly. Is an alternative to using Algorithm 2.2, but it is less efficient.

```
 1: FindAssignment2(e):
 2:     for v ∈ V:
 3:         if AssignmentAllowed(M(e,v)):
 4:             if Assign(v) = None:
 5:                 InSearch(v):=True
 6:                 Assign(v):=e
 7:                 return True
 8:
 9:     for v ∈ V:
10:         if AssignmentAllowed(M(e,v)):
11:             if not InSearch(v):
12:                 InSearch(v):=True
13:                 e2:=Assign(v)
14:                 Result := FindAssignment2(e2)
15:                 if Result = True:
16:                     Assign(v):=e
17:                     return True
18:
19:     return False
```

The algorithm assumes that a function *AssignmentAllowed* is defined before the algorithm is called: The call $AssignmentAllowed(M(e,v))$ should return *True* if and only if an assignment is allowed between equation e and variable v.

Introducing this simplification and flexibility has some costs for the bounds on computational costs. Since each call, ignoring recursion, now costs full n steps in the loop of line 3 and 10, then the computational cost can only be bounded by

$$n^2$$

Recall how the arrays *Assign* and *InSearch* are used from the description in section 2.5.5. Here a description of the algorithm can also be found.

5.6.3 Structurally regular DAE

One of the first things to discover is, whether the DAE system is a structurally regular DAE, i.e. if there exists an underlying structural ODE obtainable by differentiations. See section 5.5.3. As presented in the Theorem 5.9, structural regularity of a DAE requires that we are able to permute non-infinity elements onto the entire diagonal. This again corresponds to the existence of a full assignment, where an assignment can use any element with well defined index $M(i,j) \neq \infty$.

Detecting for structural regularity can consequently be done by Algorithm 5.2. We assume that E, V and M are given. The algorithm initially defines the function *AssignmentAllowed*. We note, that assignments can be done if the element index $M(e,v) \neq -\infty$. Then the algorithm simply runs through the individual equations in E and tries to find an assignment using the new assignment algorithm.

Algorithm 5.2 For detecting structurally regularity of a DAE system using the index matrix representation.

```
StructurallyRegular():
    def AssignmentAllowed(m):
        if m ≠ −∞:
            return True
        else:
            return False

    Assign(:):=None
    for e ∈ E:
        InSearch(:):=False
        Result:=FindAssignment2(e)
        if Result = False:
            return False

    return True
```

5.6.4 Index analysis

Algorithm 5.3 performs the actual index analysis of the DAE system.

Algorithm 5.3 For performing index analysis on the index matrix. Note that the algorithm assumes that the system is structurally regular, and that the index matrix is on integral form.

```
IndexAnalysis():
    Index_e(:):=0, Index_v(:):=0
    Assign(:) = None

    def AssignmentAllowed(m):
        if m = 0:
            return True
        else:
            return False

    for e ∈ E:
        while True:
            InSearch(:)=False
            Result=FindAssignment2(e)
            if Result = True:
                break
            else:
                V_MSS:={v ∈ V | InSearch(v)}
                E_MSS:={e} ∪ Assign(V_MSS)
                for e ∈ E_MSS:
                    Index_e(e):=Index_e(e) + 1
                    for v ∈ V:
                        M(e,v):=M(e,v) − 1
                for v ∈ V_MSS:
                    Index_v(v):=Index_v(v) + 1
                    for e ∈ E:
                        M(e,v):=M(e,v) + 1
```

The principle is to try to find assignments as if the system was a structural ODE. When no assignment can be found we have an MSS subset which should be differentiated. It is important to understand the we do not need to restart the assignment process, and correspondingly reset the $Assign$-array, after differentiating an MSS subset. This is because all the index-0 elements used in previous assignments will still be index-0 elements after differentiation and re-ordering to integral form.

The output of the algorithm are the $Index_v$ and $Index_e$ arrays. These correspond to the m- and p-vectors used in section 5.4 for representing the re-ordering and the prescribed differentiations. In this way $Index_e(e)$ gives the number of times one should differentiate equations e.

5.6.5 Computation complexity

To estimate the computation complexity we initially consider the computation complexity of Algorithm 5.1 This was found to be n^2. It is also possible to bound the number of times the algorithm has to be called: Each time it finds an MSS subset it decreases the maximum trace by one, this means that we have to call the algorithm at least $(\max trM)$ times. But we also need to call the algorithm for each successful assignment of an equation to a variable. This means that the algorithm is called exactly $\max trM + n$ times. This gives a bound on the computational complexity of

$$(\max trM + n)n^2 \ ,$$

where n is the number of equations and base variables. We can also bound the maximum trace. If the smallest (most negative) index of the index matrix is m_{min}, then the maximum trace, being the sum of n elements, is bounded by

$$\max trM \leq n m_{min} \ .$$

Note, that m_{min} indicates the order of the DAE system, i.e. the highest order of integrals appearing in the model.

This altogether indicates a bound on the computational cost proportional to

$$m_{min} n^3$$

Since the index analysis process is generally done once for each DAE model, the computations involved in the analysis will only have to be performed once. This is found to be an acceptable computation complexity.

5.6.6 Use high-order formulation

The reason why this thesis emphasizes the use of a high-order formulation, is that the index reduction process is simpler.

If we rewrite a system which is essentially a high-order system to first-order, we introduce the extra variables and equations. In a differentiation of the corresponding equations these extra variables and equations will also need differentiation. In this sense we introduce even more equations and variables which are trivial and uninteresting.

After having performed the index analysis and index reduction, one might choose to rewrite the (augmented) underlying ODE to first-order. This will not introduce as many trivial equations.

5.6.7 Requirements and failure

There are only a few condition that can cause the Algorithm 5.3 to fail or give doubtful results.

First of all the algorithm assumes that the index matrix is on integral form, i.e. that the columns are "scaled" such that the maximum element in each column is 0.

Another reason for failure is if the system is a structurally singular DAE. This should be checked before the algorithm is applied. Otherwise the algorithm will continue infinitely. As example a row of infinite elements, would continuously be an MSS subset, but nothing would happen during "differentiations".

Finally, we again stress that the underlying structural ODE, need not be a regular ODE, but can algebraically be a higher index DAE. See discussion in section 7.2.1.

5.7 Structural index

A last topic of this chapter will be the definition of the *structural index.* We will present both a definition of structural index, regarding the entire DAE system, and also define structural index of the individual equations and variables.

First the definition concerning the entire system. This definition can have some resemblance with Definition 3.1 of the differentiation index.

Definition 5.11 (Structural index) *Assume a general implicit DAE being structurally regular.*

The (minimum) number of times that we need to differentiate one or more of the equations of the general implicit DAE to obtain the underlying structural ODE, is called the structural index.

Note, that using "minimum" is not really needed, since we imply uniqueness of the underlying structural ODE, performing only the necessary and sufficient differentiations.

The term *structural index* is used for some time in various texts as early as in e.g. [DG86], in e.g. [UKM95] and lately in e.g. [RMB00].

Generally the use of the term covers the same phenomenon, but more precisely this definition follows the tradition of using the minimum number of differentiations as prescribed by [Pan88], and defined also in [FB99].

In discussion of DAE systems it is also useful to define structural index for the individual equations and variables of the DAE system.

Definition 5.12 (Structural index of equation) *Assume a general implicit DAE being structurally regular. Concentrate on a specific equation.*

Then the (minimum) number of times that the equation has to be differentiated to obtain the underlying structural ODE, is called the structural index of the equations.

Note that the maximum structural index of a single equations matches the structural index of the complete system according to Definition 5.11.

Definition 5.13 (Structural index of variable) *Assume a general implicit DAE being structurally regular. Concentrate on a variable being either a base variable or an integral/derivative of a base variables.*

Then the (minimum) number of times that the variable has to be differentiated to appear as a base variable in the final underlying structural ODE, is called the structural index of the variable.

It obvious that these indices are essentially the information contained in the vectors p and m defined in section 5.4 and returned as output by Algorithm 5.3. The following section will show how these individual indices can be used in practice.

5.7.1 Continuity and differentiability requirements

The perhaps simplest use of these structural index definitions is to discuss, which continuity and smoothness demands to expect from variables and equations/residuals in the model. In the following we assume that structural index analysis perform exactly the necessary and sufficient differentiations to obtain the "real" underlying regular ODE.

In this case the *structural index of an equation* indicates the smoothness demands for equations or residuals. If an equation has structural index 2, then it means that it should at least be two times differentiable, since it is the two times differentiated version which goes into the underlying ODE. Note that it also indicates the smoothness demands of permit-able perturbations of the equations.

Note, that if we use e.g. complicated fitted tables/functions then the structural index analysis can be done without actually knowing the representation of the function. After the analysis the smoothness demands will be obvious and the table/function can be designed with this in mind.

The *structural index of a variables* indicates the least smoothness to expect from a variables. This is also a nice property to classify the variables of a DAE model.

A variables of structural index 0, i.e. a base variables of the underlying ODE, should be at least integrable. This generally means continuous with a finite number of discontinuities. See the discussion in section 2.1.4. In this way a variable of structural index 1, 2, etc, must be 1 , 2, etc. times differentiable, since they are integrals of the base variables.

Yet another reason for dealing with this concept is that it can increase the understanding of the more traditional indices as the differentiation and the perturbation index. This is discussed in chapter 7.

5.8 Summary

Structural index analysis of DAE systems can explain which subsets of the equations should be differentiated to perform index reduction and obtain *the underlying ODE* or at least the underlying *structural* ODE.

A *structural ODE* denotes a DAE system characterized by having a structurally regular Jacobian matrix considering the non-integral variables. The structural ODE can be considered the closest parallel to the general (implicit) ODE, detectable by structural analysis. Therefore it represents a reasonable endpoint of the structural index reduction process. Using the structural ODE as endpoint is discussed in the later Chapter 7.

Structural index analysis is based on structural information of the general high-order implicit DAE system. As presented, structural index analysis is dependent on the *index matrix representation*. This representation is based on the definition of the so called *leading derivative* and *leading derivative index*, which describe the dependencies of specific equations on specific base variables including derivatives/integrals of these.

We learn how the index matrix is ideal for representing the basic operations as reordering to integral form, differentiation of equations with respect to determining the underlying structural ODE. We also learn that choosing equations to differentiate for index reduction is a question of identifying MSS subsets considering only index-0 elements of the index matrix.

We have defined the fundamental property *structural regularity* for DAE systems. We have seen that this property is valid for all reasonable DAE systems, and that this property at the same time guarantees the *existence* of an underlying structural ODE. Furthermore, we learned that differentiation of MSS subsets is sufficient and necessary for obtaining the underlying structural ODE. In this sense the underlying structural ODE is unique, since it is obtained by a minimum number of differentiations.

Algorithms for performing the index analysis process are described. These are based on the index matrix representation. Analysis of computational costs limits the work to n^3 for each analysis process.

In this respect we also motivate that one uses the high-order formulation of the DAE during the analysis process, since it minimizes the number of extra equations introduced and keeps the system smaller and easier to survey.

The results from the index analysis process is *not* the underlying ODE directly. It is solely *information* on which subsets of equations to differentiate to obtain the underlying structural ODE.

Structural index analysis also motivates the definition of the term *structural index*. This is the structural parallel to the differentiations index. We also introduced a structural index definition for the individual equations and variables. These concepts are nice concepts for discussing DAE models and can explain which smoothness to require and expect for equations and variables.

Overall, this chapter is considered a unique description of the theory for structural index analysis developed under this Ph.D. project. It is a contribution to the theory for structural analysis of DAEs, in the respect that it introduces a simple representation of the structural information on DAEs based on a matrix instead of graph theory, and gives a more consistent and complete presentation of the theory for structural analysis of DAEs and presents algorithms based on this representation.

Chapter 6

Augmented underlying ODE

This chapter describes theory and practical methods for using the *augmented underlying ODE (AUODE)* to solve the initial values problem for DAE systems.

The fundamental assumption is that we are able to perform index reduction on the original DAE and thereby obtain the underlying ODE. During the reduction process all the original an intermediate constraints are saved and used as an augmentation of the underlying ODE.

Using the augmented underlying ODE has some advantages over just using the underlying ODE, but methodologies for using the AUODE are not as developed as methods for ODEs in general. This chapter represent some general aspects of using the AUODE, but also some intermediate investigations to help developing and improving methods for the AUODE, here especially the *dummy derivative method*. These investigation are believed to be among the original contributions from the work in this Ph.D project.

6.1 Definition

By the *augmented underlying ODE* we understand the combined system of the *underlying ODE* and all the *reduced constraints* from the index reduction process. The term augmented underlying ODE is e.g. used in [MS93].

Using integral notations, the general high-order augmented underlying

ODE has the following representation:

$$\begin{aligned} f_0(y, \textstyle\int y, \iint y, \ldots, t) &= 0 \\ g(\textstyle\int y, \iint y, \ldots, t) &= 0 \end{aligned} \tag{6.1}$$

Here f_0 represents the underlying ODE and g represents all the reduced constraints, i.e. all the differentiated equations from from the index reduction process.

Sometimes, if the index reduction is done only by differentiations, as assumed in structural index analysis, then the reduced constraints represented by g can be separated into several functions g_1, g_2, ...:

$$\begin{aligned} g_1(\textstyle\int y, \iint y, \ldots, t) &= 0 \\ g_2(\textstyle\iint y, \ldots, t) &= 0 \\ &\vdots \end{aligned} \tag{6.2}$$

The constraints represented by g_1 are all the constraints that appear in f_0 after *one* differentiation, the constraints represented by g_2 are all the constraints that appear in g_1 after one differentiations and therefore they appear in f_0 after *two* differentiation, and so on for g_3, ...

Without loss of generality we will often concentrate on methods for the first-order form:

$$\begin{aligned} f_0(y, \textstyle\int y, t) &= 0 \\ g(\textstyle\int y, t) &= 0 \end{aligned} \tag{6.3}$$

We can do this without loss of generality, since we can transform a high-order AUODE to the first-order from, much like we transform a high-order ODE into a first-order ODE. See section 2.2.

6.2 Obtaining the augmented underlying ODE

How to obtain the augmented underlying ODE is almost identical to the question of how to obtaining the underlying ODE.

We can perform the index reduction "manually" by analysis of the original DAE. In this process we might use both differentiations and algebraic manipulations, but every time we reduce an equation by differentiation we should add it as a new function in g.

We can also perform the index reduction more automatically. Since we have just described the theory of structural index analysis in chapter 5, we will describe how to use this technique to obtain the AUODE.

6.2.1 Using structural index analysis

To use structural index analysis we make the important initial assumption: We assume that the result of the structural analysis *is* the real underlying regular ODE.

Recall that the information obtained from structural index analysis, i.e. Algorithm 5.3, are two index arrays $Index_e$ and $Index_v$ corresponding to the two vector p and m used in the theory of section 5.4. p_i explains the number of times that equation i has to be differentiated to obtain the underlying ODE. m_j explains how the j'th non-integral variables in $\tilde{y}$ related to the corresponding non-integral variable in y of the original DAE.

This information can be used to obtain the augmented underlying ODE as follows. Assume the original DAE system on integral form using Eich-Führer notation style:

$$f(y, y^{(-1)}, y^{(-2)}, \ldots, t) = 0. \tag{6.4}$$

Then the augmented underlying ODE is

$$\begin{aligned} f_0(\widetilde{y}, \widetilde{y}^{(-1)}, \widetilde{y}^{(-2)}, \ldots, t) &= 0 \\ g_1(\widetilde{y}^{(-1)}, \widetilde{y}^{(-2)}, \ldots, t) &= 0 \\ g_2(\widetilde{y}^{(-2)}, \ldots, t) &= 0 \\ &\vdots \end{aligned} \tag{6.5}$$

where f_0, g_1, g_2 and so on can be obtained by differentiating each equation of the original ODE a number of times, denoted by $p_i = Index_e(i)$. For $i = 1, 2, \ldots, n$ we have

$$\begin{aligned} f_{0,i} &= f_{,i}^{(p_i)} \\ g_{k,i} &= f_{,i}^{(p_i - k)}, \quad \text{for k=1,2, } \ldots, \; p_i \,. \end{aligned} \tag{6.6}$$

Here $f_{,i}$ means the $i'th$ equation of the original DAE. Note that the different g_k will not contain the same number of equations. The new base variables $\widetilde{y}$ are given by

$$\widetilde{y}_j = y_j^{(m_j)}$$

where $m_j = Index_v(j)$.

6.2.2 Considerations on differentiation

Performing the actual differentiations is the next question. Index reduction is often criticised because automating the process apparently requires some sort of symbolic differentiation. This is true, but today there are several tools available for doing this in an efficient way.

Naturally, the first elementary idea is to deduce the expressions for the differentiated equations either by hand or using some tool for symbolic computations. This will do for small systems and a few differentiations, but for larger systems, the generation and evaluation of the derivatives will be cumbersome and in-efficient.

There are especially two things to look into, when implementing the differentiations appropriately:

- Automatic differentiation.
- Use of leading derivative matrix.

Automatic differentiation. The technique called automatic differentiation is probably less known than it deserves. The authors experience suggests that the name is not sufficiently describing or at least under-emphasizes the main advantages of the technique,

- Efficient computations of derivatives,

more than automatic derivation of derivatives.

The fundamental principle is that when describing an expression in a tree-like structure, with one node for each elementary part of the expression, then either computing the corresponding derivatives at each node w.r.t. the input variables (forward automatic differentiation), or computing the derivative of the entire expression w.r.t. to the value at each node (backward automatic differentiation), can be done by a finite number of operations per node.

Taking advantage of this, and using e.g. backward automatic differentiations, it limits the cost of computing e.g. the derivatives of an expression w.r.t. to all entering variables (corresponds to obtaining the gradient) to maximally five times the cost of evaluating the expression itself. In practice this factor is even smaller - around 1.5. For this results and theory of automatic differentiation see e.g. [Gri89].

For computations regarding the augmented underlying ODE this is very useful. We need to compute the time derivative of the individual equations, and this can be done using the gradient. Assume that we want to differentiate a function $g(y,t)$:

$$\frac{d}{dt}\, g(y,t) = \bigtriangledown g \cdot (\dot{y}, 1) \ ,$$

Consequently it will cost maximally five times the cost of evaluating g to evaluate the derivative. For second-order derivatives it will be 5×5 times etc. Recall, that in practice this factor is much smaller.

There are several software tools that implements automatic differentiation. For example FADBAD, see [BS96]. Alternatively, automatic differentiation can actually be done "manually", by hand, writing up expressions for the derivatives in the appropriate sequence and style. Considering the name, it can seem almost as a paradox, that automatic differentiation can be done "manually".

Leading derivative matrix and Jacobian matrix. One of the interesting findings, is that even though the underlying ODE is obtained by possibly several differentiations of each individual equation, then the Jacobian matrix of the underlying ODE, will always consist of first-order partial derivatives of the original DAE equations. This still assumes that index reduction is done solely by differentiations.

This is due to the *unchanged leading derivatives* under differentiations as described in (5.26). We can define the *leading derivative matrix* as the functional matrix using the index matrix M:

$$L_{i,j} = \begin{cases} \dfrac{\partial f_i}{\partial y_j^{(M_{i,j})}} & \text{if } M_{i,j} \neq -\infty \\ 0 & \text{if } M_{i,j} = -\infty \end{cases}$$

In this way L consists of all the leading derivatives of the original DAE. No matter which differentiations are performed this matrix will not change.

The leading Jacobian matrix for the underlying ODE $\widetilde{J} = \partial \widetilde{f} / \partial \widetilde{y}$ can easily be found for any derived system obtained by differentiations of the original equations. It can be obtained from the index matrix M and the leading derivative matrix L:

$$\widetilde{J}_{i,j} = \begin{cases} L_{i,j} & , \text{if } \widetilde{M}_{i,j} = 0 \\ 0 & , \text{otherwise} \end{cases}$$

This can actually be used for the reduced constraints as well. This means that we will never need to derive the Jacobian matrix by considering the differentiated equations.

The conclusion is that the relevant computations for the augmented underlying ODE, can readily be done using automatic differentiation for evaluating the high-order derivatives of the original equations and using the first-order partial derivatives of the initial equations for computing the leading Jacobian matrix. This should be considered in an implementation of e.g. integration methods for AUODEs.

6.3 Using the augmented underlying ODE

Having obtained the augmented underlying ODE, this section briefly describes how the AUODE can be used for solving problems concerning the original DAE. This concerns both the problem of finding consistent initial values and solving the initial values problem (IVP).

6.3.1 Consistent initial values

As already described, finding consistent initial values is one of the central problems in the solution of IVPs for DAE systems. This is the case whether one chooses to use the BDF methods or the Runge-Kutta methods as direct methods, see section 3.5, or one chooses to use index reduction and use e.g. the underlying ODE as will be discussed shortly.

The conditions for consistent initial values are already discussed in section 3.6.2. To specify the consistent initial value problem one can simply consider either the reduced equations or the entire augmented ODE as an under-determined equation system which must be solved:

$$\begin{aligned} F(X) &= \begin{bmatrix} f(y, \int y, t) \\ g(\int y, t) \end{bmatrix} = 0 \\ X &= (y, \int y, t) \end{aligned} \tag{6.7}$$

Note that in (6.7) we have not specified which variables should be dependent or independent. This decision should be postponed to the concrete problem and situation. Methods for solving an under-determined system of equations is discussed in the background material, section 2.4.6.

A nice terminology, used in connection with the under-determined equations system for the consistent initial values, is the notion of the *dynamic degree of freedom*. This simply correspond to the degree of freedom for the under-determined system (6.7). This terminology is used in e.g. [UKM95].

6.3.2 Using only the underlying ODE

It should be emphasized that using just the underlying ODE and standard integration methods for solving the IVP, will often be a reasonable alternative to using the entire augmented underlying ODE. Theoretically, the only requirement is that consistent initial values are computed before the integration starts.

As described in section 3.6.2, the problems in practice can be drift-off, especially if differentiations have introduced instability around the reduced constraints.

Only, if it is very important that there is no drift-off and that in-stability is not too large, then one should consider using the full AUODE for solving an IVP. Using the underlying ODE alone one can still use the reduced constraints for checking the size of drift-off.

Remember that the underlying ODE will generally be an implicit ODE[1] and therefore the numerical methods should still apply to implicit ODEs.

6.3.3 Using the augmented underlying ODE

If we want to prevent the problems of drift-off when solving an IVP, then we should consider using the augmented underlying ODE. It contains the reduced constraints, and these can be used during integration to remove or reduce drift-off.

Since just using the underlying ODE, given consistent initial values, determines a unique solution, then the augmented underlying ODE, including additional constraints must be somehow (dynamically) *over-determined*. In this respect the AUODE belongs to the class of over-determined ODEs or *ODEs with invariants*.

ODEs with invariants is a classical research topic. Consequently different methodologies are investigated and described. One can split the methods into some fundamental classes:

- *Local state space methods:* One of the methods for solving over-determined ODE systems, is by choosing some local coordinates which lie on the manifold specified by the constraints $g(y,t) = 0$. This basically corresponds to reducing some of the variables from the ODE. Using the local coordinates the resulting system constitutes and ODE on the manifold.
- *Stabilization methods:* The fundamental principle of stabilisation is to build extra dynamics into the underlying ODE, so that a solution will stay on the manifold and thereby avoid drift-off.

[1] Often it will be a linearly implicit ODE.

- *Projection methods:* Typically, these methods will initially take a step using the underlying ODE, and subsequently project the solution point onto the manifold, after each step is taken. The problem is usually to choose in which direction to project.

To read more in detail about these methods, see e.g. [PW99] or [HW96, p. 472 ff.]. Considering concrete implementations, there are no defacto standard implementations of these methods.

The next section describes a special method for the augmented underlying ODE called the *Dummy derivative method* which is studied during this Ph.D. project and is used in the methodology. It has much in common with the local state space method, but can also be compared to the projection methods.

6.4 Dummy derivative method

One way to come around the over-determinedness of the augmented ODE system is by "decoupling" some of the integral variables from their derivatives, in this way making their derivatives to the *dummy derivatives.*

The notion and method of dummy derivatives appear in [MS93]. In this article the method is highly connected to the index reduction process. The approach presented here has basis in the augmented underlying ODE or general under-determined equations systems of a similar form.

6.4.1 Description of principle

Assume that we have an augmented underlying ODE on first-order integral form:

$$f_0(y, \textstyle\int y, t) = 0 \tag{6.8a}$$

$$g(\textstyle\int y, t) = 0 \; . \tag{6.8b}$$

Now we pick out the variables $\int y_{dd}$ from the integral variables $\int y$, so that

$$\frac{\partial g}{\partial \int y_{dd}} \text{ is square and } \textit{non-singular.} \tag{6.9}$$

Note that the number of variables in $\int y_{dd}$ should match the number of reduced constraints represented by g. This splits $\int y$ and correspondingly y into two sets

$$\textstyle\int y = (\int y_{dd}, \int y_n), \quad y = (y_{dd}, y_n) \; ,$$

and the system (6.8) can be rewritten as

$$f_0(y_{dd}, y_n, \int y_{dd}, \int y_n, t) = 0 \tag{6.10a}$$
$$g(\int y_{dd}, \int y_n, t) = 0 \tag{6.10b}$$

Assuming that the system is consistent, $\int y_{dd}$ can be implicitly computed from (6.10b) according to the requirement (6.9) and the *implicit function theorem* - Theorem 2.6. Therefore, we can "cheat" and rewrite (6.10) such that the variables of $\int y_{dd}$ no longer appear as integrals of y_{dd}, but as non-integral variables instead. For this purpose we introduce y_{ci}:

$$y_{ci} = \int y_{dd} \,.$$

Rewriting system (6.10) gives:

$$f_0(y_{dd}, y_n, y_{ci}, \int y_n, t) = 0 \tag{6.11a}$$
$$g(y_{ci}, \int y_n, t) = 0 \tag{6.11b}$$

Finally, we observe that this system is now simply a *general implicit ODE system*, which can be represented by the form

$$\tilde{f}(\tilde{y}, \int \tilde{y}, t) = 0 \tag{6.12}$$

where $\tilde{y} = (y_{dd}, y_n, y_{ci})$. The name *dummy derivatives* applies to the variables in y_{dd} since they are no longer used as derivatives, and the name *cheating integrals* applies to y_{ci}, since they are no longer considered as integrals of the y_{dd}'s.

Note that this system can be solved by traditional numerical integration, being able to handle general implicit ODEs. Such traditional integration routines will typically compute all the integrals in the vector $\int \tilde{y} = (\int y_{dd}, \int y_n, \int y_{ci})$, but we do not need to compute all these values. Instead of $\int y_{dd}$ we have y_{ci}, and $\int y_{ci}$ was never really needed. Therefore it would be nice, if the integration routines had a way to specify is the integrals are not needed.

The principle of choosing dummy derivatives can also be applied directly to the augmented ODE on *high-order* form:

$$f_0(y, \int y, \iint y, \ldots, t) = 0 \tag{6.13a}$$
$$g(\int y, \iint y, \ldots, t) = 0 \tag{6.13b}$$
$$\tag{6.13c}$$

To choose the cheating integrals we just have to find integral variables $y_{ci} = (\int y_{dd1}, \iint y_{dd2}, \dots)$ such that $\frac{\partial g}{\partial y_{ci}}$ is square and *non-singular*. The different indices $dd1$ and $dd2$ on $\int y_{dd1}$ and $\iint y_{dd2}$ emphasize that the choice of cheating integrals from $\int y$ is independent of the choice from $\iint y$. Having found the variables of y_{ci}, then (6.13) can be changed into a general implicit ODE on high-order form.

6.4.2 Choosing dummy derivatives

The central problem of the dummy derivative method is to choose the dummy derivatives, i.e. choose the cheating integrals.

The main objective is to fulfill the non-singularity requirement (6.9). We cannot though, expect a given choice of dummy derivatives to satisfy this requirements along the entire solutions curve. A given choice of dummy derivatives which satisfies the non-singularity requirement for some values of $\int y$, might violate the requirement for others. Following a solution curve in time, the typical situation will be that at some point the requirement is violated.

We illustrate this by looking at the pendulum system:

■ **Example 6.1 (2-D pendulum system - dummy derivatives)**
Consider the 2-D pendulum presented as Example 1.1 in the introduction. The underlying ODE is

$$M\ddot{x} + F\frac{x}{L} = 0 \tag{6.14a}$$

$$M\ddot{y} + F\frac{y}{L} + Mg = 0 \tag{6.14b}$$

$$\dot{x}^2 + x\ddot{x} + \dot{y}^2 + y\ddot{y} = 0 \tag{6.14c}$$

and the reduced constraints are:

$$x\dot{x} + y\dot{y} = 0 \tag{6.14d}$$

$$x^2 + y^2 - L^2 = 0 \tag{6.14e}$$

We have used derivative notation, but we will can easily subscribe the model variables to the second-order augmented underlying ODE on integral form:

$$\begin{aligned} f_0(Y, \textstyle\int Y, \iint Y, t) &= 0 \\ g(\textstyle\int Y, \iint Y, t) &= 0 \end{aligned} \tag{6.15}$$

We assign the variables as follows:

$$Y = (\ddot{x}, \ddot{y}, F), \quad \textstyle\int Y = (\dot{x}, \dot{y}, \int F), \quad \iint Y = (x, y, \iint F)$$

Now we have to find the cheating integrals Y_{ci} among the all integral variables in $\int Y$ and $\iint Y$, such that $\frac{\partial g}{\partial Y_{ci}}$ becomes non-singular. First, we compute the functional Jacobian matrix of the reduced constraints in g with respect to all *the integral variables:*

$$\left[\begin{array}{c|c} \dfrac{\partial g}{\partial \int Y} & \dfrac{\partial g}{\partial \iint Y} \end{array}\right] = \begin{array}{c} \begin{array}{cccccc} \dot{x} & \dot{y} & \int F & x & y & \iint F \end{array} \\ \left[\begin{array}{ccc|ccc} x & y & 0 & \dot{x} & \dot{y} & 0 \\ 0 & 0 & 0 & 2x & 2y & 0 \end{array}\right] \end{array}$$

Based on this, we can now consider different choices of the two integral variables for the vector Y_{ci}, the so-called cheating integrals:

- $Y_{ci} = (\dot{x}, \dot{y})$: *This choice of cheating integrals gives*

  $$\frac{\partial g}{\partial Y_{ci}} = \begin{bmatrix} x & y \\ 0 & 0 \end{bmatrix}.$$

  This choice is obviously inappropriate since $\frac{\partial g}{\partial Y_{ci}}$ can never be regular.

- $Y_{ci} = (\dot{x}, x)$: *This choice gives*

  $$\frac{\partial g}{\partial Y_{ci}} = \begin{bmatrix} x & \dot{x} \\ 0 & 2x \end{bmatrix}.$$

  By computing the determinant we see that this functional matrix is regular when $2x^2 \neq 0$. This means that this choice can be used for an solution as long at it does not cross $x = 0$.

- $Y_{ci} = (\dot{x}, y)$: *This choice gives*

  $$\frac{\partial g}{\partial Y_{ci}} = \begin{bmatrix} x & \dot{y} \\ 0 & 2y \end{bmatrix}.$$

  Again the determinant gives us that this matrix is regular whenever $2xy \neq 0$. This means that the solution must not cross $x = 0$ nor $y = 0$. In this respect this choice is worse than the previous.

- $Y_{ci} = (x, y)$: *This choice gives*

  $$\frac{\partial g}{\partial Y_{ci}} = \begin{bmatrix} \dot{x} & \dot{y} \\ 2x & 2y \end{bmatrix}. \tag{6.16}$$

  This matrix is regular whenever $2\dot{x}y - 2x\dot{y} \neq 0$. Considering also the reduced constraints $\dot{x}x + y\dot{y} = 0$ and $x^2 + y^2 - L^2 = 0$, this means that the solution can go everywhere except the point $(\dot{x}, \dot{y}) = (0, 0)$, where the pendulum is at rest.

From start was already obvious that we did not have to consider the variables $\int F$ or $\iint F$, since these does not appear in any of the equations. We do not either need to investigate the remaining combination since they look like the ones already treated.

If we e.g. choose $Y_{ci} = (x, y)$, then we can locally, i.e. when $2\dot{x}y - 2x\dot{y} \neq 0$, transform the augmented underlying ODE to the second-order ODE on integral form

$$\tilde{f}(\tilde{Y}, \textstyle\int\tilde{Y}, \iint\tilde{Y}, t) = 0 \ .$$

The variables are assigned as follows:

$$\tilde{Y} = (\ddot{x}, \ddot{y}, F, x, y), \quad \textstyle\int\tilde{Y} = (\dot{x}, \dot{y}, \cdot, \cdot, \cdot), \quad \iint\tilde{Y} = (\cdot, \cdot, \cdot, \cdot, \cdot)$$

Here $\cdot$ denotes that we do not need to compute this integral variable. We see that it is only necessary to consider an ODE of first-order. ■

The conclusion from this example must be that no choice is really appropriate as a general choice, but that many of them will have a local domain of validity. We can say that the augmented underlying ODE is transformed into a local ODE by choosing dummy derivatives.

Consequently, we need to focus on what happens when we reach this limit of validity for the local ODE, and possibly how to adaptively change the choice of dummy derivatives. This is studies in the following sections.

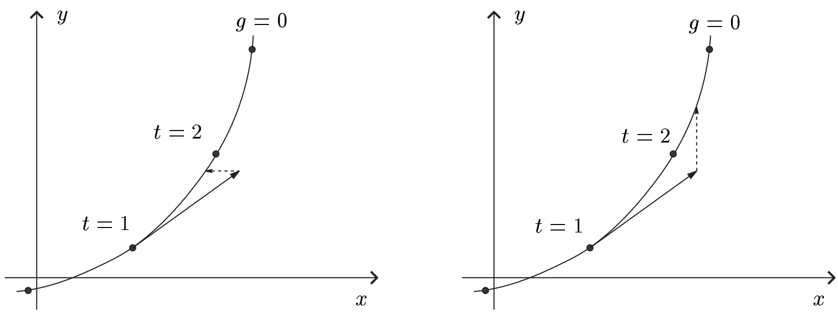
6.4.3 Minimizing projection length

It is interesting to investigate what effect the choice of dummy derivatives can have on the local error committed by numerical integration. This can be done by e.g. comparing a step using the underlying ODE and a step using the local ODE obtained by choosing dummy derivatives.

Since the numerical solution of the local ODE is similar to integration of any implicit ODE the order of convergence must be the same. That the cheating integrals are computed as an implicit function of the other integrals will not change the order of convergence.

In the following we will illustrate that using the dummy derivative method, the choice of dummy derivatives, can have an considerable impact on the step and the local error.

Using dummy derivatives with simple integration schemes can be compared to using a projection method. We will shortly see that the choice of dummy derivatives, can have great impact on the projection lengths, and consequently influences the step and local error.



(a) x is cheating integral. (b) y is cheating integral.

FIGURE 6.1: Taking an *explicit* Euler step using the local ODE with dummy derivatives/cheating integrals instead of the underlying ODE.

We investigate what happens to a small augmented underlying ODE system when taking a step using the explicit and implicit Euler method: The system is given by

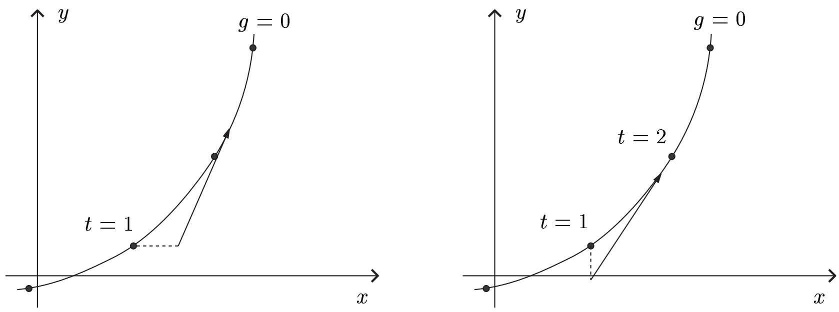
$$\begin{aligned} \dot{x} &= f_x(x, y, t) \\ \dot{y} &= f_y(x, y, t) \\ 0 &= g(x, y) \ . \end{aligned} \tag{6.17}$$

First, we observe what happens when we take a step from $t = 1$ to $t = 2$ using and explicit Euler step. This is illustrated in Figure 6.1: The dots at $t = 1$ and $t = 2$ indicate the exact results. The solid arrows on figures (a) and (b) corresponds to a step using the (underlying) ODE alone. If we use x as cheating integral, see Figure (a), the explicit Euler step corresponds to projecting the step of the underlying ODE along the x-axis to the constraint $g = 0$. Alternatively, choosing y as the cheating integral, we project the step along the y-axis onto $g = 0$.

By observing the endpoints of the two projections, it becomes clear that the choice of dummy derivative/cheating integral has consequence for the result and the local error. We realise that the more near we are to a singularity the longer the projection lengths will be.

Figure 6.2 (a) and (b) shows the same situation for the implicit Euler method. Again, we notice that different choices gives different results and "projection" lengths. We again realise that around a singularity, the choice becomes increasingly important.

Note that using dummy derivatives can not always be considered as a



(a) x is cheating integral. (b) y is cheating integral.

FIGURE 6.2: Taking an *implicit* Euler step using the local ODE with dummy derivatives/cheating integrals.

projection method in the traditional understanding. An alternative to the projection length can be the difference between taking a step using the underlying ODE and the local ODE.

6.4.4 Selection strategy

For coping with the problems of choosing dummy derivatives we have to find a suited strategy. Three strategies are:

1. Make a selection which avoids to violate the non-singularity requirement, but this is not necessarily always possible.
2. Make local selection of dummy derivatives and update the selection when problems arise, i.e. near the singularity.
3. Frequently update the selection of dummy derivatives to e.g. continuously minimize the projection length.

The principle studied in this project is strategy 2.

The problem of choosing cheating integrals is similar, or identical, to the problem of finding a dependent variables in an under-determined equation system: The objective is to make the Jacobian matrix $\partial g/\partial y_{ci}$ regular. We can locally compute the rectangular Jacobian matrix of g w.r.t. all the integral variables, and then identify a set of columns which are linearly independent, and subsequently choose the corresponding variables as the cheating integrals.

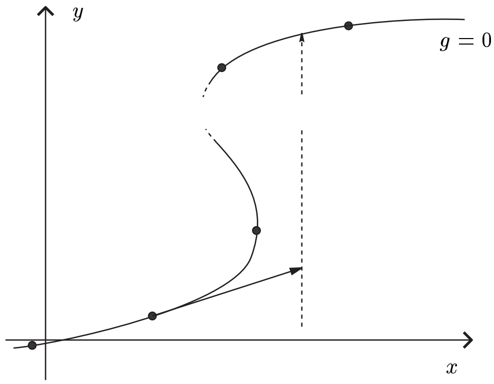


FIGURE 6.3: Illustration of how the dummy derivative method might fail near a singularity. Either there exist *no* solution or there might exist a solution on a distant part of the $g = 0$-manifold.

The method for choosing linearly independent columns can be varied. In the experimental tool Simpy described in Chapter 8, Gaussian elimination with full pivotation is used. The columns are scaled before the elimination process starts. This method works for the tested systems. For details on this and other principles for choosing independent columns look into [GL96] or [GMW91].

It is not guaranteed that this method minimizes the projection length, but at least from a numerical linear algebra point of view, this should ensure that e.g. back-solution of the linear equations which are used in the Newton iterations are well-conditioned and stable.

6.4.5 Near a singularity

What happens when the numerical solution of the local ODE that uses a certain choice of dummy derivatives, comes near or passes a singularity? This is illustrated on Figure 6.3. Either there will be no solution for the cheating integral, or there might exist a solution on a distant point of the manifolds of the constraints.

From the point of view of the local ODE, this jump to a distant part of the manifold is in principle a valid solution. Seen as a local ODE, there is no demand that this variable should not have discontinuities, and therefore we cannot expect the numerical integration routines to give any clear warning signals.

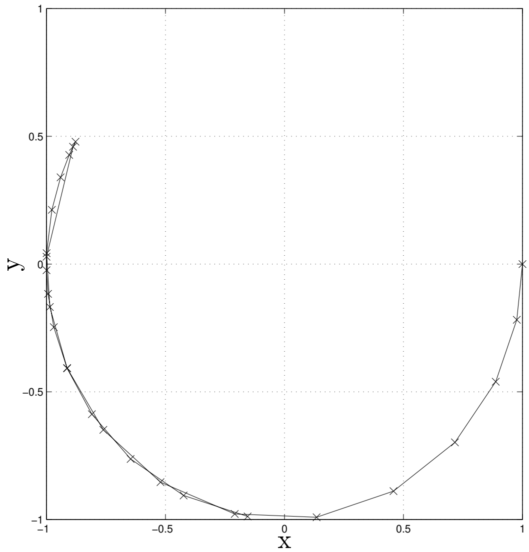


FIGURE 6.4: State space graph of the numerical solution for the 2-D pendulum system solved using x and y positions as cheating integrals. Note that a discontinuity suddenly arises in the position variables in the top left corner.

Of course seen from the original augmented ODE, we would expect the cheating integrals to be continuous although they are "cheating". Therefore we are interested in way to detect when such illegal jumps appear. The situation where no solution is found is not a problem, since it will be discovered by the non-linear solver.

Figure 6.4 and Figure 6.5 shows what happens in a specific example of solving an IVP for the pendulum system. See Example 6.1. The choice of cheating integrals are the two position variables p_x and p_y. As we learned from Example 6.1, the singularity will occur when the velocity is zero. This corresponds to the upper left corner in Figure 6.4 and the approximate time $t = 3.3$ in Figure 6.5.

From the figures we see that at the point, we have a discontinuity in the position variables as well as in the acceleration variables.

The initial value is $(\dot{x}, \dot{y}, x, y) = (0, -1, 1, 0)$. The integration is done by explicit steps using the 5. order stage of a DOPRI54 with step-length $h = 0.2$ and Newton iterations for each computation of the implicitly given derivatives.

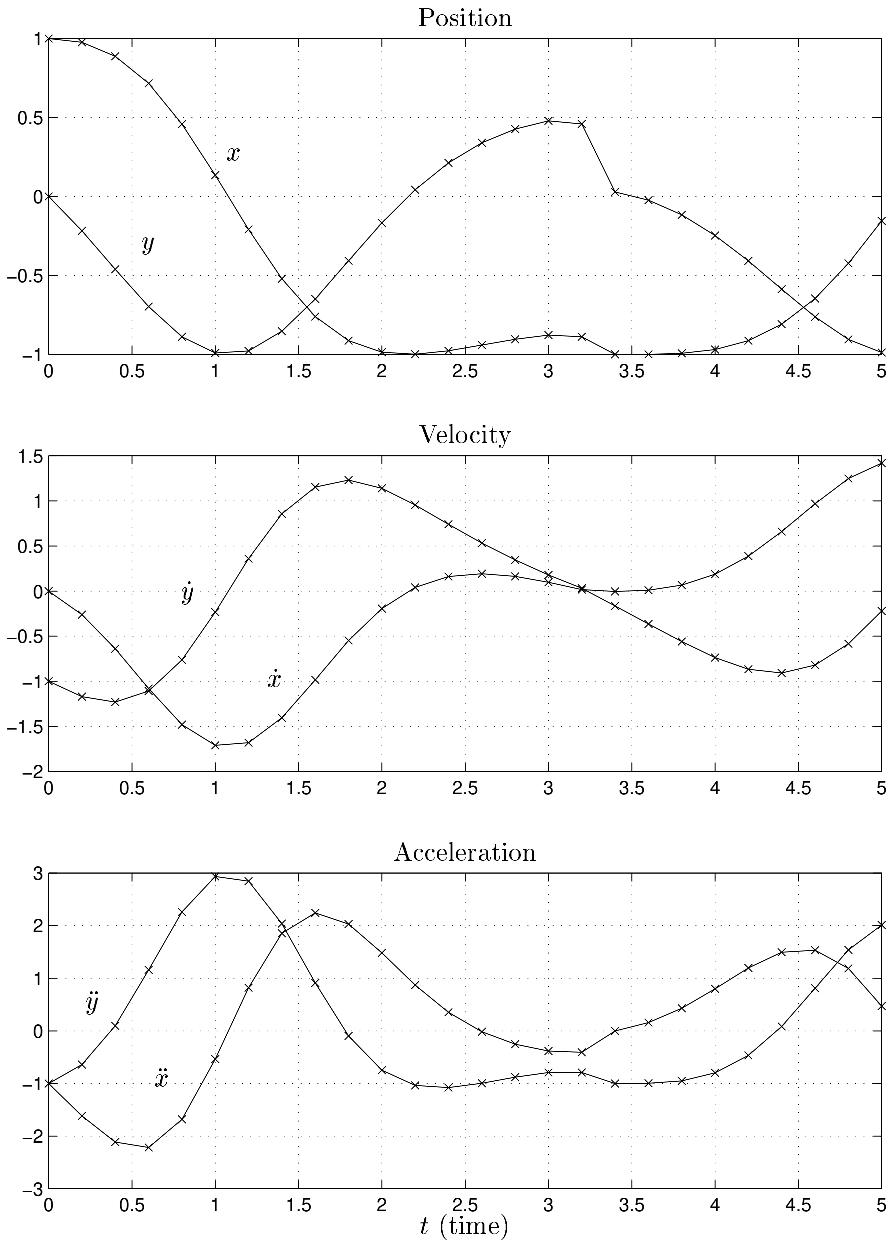


FIGURE 6.5: Time graph of the numerical solution for the 2-D pendulum system solved using x and y positions as cheating integrals. Note the discontinuity in the position and acceleration and the continuity of the velocity variables at approx. $t = 3.3$.

6.4.6 Detecting singularity

It is interesting to investigate what happens during the numerical integration of the local ODE, especially when approaching the bounds of the validity domain, i.e. begin nearly singular.

Four quantities are examined as a basis for detecting when the solution is nearly singular and it is time to change the dummy-derivatives:

- The condition number for the leading Jacobian matrix.
- The embedded error estimates. (Runge-Kutta methods)
- The deviation from underlying ODE step.
- The special embedded underlying ODE stage. (Runge-Kutta methods)

These quantities are studied in practice as a singularity is approached. The experiments are again performed with the pendulum system using p_x and p_y as cheating integrals, corresponding to section 6.4.5.

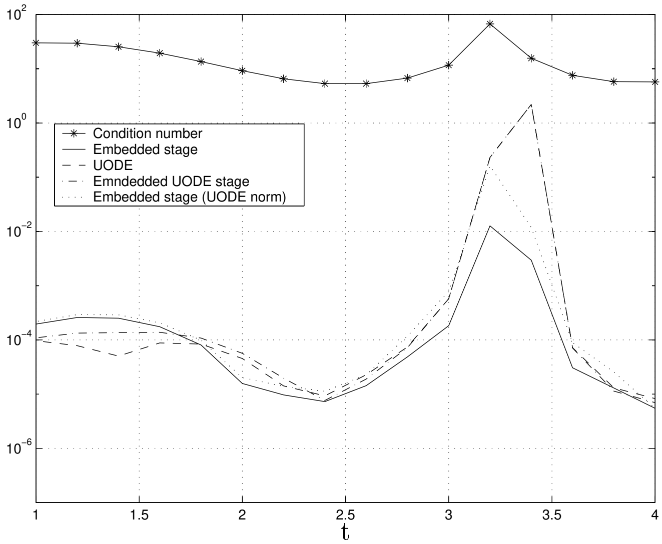
The results are shown in Figure 6.6. The DOPRI54 method is used with step-sizes (a) $h = 0.2$ and (b) $h = 0.1$. The singularity appears at approximately $t = 3.3$.

Condition number. The condition number of the leading Jacobian matrix for the local ODE is a straight forward "measure" of the regularity of the Jacobian. There are several ways to compute an estimate for the condition number. Choosing an appropriate estimate is important for the efficiency of the code. Many linear algebra codes supply some estimate for the condition number. In the following we have used the ratio between the maximum and minimum singular value.

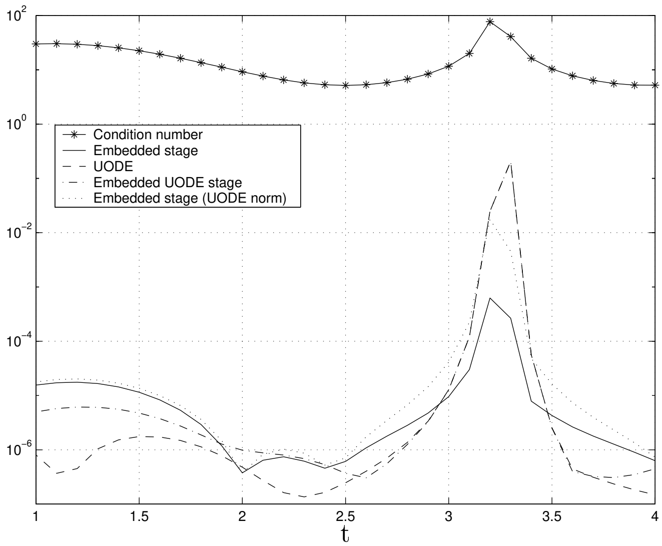
From Figure 6.6 we can see that this property in not well suited for indicating that we are near a singularity. It does not change considerably.

Embedded error estimate. The embedded error estimate is available for many Runge-Kutta methods. In Figure 6.6, the graphs labeled "Embedded stage", we can see how the 5. order embedded error estimate of DOPRI54 reacts near the singularity. Some impact can be seen.

Normally this norm of the embedded error estimator only concerns the integral variables, but since some of the integral variables are "cheating", it is interesting to see what happens, if we instead use a norm where the cheating integrals are included. This is investigated and can be seen in Figure 6.6 as



(a) Step-size $h = 0.2$.



(b) Step-size $h = 0.1$.

FIGURE 6.6: Special properties when stepping near a singularity (see Figure 6.5) of the localized ODE for the 2-D pendulum system. Except the condition number, the graphs describe some norm of the deviation from the final (5. order) stage of a DOPRI54 step, and different special stages. One might consider using (some of) these properties as an indication of when to choose new dummy derivatives/cheating integrals.

the graphs labeled "Embedded stage (UODE norm)"[2]. From the results we can see that the impact on the error estimator, from the singularity, is larger using this norm.

Deviation from underlying ODE. By this concept we try to measure the difference of taking a step using the local ODE obtained by using dummy derivatives and a step using the underlying ODE. From Figure 6.6, the graphs labeled "UODE", it can be seen that the singularity have quite some impact on this quantity, which therefore forms a reasonable basis for detecting singularities. The norm corresponds to the underlying ODE, i.e. includes also cheating integrals.

Embedded underlying ODE stage. The problem of computing the deviation from the underlying ODE step, is that we essentially have to compute the step twice: one time for the local ODE and one time for the underlying ODE.

Therefore an estimate of this deviation is proposed. Assume that we have computed the all stages including the final stage of a Runge-Kutta method. Then the cheating integrals are not integrated. Now recompute only the final stage but this time compute also the cheating integrals as if they were normal integrals. This is a quite cheap operation. Now we take the norm of the deviation between these two results.

In Figure 6.6, the graphs labeled "Embedded UODE stage", we can see that this quantity is remarkably affected by the singularity, and therefore it is a potential candidate as an indicator of when to change dummy derivatives.

Conclusion and future work. These experiments and investigations are only preliminary. More experiments need to be done, but generally the impression is that especially the embedded underlying ODE stage is relevant.

Yet investigations regarding appropriate implementation of this update in more advanced adaptive stepping schemes are necessary.

In the experimental software tool Simpy, described in Chapter 8, an integration scheme for the augmented underlying ODE, based on dummy derivatives, updated by indication of the embedded underlying ODE stage is implemented. By appropriate choice of tolerances the integration succeeds. The update tolerance on this quantity is set to 1/10 of the local error tolerance, and the step is recomputed using new dummy derivatives if the deviation exceeds the local error tolerance.

[2]This norm will correspond to the norm used for the underlying ODE (UODE).

6.5 Summary

This chapter described how the *augmented underlying ODE (AUODE)*, can be used to solve the initial value problem for the original DAE system. The augmented underlying ODE is the combination of the *underlying ODE* and the all *reduced constraints* which are differentiated in the index reduction process.

The AUODE can be obtained from the original DAE by use of the results from structural index analysis. Representing and performing the differentiation should be don appropriately. One should e.g. consider using automatic differentiation and the leading derivative matrix.

We acknowledge that the problem of finding *consistent initial values* for the original DAE system, corresponds to finding values, that satisfies all the equations of the augmented underlying ODE including the reduced constraints. Essentially these equations form an under-determined system of non-linear equations when considering all variables as independent.

Generally, it is possible to solve an IVP for the original DAE, only by using the underlying ODE and consistent initial values, but for some systems one might experience problems with certain phenomena: (1) Drift-off and (2) poor stability properties of the underlying ODE.

To overcome these problems, we consider including the constraints in the solution process, and use the complete augmented underlying ODE system. The augmented underlying ODE essentially belongs to a classical topic from theory of ODEs called *ODEs with invariants*. For this kind of system, several methods exist. Some of the basic principles are (1) Local state space form, (2) Stabilization and (3) Projection.

We describe and investigate a specific methodology called the *dummy derivatives method*. The basic idea is to decouple a number of derivative/integral variable-pairs (*dummy derivatives/cheating integrals*), such that the *cheating integrals* are determined by the reduced constraints instead of the discretization formulas of the numerical method. After choosing dummy derivatives, the augmented ODE system can essentially be transformed into an implicit ODE, and simply solved by standard numerical methods for implicit ODEs.

The problem of choosing dummy derivatives/cheating integrals is treated. Possible strategies are discussed. One problem is, that the implicit ODE might become singular at a point where the cheating integrals are no longer well-determined by the reduced constraints. This means that the choice of cheating integrals only has local validity. Therefore at this point of singularity (or before) the choice of dummy derivatives/cheating integrals should be reconsidered.

Special measures for detecting this point are investigated. It has been seen that e.g. the condition number of the leading Jacobian matrix from the implicit ODE system can be changing rapidly so that one might step over this point, with strange behaviour of the solution as a result. Much more appropriate seems, in some form, the difference between taking a step using the pure underlying ODE and using the dummy derivative system. It is discussed how an estimate of this kind could be introduced at a low cost in a Runge-Kutta step, and in this respect be incorporated in a stepping algorithm.

The original contributions in the work presented, is the idea of defining and using the Leading derivative matrix when deriving the Jacobian matrix. Also the investigations concerning of the dummy derivative method and detection of singularities, is hopefully a valuable contribution to the continued development of the dummy derivative method and, in general, methods for the augmented underlying ODE.

Chapter 7

Discussion of methodology

This chapter is intended a more general discussion of different aspects concerning the methodology presented in this thesis. This concerns both advantages and drawbacks and special requirements for the method to work.

We start by giving an overview of the methodology.

7.1 Methodology overview

During the thesis a general approach or methodology for analysis of DAEs has been proposed. This section will give an outline of the methodology. It has three major steps:

- Index analysis and index reduction.
- Finding consistent initial values.
- Solving an IVP corresponding to the DAE.

7.1.1 Index analysis and reduction

First, assume that a *general implicit DAE system* is given. As described in Chapter 4 it is non-trivial to assign model variables to the general implicit DAE. Therefore, it is recommended that one considers using the *integral form*. This should improve the possibility that we detect if the system is essentially an *implicit ODE* and not a *proper DAE system*. See Figure 7.1.

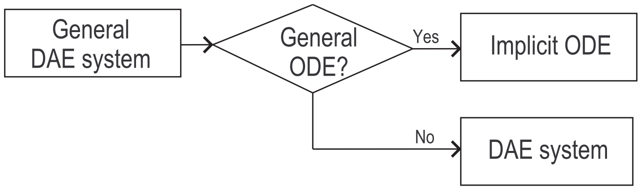


FIGURE 7.1: Check if the general DAE system is essentially a general implicit ODE.

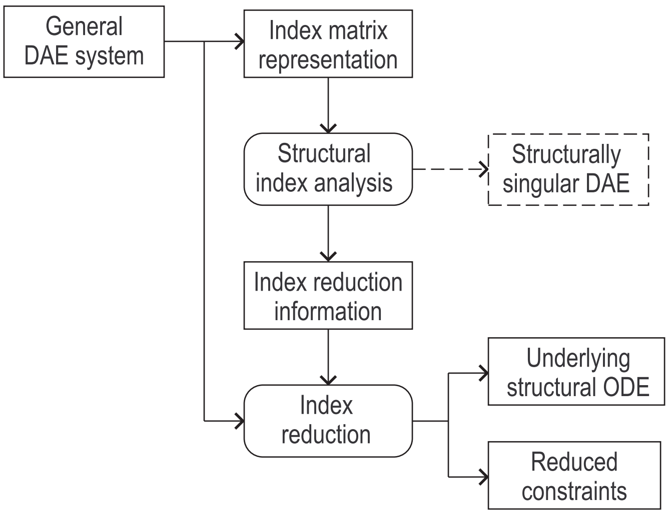


FIGURE 7.2: Performing structural index analysis and index reduction.

Next, if the system is a proper DAE system, a fundamental element of the methodology, is the use of *index reduction*, i.e. successive differentiations of some of the equations until we obtain the *underlying ODE*.

Index reduction consist of two steps: (1) *structural index analysis* and (2) the actual *index reduction*, i.e. differentiations of equations. The process is described in Figure 7.2.

The result of the index analysis is the *index reduction information*, i.e. information on which equations to differentiate. Based on this information the actual index reduction can be performed. It is not necessary to perform the relevant differentiations "explicitly" in traditional symbolic understanding. When efficiency is important, one should rather consider alternatives as e.g. *automatic differentiation*.

As seen the analysis might conclude that the DAE system is *structurally singular*. It is one of the fundamental methodology assumptions that every reasonable DAE system is *structurally regular*. It is argued why this is a reasonable condition in later section 7.2.2.

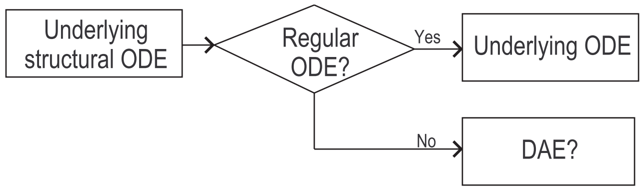


FIGURE 7.3: Is the underlying structural ODE a regular ODE and consequently the underlying ODE? If not, it might be a DAE of higher index for which structural analysis cannot be used further.

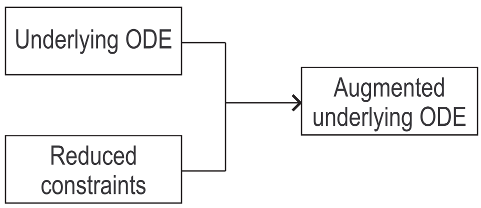


FIGURE 7.4: Constructing the augmented underlying ODE (AUODE).

As seen the reduction process results in two sets of equations: (1) The *underlying structural ODE* and (2) the *reduced constraints*. Here, the general assumption in the methodology is that the underlying structural ODE is a regular ODE, and consequently the underlying structural ODE is the actual underlying ODE. See Figure 7.3. The reduced constraints represent the equations which are differentiated (or reduced) under the .

Finally, we introduce the notion of the *augmented underlying ODE (AUODE)*, consisting of the underlying ODE *augmented* with the reduced constraints. See Figure 7.4.

7.1.2 Consistent initial values

To find the consistent initial values (CIVs), the methodology proposes that one sets up an *under-determined equation system*.

This equation system should as a minimum include the reduced constraints. In this case we only consider initial values for the integral variables as unknown. After finding the consistent values for the integral variables, the non-integral variables will be determined by the underlying ODE.

If we, alternatively, want to consider the non-integral variables as unknown, then we should also include the underlying ODE in the under-determined equation system. One might also choose to include extra equations for specifying additional constraints on the initial values. See Figure 7.5.

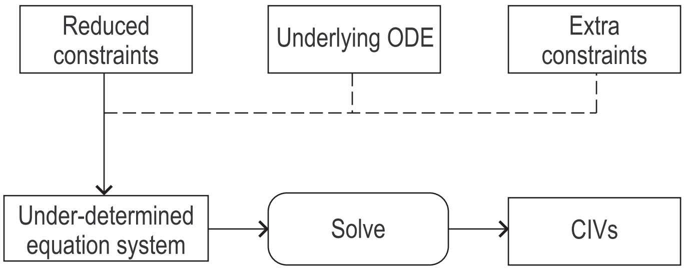


FIGURE 7.5: Finding consistent initial values (CIVs).

The general methodology proposes solving under-determined equation systems, by choosing dependent/independent variables.

7.1.3 Solving the IVP

Basically, there are two fundamental strategies in the methodology for solving the IVP, assuming that consistent initial values are found.

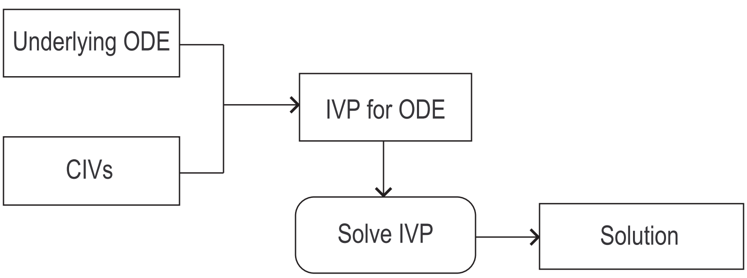


FIGURE 7.6: Solving the IVP using *only* the underlying ODE with consistent initial values.

The first approach, see Figure 7.6, is included since it is very simple, and can take advantage of existing numerical software. The method is to use the underlying ODE as a normal (possibly implicit) ODE, and the solve the traditional IVP, and remember to use consistent initial values, found as described above. This method can though have problems with drift off and stability for some cases of DAEs and for some purposes of the model.

In the case where the simple approach gives problems, the methodology recommends use of the augmented underlying ODE (AUODE) as described in Chapter 6. See Figure 7.7.

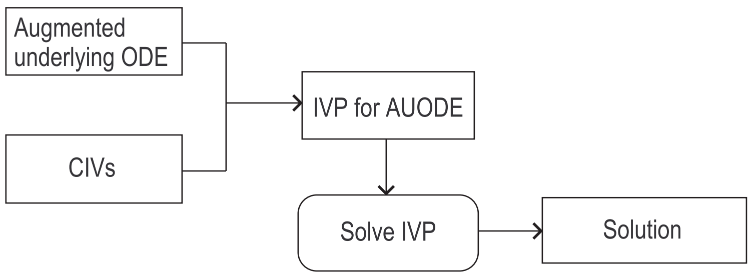


FIGURE 7.7: Solving the IVP using the augmented underlying ODE.

The AUODE is essentially an over-determined systems of *ODEs*, also known as *ODEs with invariants*. For solving the IVP for these problem, a number of general principles exist, but practical implementations are hard to find.

In the thesis, a certain approach called the *dummy-derivative* methods is investigated and recommended. Credits for the original idea should be given to [MS93]. To reach practical implementation of this method, some research and experiments still have to be done, but the basic outlines are sketched.

7.2 Discussion of structural index analysis

This section is intended to discuss various aspects of using structural index analysis as prescribed by the methodology.

One of the most important discussion regard applicability and the question of "generality" of structural index analysis. Here it is discussed if structural index analysis of DAEs can be considered a more general methodology than it is normally found.

7.2.1 When structural ODE is singular

The core point of criticism for structural analysis of DAEs, is that structural index analysis does not guarantee that we end up finding the underlying ODE. It just finds the underlying *structural ODE*.

This structural ODE might still be singular, and algebraic manipulations might show that the system is actually a higher index DAE. There are numerous example of this in literature. We can construct an almost trivial, though explanatory example:

■ **EXAMPLE 7.1 (STRUCTURAL ODE IS DAE OF INDEX 1)** *Assume that we have an index 1 system using integral notation:*

$$x + y = f(t) \tag{7.1a}$$

$$\int x - \int y = g(t) \tag{7.1b}$$

This system is quite obviously a DAE system of differentiation index 1, since by differentiation of (7.1b), *the system becomes an implicit ODE. Looking at the structural index matrix*

$$M = \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix},$$

structural index analysis would also prescribe differentiation of the last equation, and therefore end up at the same underlying ODE, and hence the structural index is 1. For the system (7.1) *structural index analysis works perfectly.*

Now what happens if we construct a new system by substituting equation (7.1b) *with the sum of* (7.1a) *and* (7.1b)*:*

$$x + y = f(t) \tag{7.2a}$$

$$x + y + \int x - \int y = f(t) + g(t) \tag{7.2b}$$

This system is still a DAE system of index 1.

Now the problem is that, using structural analysis, (7.2) *is a structural ODE. This can be seen from the index matrix representation:*

$$M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

This index matrix is obviously structurally regular considering the index 0 elements.

This means that structural index analysis will prescribe no further differentiations and assume that (7.2) *is the underlying ODE. Therefore the structural index of* (7.2) *is 0!*

We see that we can blur the structural information needed by structural index analysis, by rewriting the a DAE system or simply stating the system in an unfortunate manner. ■

This point of criticism is mentioned already in the article [Pan88], but can be found in general texts on DAEs as e.g. [BCP96]. This section is intended to oppose to this criticism by discussing the nature of systems from which structural analysis fails to reveal the underlying ODE:

The first thing to note is that a numerical representation of a system as (7.2), seen from a numerical viewpoint, will generally be *nearly* singular, and not exactly singular.

In section 3.4.5 we discussed nearly singular ODEs. We learned that the solutions of a nearly singular ODEs can not in general be expected to converge towards the solution of a corresponding DAE, and therefore the numerical representation of any nearly singular ODE, as e.g. (7.2) can not be expected to have solutions converging towards the solution of the corresponding DAE. As the numerical precision increases the solution will be even more diverging. The major conclusion is that numerical representation of a higher index DAE system is generally an ill-conditioned problem seen from a numerical viewpoint. The only way to solve a numerical DAE is by using some stabilizing method as e.g. the BDF methods or introducing some regularization of the system in one way or the other.

Therefore we propose that well-posed DAEs, should generally contain algebraic information from which the specified singularity and rank-deficiency can be analysed. In this respect, structural information is exactly algebraic information, very simple to obtain, and which can be used for analysing singularity of the DAE. One could imagine other kinds of algebraic information, for example symbolic representation of parts of the DAE system.

For practical implementations of tools using structural index analysis, it is proposed that one builds in a check for regularity of the underlying structural ODE.

7.2.2 "Sound" DAEs are structurally regular

Another important property of structural analysis, is that we expect every "sound"/reasonable DAE model to be *structurally regular*, see definition 5.8.

If the DAE is not structurally regular, then there will exist a subset of equations which concerns less base variables[1] than there are equations. This subset will normally be over-determined, and at the same time the remaining equations will normally be under-determined. Hence the DAE is generally "unsound".

[1] Here a base variable, including the derivatives and integrals of the variables, only count as one.

That every “sound” DAE is structurally regular, means that there exist an underlying structural ODE for every “sound” DAE system, and subsequently we can then use structural index analysis to identify the underlying structural ODE.

7.2.3 General applicability of structural analysis

This section presents a viewpoint which the author uses when trying to argue for the “generality” of the structural methods, despite the counter examples mentioned earlier.

The basic idea behind this “generality” is based on two observations: (1) We can assume that we can obtain an underlying structural ODE for most reasonable DAE models. If the underlying structural ODE is an ODE, then we use methods for the underlying ODE or augmented underlying ODE as described in Chapter 6. (2) The assumption is, that if we unfortunately end up at a structural ODE, which is not an ODE but actually a DAE system, then this system should at least contain algebraic information, and not only numerical information, which describes the singularity and rank deficiency of the DAE system.

It is assumed that either these models can be rewritten or formulated more appropriately, so that this algebraic information will be reflected by the structural properties of the rewritten DAE. The alternative is to develop methods which can use even more algebraic information than just structural information.

On the other hand if we end with a structural ODE which is not a ODE, then one can always hope that structural index analysis, either way, have partly reduced the index of the system. This should generally improve performance of the direct methods for the fully implicit DAE.

7.2.4 Separating index analysis from the index reduction process

One of the major practical advantages of structural analysis of DAEs, is that the analysis can be done without knowing the exact representation of the equations, and therefore neither the derivatives of these equations. The analysis can be based solely on structural information.

This can have advantages in practical modelling where special functions and tables have to be developed. Structural analysis can be done without knowing these function in advance. The results of structural index analysis can then be used to obtain smoothness demands on functions and tables. This is very useful for practical modelling situations.

7.3 Discussion of structural index

In this section, different aspects of using the structural index are discussed.

7.3.1 Reduced DAE index

The structural index analysis described in Chapter 5, is based on a principle of stopping the index reduction process as soon at we have reached a system which can be interpreted as a general ODE.

As described in Chapter 4, this means that, for some systems, the index reduction process will stop at a system, which is traditionally considered a considered a DAE system of index 1. This can be related to the fact that choosing appropriate assignment of variables might reduce the index by one.

In this respect the methodology described in this thesis uses the lowest possible index. It is recommended that generally one should index classify DAE systems by the least index obtainable from appropriate assignments of the variables.

7.3.2 Comparing to differentiation index

A topic often discussed is how the structural index relates to the differentiation index. Generally, the difference comes from structural analysis being prevented from using algebraic manipulations combined with differentiations under the index reduction process.

Generally, from remarks in [Pan88], we learn that structural index analysis might underestimate the number of needed differentiations to obtain the underlying ODE, and therefore the structural index can be lower than the differentiation index. This is related to the fact the the underlying structural ODE, does not necessarily have to be an ODE, but can be a higher index DAE. See discussion section 7.2.1.

Alternatively, we can say that if the underlying structural ODE is really a consistent and regular (implicit) ODE, then the differentiation index must at least be equal or lower than the structural index. Therefore we often assume that, in this case, the indices match.

But more recently, in [RMB00], they show that certainly the differentiation index can be lower that the structural index. They come up with examples where structural analysis will perform extra differentiations, although simple algebraic manipulations could have prevented most differentiations.

They point out that differentiating MSS subset is not necessarily minimal. This can seem to contradict Theorem 5.10 which states the necessity of differentiating MSS subsets. But this is not a contradiction. If one allow algebraic manipulation, this gives the possibility of changing the structural properties of an MSS subset, thereby changing the subset into different independent subsets, some of which are not necessarily MSS subsets, and which for this reason does not have to be differentiated.

The consequence for applicability of structural analysis is that structural analysis will sometimes end up with a system which is overly index reduced. This is not necessarily a problem since the solutions of this system will still be the solutions of the old system, but it will introduce additional reduced constraints in the augmented underlying ODE.

7.3.3 Comparing to perturbation index

Regarding the perturbation index, some interesting information can be concluded from the structural index. The following considerations are not very elaborate, but it is hoped that they can sketch a result which can be formalized at a later stage. Again we assume that the underlying structural ODE is really a consistent and regular (implicit) ODE.

Assume that we introduce perturbations in every equation of the original DAE, then these perturbations will enter the corresponding equations of the underlying ODE, but differentiated a number of times according to the individual structural indices of the equations.

Since all base variables of the underlying ODE are computed directly from the implicit function theorem, then the permutations will affect these corresponding to their differentiated presence in the underlying ODE, i.e. the error corresponding to the perturbations, can be bounded by a constant (depending on the time and some Lipschitz constant) times a bound on these *differentiated* perturbations.

Now if the system has structural index 2, i.e. the maximum structural index for the individual equations is 2, we can conclude that at least one of the perturbations will be differentiated 2 times. When solving for the non-integral variables, i.e. the base variables, at least one of the variables will depend on the second derivative of a perturbation, and therefore have perturbation index 2+1=3. The "plus one" is due to the traditional definition of perturbation index. The others base variables will be of perturbation index 3 or lower.

Considering the integral variables of the model, these are all first or higher order integrals of the base variables, depending on their structural index. If we have a variable of structural index 2, i.e. a second-order integral of a base variable, we expect its perturbation index of this variable lower correspondingly compared to the base variable.

7.4 Summary

This chapter contains an overall discussion of advantages and drawbacks of the described methodology. The chapter start by summarizing the methodology, and then continues to discussion of various topics.

The methodology for analysing DAE systems presented in this thesis has some major steps. These are *structural index analysis* using the index matrix representation, *index reduction* for obtaining the augmented underlying ODE (AUODE), solving for consistent initial values of the DAE using the AUODE, and finally *solving the initial value problem* for the DAE using either the underlying ODE with traditional methods for (possibly implicit) ODEs, or special methods for the augmented underlying ODE, as e.g. the dummy derivative method.

The first discussion concerns the problem that a structural ODE is not necessarily an ODE. An example is constructed and it is discussed that from a numerical point of view DAEs are in general ill-conditioned nearly singular ODEs, since small errors in the representation will generally disturb the singularity. Therefore it is concluded that generally well-posed DAE system should contain algebraic information for exact specification of the singularity. Structural information is an example of this type of information.

It is described that we must expect “sound”/reasonable DAEs model to be structurally regular, and consequently have an underlying structural ODE.

This outlines the arguments for a more “general” applicability of structural analysis than is usually understood. Since every reasonable DAE has an underlying structural ODE, and that we normally expects a structural ODE to be regular or have algebraic information specifying the singularity.

There are several advantages of separating the index analysis from the concrete index reduction process. One is that we are not forced to consider derivatives and smoothness of complicated functions, expressions and fitted tables, before the index analysis tells that they are needed.

The algorithm for structural index analysis uses the general ODE as an endpoint. This means that the first time the index reduction process reaches a system which can be interpreted as an structural ODE it will stop. For some system, the underlying ODE proposed by the algorithm will traditionally be considered an index-1 DAE. Interpreting this system as a general ODE is just a matter of definition and assigning variable as described in Chapter 4. This also implies that the structural index suggested by structural index analysis might be one lower that the traditional index.

We also discuss and confirm that the differentiation index can be both less that, equal or greater that the structural index. Also the relation between the structural index and the perturbation index is discussed.

CHAPTER 8

Simpy

Simpy is a mathematical software tool box, developed under this Ph.D.-project to demonstrate and experiment with the presented methodology for analysing DAEs. Using Simpy, one can build non-linear equation systems, ODEs and DAEs, and solve mathematical problems concerning these, especially the initial value problem for a range of DAEs[1].

Simpy also represents experiments and investigations in object oriented design of mathematical and numerical software. In this respect it is a contribution to the research in object oriented design of numerical software. See e.g. [HAL99].

8.1 Description of Simpy

Simpy is written in the programming and scripting language Python. Simpy can be used interactively from the Python command line and by using Python scripting files.

Simpy is written as a package for Python, such that by loading the appropriate Simpy-modules, Simpy extends the Python language with classes and functions for building equation systems, ODEs and DAEs.

Consequently, to use Simpy, it is important to have some knowledge of Python. It is believed that users familiar with programming languages as

[1] See the discussion on applicability of structural index analysis in section 7.3

C++, Java or MATLAB, will have no major problems learning Python.

8.1.1 Design objectives and requirements

Simpy is designed with a number of objectives and requirements in mind. These requirements can roughly be separated in two: (1) Some concerning Simpy as a proof of principle for the methodology presented in this thesis and (2) some concerning Simpy as an experiment regarding design, especially object oriented design, of mathematical software tools.

Requirements for demonstrating DAE methodology. We can list the requirement concerning Simpy as a tool for investigating and demonstrating the methodology and as a proof of principle.

The following list briefly sketch which data and methods are necessary and states the arguments:

- *DAE systems:* Representation of DAE systems is a fundamental requirement.
- *Index matrix and structural index analysis* Should produce index matrix from DAE and perform structural index analysis on index matrix.
- *Symbolic expressions and differentiation:* Equations and expressions must be represented symbolically, such that differentiation of some kind can be performed automatically.
- *Augmented underlying ODE:* Representation of the augmented underlying ODE, obtained from DAE systems as the result of the structural index analysis, by symbolic differentiation of equations.
- *Under-determined non-linear equation system:* Representation of under-determined equation systems is necessary for the representing and solving the consistent initial value problem.
- *Find dependent variables:* Automatically suggest a set of dependent and independent variables for changing an under-determined equation system into a determined equation system and for automatically choosing dummy derivatives.
- *Non-linear equation system and implicit function:* Representation of non-linear equations and implicit functions, for solving under-determined equation systems and computing the derivatives of implicit ODEs.

- *Newton solver:* For solving non-linear equations and implicit functions.
- *Implicit ODE:* For representing underlying ODE or augmented underlying ODE after choice of dummy derivatives.
- *Integration of implicit ODE:* Implement methods for integration of implicit ODEs. Since we do not generally expect problems to be stiff we can e.g. use explicit Runge-Kutta methods.

These requirements are all full-filled by the current version of Simpy.

General objectives and requirements. Continuing the work described in [PWTH99], Simpy is a new contribution in the research concerning object oriented tools for mathematical modelling and numerical computations. The following list is the objectives and requirements, which have shaped the construction of Simpy:

- *Interactivity and scripting:* Model building, conducting experiments and analysis of models, and handling result data should both be possible interactively and from a scripting file. Interaction can both be graphical and use a command line.
- *Object orientation:* The tool should use object orientation. There should generally be classes for all mathematical objects, and these should be interactively modifiable via the command line or graphically.
- *Multi-purpose language:* The tools should be designed as an extension to an existing multi-purpose language. This leaves the best opportunities that users can flexibly treat model and solution data, and implement extensions to the functionality of the tool.
- *Open design of methods:* No black-boxes. It should be possible to go into methods and examine what they do and how they behave. In this way it should be possible also to copy, modify and experiment with methods.
- *Extensible:* New mathematical models, problems and methods should easily be added to Simpy. Especially there should be a way of having multiple method for the same problem, and along with this have some functionality for choosing the standard method.

8.1.2 Python

Python is a relatively new programming language. It is an interpreted scripting language available for most platforms as public domain. It is an object oriented language and it has a number of high-level data types as e.g. lists and associative arrays, which have shown useful when aiming at readability and clarity of the implementation.

There is a large community around Python. A central places for this community is the Internet site, *www.python.org*. Here one finds links to download areas, documentation, and special purpose packages, which extends the standard packages that already come with the Python distribution. These implement a large number of high-level features for Python as e.g. graphics, basic numerics and Internet tools. A vast number of people are continuously active in developing the language, not least the original author of Python, Guido van Rossum.

Newcomers to Python could start at this site and will find information which will improve the understanding and use of Simpy. Recommended (mandatory) reading for newcomers is the Python tutorial [vRD01]. Optionally, a number of books can give the next level of introduction to Python, e.g. [Lut96].

Choice of Python. During this Ph.D. much effort has been put into investigating and evaluating programming languages. In the search a number of qualities and requirements for the language was sought and evaluated. Some are listen in section 8.1.1. Others are listed below:

- *Platform in-dependency* to be broadly available and usable.
- *Obtainable efficiency/performance* to be able to tune performance when methods are developed and understood,
- *Syntax close to traditional syntax* e.g. MATLAB, C++, Fortran, which are assumed to be popular among a large forum of engineers and scientists.

Python evaluated good in all these qualities, and this was unique compared to the alternatives also evaluated.

Being able to write extension packages in C or C++ is one of the basic principles in the Python ideology. The current version of Simpy makes use of the extension package NumPy (Numerical Python) [ADH$^+$99] which implements the basic vector and matrix operations so that some efficiency is obtained in these operations.

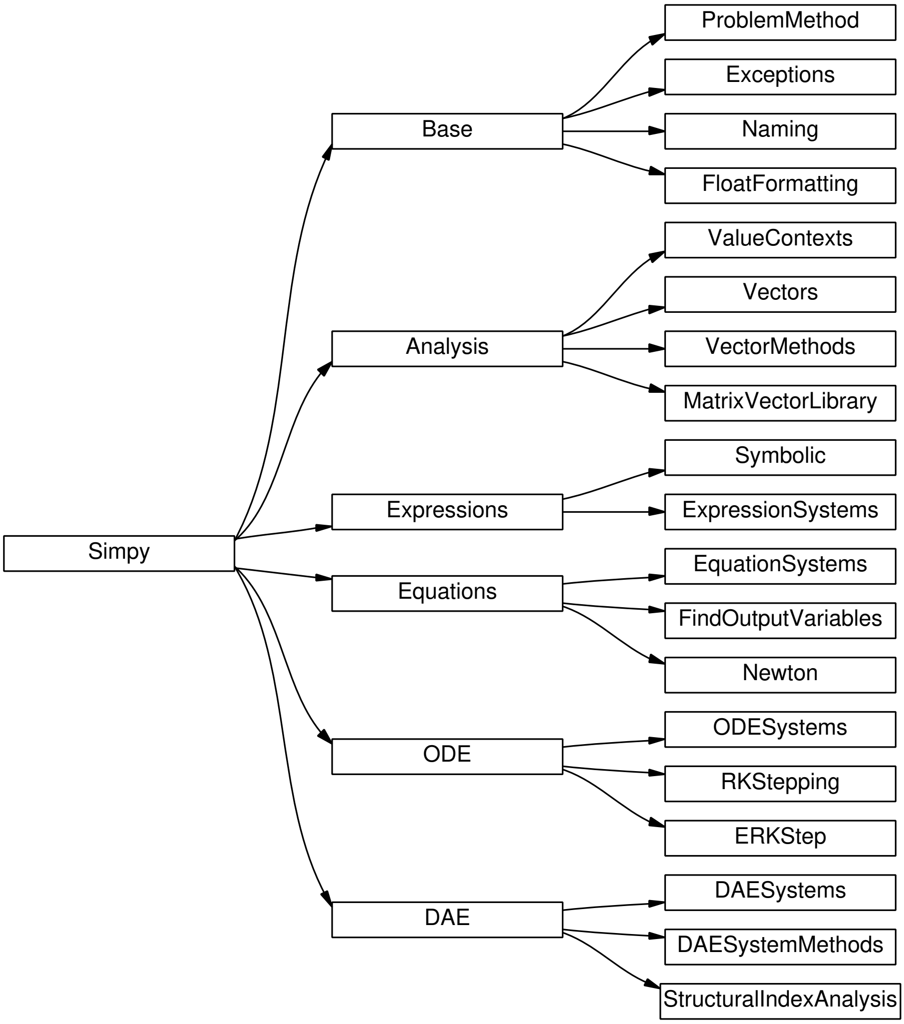
The syntax of Python is natural to programmers used to imperative languages, and at the same time Python supports functional programming style, by supporting high level data types as lists, associative arrays and function data types.

8.1.3 Simpy modules

Simpy is a standard Python package organized in several modules and submodules. Modules in Python are definitions of functions, variable and class definitions gathered in a code file or are directories containing other (sub-)modules[2]. Figure 8.1 gives an overview of the modules in Simpy

- *Base:* Implements some basic functionality for Simpy: Abstract problem- and method classes, top-level exceptions, object naming functionality and float formatting.
- *Analysis:* Implements basic mathematical quantities: Value contexts (associative arrays used much like vectors), vectors, matrices, and various methods for working with vectors and matrices (using the standard Python module NumPy).
- *Expressions:* Implements symbolic representation of real value expressions and classes for representing systems of expressions, vector functions and a class for Jacobian matrix functions.
- *Equations:* Implements classes for representing non-linear equation systems, and implicit functions. Implements a Newton solver class for solution of the equations (implicit functions) and a method for finding/suggesting dependent variables for an under-determined equation system.
- *ODE:* Implements classes for representation of explicit ODEs and implicit ODEs. Also implements classes for explicit Runge-Kutta steps and for static step integration of both ODEs and IODEs using Runge-Kutta steps.
- *DAE:* Implements class for representing a general DAE system, and a class for representing the augmented underlying ODE. Implements methods for structural index analysis and integration of the augmented underlying ODE using the dummy-derivative method.

[2]Top-level directories containing modules are called packages in Python.



This graph is created using "dot". See [KN99].

FIGURE 8.1: Modules and sub-modules in Simpy.

The modules can be loaded when needed using the general python command for importing a module. If we want to create an non-linear equation system we can load the appropriate classes by issuing the command

```
>>> from Simpy.Equations.EquationSystems import *
```

Hereafter the `EquationSystem`-class will be available, including objects for building symbolic expressions based on the `Variable`-class.

8.1.4 Documentation and tests

Documentation for Simpy can be found from mainly two sources: The manual [Pou01] and from the doc-strings embedded in the python code.

The manual gives an description of Simpy from both a usage perspective, but it also tries to go behind some of the design ideas and explain some of the more advanced designs.

A feature of Python is the support of doc-strings available at run-time. Simpy tries to make consequent use of these doc-strings, and a more detailed documentation of Simpy can be obtained from these.

These doc-strings concerns both functions, classes, objects and modules. The simplest way to get to these doc-strings are by issuing commands like

```
>>> print EquationSystem.__doc__
```

which will print the doc-string for the class `EquationSystem`. There are a number of tools which makes the doc-strings available in a compiled and more nice layout. One such tool is the Python package `pydoc`. Included in the Simpy-distribution is a directory containing html-documentation created by `pydoc`.

There are also a number of examples and test files which can be examined and executed to learn more about Simpy. The examples are found in a separate directory in the Simpy-distribution, while the test files can be found under each module.

These examples and test files also represent the test material for Simpy done at this stage. The tests are designed for each modules and its problems. Simpy has produced the expected results in these cases.

8.1.5 Getting and installing Simpy

Simpy can be obtained by contacting the author. Simpy is implemented purely in Python. This means that all the Python code is available and distributed in a simple directory structure.

For using Simpy one needs to initially install Python. One can down-load free distributions from e.g. *www.python.org*. Simpy is programmed and tested using Python version 2.1. Next, one should get and install a version of the NumPy- (Numerical Python-) distribution. Simpy is programmed using version 11. Finally, place Simpy-distribution directory somewhere in the standard Python directory path list.

8.2 Experimental design principles

Simpy also represents research in object oriented implementation of mathematical modelling and numerical tools. A number of design principles have been tried out for Simpy. In the following some of the more successful or interesting are descried.

8.2.1 Objects for mathematical models

One of the most obvious design principles applied in Simpy concerns design of object classes for the mathematical model types, i.e. non-linear equations, ODEs, DAEs etc.

From traditional tools we are used to mathematical models being specified by just functions, but there is some advantages of representing these using objects: Objects can be changed interactively, and then they can be provided with methods for displaying themselves and returning advanced information. Basically, they can be used to gather all functions relevant for the specific model type in the model class.

Examples on use of Simpy-objects for mathematical models, can be seen in the example section 8.3. Generally, this object design principle is found quite useful.

8.2.2 Models, problems and methods

One of the core design ideas in Simpy is to clearly separate the objects in three basic types:

- *Models:* Corresponds to the mathematical model objects discussed in the previous section: Non-linear equations, ODEs and DAEs.
- *Problems:* These are the "questions" or queries one can ask/pose based on the models. A example of a problem would be to solve for

the dependent variables in an equation system. Another one could be to evaluate an ODE in a certain point given an initial value.

- *Methods:* Methods are the functions which can be used to solve a given problem. Generally we must expects that there can be several methods for each problem, but this is treated in the following section 8.2.3.

This distinction is found to be an important design principle. It describes three types of objects which are fundamentally different and this helps structuring the code: Models are quite large classes, characterized by having lots of methods for building, displaying and modifying themselves. Problems are small classes, and are merely the abstract statements of what (possibly implicit) information is sought/required based on references to a given model. Methods represents the practical routines/functions for attempting to answer and solve the problems. These will sometimes only give approximate solutions.

One can do a comparison with the Prolog language (See [Bra86]) for logic programming. We can compare *facts* and *rules* with models, and compare *questions* with problem. Furthermore methods must be compared to the inference engine.

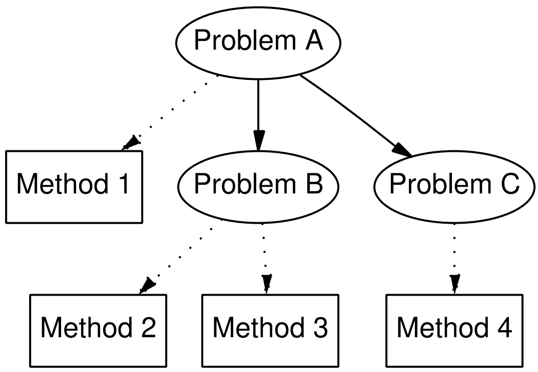
In Simpy it is easy to see the separation in these types. Considering a module, then typically one sub-module defines the models and the problem classes, and one or more other sub-modules defines the methods which solves the problems. During initialisation of the module the methods will be loaded and attached to the problem classes.

8.2.3 Method-problem relation

During the Ph.D. project some effort has also gone into investigating how one can introduce expert-system-like behaviour of the mathematical tools. This is a vast topic, and the focus has been on structuring data about which methods can solve which problems.

Basically it is assumed that we have a problem hierarchy, where similar problems are organized. See Figure 8.2. The hierarchy describes that e.g. Problem B and Problem C are types of Problem A. This means that these problems can be converted into Problem A types. Reversed, a problem of type A can possibly be of either type C, type B or none of these.

The idea is furthermore, that we attach methods to the most general problem class that each method can handle. In Figure 8.2 we see that Method 1 is the most general since it can handle problems of type A and therefore also problems of type B and C. This also means that Problem B can be solved by



This graph is created using "dot". See [KN99].

FIGURE 8.2: The relation between the problem hierarchy and the methods.

either Method 1,2 or 3, while C can be solved by Method 1 or 4.

This design is implemented in Simpy. In the initialisation of the modules, the methods are loaded an attached to the problems and a default method is chosen. Issuing a command like

```
>>> evalproblem.getMethods()
```

returns a dictionary containing links to the appropriate methods for this problem class including methods for all super classes.

This design seems to have some nice features. As example it is possible by a simple command to change the default method for solving equation systems, generally affecting the solution of equation systems for all running modules.

The principle need to be investigated further. The relevance of representing the problem hierarchy along with the corresponding methods is not obvious from a small system like Simpy. Interested readers can have a look at [Pou00] where the relations between problems an methods as described are studies in terms of algebraic lattice theory.

8.3 Example: Mechanical spinner

The use of Simpy is probably best illustrated using an example. This section exemplifies how Simpy can be used to analyse and simulate the behaviour of a classical mechanical multi-body system. The mathematical model is a DAE system, and the example also shows how DAEs arise from a component based description of the system.

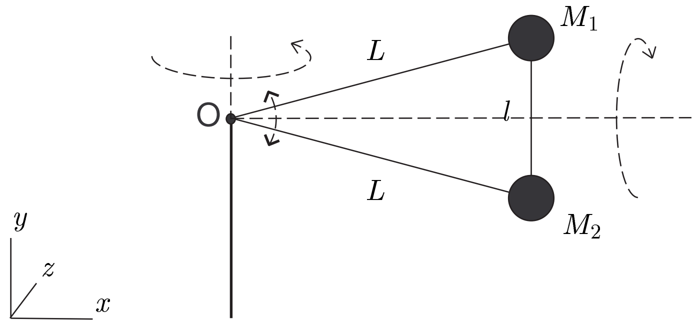


FIGURE 8.3: Illustration of the mechanical spinner system. The rotation arrows basically illustrates the rotational freedom in the system.

The system is illustrated in Figure 8.3. At the top of a fixed vertical pole, O denotes the origin of the coordinate system. The two massless poles of length L, each connected with one end to the origin, can rotate freely in the 3-D space, and the mass particles, with masses M_1 and M_2, at the end of these poles, are joined by yet another massless pole of length l. The rotation arrows illustrates some of the possible rotations in the system.

One of the interesting phenomena this system can be expected to show is the *precession movement*: When the two masses rotate around a horizontal axis, going trough the origin and their common center of mass, the gravitational force will cause a horizontal rotation of the same axis describing a disc or cone emanating from the origin. Precession is described in most textbooks on mechanics, and since it is often introduced in connection with rigid body dynamics, the examples often uses a solid disc instead of two masses. Using particle masses, we avoid using rigid body formalisms and handling moment of inertia matrices etc.

8.3.1 The mathematical model

We will now set up the equations for the two mass particles. It is a classical kinetic problem in three dimensions (x-, y-, z-axis), and we describe the movement of the particles using position , velocity and acceleration. For the particle M_1 these variables are (p_{x1}, p_{y1}, p_{z1}), (v_{x1}, v_{y1}, v_{z1}) and (a_{x1}, a_{y1}, a_{z1}) respectively.

We use Newtons second law of motion on each particle. First we identify all forces working on the particles: The scalar forces F_1 and F_2 from the two poles attached to the origin and particles M_1 and M_2 respectively, the force

F_{12} from the pole connecting the two particles and finally gravitation force g.

Since we work in three dimensions, Newtons second law results in three scalar equations for each particle:

$$\begin{aligned}
M_1 a_{x1} &= -F_1 p_{x1}/L \ + \ F_{12}(p_{x2} - p_{x1})/l \\
M_1 a_{y1} &= -F_1 p_{y1}/L \ + \ F_{12}(p_{y2} - p_{y1})/l \ - \ M_1 g \\
M_1 a_{z1} &= -F_1 p_{z1}/L \ + \ F_{12}(p_{z2} - p_{z1})/l
\end{aligned}$$

$$\begin{aligned}
M_2 a_{x2} &= -F_2 p_{x2}/L \ - \ F_{12}(p_{x2} - p_{x1})/l \\
M_2 a_{y2} &= -F_2 p_{y2}/L \ - \ F_{12}(p_{y2} - p_{y1})/l \ - \ M_2 g \\
M_2 a_{z2} &= -F_2 p_{z2}/L \ - \ F_{12}(p_{z2} - p_{z1})/l
\end{aligned}$$

In each equation the scalar forces are projected onto each axis, as e.g. the force F_1 working on the particle M_1 are projected using $F_1 p_{x1}/L$.

These equations could look like a part of an ODE System, but we need to add some equations modelling the constraints on the movement imposed by the three poles:

$$\begin{aligned}
L^2 &= p_{x1}^2 + p_{y1}^2 + p_{z1}^2 \\
L^2 &= p_{x2}^2 + p_{y2}^2 + p_{z2}^2 \\
l^2 &= (p_{x1} - p_{x2})^2 + (p_{y1} - p_{y2})^2 + (p_{z1} - p_{z2})^2
\end{aligned}$$

We realise that this system is not a general ODE system, since this equation system is not determined w.r.t the non-integral variables. It is in-fact a DAE system.

8.3.2 Defining the model in Simpy

When defining the model in Simpy one can do this in a file, or directly at the interpreters command line. The first thing to do is to import the appropriate module:

```
>>> import Simpy.DAE.DAESystems *
```

This line will import a set of classes for representing DAEs, using symbolic variables and expressions.

Then we define the variables of the model:

```
>>> t=Variable()
>>> px1=Variable(); py1=Variable(); pz1=Variable()
```

```
>>> px2=Variable(); py2=Variable(); pz2=Variable()
>>> vx1=Variable(); vy1=Variable(); vz1=Variable()
>>> vx2=Variable(); vy2=Variable(); vz2=Variable()
>>> ax1=Variable(); ay1=Variable(); az1=Variable()
>>> ax2=Variable(); ay2=Variable(); az2=Variable()
>>> F12=Variable()
>>> F1=Variable()
>>> F2=Variable()
```

This defines the dynamic variables of the model. We define also have to define the parameters:

```
>>> M1=Variable()
>>> M2=Variable()
>>> g=Variable()
>>> l=Variable()
>>> L=Variable()
```

We choose to define these as `Variables` as well, in contrast to e.g. `Constant`'s. This means that we can choose to treat these as unknown and e.g. solve for the parameters if this is needed at some point.

Now we are ready to define the DAE system: First we initialise an object of the DAE system class. This is done by

```
>>> dae=DAESystem()
```

Then the specification of independent variable (time) and dependent variables (dynamical variables) are done:

```
>>> dae.independentVariable=t
>>> dae.dependentVariables=[
       px1,py1,pz1,
       px2,py2,pz2,
       vx1,vy1,vz1,
       vx2,vy2,vz2,
       ax1,ay1,az1,
       ax2,ay2,az2,
       F12,F1,F2
       ]
```

Next, we specify how the dependent variables are related as derivatives.

```
>>> dae.derivatives={
        px1:vx1,py1:vy1,pz1:vz1,
        px2:vx2,py2:vy2,pz2:vz2,
        vx1:ax1,vy1:ay1,vz1:az1,
        vx2:ax2,vy2:ay2,vz2:az2,
        }
```

Finally, we define the model equations as residuals:

```
>>> dae.equations=[
        -F1*px1/L + F12*(px2-px1)/l - M1*ax1,
        -F1*py1/L + F12*(py2-py1)/l - M1*g - M1*ay1,
        -F1*pz1/L + F12*(pz2-pz1)/l - M1*az1,
        -F2*px2/L - F12*(px2-px1)/l - M2*ax2,
        -F2*py2/L - F12*(py2-py1)/l - M2*g - M2*ay2,
        -F2*pz2/L - F12*(pz2-pz1)/l - M2*az2,
        px1**2 + py1**2 + pz1**2 - L**2,
        px2**2 + py2**2 + pz2**2 - L**2,
        (px1-px2)**2 + (py1-py2)**2 + (pz1-pz2)**2 - l**2
        ]
```

This ends the specification of the DAE system. If we want to check the DAE system we can get a string representation using the `string` method or just issue

```
>>> print dae
```

which gives a text based description of the system, including equations.

8.3.3 Index-matrix

Since Simpy is intended to illustrate the methodology for treating DAEs presented in this thesis, one can obtain and examine the index-matrix representation.

```
>>> M = dae.IndexMatrix()
>>> print M
  0            -2              0    0
       0            -2         0    0
            0            -2    0    0
 -2             0              0         0
      -2             0         0         0
```

```
           -2              0    0         0
 -2   -2   -2
                -2   -2   -2
 -2   -2   -2   -2   -2   -2
```

We can perform index reduction on this index-matrix by issuing the command:

```
>>> M.IndexReduction()
>>> print M
  0             -2            ( 0)   0
       0             -2           0  ( 0)
          ( 0)            -2      0    0
 -2              0                0       ( 0)
      -2            ( 0)          0         0
           -2            ( 0)     0         0
  0  ( 0)   0
               ( 0)   0    0
( 0)   0    0    0    0    0
```

The parentheses mark the full-assignment illustrating that the reduced index-matrix represents a structural ODE. It is quite obvious that the index reduction is simply done by two differentiations of the last three equations.

8.3.4 Obtaining the augmented underlying ODE

We can also obtain the augmented underlying ODE by issuing the simple command:

```
>>> auode=dae.AUODESystem()
```

In this version of Simpy the augmented underlying ODE is obtained by symbolic differentiation of each equation. It does not use automatic differentiation. This means that the expressions for the differentiated equations can potentially be quite complex, and inappropriate for evaluation.

This can be seen by having the AUODE written on output by issuing the command

```
>>> print auode
```

8.3.5 Finding consistent initial values

Note that the Figure 8.3 illustrates a possible choice of initial state of the system. For computing these initial condition we set up an equation system consisting of the AUODE and some extra equations specifying exactly these initial conditions. Since the degrees of freedom are 6, we extend the equation system with 6 extra conditions:

```
>>> print auode.DegreesOfFreedom()
6
>>> eqnsys=auode.EquationSystem()
>>> eqnsys.extend([
        pz1, pz2, py1+py2,      #position residuals
        vy1+vy2, vz1+vz2-3.0, vz1+4.0  #velocity residuals
        ])
```

Since we now have no degrees of freedom left, we should be able solve the equation system. We initially construct a value context to contain the solution:

```
>>> s=ValueContext(RandomGaussValueContext(0.0,0.3))
```

The special name *value context* describes a data type which works a bit like a vector. Instead of integer indices, this data type maps variables to their values, and instead of a fixed size the `ValueContext` can have values for an arbitrary number of variables. One can say that the data type holds values for the variables in a certain context. The initialisation using a `RandomGaussValueContext`, implies that the value of a variable which is not set, will be taken from this value context, which chooses values according to a gauss distribution, here with mean value 0.0 and variance 0.3.

Before solving for the initial values, we have to set the values of the parameters

```
>>> s[M1]=1.0
>>> s[M2]=1.0
>>> s[L]=2.0
>>> s[l]=1.0
>>> s[g]=1.0
```

and the initial time

```
>>> s[t]=0.0
```

Since solving for consistent initial values, i.e. solving the non-linear equation system, is based on converging iterations, it is helpful and sometimes necessary to specify some relatively good initial guesses on the values:

```
>>> s[px1]=2.0
>>> s[vx1]=0.0
>>> ...
```

Now the only thing left is to solve the equation system. We have to specify which variables should be considered dependent:

```
>>> init=eqnsys.SolveFor(auode.dependentVariables,s)
```

Hereafter the value context `init` contains the consistent initial values. If we want to know the size of the force `F12` between the mass-particles, we simply issue

```
>>> init[F12]
60.200000000029966
```

8.3.6 Solving the IVP

When we want to solve an initial value problem, we have to initially construct the initial value problem and then evaluate the IVP at certain points. This can be done by the following commands:

```
>>> auode.LocalUnitErrorTOL=1.0e-1
>>> ivp=auode.InitialValueProblem(init)
>>> evalpoints=[(2.5*x)/50 for x in range(0,51)]
>>> sol=ivp.eval(evalpoints)
```

The first line is intended to set the unit step error tolerance, the second line initializes the problem, then the evaluation points are defined, and finally the solution is computed in these points.

The current version of Simpy only uses static stepping defined by the evaluation points. Therefor the tolerance is only used to state a warning if error estimation at a point in time violates this unit step tolerance.

The solution specifies the movement of the spinner system. In this way `sol[0.1]` is the value context at the point `t=0.1`. Figure 8.4 is constructed from the computed solution.

If we subsequently want to analyse the data, we can use the fundamental functionality of Python combined with the functionality of Simpy. For one we

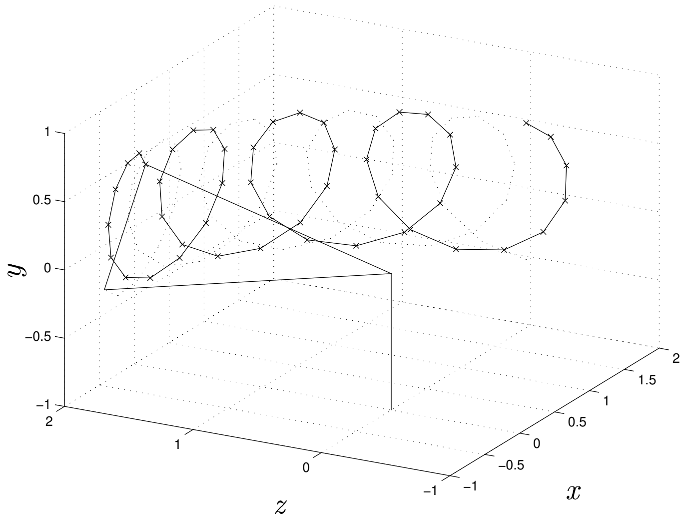


FIGURE 8.4: Illustration of the movement of the mechanical spinner system solved by using dummy derivatives for the augmented underlying ODE and DOPRI54 for numerical stepping.

can save data to a file using basic Python functionality, and if we e.g. want to check the conservation of the mechanical energy we can define the expression for the energy and compute the relative increase in energy:

```
>>> energy=M1*g*py1+M2*g*py2+0.5*M1*(vx1**2+vy1**2+vz1**2) \
... + 0,5*M2*(vx2**2+vy2**2+vz2**2)
>>> (energy.eval(sol[2.5])/energy.eval(init)-1)
0.020264967502194198
```

This increase in mechanical energy of 2 p.c. is due to the local error of the integration scheme.

If we want to change method of integration we can do this by changing the method linked to the `AUODEIVPEvalProblem`, i.e. the problem class for evaluation of an initial value problem for the augmented underlying ODE. We can e.g. change it to use a modified Euler step:

```
>>> AUODEIVPEvalProblem.defaultMethod= \
... AUODEIVPEvalProblem.ModifiedEuler21StaticStepping
```

The default method is at startup, the `DOPRI54StaticStepping`-method, a method using DOPRI54, with static stepping, i.e. steps given by the evaluation points.

8.4 Summary

As *proof of principle* the tool box Simpy was developed. Simpy can be used for building and analysing DAE systems. This includes structural analysis of DAEs, by analysis of the index matrix representation, deriving the augmented underlying ODE, by use of structural analysis, use the AUODE for finding consistent initial values, and solve the initial value problem for the AUODE (i.e. DAE) using the dummy derivative method.

Simpy also represents experiments with software design of mathematical and numerical modelling tools:

For one, all mathematical models, as equation system, ODEs and DAEs are objects, and as such they can be changed, updated interactively and displayed on the screen. Every object class has methods for obtaining information about the object, or transforming it into another related mathematical object class. This is found a natural requirement for an object oriented mathematical modelling tool.

As a more fundamental design idea, the Simpy design tries to separate the objects in some fundamental classes: Models, Problems and Methods.

As yet another design idea, Simpy supports implementing a problem hierarchy where several methods can be attached the same problem. The user can list these and choose the desired method.

Simpy is a proof of principle since it implements the methodology described in this thesis and based on this solves a range of DAE systems. Simpy is found unique in the sense that it is based on the theory for structural index analysis presented in this thesis, and also for the experimental object oriented designs.

Chapter 9

Conclusion

This thesis has presented a practical approach and methodology for analysing differential algebraic equations (DAEs), and particularly methods for finding consistent initial values and subsequently solving the initial value problem. The methodology is demonstrated by implementing the Simpy tool box. The thesis also describes an effort in experimenting with designing of software for numerical methods.

9.1 DAEs on integral form

The general implicit form for representing DAEs is

$$f(\dot{y}, y, t) = 0\ . \tag{9.1}$$

In most DAE texts the general form (9.1) is presented, though there are no instructions in how to interpret this form, and especially how to assign model variables to the vectors. This means that the task of interpreting a concrete system in terms of (9.1) is ambiguous.

This is discussed in Chapter 4 and a solution to resolve this ambiguity is suggested by the introduction of the *integral form*. Using the integral form, an ODE or a DAE model is formulated using integrals instead of derivatives, and consequently the integral notation is used for representing the general implicit DAE:

$$f(y, \textstyle\int y, \iint y, \ldots, \int^{(n)} y, t) = 0\ . \tag{9.2}$$

Here the integral form of n'th-order is presented instead of first-order form.

It is argued that using the integral form (9.2), it is easier to reveal if a general implicit DAE is essentially just a general (implicit) ODE and not a proper DAE: We define that, if y can be expressed as a (possibly implicit) function of the integral variables and the time t, then (9.2) is essentially a *general ODE*.

In general, discussion of indices and other classifications of DAE systems often inherit this ambiguity of the form (9.1), because variable assignment is not discussed. It suggested that this ambiguity is removed, by generally assuming that a DAE system is on integral form. For some DAE systems this can result in the index begin reduced by one.

The original contributions in Chapter 4, are the emphasis on and discussion of assignment of variables and the introduction of the unambiguous integral form.

9.2 Structural analysis of DAEs

As a methodology for analysis of DAE systems it is recommended to try using *structural index analysis*. This is described in Chapter 5.

The purpose of structural index analysis is to identify the *underlying ODE* by *index reduction*. More precisely, we want to identify the *augmented underlying ODE*. This augmented system can be used for e.g. consistent initial value determination.

It is shown that for performing structural index analysis it is sufficient to use only structural information of the original system alone. This information can be represented using the new *index matrix*, defined in this thesis. This representation and the corresponding algorithms are found to be an contributions to structural analysis of DAE, since it uses a simple matrix oriented approach and representation instead of a graph theoretic approach, and moreover it can represent higher oder DAEs.

Very essential is the property, *structural regularity* of a DAE. This property is simple to detect by structural analysis. One of the fundamental theorems of structural index analysis, is that structural regularity of a DAE, is identical to existence of an underlying structural ODE obtainable by differentiations. This is not a new result, but as a contribution, the proof presented in this thesis is assumed to be simpler, because it uses the index matrix and the usefull quantity, the *maximum trace*. The proofs also incorporates higher order DAEs.

As one of the pioneers of structural analysis, Pantelides has in [Pan88] proposed the principle, used in structural index analysis, for detection of subsets to be differentiated. In [Pan88] it is stated that this principle gives the minimal number of differentiations, which indicates *uniqueness* of the underlying structural ODE. This thesis contains a proof why this principle identifies the minimal number of differentiations. This is also considered a contribution since the arguments behind this statement are found hard to read directly from [Pan88].

In Chapter 7 we discuss different aspects of using structural analysis. One of the conclusion is that one can reasonably say that every “sound” DAE system, is a structurally regular DAE, and therefore an underlying structural ODE exist for every “sound” DAE system.

Also the Achilles’ heel of structural analysis is discussed: The problem of using the structural ODE property as an endpoint of index reduction. It is argued that a structural ODE, will also be a regular ODE for a large class of numerically well-posed systems, or that otherwise a well-posed DAE system should contain alternative algebraic information about singularities.

It is also attempted to contribute to the discussion of the relation between differentiation index and structural index. It is confirmed that the structural index can be both larger and smaller than the differentiation index.

9.3 Augmented underlying ODE

It is well-known that just using the underlying ODE, even though initial values are consistent, we can have drift-off since, i.e. have that the defect of the reduced constraints grow as the numerical solution process advances.

These problems can be helped by considering the underlying ODE, together with all the constraints which were differentiated to get from the DAE, we call the combined system the *augmented underlying ODE*. We describe how this system can be used for both detection of consistent initial values and solving the IVP for the original DAE system.

In this thesis we investigate an existing methodology for the augmented underlying ODE. It is called the *dummy derivative method*.

Using this method, the choice of dummy derivatives might need to be changed over the integration period. Some properties indicating when to change the cheating integrals are proposed and investigated. A certain embedded underlying ODE stage for Runge-Kutta methods is found promising, but much more research has to be done to make a practical running implementation of the dummy derivative method.

9.4 Simpy

During this project the experimental software tool box Simpy was implemented. Simpy in written in the Python programming language and runs interactively from the Python command line or, but can also be controlled from script files.

In Simpy it is possible to specify objects for symbolic representation of nonlinear equation systems, ODEs, and DAEs, and solve the relevant problems for these model types.

Simpy implements the methodology and the approach for solving DAEs as presented in this thesis using structural index analysis on the index matrix and using the dummy derivative method for solving the augmented underlying ODE. Simpy is tested for a number of simpler systems, and should solve a large range of general DAEs, independent on DAE index, though it is in the current version not very efficient for solving larger systems. Therefore Simpy constitutes a *proof of principle* for the methodology presented in the thesis.

Generally, Simpy is also an experiment of trying out some design ideas for objects and classes for object oriented mathematical and numerical modelling tools. We shortly mention the most important:

All mathematical models, as equation system, ODEs and DAEs are represented by objects. This is found a natural an basic requirement of an object oriented interactive tool. Next, the design of Simpy uses a separation of object in some fundamental classes: Models, Problems and Methods. This is found important for separating objects which are fundamentally different and clarifying interdependencies between object classes. Last, Simpy design supports implementing several methods for the same problem class. This design is though not fully developed yet.

In this respect, it is hoped that the design experiments represented by Simpy, will prove valuable for future design of mathematical modelling tools.

Appendix A

Some theory and proofs on structural analysis

This appendix describes some of the more specialised theory and proofs for structural analysis which is not contained in the background section.

A.1 Theorem of Hall

To understand the implications of structural singularity, we use a famous theorem from combinatorics of Hall [Hal35], here presented in a form equivalent to [Pan88].

Theorem A.1 (Systems of distinct representatives (SDR)) *Let $V = \{V_1, V_2, \ldots, V_m\}$ be a set of objects and $S = \{S_1, S_2, \ldots, S_n\}$ a set of subsets of V.*

Then each element of S can be assigned a different element of V, if and only if every collection of k $(\leq n)$ subsets from S contains at least k distinct elements of V.

This theorem relates to structural analysis in the following way: Consider that the subsets in S represents equations "containing" variables represented by V. An equation-variable assignment can only be done if the equation contains the variable. The theorem says that every equation can be assigned a unique variable, i.e. the system is structurally regular, if and only if every set of k equations depends on at least k different variables.

The theorem has a straight forward equivalence, which is simply the negated version of Theorem A.1.

Corollary A.2 *There exist at least one collection of k ($\leq n$) subsets of S which contains less that k elements of V, if and only if at least one element of S cannot be assigned a (distinct) element of V.*

This corollary relates to structural analysis the following way: An equation system, only contains a subsystem containing less variables than equations if and only if the system is structurally singular.

A.2 Structurally singular subset

Corollary A.2 is useful, since in terms of structural analysis it states, that if no full assignments exist, then and only then there will exist a subset of k equations which depends on less than k variables, i.e. it has less variables than equations. We call such a set *structurally singular* and restate the definition from section 2.5.4.

Definition A.3 (Structurally singular subset) *A subset of equations is* structurally singular *if it has more equations than variables.*

Recall that in this definition, the term structurally singular relates to a subset of the equations, not the entire system.

We state the following lemma:

Lemma A.4 *A system is structurally singular, if and only if there exist a subset of equations which is structurally singular.*

Proof. Follows directly from the definition of structural singularity and corollary A.2, when using the interpretation for structural analysis. □

A.3 Minimally structurally singular (MSS)

Now examine the "minimal-size" property of a structurally over-determined equation set. We restate the definition:

Definition A.5 (Minimally structurally singular) *A subset of equations is* minimally structurally singular (MSS) *if it is structurally singular, and none of its proper subsets are structurally singular.*

We state the following lemma:

Lemma A.6 *Every structurally singular subsystem, has at least one subset of equations which is minimally structurally singular.*

Proof. Assume a structurally over-determined subset. Then count all proper subsets which are structurally over-determined. There must exist a finite number of these subsets and at least one, namely the set it self.

Now, at least one of these subsets will not have any of the others as a subset, and therefore it is minimally structurally singular. The existence is proven. □

We observe that there exist a minimally structurally singular subsets, of every equation system which is structurally singular:

Theorem A.7 *An equation system is structurally singular, if and only if there exist a subset of equations which is minimally structurally singular.*

Proof. Assume a structurally singular equation system. This system has a subset which is structurally over-determined according to Lemma A.4, which again implies existence of a MSS subset according to Lemma A.6.

The other way around. Assume that a subset exist which is MSS. Then it is structurally over-determined according to the definition. This implies according to Lemma A.4 that the system is structurally singular. □

Lemma A.8 *If a subset of equations is minimally structurally singular, then it has* exactly one *equation more than variables.*

Proof. We prove this indirectly: (1) If the subset *has less* or equally many equations compared to variables, then it is not structurally singular. (2) If it *has more than one "extra" equation*, then by removing one equation we still have a structurally singular subset, which is in contradiction to the definition 2.10, saying that no subset must be structurally singular. □

Appendix B

List of symbols

f	Residual function for implicit DAE/ODE
f_0	Underlying ODE
g	Implicit function, often representing constraints
L	Leading derivative matrix
M	Index matrix of DAE. $M_{i,j}$ is the leading derivative index of variable j considering equation i
m	Leading derivative index
m	Vector containing integers for rearranging to integral form
n	Dimension of state vector y
n	Order of differential equation
p	Vector of integers, describing number of differentitions of a specific equation
S	Structural representation of matrix
t	Time. Independent variable
trM	Trace of M
y	Base variables. Dependent variables
$\int y, \iint y$	1. and 2. order integral of y w.r.t. time t
$\dot{y}, \ddot{y}$	1. and 2. order derivative of y w.r.t. time t
$y^{(n)}$	n'th order derivative (integrals have negative n)

Bibliography

[ADH+99] David Asher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, and Travis Oliphant. *Numerical Python*. Lawrence Livermore National Laboratory, University of California, 1999.

[And94] Mats Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, 1994.

[AP99] Anton Antonov Antonov and Mikael Zebbelin Poulsen. *Runge-Kutta methods for DAE problems*, chapter 2, pages 9–22. In Bendtsen and Thomsen [BT99], 1999.

[BCP96] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, 1996.

[BHP98] Peter N. Brown, Alan C. Hindmarsh, and Linda R. Petzold. Consistent initial condition calculation for differential-algebraic systems. *SIAM journal of Scientific Computing*, 19(5):1495–1512, 1998.

[Bra86] Ivan Bratko. *Prolog programming for artificial intelligence*. Addison-Wesley Publishers Limited, 1986.

[BS96] Claus Bendtsen and Ole Stauning. FADBAD, a flexible c++ package for automatic differentiation. Technical report, IMM, Technical University of Denmark, 1996.

[BT99] Claus Bendtsen and Per Grove Thomsen, editors. *Numerical Solution of Differential Algebraic Equations, A Ph.D. Course*. Department of Mathematical Modelling, Technical University of Denmark, 1999.

[CG95] Stephen L. Campbell and C. William Gear. The index of general nonlinear DAEs. *Numerische Mathematik*, 72:173–196, 1995.

[CM95] Stephen L. Campbell and Edward Moore. Constraint preserving integrators for general nonlinear higher index DAEs. *Numerische Mathematik*, 69:383–399, 1995.

[DER86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[DG86] I. S. Duff and C. W. Gear. Computing the structural index. *SIAM Journal on Alg. Disc. Meth*, 7(4):594–603, 1986.

[EHL97] B. Elmegaard, N. Houbak, and B. Lorentzen. DNA—a network based energy system simulator. In Giampaolo Manfrida, editor, *FLOWERS'97—Florence World Energy Research Symposium*, pages 239–246. SGEditoriale, 1997.

[ESF98] Edda Eich-Soellner and Claus Führer. *Numerical Methods in Multibody Dynamics*. B. G. Teubner Stuttgart, 1998.

[FB99] Wiliam F. Feehery and Paul I. Barton. Dynamic optimization with equality path constraints. *Ind. Eng. Chem. Res., American Chemical Society*, 38:2350–2363, 1999.

[FL91] Claus Führer and B. J. Leimkuhler. Numerical solution of differential-algebraic equations for constrained mechanical motion. *Numerische Mathematik*, 59:55–69, 1991.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[GMW91] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Allan M. Wylde, 1991.

[Gri89] Andreas Griewank. On automatic differentiation. *Mathematical Programming*, pages 83–107, 1989. KTK Scientific Publishers, Tokyo.

[GS97] Kjell Gustafsson and Gustaf Söderlind. Control strategies for the iterative solution of non-linear equations in ODE solvers. *SIAM Journal of Scientific Computing*, 18(1):23–40, 1997.

[Gus92] Kjell Gustafsson. *Control of Error and Convergence in ODE Solvers*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, 1992.

[Hal35] P. Hall. On representatives of subsets. *Journal of London Math-*

ematical Society, 10:26–30, 1935.

[HAL99] Michael E. Henderson, Christopher R. Anderson, and Stephen L. Lyons, editors. *Object Oriented Methods for Interoperable Scientific and Engineering Computing*. SIAM, 1999.

[Har69] F. Harary. *Graph Theory*. Addison-Wesley, Reading MA, 1969.

[HW96] Ernst Hairer and Gerhard Wanner. *Solving ordinary differential equations II (Stiff and differential-algebraic problems)*. Springer-Verlag, 1996.

[HWN93] Ernst Hairer, Gerhard Wanner, and Syvert Paul Nørsett. *Solving ordinary differential equations I (Nonstiff problems)*. Springer-Verlag, second edition, 1993.

[KN99] Eleftherios Koutsofios and Stephen C. North. *Drawing graphs with "dot"*. AT&T Labs - Research, Murray Hill, NJ, 1999.

[Kvæ90] Anne Kværnø. Runge-Kutta methods applied to fully implicit differential-algebraic equations of index 1. *Mathematics of Computations*, 54(190):583–625, 1990.

[Lam91] J. D. Lambert. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons Ltd., 1991.

[Lut96] Mark Lutz. *Programming Python*. O'Reilly, first edition, 1996.

[Mat95] Sven Erik Mattson. Simulation of object-oriented continuous time models. *Mathematics and Computers in Simulation*, (39):513–518, 1995.

[MG98] C. Maffezzoni and R. Girelli. MOSES: Modular modelling of physical systems in an object-oriented database. *Mathematical and Computer Modelling of Dynamical Systems*, 4(2):121–147, 1998.

[Mod99] Modelica Design Group, http://www.modelica.org. *Modelica^{TM} - A Unified Object-Oriented Language for Physical Systems Modeling*, 1.3 edition, December 1999.

[MOE99] Sven Erik Mattson, Martin Otter, and Hilding Elmqvist. Modelica hybrid modeling and efficient simulation. Presented at the 38th IEEE Conference on Decision and Control, 1999.

[MS93] Sven Erik Mattsson and Gustaf Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal of Scientific Computing*, 14(3):677–692, 1993.

[Ols98] Hans Olsson. *Runge-Kutta solution of Initial Value Problems*. PhD thesis, Department of Computer Science, Lund University, 1998.

[Pan88] Constantinos C. Pantelides. The consistent initialization of differential-algebraic equations. *SIAM Journal of Scientific and Statistical Computing*, 9(2):213–231, 1988.

[Pou00] Mikael Zebbelin Poulsen. Studies in lattice theory with application to knowledge representation in numerical analysis. Technical report, Department of Informatics and Mathematical Modelling, 2000.

[Pou01] Mikael Zebbelin Poulsen. *Simpy - a modelling tool box*. Informatics and Mathematical modelling, Technical University of Denmark, http://www.imm.dtu.dk/∼mzp/simpy/, 1.0 edition, 2001.

[PW97] Mikael Zebbelin Poulsen and Falko Jens Wagner. Objekt-orienteret modellering og dynamisk simulering. Master's thesis, Technical University of Denmark, 1997. In Danish.

[PW98] Mikael Zebbelin Poulsen and Falko Jens Wagner. Objekt-orienteret modellering og dynamisk simulering (2. udgave). Technical report, Technical University of Denmark, 1998. In Danish.

[PW99] Mikael Zebbelin Poulsen and Falko Jens Wagner. *Projection methods*, chapter 3, pages 23–32. In Bendtsen and Thomsen [BT99], 1999.

[PWTH99] Mikael Zebbelin Poulsen, Falko Jens Wagner, Per Grove Thomsen, and Niels Houbak. Object oriented toolbox for modelling and simulation of dynamical systems. In Henderson et al. [HAL99], pages 155–164.

[RMB00] Gunther Reissig, Wade S. Martinson, and Paul I. Barton. Differential-algebraic equations of index 1 may have an arbitrarily high structural index. *SIAM Journal of Scientific Computing*, 21(6):1987–1990, 2000.

[Rud76] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, Inc., 3 edition, 1976.

[Sjö99] Anders Sjö. *Analysis of Computational Algorithms for Linear Multistep Methods*. PhD thesis, Lund University, 1999.

[Söd92] Gustaf Söderlind. Remarks on the stability of high-index DAEs with respect to parametric perturbations. *Computing, Springer-Verlag*, 49:303–314, 1992.

[TKE97] H. Tummescheit, M. Klose, and T. Ernst. Modelica and Smile - a case study applying object-oriented concepts to multi-facet modelling. In *Proc. European Simulation Symposium (ESS97)*, 1997.

[UKM95] J. Unger, A. Kröner, and W. Marquardt. Structural analysis of differential-algebraic equation systems and application. *Computers chem. Engng.*, 19(8):867–882, 1995.

[vRD01] Guido van Rossum and Fred L. Drake. *Python Tutorial.* PythonLabs, 2.1 edition, April 2001. Available at http://www.python.org.

[Wag00] Falko Jens Wagner. *Object-oriented Modeling of Energy Systems.* PhD thesis, Technical University of Denmark, December 2000.