Methods for the Solution and Filtering of Non-Linear Differential Algebraic Systems

Aquib Jahangir

Department of Electrical & Computer Engineering
McGill University
Montréal, Québec, Canada

November 15, 2021

A thesis presented for the degree of Masters of Electrical Engineering

Abstract

Differential-algebraic equations over the past years have become a widely accepted tool for the modeling and simulation of constrained dynamical systems in numerous applications, such as mechanical multibody systems, electrical circuit simulation, chemical engineering, control theory, fluid dynamics, and many other areas. In this thesis we have explored the theory and numerical methods for solving complex DAEs. In particular, we provide a systematic and detailed analysis of initial and boundary value problems for differential-algebraic equations. We also discuss numerical methods and software for the solution of these problems. This includes linear and nonlinear problems, over and underdetermined problems as well as control problems. Methods of incorporating constraints in the kalman filtering of DAEs are studied. This significantly improves the prediction accuracy of the filter. The use of filtering algorithm is explained with simple nonlinear vehicle tracking problem.

Abrégé

Au cours des dernières années, les équations algébriques différentielles sont devenues un outil largement accepté pour la modélisation et la simulation de systèmes dynamiques contraints dans de nombreuses applications, telles que les systèmes mécaniques multicorps, la simulation de circuits électriques, le génie chimique, la théorie du contrôle, la dynamique des fluides et de nombreux autres domaines. Dans cette thèse, nous avons exploré la théorie et les méthodes numériques pour résoudre les DAE complexes. Dans en particulier, nous fournissons une analyse systématique et détaillée des problèmes de valeurs initiales et limites pour les équations différentielles-algébriques. Nous discutons également des méthodes numériques et des logiciels pour la solution de ces problèmes. Cela inclut les problèmes linéaires et non linéaires, les problèmes surdéterminés et sous-déterminés ainsi que les problèmes de contrôle. Les méthodes d'incorporation des contraintes dans le filtrage kalman des DAE sont étudiées. Cela améliore considérablement la précision de prédiction du filtre. L'utilisation de l'algorithme de filtrage est expliquée par un simple problème de suivi non linéaire des véhicules.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Hannah Michalska for her continuous support and constant encouragements during my Master's thesis. I am really grateful for her motivation, enthusiasm, and immense knowledge. She guided and moulded me patiently through the entire research and helped to shape this research thesis elegantly.

I thank the current and also past students of our group at McGill University. I would like to give special thanks to my fellow research partner Raghuveer Rao Thoka for his feedback, cooperation, and of course for the friendly academic atmosphere during my research. I truly thank Mobeen Mahmood, Arham Uddin Syed and Arpit Aggarwal for their friendship, constant support and encouragement during my academic journey and the pandemic.

Last but not the least, I am forever grateful of my father Mohammed Jahangir Alam and my mother Shama Jahangir as words do not exist to describe how thankful and blessed I feel for everything they did and are still doing for me. They have kept me going in the times of ups and downs.

Contents

List of Figures

List of Tables

List of Acronyms

BVP	Bondary Value Problem.
CF	Canonical form.
ck	Cash-Karp.
DAE	Diiferential Algebraic Equations.
DAS	Diiferential Algebraic Systems.
DASPK	Solver for Differential Algebraic Equations.
DASSL	Differencial Algebraic System Solver.
DMM	Diagonal Matrix Multiplication.
EKF	Extended Kalman Filter.
GELDA	GEneral Linear Differential-Algebraic equation solver.
GENDA	GEneral Nonlinear Differential-Algebraic equation solver.
GUI	Graphical User Interface.
IDA	Implicit Differential-Algebraic solver.
IVP	Initial Value Problem.
mebdfi	Modified Extended Backward-Differentiation Formula Implicit Method.
MM	Mass Matrix.
NS	Non-Singular.
ODE	Ordinary Differential Equations.

odeplot	Ordinary Differential Equation Plot.
PDE	Partial Differential Equations.
PW	Pointwise.
rk	Runge-Kutta.
rkf	Runge-Kutta Fehlberg.
SUNDIALS	SUite of Nonlinear and DIfferential/ALgebraicequation Solvers.
SVD	singular value decomposition.
UKF	Unscented Kalman Filter.
vars	Variables.
VC	Variable Coefficients.

Chapter 1

Introduction and Literature Review

1.1 Introduction

The dynamical behavior of physical processes is usually modeled via differential equations. However, if the states of the physical system are in some ways constrained, like for example by conservation laws such as Kirchhoff's laws in electrical networks, or by position constraints such as the movement of mass points on a surface, then the mathematical model also contains algebraic equations to describe these constraints. Such systems, consisting of both differential and algebraic equations are called differential-algebraic systems, algebro-differential systems, implicit differential equations or singular systems [1].

The most general form of a differential-algebraic equation is

$$F(t, x, \dot{x}) = 0 \tag{1.1}$$

with

$F : \mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_{\dot{x}} \to R^m$

where $I \subseteq R$ is a (compact) interval and $D_x, D_{\dot{x}} \subseteq R^n$ are open, m, $n \in N$. The meaning

of the quantity $\dot{x}$ is ambiguous as in the case of ordinary differential equations. On one hand, it denotes the derivative of a differentiable function $x : I \to R^n$ with respect to its argument $t \in I$. On the other hand, in the context of (1.1), it is used as an independent variable of F. The reason for this ambiguity is that we want F to determine a differentiable function x that solves (1.1) in the sense that $F(t, x(t), \dot{x}(t)) = 0$ for all $t \in I$.
In connection with (1.1), we will discuss the existence of solutions. Uniqueness of solutions will be considered in the context of initial value problems, when we additionally require a solution to satisfy the condition

$$x(t_0) = x_0 \tag{1.2}$$

with given $t_0 \in I$ and $x_0 \in R^n$, and boundary value problems, where the solution is supposed to satisfy

$$b(x(t_0), x(t_f)) = 0 \tag{1.3}$$

with $b : D_x \times D_x \to R^d, I = [t_0, t_f]$ and some problem dependent integer d. It will turn out that the properties of differential-algebraic equations reflect the properties of differential equations as well as the properties of algebraic equations, but also that other phenomena occur which result from the mixture of these different types of differential and algebraic equations.

The basic theory for linear differential-algebraic equations with constant coefficients is given by

$$E\dot{x} = Ax + f(t) \tag{1.4}$$

where $E, A \in R^{(m,n)}$ and $f : I \times R^m$. This type of system is first used in the nineteenth century by the fundamental work of Weierstraß [2], [3] and Kronecker [4] on matrix pencils, until the pioneering work of Gear [5] for modeling dynamical systems. The subsequent developments in numerical methods for the solution of differential-algebraic equations made differential-

algebraic equations to be used directly in numerical simulation. Since then an explosion of the research in this area has taken place and has led to a wide acceptance of differential-algebraic equations in the modeling and simulation of dynamical systems. Despite the wide applicability and the great importance, only few monographs and essentially no textbooks are so far devoted to this subject, see [6], [7], [8], [9]. Partially, differential-algebraic equations are also discussed in [10], [11], [12], [13], [14], [15], [16]. However, the implicit systems of the form (1.1) were usually transformed into ordinary differential equations via analytical transformations.

$$\dot{y} = g(t, y) \tag{1.5}$$

One way to achieve this is to explicitly solve the constraint equations analytically in order to reduce the given differential-algebraic equations to an ordinary differential equations with fewer variables. However, this approach heavily relies on either transformations by hand or symbolic computation software which are both not feasible for medium or large scale systems.

Another possibility is to differentiate the algebraic constraints in order to get an ordinary differential equation with the same number of variables. Due to the necessary use of the implicit function theorem, this approach is often difficult to perform. Moreover, due to possible changes of coordinate bases, the resulting variables may have no physical meaning. In the context of numerical solution methods, it was observed in Gear [5] approach that the numerical solution may drift off from the constraint manifold after a few integration steps. For this reason, in particular in the simulation of mechanical multibody systems, stabilization techniques were developed to address this difficulty. But it is in general preferable to develop methods that operate directly on the given differential-algebraic equation.

In view of the described difficulties, the development of numerical methods that can be

directly applied to the differential-algebraic equation has been the subject of a large number of research projects in the last thirty years and many different directions have been taken. In particular, in combination with modern modeling tools (that automatically generate models for substructures and link them together via constraints), it is important to develop generally applicable numerical methods as well as methods that are tailored to a specific physical situation. It would be ideal if such an automatically generated model could be directly transferred to a numerical simulation package via an appropriate interface so that in practical design problems the engineer can optimize the design via a sequence of modeling and simulation steps. To obtain such a general solution package for differential-algebraic equations is an active area of current research that requires strong interdisciplinary cooperation between researchers working in modeling, the development of numerical methods, and the design of software. A major difficulty lies in understanding the analytical and numerical properties of differential-algebraic systems. In particular, the treatment of bifurcations or switches in nonlinear systems and the analysis and numerical solution of heterogeneous (coupled) systems combined of differential-algebraic equations and partial differential equations (sometimes called partial differential-algebraic equations) represent major research tasks [1].

Our thesis gives a coherent introduction to the theoretical analysis of differential-algebraic equations and to present some appropriate numerical methods for initial and boundary value problems. For the analysis of differential-algebraic equations, there are several paths that can be followed. A very general approach is given by the geometrical analysis initiated by Rheinboldt [17], see also [9], to study differential-algebraic equations as differential equations on manifolds. Our main approach, however, will be the algebraic path that leads from the theory of matrix pencils byWeierstraß and Kronecker via the fundamental work of Campbell on derivative arrays [44] to canonical forms for linear variable coefficient systems [18], [19]

and their extensions to nonlinear systems in the work of the authors ([20], [21], [22], [23]).

The algebraic approach allows the study of generalized solutions and the treatment of over and under determined systems as well as control problems. At the same time, it leads to new discretization methods and new numerical software.

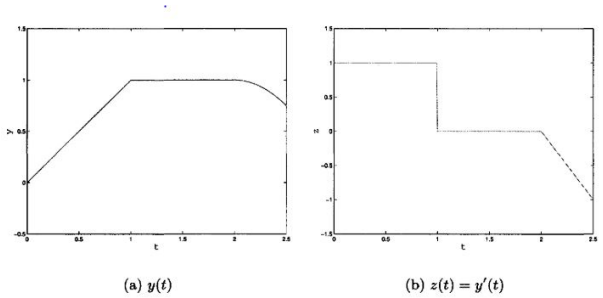
Unfortunately, the simultaneous development of the theory in many different research groups has led to a large number of slightly different existence and uniqueness results, particularly based on different concepts of the so-called index. The general idea of all these index concepts is to measure the degree of smoothness of the problem that is needed to obtain existence and uniqueness results. To clarify exact implementation of our thesis , we now briefly discuss the most common approaches.

1.2 ODE and DAE

To understand the similarity and the difference between DAEs and ODEs, consider two functions *y(t)* and *z(t)* which are related on some interval $[0, b]$ by

$$\dot{y(t)} = z(t), \quad 0 \leq t \leq b \tag{1.6}$$

and the task of recovering one of these functions from the other via the differential equation (1.6). To recover z from y, one needs to differentiate *y(t)*, an automatic process familiar to us from a first calculus course. To recover y from z one needs to integrate $z(t)$— a less automatic process necessitating an additional boundary condition (such as the value of $y(0)$). This would suggest that differentiation is a simpler, more straightforward process than integration. On the other hand, though, note that *y(t)* is generally a smoother function than *z(t)*. For instance, if *z(t)* is bounded but has jump discontinuities, then *y(t)* will not have any discontinuities; see Figure 1.1. Thus, integration is a smoothing process, while



(a) $y(t)$ (b) $z(t) = y'(t)$

Figure 1.1: A function and its less smooth derivative.

differentiation is an anti-smoothing process. The differentiation process is unstable[1] to noisy perturbations, although it is often very simple to carry out analytically.

A differential equation involves integration, hence smoothing: the solution *y(t)* of the linear system $\dot{y} = A_y + q(t)$ is one degree smoother than *q(t)*. A DAE, on the other hand, involves both differentiations and already integrated equations. The class of DAEs contains all ODEs but it also contains problems where both differentiations and integrations are intertwined in a complex manner when simple differentiations may no longer be possible, but their effect complicates the numerical integration process, potentially well beyond what we have seen so far [1].

Implicit Function Theorem:

Let $f : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function, and let $\mathbb{R}^{n+m}$ have coordinates (x, y). Fix a point $(a, b) = (a_1, \ldots, a_n, b_1, \ldots, b_m)$ with $f(a, b) = 0$, where $0 \in \mathbb{R}^m$ is the zero vector. If the Jacobian matrix

$$J_{f,\mathbf{y}}(a, b) = \left[\frac{\partial f_i}{\partial y_j}(a, b)\right]$$

[1]If we add to *y(t)* a small perturbation cos wt, where $|e| \ll 1$ and $w; > |\epsilon^{-}1|$, then *z(t))* is perturbed by a large amount $|w\epsilon|$.

is invertible, then there exists an open set $U \subset \mathbb{R}^n$ containing a such that there exists a unique continuously differentiable function $g : U \to \mathbb{R}^m$ such that $g(a) = (b)$, and $f(x, g(x)) = 0$ for all $x \in U$. Moreover, the partial derivatives of g in U are given by the matrix product

$$\frac{\partial g}{\partial x_j}(x) = -\left[J_{f,y}(x, g(x))\right]^{-1}_{m\times m} \left[\frac{\partial f}{\partial x_j}(x, g(x))\right]_{m\times 1}$$

1.3 Solvability Concepts

To analyze the DAE system (1.1) from the view point of existence of solutions we have to specify the function space in which the solution should lie.

Definition 1.1. (Classical Solutions) Let $C^k(\mathbb{I}, R^n)$ denote the vector space of all k-times continuously differentiable functions from the real interval $\mathbb{I}$ into the real vector space R^n.

1. A function $x \in C^1(\mathbb{I}, R^n)$ is called a solution of (1.1), if it satisfies (1.1) pointwise.

2. The function $x \in C^1(\mathbb{I}, R^n)$ is called a solution of the initial value problem (1.1) with initial condition (1.2), if it furthermore satisfies (1.2).

3. An initial condition (1.2) is called consistent with F, if the associated initial value problem has at least one solution.

A problem is called solvable if it has at least one classical solution. Note that in the previous section solvability is used only for systems which have a unique solution when consistent initial conditions are provided. If the solution of the initial value problem is not unique, which is in the case of control problems, then further conditions have to be specified to single out specific desired solutions.

Generalized Solutions

The smoothness requirements for the forcing function f in (1.4) can be mildly relaxed if the solution is allowed to be less smooth. The consistency conditions for the initial values, however cannot be relaxed when considering classical solutions (Definition 1.1). Other way to remove consistency conditions and to relax smoothness requirements is to allow generalized functions (or distributions) [1], as solutions of (1.4). For the analysis of differential-algebraic equations, this approach is relatively recent. Several different directions can be followed that allow to include non-differentiable forcing functions f or non-consistent initial values. A very elegant and completely algebraic approach was introduced in [1] to treat the problem by using a particular class of distributions introduced first in [1] in the study of control problems. We essentially follow this [1] approach.

Example: The discharging of a capacitor via a resistor, in Figure 1.2, can be modeled by the system as shown below

$$x_1 - x_3 = u(t), \quad C(\dot{x}_3 - \dot{x}_2) + \frac{x_1 - x_2}{R} = 0, \quad x_3 = 0$$

where x_1, x_2, x_3 denote the potentials in the different parts of the circuit. This system can be reduced to the ordinary differential equation

$$\dot{x}_2 = -\frac{1}{RC}x_2 + \frac{1}{RC}u(t).$$

Let the input voltage u be defined by $u(t) = u_0 > 0$ for $t < 0$ and $u(t) = 0$ for $t \geq 0$. Thus, we want to study the behavior of the circuit when we close a switch between x_1 and x_3. As initial condition, we take $x_2(0) = u_0$. The differential equation can then be solved separately for $t < 0$ and $t > 0$. Since both parts can be joined together into a continuous function, we may view x_2 defined by

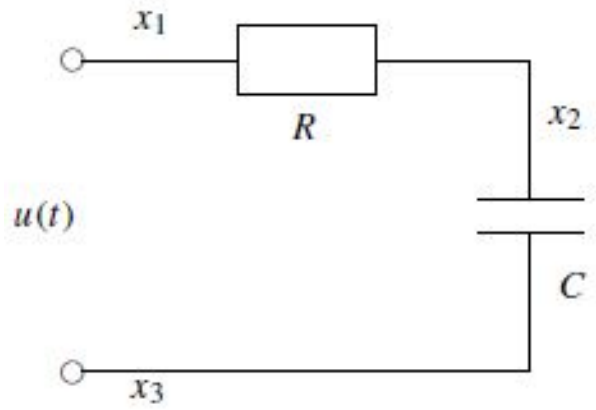


Figure 1.2: Discharging a capacitor.

$$x_2 = \begin{cases} u_0 & for \quad t < 0 \\ u_0 e^{-t/RC} & for \quad t \geq 0 \end{cases}$$

as solution everywhere in R. This procedure can be formalized for linear differential equations working with piecewise continuous forcing function and piecewise continuously differentiable solutions. However, such a solution is not differentiable at points where the forcing function is discontinuous.

1.4 Mathematical Structure and Index Concepts

Since a DAE involves a mixture of differentiable functions and algebraic equations, we may hope that applying analytical differentiations to a given system and eliminating constraints as needed will yield an explicit ODE system. This is true unless the DAE problem is singular. The number of differentiations needed for this transformation is called the index of the DAE. Thus, ODEs have index 0 .

We must specify m initial or boundary conditions to find the solution of a DAE of order

m. Complex DAE systems most probably include ODE subsystems. Thus, the DAE system will have l degrees of freedom, where l is anywhere between 0 and m [24].

It may be difficult to find the necessary information about the constraints needed to solve the DAE system. Often the entire initial solution vector is known. Initial or boundary conditions which are specified for the DAE must be consistent i.e they must satisfy the constraints of the system. The important difference between index-1 and higher-index (index greater than 1) is that DAE's with higher-index include some hidden constraints (1.8b). These hidden constraints (1.8b) are the derivatives of the explicitly stated constraints in the system. Index-2 systems include hidden constraints which are the first derivative of explicitly stated constraints. Higher-index systems include hidden constraints which correspond to higher-order derivatives.

The most general form of a fully implicit DAE is given by

$$F(t, y, \dot{y}) = 0 \tag{1.7}$$

where $\frac{\partial F}{\partial \dot{y}}$ is the Jacobian matrix and may be singular. The rank and structure of the Jacobian matrix may depend on the solution *y(t)*, and for simplicity we will always assume that it is independent of t. Consider the following special case of a semi-explicit DAE with hidden constraints

$$\dot{x} = f(t, x, z) \tag{1.8a}$$

$$0 = g(t, x, z) \tag{1.8b}$$

The index for (1.7) is 1, if $\frac{\partial g}{\partial z}$ is non-singular, because then one differentiation of (1.8b) yields $\dot{z}$. For the semi-explicit index-1 DAE we can distinguish between differential variables *x(t)* and algebraic variables *z(t)*. The algebraic variables may be less smooth than the differential

variables by one degree (e.g., the algebraic variables may be non-differentiable).

Generally, y in (1.7) may contain a mix of differential and algebraic components, which makes the numerical solution of such high-index problems (see section 3.5) complex. DAE of (1.7) can be written in the semi-explicit form (1.8) but with the index increased by 1, upon defining $\dot{y} = z$, which gives

$$\dot{y} = z \tag{1.9a}$$

$$0 = F(t, y, z) \tag{1.9b}$$

The rearranged equations (1.9) does not make the problem easier to solve. Consider a semi-explicit index-2 DAE system (1.10), where $\dot{w} = z$. It is easily shown that the system

$$\dot{x} = f(t, x, \dot{w}) \tag{1.10a}$$

$$0 = g(t, x, \dot{w}) \tag{1.10b}$$

is an index-1 DAE and yields exactly the same solution for x as (1.8). The class of fully implicit index-1 DAEs of the form (1.7) and semi-explicit index-2 DAEs of the form (1.8) are therefore equivalent.

In the analysis of linear differential-algebraic equations with constant coefficients (1.4), all properties of the system can be determined by computing the invariants of the associated matrix pair (E, A). In particular, the size of the largest Jordan block [25] to an infinite eigenvalue in the associated Kronecker canonical form [25], called **index**, plays a major role in the analysis and determines (at least in the case of so-called regular pairs) the smoothness that is needed for the forcing function f in (1.4) to guarantee the existence of a classical solution (Definition 1.1).

Inspired by this case, it was first defined as an analogous index for linear time-varying systems and then for general implicit systems, see [26]. However, it was soon realized that a direct generalization by linearization and consideration of the local linearized constant coefficient system does not lead to a reasonable concept. The reason is that important invariants of constant coefficient systems are not even locally invariant under nonconstant equivalence transformations (Definition 1.2). This observation led to a multitude of different index concepts even for linear systems with variable coefficients, see [27].

The differentiation index is the minimum number of times that an equation (1.1) must be differentiated with respect to t in order to determine x as a continuous function of t and x. The procedure to solve the algebraic equations (using their derivatives if necessary) is by transforming the implicit system (1.1) to an ordinary differential equation. Although the concept of the differentiation index is widely used, it has a major drawback, since it is not suited for over and underdetermined systems. The reason for this is that it is based on a solvability concept that requires unique solvability. In our thesis, we will focus on the concept of the strangeness index [18], [20], [21], [23], which generalizes the differentiation index to over and underdetermined systems. We will not discuss other index concepts such as the geometric index [17], the tractability index [7], [28], [29] or the structural index [30] in our thesis. Perturbation index is of great importance in the numerical treatment of differential-algebraic equations that was introduced in [8] to measure the sensitivity of solutions with respect to perturbations of the problem. For a detailed analysis and a comparison of various index concepts with the differentiation index, see [27], [31], [32], [33], [34], [35].

The purpose of index of DAE is to categorise different types of differential-algebraic equations with respect to the difficulty to solve them analytically as well as numerically. In view of the above classification aspect, the differentiation index was introduced to determine how far the differential-algebraic equation is away from an ordinary differential equation, for

which the analysis and numerical techniques are well established. However, pure algebraic equations of (1.1) are equally well analyzed. Furthermore, it would certainly not make sense to turn a uniquely solvable classical linear system $Ax = b$ into a differential equation, since the solution would not be unique anymore without specifying initial conditions. In view of above discussion, it seems desirable to classify differential-algebraic equations by their distance between system of ordinary differential equations and purely algebraic equations. However, it can be concluded the index of an ordinary differential equation and that of a system of algebraic equations should be the same.

1.5 Linear and Non-Linear DAEs

1.5.1 Linear DAEs with Constant Coefficients

Linear differencial-algebraic equations with constant coefficients are of the form

$$E\dot{x} = Ax + f(t), \tag{1.11}$$

with E, $A \in R^{m,n}$ and $f \in R(\mathbb{I}, R^m)$, possibly together with an initial condition

$$x(t_0) = x_0. \tag{1.12}$$

Such equations occur by linearization of autonomous nonlinear problems with respect to constant (or critical) solutions, where f plays the role of a perturbation [36].

Canonical forms [36]

The properties of the above equations (1.11) can be well understood in the works of Weierstraß [31] and Kronecker [32]. In the following, we describe the main aspects of solving equation (1.11) by purely algebraic techniques. Scaling (1.11) by a nonsingular matrix $P \in R^{m,n}$, and the function x according to $x = Q\dot{x}$ with a nonsingular matrix $Q \in R^{n,n}$, we obtain

$$\tilde{E}\dot{\tilde{x}} + \tilde{f}(t), \quad \tilde{E} = PEQ, \quad \tilde{A} = PAQ, \quad \tilde{f} = Pf, \tag{1.13}$$

which is again a linear differential-algebraic equation with constant coefficients. Moreover, the relation $x = Q\tilde{x}$ gives a one-to-one correspondence between the corresponding solution sets. This means that we can consider the transformed problem (1.13) instead of equation (1.11) with respect to solvability. The following definition (1.2) of equivalence is now evident.

Definition 1.2. Two pairs of matrices (E_i, A_i), E_i,$A_i \in R^{m,n}$, i = 1, 2, are called (strongly) equivalent if there exist nonsingular matrices $P \in R^{m,m}$ and $Q \in R^{n,n}$, such that

$$E_2 = PE_1Q, \quad A_2 = PA_1Q. \tag{1.14}$$

If this is the case, we write $(E_1, A_1) \sim (E_2, A_2)$ [36].

Lemma 1.3. *The relation introduced in Definition 1.2 is an equivalence relation.*

Having defined the equivalence relation (Definition 1.2), the standard procedure is to look for a canonical form, i.e. to look for a matrix pair (A in 1.14) which is equivalent to a given matrix pair (E in 1.14) and which has a simple form from which we can directly read off the properties and invariants of the corresponding differential-algebraic equation. In (1.14), such a canonical form is represented by the so-called Kronecker [32] canonical form. We therefore restrict ourselves to a special case i.e we present only the general result without proof. In

the next section, we will derive a canonical form for linear differential-algebraic equations with variable coefficients which will generalize the Kronecker [32] canonical form at least in the sense that existence and uniqueness results can be obtained in the same way as from the Kronecker canonical form for the case of constant coefficients.

Theorem 1.4. *Let E, $A \in R^{m,n}$, then there exist nonsingular matrices $P \in R^{m,m}$ and $Q \in R^{n,n}$ such that (for all $\lambda \in R$)*

$$P(\lambda E - A)Q = diag(\mathcal{L}_{\epsilon 1}, ..., \mathcal{L}_{\epsilon p}, \mathcal{M}_{\eta 1}....., \mathcal{M}_{\eta q}, \mathcal{J}_{\rho 1}, ...\mathcal{J}_{\rho r}, \mathcal{N}_{\sigma 1}, ...\mathcal{N}_{\sigma s}) \tag{1.15}$$

Where the block entries have the following properties:

1. *Every entry $\mathcal{L}_{\epsilon j}$ is a bidiagonal block of size $\epsilon \times (\epsilon + 1)$, $\epsilon_j \in \mathbb{N}_0$, of the form*

$$\lambda \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}$$

2. *Every entry $\mathcal{M}_{\eta j}$ is a bidiagonal block of size $(\eta_j + 1) \times \eta_j, \eta_j \in \mathbb{N}_0$, of the form*

$$\lambda \begin{bmatrix} 1 & & & \\ 0 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & & 0 \end{bmatrix} - \begin{bmatrix} 0 & & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 0 \\ & & & 1 \end{bmatrix}$$

3. *Every entry :* $\mathcal{J}_{\rho j}$ *is a Jordan block of size* $\rho_j \times \rho_j, \rho_j \in \mathbb{N}, \lambda_j \in \mathbb{N}$*, of the form*

$$\lambda \begin{bmatrix} 1 & & & \\ & \ddots & \ddots & \\ & & & 1 \end{bmatrix} - \begin{bmatrix} \lambda_j & 1 & & \\ & \ddots & \ddots & 1 \\ & & & \lambda_j \end{bmatrix}$$

4. *Every entry :* $\mathcal{N}_{\sigma j}$ *is a nilpotent block of size* $\sigma_j \times \sigma_j, \sigma_j \in \mathbb{N}$*, of the form*

$$\lambda \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & 1 \\ & & & 0 \end{bmatrix} - \begin{bmatrix} 1 & & & \\ & \ddots & \ddots & \\ & & & 1 \end{bmatrix}$$

The Kronecker canonical [36] form is unique up to permutation of the blocks, i.e., the kind, size and number of the blocks are characteristic for the matrix pair (E, A) [36].

Definition 1.5. Let E, $A \in R^{m,n}$. The matrix pair (E, A) is called regular if $m = n$ and so-called characteristic polynomial p defined by

$$p(\lambda) = det(\lambda E - A) \tag{1.16}$$

is not zero polynomial. A matrix pair which is not regular is called singular [36].

Lemma 1.6. Every matrix pair which is strongly equivalent to a regular matrix (Definition 1.5) pair is regular [36].

Definition 1.7. Let the matrix pair (E, A) be regular and $E, A \in R^{n,n}$. Then

$$(E, A) \sim \left(\begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix}, \begin{bmatrix} J & 0 \\ 0 & I \end{bmatrix} \right) \tag{1.17}$$

where J is a matrix in Jordan canonical form [36] and N is a nilpotent matrix also in Jordan canonical form. Moreover, it is allowed that one or the other block is not present [36].

Theorem 1.8. Let the pair of square matrices (E, A) be regular. Let P and Q be nonsingular matrices which transform (1.11) and (1.12) to Weierstraß canonical form (E, A) [36] i.e.,

$$PEQ = \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix}, \quad PAQ = \begin{bmatrix} J & 0 \\ 0 & I \end{bmatrix}, \quad P_f = \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \end{bmatrix} \tag{1.18}$$

and set

$$Q^{-1}x = \begin{bmatrix} \tilde{x_1} \\ \tilde{x_2} \end{bmatrix}, \quad Q^{-1}x_0 = \begin{bmatrix} \tilde{x_{1,0}} \\ \tilde{x_{2,0}} \end{bmatrix} \tag{1.19}$$

Furthermore, let $v = ind(E, A)$ where ind is the index of the matrix pair and $f \in \mathsf{R}^v(\mathbb{I}, R^n)$ .Then we have the following:

1. The differential-algebraic equation (1.11) is solvable.

2. An initial condition (1.12) is consistent if and only if

$$\tilde{x}_{2,0} = -\sum_{i=0}^{v-1} N^i \tilde{f}_2^{(i)}(t_0)$$

   In particular, the set of consistent initial values x_0 is nonempty.

3. Every initial value problem with consistent initial condition is uniquely solvable [36].

Theorem 1.9. Let E, $A \in R^{m,n}$ and suppose that (E, A) is a singular matrix pair.

1. If rank $(\lambda E - A) < n$ for all $\lambda \in R$, then the initial value problem

$$E\dot{x} = Ax, \quad x(t_0) = 0 \tag{1.20}$$

has a nontrivial solution.

2. If rank $(\lambda E - A) = n$ for some $\lambda \in R$ and hence $m > n$, then there exist arbitrarily smooth forcing function f for which the corresponding differential algebraic equation is not solvable [36].

1.5.2 Linear DAEs with variable coefficients

Linear differencial-algebraic equations with variable coefficients are of the form

$$E(t)\dot{x} = A(t)x + f(t), \tag{1.21}$$

with E, $A \in R(\mathbb{I}, R^{m,n})$ and $f \in C(\mathbb{I}, R^m)$, again possibly together with an initial condition [36]

$$x(t_0) = x_0. \tag{1.22}$$

Canonical forms [36]

Comparing with the case of constant coefficients in previous section , in view of Theorem 1.8, an obvious idea in dealing with (1.21) for $m = n$ would be requiring regularity of the matrix pair $(E(t), A(t))$ for all $t \in \mathbb{I}$. Unfortunately, this does not guarantee unique solvability of the initial value problem. Moreover, it turns out that these properties are completely independent of each other.

Definition 1.10. Two pairs $(E_i, A_i), E_i, A_i \in R(\mathbb{I}, R^{m,n})$, i = 1, 2, of matrix functions are called (globally) equivalent if there exist pointwise nonsingular matrix functions $P \in R(\mathbb{I}, R^{m,m})$ and $Q \in R^1(\mathbb{I}, R^{n,n})$ such that

$$E_2 = PE_1Q, \quad A_2 = PA_1Q - PE_1\dot{Q} \tag{1.23}$$

as equality of functions. We again write $(E_1, A_1) \sim (E_2, A_2)$ [36].

Lemma 1.11. The relation introduced in Definition 1.10 is an equivalence relation.

Definition 1.12. Two pairs of matrices $(E_i, A_i), E_i, A_i \in R^{m,n}$ i = 1, 2, are called (locally) equivalent if there exist matrices $P \in R^{m,m}$ and $Q, R \in R^{n,n}$ where P, Q are nonsingular, such that

$$E_2 = PE_1Q, \quad A_2 = PA_1Q - PE_1R \tag{1.24}$$

Again, we write $(E_1, A_1) \sim (E_2, A_2)$ and distinguish from the equivalence relation in Definition 1.9 by the type of pairs (matrix or matrix function) [36].

Lemma 1.13. The relation introduced in Definition 1.12 is an equivalence relation.

Definition 1.14. Let $E, A \in R^{m,n}$ and introduce the following spaces and matrices:

$$T \quad \text{basis of kernel E},$$

$$Z \quad \text{basis of corange E} = \text{kernel } E^H,$$

$$T' \quad \text{basis of cokernel E} = \text{range } E^H,$$

$$V \quad \text{basis of corange } (Z^H \text{ A T})$$

Then the quantities

$$r = rankE, \quad (rank)$$

$$a = rank(Z^H AT) \quad (algebraicpart)$$

$$a = rank(V^H Z^H AT') \quad (strangeness)$$

$$d = r - s, \quad (differentialpart)$$

$$u = n - r - a, \quad (undeterminedvariables)$$

$$v = m - r - a - s, \quad (vanishingequations)$$

are invariant under (1.24), and (E, A) is (locally) equivalent to the canonical form

$$\left(\begin{bmatrix} I_s & 0 & 0 & 0 \\ 0 & I_d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & I_d & 0 & 0 \\ 0 & 0 & I_a & 0 \\ I_s & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) \begin{pmatrix} s \\ d \\ a \\ s \\ v \end{pmatrix}, \tag{1.25}$$

where all diagonal blocks with the exception of the last block are square and the column in the last block in both matrices (1.25) has size u [36].

Theorem 1.15. Let $E \in R^l(\mathbb{I}, R^{m,n}), l \in \mathbb{N}_0 \cup \{\infty\}$, with rank $E(t) = r$ for all $t \in \mathbb{I}$. Then there exist pointwise unitary (and therefore nonsingular) functions $U \in R^l(\mathbb{I}, R^{n,n})$, such that

$$U^H E V = \begin{bmatrix} \sum & 0 \\ 0 & 0 \end{bmatrix} \tag{1.26}$$

with pointwise nonsingular $\sum \in C^l(\mathbb{I}, \mathbb{C}^{r,r})$ [36].

Theorem 1.16. Let E,$A \in R(\mathbb{I}, R^{m,n})$ be sufficiently smooth and suppose that

$$r(t) \equiv r, \quad a(t) \equiv a, \quad s(t) \equiv s \tag{1.27}$$

for the local characteristic values of $(E(t), A(t))$. Then, (E, A) is globally equivalent to the

canonical form

$$\left(\begin{bmatrix} I_s & 0 & 0 & 0 \\ 0 & I_d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & A_{12} & 0 & A_{14} \\ 0 & 0 & 0 & A_{24} \\ 0 & 0 & I_a & 0 \\ I_s & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) \begin{pmatrix} s \\ d \\ a \\ s \\ v \end{pmatrix}, \tag{1.28}$$

All entries A_{ij} are again matrix functions on $\mathbb{I}$ and the column in the last block of both matrix functions of (1.28) has size $u = n - s - d - a$ [36].

Theorem 1.17. Assume that the pairs (E, A) and $(\tilde{E}, \tilde{A})$ of matrix functions are (globally) equivalent and in global canonical form (1.28). Then the modified pairs (E_{mod}, A_{mod}) and $(\tilde{E_{mod}}, \tilde{A_{mod}})$ obtained by passing from (1.28) are also (globally) equivalent [36].

Definition 1.18. Let (E, A) be a pair of sufficiently smooth matrix functions. Let the sequence $(r_i, a_i, s_i), i \in \mathbb{N}_0$, be well defined. In particular, let (1.27) hold for every entry (E_i, A_i) of the above sequence. Then, we call

$$\mu = min\{i \in \mathbb{N}_0 | s_i = 0\} \tag{1.29}$$

the strangeness index of (E, A) in (1.21). In case that $\mu = 0$ we call (E, A) in (1.21) strangeness-free [36].

Theorem 1.19. Let the strangeness index μ of (E, A) as in (1.29) be well defined (i.e., let (1.27) hold for every entry (E_i, A_i) of the above sequence and let $f \in R^{\mu}(\mathbb{I}, R^m)$. Then the differential-algebraic equation (1.21) is equivalent (in the sense that there is a one-to-one correspondence between the solution spaces via a pointwise nonsingular matrix function) to

a differential-algebraic equation of the form

$$\dot{x}_1 = A_{13}(t)x_3 + f_1(t), \quad d_\mu \tag{1.30a}$$

$$0 = x_2 + f_2(t), \quad a_\mu \tag{1.30b}$$

$$0 = f_3(t), \quad v_\mu \tag{1.30c}$$

where $A_{13} \in R(\mathbb{I}, R^{d_\mu, u_\mu})$, and the forcing function f_1, f_2, f_3 are determined by $f^0, ..., f^\mu$ [36].

Corollary 1.20. Let the strangeness index μ of (E, A) as in (1.29) be well defined and let $f \in R^{\mu+1}(\mathbb{I}, R^m)$. Then we have:

1. The problem (1.21) is solvable if and only if the v_μ functional consistency conditions

$$f_3 = 0 \tag{1.31}$$

   are fulfilled.

2. An initial condition (1.22) is consistent if and only if in addition the a_μ conditions

$$x_2(t_0) = -f_2(t_0) \tag{1.32}$$

   are implied by (1.22).

3. The corresponding initial value problem is uniquely solvable if and only if in addition

$$u_\mu = 0 \tag{1.33}$$

Observe that the stronger assumption on the smoothness of the forcing function, i.e., that $f \in R^{\mu+1}(\mathbb{I}, R^m)$ rather than $f \in R^\mu(\mathbb{I}, R^m)$, is only used to guarantee that x_2 is continuously

differentiable. The structure of (1.30), however, suggests that it is sufficient to require only continuity for the parts x_2 and x_3 of the solution [36].

Lemma 1.21. Let the strangeness index μ of (E, A) as in (1.29) be well defined. Let the process leading to Theorem 1.19 yield a sequence $(E_i, A_i), i \in \mathbb{N}_0$, with characteristic values $(r_i, a_i, s_i, d_i, u_i, v_i)$ according to the quantities in Definition 1.14 and

$$(E_i, A_i) \sim \left(\begin{bmatrix} I_{s_i} & 0 & 0 & 0 \\ 0 & I_{d_i} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & A_{12}^i & 0 & A_{14}^i \\ 0 & 0 & 0 & A_{24}^i \\ 0 & 0 & I_{a_i} & 0 \\ I_{s_i} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) \begin{pmatrix} s_i \\ d_i \\ a_i \\ s_i \\ v_i \end{pmatrix}, \tag{1.34}$$

where the last block column in both matrix functions has size u_i. Defining

$$b_0 = a_0, \qquad b_{i+1} = \operatorname{rank} A_{14}^{(i)} \tag{1.35a}$$

$$c_0 = a_0 + s_0, \qquad c_{i+1} = \operatorname{rank} \begin{bmatrix} A_{12}^{(i)} & A_{12}^{(i)} \end{bmatrix} \tag{1.35b}$$

$$w_0 = v_0, \qquad w_{i+1} = v_{i+1} - v - i \tag{1.35c}$$

we have

$$r_{i+1} = r_i - s_i, \tag{1.36a}$$

$$a_{i+1} = a_i + s_i + b_{i+1} = c_0 + ... + c_{i+1} - s_{i+1}, \tag{1.36b}$$

$$s_{i+1} = c_{i+1} - b_{i+1}, \tag{1.36c}$$

$$d_{i+1} = r_{i+1} - s_{i+1} = d_i - s_{i+1}, \tag{1.36d}$$

$$w_{i+1} = s_i - c_{i+1}, \tag{1.36e}$$

$$u_{i+1} = u_0 - b_1 - .... - b_{i+1}, \tag{1.36f}$$

$$v_{i+1} = v_0 + w_1 - .... + w_{i+1}, \tag{1.36g}$$

$$\tag{1.36h}$$

Theorem 1.22. Let the strangeness index μ of (E, A) as in (1.29) be well defined [36]. Then, (E, A) is (globally) equivalent to a pair of the form

$$\left(\begin{bmatrix} I_{d_\mu} & 0 & W \\ 0 & 0 & F \\ 0 & 0 & G \end{bmatrix}, \begin{bmatrix} 0 & * & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_{a_\mu} \end{bmatrix} \right), \tag{1.37}$$

with

$$F = \begin{bmatrix} 0 & F_\mu & & * \\ & \ddots & \ddots & F_1 \\ & & \ddots & 0 \end{bmatrix}, \quad G = \begin{bmatrix} 0 & G_\mu & & * \\ & \ddots & \ddots & G_1 \\ & & \ddots & 0 \end{bmatrix} \tag{1.38}$$

where F_i and G_i have sizes $w_i \times c_{i-1}$ and $c_i \times c_{i-1}$, respectively, with w_i,c_i as in (1.27),

and W = [0 · · ·] is partitioned accordingly. In particular, F_i and G_i together have full row rank, i.e.,

$$rank \begin{bmatrix} F_i \\ G_i \end{bmatrix} = c_i + w_i = s_{i-1} \leqslant c_{i-1} \tag{1.39}$$

1.5.3 Nonlinear differential-algebraic equations

In general nonlinear systems of differential-algebraic equations are of the form

$$F(t, x, \dot{x}) = 0 \tag{1.40}$$

However, we first study the case where, the number of equations equals the number of unknowns. Thus, we consider with $F \in R(\mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_{\dot{x}}, \mathbb{R}^n)$ with $\mathbb{D}_x, \mathbb{D}_{\dot{x}} \subseteq \mathbb{R}^n$ open. Again, we may have an initial condition

$$x(t_0) = x_0. \tag{1.41}$$

together with (1.40) [36] [37].

Existence and uniqueness of solutions [36]

A typical approach to analyze nonlinear problems is to use the implicit function theorem in order to show that a (given) solution is locally unique. To apply the implicit function theorem, one must require that for the given solution the Fréchet derivative of the underlying nonlinear operator is regular (i.e., has a continuous inverse). Using the Fréchet derivative it can be interpreted as linearization of the nonlinear problem. In (1.40) the linearization will lead to linear differential-algebraic equations with variable coefficients. Also, we must deal with inflated systems that are obtained by successive differentiation of the original

equation with respect to time. We are then concerned with the question that how these differentiated equations and their linearizations look like. We must investigate whether these two processes (differentiation and linearization) commute, i.e., whether it makes a difference first to differentiate and then to linearize or vice versa.

Theorem 1.23. Let $E \in R^l(\mathbb{D}, \mathbb{R}^{m,n}), l \in \mathbb{N}_0 \cup \{\infty\}$, with rank $E(x) = r$ for all $x \in \mathbb{M} \subseteq \mathbb{R}^k$. For every $\hat{x} \in \mathbb{M}$ there exist a sufficiently small neighborhood $\mathbb{V} \subseteq \mathbb{D}$ of $\hat{x}$ and matrix functions $T \in R^l(\mathbb{V}, \mathbb{R}^{n,n-r}), Z \in R^l(\mathbb{V}, \mathbb{R}^{m,m-r})$ with pointwise orthonormal columns such that

$$ET = 0, \quad Z^T E = 0 \tag{1.42}$$

on $\mathbb{M}$

Definition 1.24. Given a differential-algebraic equation as in (1.40), the smallest value of μ is called the strangeness index of (1.40). If $\mu = 0$, then the differential-algebraic equation is called strangeness-free. The definition of the strangeness index for a nonlinear differential-algebraic equation is a straightforward generalization of the strangeness index for a linear differential-algebraic equation.

Assumption 1.25. The function $f : \mathbb{R}^n \times \mathcal{D}_f \times \mathcal{I}_f \to \mathbb{R}^k$ is continuous on the open set $\mathbb{R}^n \times \mathcal{D}_f \times \mathcal{I}_f \subseteq \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}$ and has continuous partial derivatives f_y, f_x with respect to the first two variables $y \in \mathbb{R}^n, x \in \mathcal{D}_f$.

The function $d : \mathcal{D}_f \times \mathcal{I}_f \to \mathbb{R}^n$ is continuously differentiable.

DAEs in the form (1.40) arise, for instance, in circuit simulation by means of the modified nodal analysis. Involving the derivative (1.40) by means of an extra function into the DAE brings benefits in view of solvability.

Definition 1.26. A solution x_* of equation (1.40) is a continuous function defined on an interval $\mathcal{I}_* \subseteq \mathcal{I}_f$, with values $x_*(t) \in \mathcal{D}_f, t \in \mathcal{I}_s$, such that the function $u_*(.) := d(x_*(.), .)$ is continuously differentiable, and x_* satisfies the DAE (1.40) pointwise on $\mathcal{I}_*$.

In general, if a DAE is solvable then it means their exist a solution for the given DAE. In contrast, mostly in the literature on DAEs, solvability of a DAE means the existence of a continuously differentiable function satisfying the DAE pointwise. As for linear DAEs, we can expect lower smoothness solvability results for for nonlinear DAEs (1.40).

1.6 Linear and Non-Linear Filtering of DAEs

1.6.1 Kalman Filter for Linear System

The linear discrete-time Kalman filter provides an algorithm that is optimal for filtering noise. This subsection presents the Kalman filter algorithm for a linear discrete-time system. There exists a Kalman filter which can be applied on linear continuous-time systems and hybrid systems. A system is called hybrid if the process is continous and the measurements are available at discrete times, these systems are also known as continuous-discrete. It should be noted that a system with a continuous process may be transformed into a discrete system. Given:

$$\dot{x} = Ax, \quad x(0) = x_{initial} \tag{1.43}$$

then a discrete-time system is defined as

$$x_k = exp(h_k A)x_{k-1}, \quad x_0 = x_{initial}, \tag{1.44}$$

where the time between x_k and $h = t_k - t_{k-1}$ Given a linear discrete-time system:

$$x_k = A_k x_{k-1} + w_k, \tag{1.45a}$$

$$y_k = H_k x_k + v_k. \tag{1.45b}$$

where $x_k \in \mathbb{R}$ is the state at time k, $A_k \in \mathbb{R}^{n\times n}$ is the process matrix at time k, $y_k \in \mathbb{R}^m$ is the measurement at time k, $H_k \in \mathbb{R}^{m\times n}$ is the measurement matrix at time k. The noises,w_k and v_k, are white, zero-mean, uncorrelated and have known covariance matrices Q_k and R_k

$$w_k \sim N(0, Q_k) \tag{1.46a}$$

$$v_k \sim N(0, R_k) \tag{1.46b}$$

$$\mathbb{E}[w_k w_j^T] = \begin{cases} Q_k & k = j \\ 0 & k \neq j \end{cases} \tag{1.46c}$$

$$\mathbb{E}[v_k v_j^T] = \begin{cases} R_k & k = j \\ 0 & k \neq j \end{cases} \tag{1.46d}$$

$$\mathbb{E}[w_k v_j^T] = 0 \tag{1.46e}$$

The discrete-time Kalman filter can be applied on the above system (1.49) to filter out the noise and provide an optimal estimate of the true state. After initialization, the algorithm progresses one time step at a time, using only the updated estimates from the previous step, along with the given information about the system. Each time step has two phases, the predict phase and the update phase. The predicted (pre-updated) estimates are indicated by a minus sign in the superscript and the updated estimates are indicated by a plus sign in the superscript. Below is the Kalman filter algorithm:

Initialize:

$$\hat{x}_0^+ = \mathbb{E}(x_0)$$

$$P_0^+ = \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$$

1. Predict phase:

   (a) Predict state estimate

$$\hat{x}_k^- = A_k \hat{x}_{k-1}^+$$

   (b) Predict estimate covariance

$$P_k^- = A_k P_{k-1}^+ + Q_k$$

2. Update phase:

   (a) Obtain the optimal Kalman gain,

$$K_k = P_k^- H_{k-1}^T (H_{k-1} P_k^- H_{k-1}^T + R_k)^{-1}$$

   (b) Update state estimate

$$\hat{x}_{k-1}^+ = \hat{x}_k^- + K_k(y_k - H_k \hat{x}_k^-)$$

   (c) Update estimate covariance

$$P_k^+ = (I - K_k H_k) P_k^-$$

The Kalman filter is applied recursively to produce state estimates $x_1^+, x_1^+, ...$ at times $t = 1, 2, ...$ The outputs of the Kalman filter can be tuned by changing the covariance matrices Q and R. If the measurement is to be trusted more than the model then the norm $\|R\|$ should be lower than $\|Q\|$ and vice versa, the greater the trust in the measurement over the model the greater the ratio $\|Q\|/\|R\|$. The formula forP_k^+ may be replaced by the

so-called Joseph stabilized version

$$P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T$$

This is the original equation for covariance derived in [38]. The Joseph stabilized version is more robust and stable [38] and guarantees that despite numerical inaccuracies P_k^+ will be symmetric and positive-definite as long as P_k^- and R_k are. For more information see [38].

1.6.2 Kalman Filter for Nonlinear System

The algorithm presented in the previous section is only applicable on linear systems. However, many systems are nonlinear. This section discusses generalizations of the standard Kalman filter: the Extended Kalman Filter (EKF). Other such generalizations exist such as Unscented Kalman Filter (UKF), Particle Filter/Sequential Monte Carlo, Ensemble Kalman Filter and Iterative Extended Kalman Filter [38].

Extended Kalman Filter

The Extended Kalman filter is a generalization of the Kalman Filter to nonlinear systems. The EKF requires the Jacobian matrices of the nonlinear process and measurement functions with respect to the state and noise to be available. The idea of the EKF is to linearize at each time step and treat the system as a time-varying linear system. Suppose we have a

non-linear discrete time system:

$$x_k = f(t_k, x_{k-1}, w_k) \tag{1.47a}$$

$$y_k = h(t_k, x_k, v_k) \tag{1.47b}$$

$$w_k \sim N(0, Q_k) \tag{1.47c}$$

$$v_k \sim N(0, R_k) \tag{1.47d}$$

$$\tag{1.47e}$$

The noises, w_k and v_k, are allowed to act nonlinearly on the process and measurement. Below is the Extended Kalman filter algorithm:

Initialize:

$$\hat{x}_0^+ = \mathbb{E}(x_0)$$

$$P_0^+ = \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T]$$

1. Predict phase:

   (a) Predict state estimate

   $$\hat{x}_k^- = f(t_k, \hat{x_{k-1}}, 0).$$

   (b) Compute the Jacobians to linearize the process.

   $$A_k = \left.\frac{\partial f}{\partial x}\right|_{(t_k, x_{k-1}^+, 0)} \qquad \text{and} \qquad L_k = \left.\frac{\partial f}{\partial w}\right|_{(t_k, x_{k-1}^+, 0)}$$

   (c) Predict estimate covariance,

   $$P_k^- = A_k P_{k-1}^+ A_k^T + L_k Q_k L_k^T$$

2. Update phase

   (a) Compute the Jacobians to linearize the measurement,

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{(t_k, x_k^-, 0)} \qquad \text{and} \qquad M_k = \left.\frac{\partial h}{\partial w}\right|_{(t_k, x_k^-, 0)}$$

   (b) Obtain the optimal Kalman gain,

$$K_k = P_k^- H_{k-1}^T (H_k P_k^- H_k^T + M_k R_k M_k^T)^{-1}$$

   (c) Update state estimate

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - h(t_k, \hat{x}_k^-, 0))$$

   (d) Update estimate covariance,

$$P_k^+ = (I - K_k H_k) P_k^-$$

The EKF uses a first-order approximation to obtain a linear estimate to the system. This is a clear limitation and as it may fail to provide a good estimate when the system is highly nonlinear. Higher order approximations exist and can be used to obtain better estimates. This is the idea behind the Iterated EKF and the Second-Order EKF, a discussion of these is beyond the scope of this thesis but for more information see [38].

1.7 Applications of DAEs

1. Multibody systems is one area, in which methods for solving DAEs are of special interest. A multibody system is a mechanical system, that consists of one or more bodies. The bodies can either be rigid or elastic. They can have a mass and/or a torque. Somehow they are connected with each other. These mass-less connections can either be force-elements, like springs, dampers or friction, or non-elastic joints, i.e. translational or rotational joints. Examples of such systems are The multibody truck, the pendulum [39]

2. DAEs in Energy System Models consists of technical installations in which fluids are employed to transport energy between the mechanical devices and thereby determine the work of the installation. Common examples of energy systems are power plants, combustion engines, refrigerators, air conditioning, district heating and many industrial production processes. The devices used in energy systems are for instance heat exchangers, pumps, turbines, valves, compressors and fans. Many models of these systems are used for optimization and steady state operation, but more often dynamic simulation are of interest [39].

3. Modeling on DAEs contributes to systems of rigid bodies, network, electrical circuits, chemical reactionns and discretization of PDEs [40].

4. One of the ways that differential algebraic equations (DAEs) naturally arise is with tracking problems. A robotics example is explained in details in [41].

5. Applications occurring in path planning tasks for mobile robots and vehicle dynamics which involve differential algebraic equations (DAEs). The modeling aspects and issues arising from the DAE formulation such as hidden constraints, determination of

algebraic states, and consistency is best explained with exmple in [41]

6. Systems of Differential Algebraic Equations in Computational Electromagnetics. Starting from space-discretisation of Maxwell's equations, various classical formulations are proposed for the simulation of electromagnetic fields. They differ in the phenomena considered as well as in the variables chosen for discretisation [41].

7. Gas Network Benchmark Models. The simulation of gas transportation networks becomes increasingly more important as its use involve more complex applications. Classically, the purpose of the gas network was the transportation of predominantly natural gas from a supplier to the consumer for long-term scheduled volumes. With the rise of renewable energy sources, gas-fired power plants are often chosen to compensate for the fluctuating nature of the renewables, due to their on-demand power generation capability. Such a short-term plannable supply and demand setting requires sophisticated simulations of the gas network prior to the dispatch to ensure the supply of all customers for a range of possible scenarios and to prevent damages to the gas network [41].

8. Topological Index Analysis Applied to Coupled Flow Networks. The multi-physical model consists of (simple connected) networks of different or the same physical type (liquid flow, electric, gas flow, heat flow) which are connected via interfaces or coupling conditions. Since the individual networks result in differential algebraic equations (DAEs), the combination of them gives rise to a system of DAEs. While for the individual networks existence and uniqueness results, including the formulation of index reduced systems, is available through the techniques of modified nodal analysis or topological based index analysis, topological results for coupled system are not available so far [41].

9. Nonsmooth DAEs with Applications in Modeling Phase Changes. A variety of engineering problems involve dynamic simulation and optimization, but exhibit a mixture of continuous and discrete behavior. Such hybrid continuous/discrete behavior can cause failure in traditional methods; theoretical and numerical treatments designed for smooth models may break down. Recently it has been observed that, for a number of operational problems, such hybrid continuous/discrete behavior can be accurately modeled using a nonsmooth differential-algebraic equations (DAEs) framework, now possessing a foundational well-posedness theory and a computationally relevant sensitivity theory. Numerical implementations that scale efficiently for large-scale problems are possible for nonsmooth DAEs. Moreover, this modeling approach avoids undesirable properties typical in other frameworks (e.g., hybrid automata); in this modeling paradigm, extraneous (unphysical) variables are often avoided, unphysical behaviors (e.g., Zeno phenomena) from modeling abstractions are not prevalent, and a priori knowledge of the evolution of the physical system (e.g., phase changes experienced in a flash process execution) is not needed [41].

1.8 Thesis Contribution and Organization

In this thesis, both theory and numerical methods for the solving DAEs have been studied. In particular, we provide a systematic and detailed analysis of initial and boundary value problems for differential-algebraic equations. We also discuss numerical methods and simulation software for finding the solution of DAE systems. This includes linear and nonlinear systems, over and underdetermined systems as well as control problems.

The goals of the thesis are:

- **Goal 1:** Solvability concepts of DAE have been studied and DAE index has been described. Literature review of Linear and nonlinear DAE has been mentioned. Filtering techniques of DAEs have been summarized.
- **Goal 2:** Numerical methods for solving DAEs have been described. Simulation software for numerical solution of DAE have been studied. Studied and described the ways to use software packages to solve DAE with IVP, BVP and through order reduction technique.
- **Goal 3:** Implementation of MapleSoft and Mathematica software packages for solution of complex DAE system via various integration algorithms.
- **Goal 4:** Nonlinear filtering technique of DAE, known as EKF has been studied through an example.
- **Goal 5:** Recommendation of the simulation packages based on the index of DAE has been concluded. Preference of the simulation package based on the required solution of the DAE has been described.

The thesis is organized as follows:

- **Chapter 1** discuss the basic introduction to ordinary differential equations and differential algebraic equations. Index and solvability concepts of Linear and Non-linear DAEs have been discussed. Some of the application of DAEs are also mentioned. Provides with insights into filtering of DAEs. It also states the objective and organization of the thesis.
- **Chapter 2** provides the description of DAE solutions. Different methods have been studied in the thesis for numerical solution of DAEs. Inspection of simulation software used for solving complex DAEs are also discussed.

- **Chapter 3** provides the comparision of DAE system solvers using simulation software such as MapleSoft, Mathematical and Matlab.
- **Chapter 4** discusses the filtering methods used in DAE system. Kalman filtering for linear and non linear system is discussed with an example to illustrate the need of filtering the semi-explicit index-1 DAE system.
- **Chapter 5** provides the conclusion of the thesis

Chapter 2

Solutions of Differential Algebraic Equations

2.1 Numerical Methods for Solving DAE

2.1.1 Runge-Kutta Methods for DAE [44]

The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method" or simply as "the Runge–Kutta method". Consider a following initial value problem, where y is an unknown function (scalar or vector) of time t, which we would like to approximate, the rate at which y changes is a function of t and of y itself, be specified as:

$$\dot{y} = f(t, y), \quad y \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n$$
$$y(t_0) = y_0$$

The function f and the initial conditions t_0, y_0 are assumed to be given. Consider step-size $h > 0$ and RK4 methods is defined for $n = 0, 1, 2, 3, ...,$

$$y_{n+1} = y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right),$$
$$t_{n+1} = t_n + h$$

where

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned}$$

In the above equation, y_{n+1} is the RK4 approximation of $y(t_{n+1})$, and the next value (y_{n+1}) is determined by the present value (y_n) plus the weighted average of four increments, where each increment is the product of the size of the interval h, and an estimated slope (k_1, k_2, k_3, k_4) specified by function f on the right-hand side of the differential equation are defined as:

1. k_1 is the slope at the beginning of the interval, using y (Euler's method);
2. k_2 is the slope at the midpoint of the interval, using y and k_1;
3. $k_3$is again the slope at the midpoint, but now using y and k_2;
4. k_4 is the slope at the end of the interval, using y and k_3.

In averaging the four slopes, greater weight is given to the slopes at the midpoint. If f is independent of y, so that the differential equation is equivalent to a simple integral, then RK4 is known Simpson's rule [42].

The RK4 method is a fourth-order method, where the local truncation error is of the order of $O(h^5)$, while the total accumulated error is of the order of $O(h^4)$.

In many practical applications the function f is independent of t (so called autonomous system, or time-invariant system, especially in physics), and their increments are not computed at all and not passed to function f, with only the final formula for t_{n+1} used [42].

Collocation methods [43]

Collocation method is a method for the numerical solution of ODE, PDE and integral equations. The idea is to choose a finite-dimensional space of candidate solutions (usually polynomials up to a certain degree) and a number of points in the domain (called collocation points), and to select that solution which satisfies the given equation at these collocation points [43].

Consider the following ODE which is to be solved over the interval $[t_0, t_0 + c_k h]$. The corresponding (polynomial) collocation method approximates the solution y by the polynomial p of degree n which satisfies the initial condition $p(t_0) = y_0$, and the differential equation $\dot{p}(t_k) = f(t_k, p(t_k))$ at all collocation points $t_k = t_0 + c_k h$ for $k = 1, \ldots, n$. This gives $n + 1$ conditions, which matches the $n + 1$ parameters needed to specify a polynomial of degree n.

$$\dot{y}(t) = f(t, y(t)), \quad y \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n$$
$$y(t_0) = y_0$$

Choose c_k from $0 \leq c_1 < c_2 < \ldots < c_n \leq 1$.

All these collocation methods are in fact implicit Runge–Kutta methods. The coefficients c_k in the Butcher tableau [24] of a Runge–Kutta method are the collocation points. However,

not all implicit Runge–Kutta methods are collocation methods [42].

Example: The trapezoidal rule

Consider the following example where the two collocation points are $c_1 = 0$ and $c_2 = 1$(so $n = 2$). The collocation conditions are

$$\begin{aligned} p(t_0) &= y_0 \\ p'(t_0) &= f(t_0, p(t_0)) \\ p'(t_0 + h) &= f(t_0 + h, p(t_0 + h)) \end{aligned}$$

There are three conditions, so p should be a polynomial of degree 2. Consider p is written in the form

$$p(t) = \alpha(t - t_0)^2 + \beta(t - t_0) + \gamma$$

to simplify the computations. Then the collocation conditions can be solved to give the coefficients

$$\begin{aligned} \alpha &= \frac{1}{2h}\Big(f(t_0 + h, p(t_0 + h)) - f(t_0, p(t_0))\Big) \\ \beta &= f(t_0, p(t_0)) \\ \gamma &= y_0 \end{aligned}$$

The collocation method is now given (implicitly) by

$$y_1 = p(t_0 + h) = y_0 + \frac{1}{2}h\Big(f(t_0 + h, y_1) + f(t_0, y_0)\Big)$$

where $y_1 = p(t_0 + h)$ is the approximate solution at $t = t_0 + h$.

This method is known as the "trapezoidal rule" for differential equations. Indeed, this

method can also be derived by rewriting the differential equation as

$$y(t) = y(t_0) + \int_{t_0}^{t} f(\tau, y(\tau))\,\mathrm{d}\tau$$

and approximating the integral on the right-hand side by the trapezoidal rule for integrals.

Explicit Runge–Kutta methods [42]

The family of explicit Runge–Kutta methods is a generalization of the RK4 method. It is given by

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i,$$

where

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f(t_n + c_2 h, y_n + h(a_{21}k_1)),\\
k_3 &= f(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2)),\\
&\vdots\\
k_s &= f(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s-1}k_{s-1})).
\end{aligned}$$

To specify a particular method, one needs to provide the integer s (the number of stages), and the coefficients a_{ij} (for $1 \leq j < i \leq s$), b_i (for $i = 1, 2, ..., s$) and c_i (for $i = 2, 3, ..., s$). The matrix $[a_{ij}]$ is called the Runge–Kutta matrix, while the b_i and c_i are known as the weights and the nodes. These data are usually arranged in a mnemonic device, known as a Butcher tableau (after John C. Butcher):

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\ \hline & b_1 & b_2 & \cdots & b_{s-1} \quad b_s \end{array}$$

A Taylor series expansion shows that the Runge–Kutta method is consistent if and only if

$$\sum_{i=1}^{s} b_i = 1$$

There are also accompanying requirements if one requires the method to have a certain order p, meaning that the local truncation error is $O(h^{p+1})$. These can be derived from the definition of the truncation error itself. For example, a two-stage method has order 2 if $b_1 + b_2 = 1$, $b_2 c_2 = 1/2$, and $b_2 a_{21} = 1/2$. Note that a popular condition for determining coefficients is

$$\sum_{j=1}^{i-1} a_{ij} = c_i \text{ for } i = 2, \ldots, s$$

This condition alone, however, is neither sufficient, nor necessary for consistency [42].

In general, if an explicit s-stage Runge–Kutta method has order p, then it can be proven that the number of stages must satisfy $s \geq p$, and if $p \geq 5$, then $s \geq p + 1$. However, it is not known whether these bounds are sharp in all cases; for example, all known methods of order 8 have at least 11 stages, though it is possible that there are methods with fewer stages. (The bound above suggests that there could be a method with 9 stages; but it could also be that the bound is simply not sharp.) Indeed, it is an open problem what the precise minimum number of stages s is for an explicit Runge–Kutta method to have order p

in those cases where no methods have yet been discovered that satisfy the bounds above with equality. Some values which are known are:

p	1	2	3	4	5	6	7	8
$\min s$	1	2	3	4	6	7	9	11

The provable bounds above then imply that we can not find methods of orders $p = 1, 2, \ldots, 6$ that require fewer stages than the methods we already know for these orders. However, it is conceivable that we might find a method of order $p = 7$ that has only 8 stages, whereas the only ones known today have at least 9 stages as shown in the above.

Implicit Runge–Kutta methods [42]

Explicit Runge–Kutta methods are generally unsuitable for the solution of stiff equations because their region of absolute stability is small, in particular, it is bounded. This issue is especially important in the solution of partial differential equations.

The instability of explicit Runge–Kutta methods motivates the development of implicit methods. An implicit Runge–Kutta method has the form

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

where

$$k_i = f\left(t_n + c_i h,\ y_n + h \sum_{j=1}^{s} a_{ij} k_j\right), \quad i = 1, \ldots, s$$

The difference with an explicit method is that in an explicit method, the sum over j only goes up to $i - 1$. This also shows up in the Butcher tableau the coefficient matrix a_{ij} of an explicit method is lower triangular. In an implicit method, the sum over j goes up to s and

the coefficient matrix is not triangular, yielding a Butcher tableau of the form:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \\ & b_1^* & b_2^* & \dots & b_s^* \end{array} = \begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b^T} \end{array}$$

The consequence of this difference is that at every step, a system of algebraic equations has to be solved. This increases the computational cost considerably. If a method with s stages is used to solve a differential equation with m components, then the system of algebraic equations has ms components. This can be contrasted with implicit linear multistep methods (the other big family of methods for ODEs): an implicit s-step linear multistep method needs to solve a system of algebraic equations with only m components, so the size of the system does not increase as the number of steps increases [42].

The Segregated Runge–Kutta Method [44]

The Segregated Runge–Kutta (SRK) method[1] is a family of IMplicit–EXplicit (IMEX) Runge–Kutta methods that were developed to approximate the solution of differential algebraic equations (DAE) of index 2. Consider an index 2 DAE defined as follows:

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

where $y(t) \in \mathbb{R}^n, z(t) \in \mathbb{R}^m, f : \mathbb{R}^{n+m} \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^m$.

In the above equations, y is known as the differential variable, while z is known as the algebraic variable. The time derivative of the differential variable, $\dot{y}$ depends on itself y on the algebraic variable z and on the time t. The second equation can be seen as a constraint on differential variable y [44].

Let us take the time derivative of the second equation. Assuming that the function g is linear and does not depend on time, and that the function f is linear with respect to z, we have that

$$0 = \dot{g}(y) = g(\dot{y}) = g(f(y, z)) = g(f(y) + f(z)) = g(f(y)) + g(f(z)))$$

A Runge–Kutta time integration scheme is defined as a multistage integration in which each stage is computed as a combination of the unknowns evaluated in other stages. Depending on the definition of the parameters, this combination can lead to an implicit scheme or an explicit scheme. Implicit and explicit schemes can be combined, leading to IMEX schemes. Suppose that the function f can be split into two operators $\mathcal{F}$ and $\mathcal{G}$ such that

$$\dot{y}(t) = \mathcal{F}(y(t)) + \mathcal{G}(y(t), z(t)),$$

where $\mathcal{F}(y(t))$ and $\mathcal{G}(y(t), z(t))$ are the terms to be treated implicitly and explicitly, respectively.

The SRK method is based on the use of IMEX Runge–Kutta schemes and can be defined by the following scheme:

Given a time step size $h > 0$, at a time $t_{n+1} = t_n + h$, for each Runge-Kutta stage i, with

$0 \leq i \leq s$, solve:

$$y_i = y_n + h\sum_{j=1}^{i} a_{ij}\mathcal{F}(y_j) + h\sum_{j=1}^{i-1} \hat{a}_{ij}\mathcal{G}(y_j, z_j),$$
$$g(f(z_i)) = -g(f(y_i))$$

Update the variables at t_{n+1} solving:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i\mathcal{F}(y_i) + h\sum_{i=1}^{s} \hat{b}_i\mathcal{G}(y_i, z_i),$$
$$g(f(z_{n+1})) = -g(f(y_{n+1}))$$

2.1.2 BDF-methods for DAE [47]

The backward differentiation formula (BDF) is a family of implicit methods for the numerical integration of ordinary differential equations. They are linear multistep methods that, for a given function and time, approximate the derivative of that function using information from already computed time points, thereby increasing the accuracy of the approximation. These methods are especially used for the solution of stiff differential equations. A BDF method is used to solve the initial value problem.

The general formula for a BDF for an ODE of the form

$$\dot{y} = f(t, y), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n$$

can be written as, where f is evaluated for the unknown y_{n+s} and h denotes the step size and $t_n = t_0 + nh$:

$$\sum_{k=0}^{s} \alpha_k y_{n+k} = h\beta f(t_{n+s}, y_{n+s}),$$

The coefficients α_k and β are chosen so that the method achieves order s, which is the maximum possible. BDF methods are implicit and possibly require the solution of nonlinear equations at each step.

The constant step-size BDF method applied to a nonlinear DAE of the form $F(t, x, \dot{x}) = 0$ where $(t_0 \leq t \leq t_f)$ and β_0, α_j, $(j = 0, 1, ..., k,)$ are the coefficients in the BDF method, is given by

$$F\left(t_n, x_n, \frac{1}{\beta_0 h}\sum_{j=0}^{k}\alpha_j x_{n-j}\right) = 0 \tag{2.13}$$

It has been shown that the k-step BDF method of fixed step-size h is convergent of order $O(h^k)$ if all initial values are correct to $O(h^k)$ and if the Newton iteration on each step is solved to accuracy $O(h^{k+1})$. [45]

2.2 Software for Numerical Solution of DAE

Due to the great importance of DAE in applications (see section 1.6), many of the numerical techniques for differential-algebraic equations have been implemented in software packages. In this section, we will describe about the available software packages to solve DAEs.
Most of the available software packages for differential-algebraic equations are FORTRAN or C subroutine libraries and can be found via the internet from software repositories or homepages. We have listed all relevant web addresses in below table (2.2). Following is the description for Table 2.2:

1. Computational packages such as Maple(Table 2.2, 8) and Mathematica(Table 2.2,9) contain calling routines/functions for the solution of differential-algebraic equations. MAPLE uses the following routines/methods for solving DAE:

   (a) $rk45\_dae$

(b) *mebdfi*

(c) *ck*45_*dae*

(d) Rosenbrock method for stiff problems.

Mathematica uses the following integration algorithms to solve DAE :

(a) Pantellides algorithm

(b) Structural Matrix

(c) Backward Differential Formula for stiff problems

(d) IDA(Implicit Differential Algebraic Solver)

(e) Projection Method

(f) Collocation Method

(g) Block Lower Triangular(BLT) Form

2. MEXAX(Table 2.2, 11) [46](short for MEXAnical systems eXtrapolation integrator) is a Fortran code for time integration of constrained mechanical systems. MEXAX is suited for direct integration of the equations of motion in descriptor form. It is based on extrapolation of a time stepping method that is explicit in the differential equations and linearly implicit in the nonlinear constraints. It only requires the solution of well-structured systems of linear equations which can be solved with a computational work growing linearly with the number of bodies, in the case of multibody systems with few closed kinematic loops. Position and velocity constraints are enforced throughout the integration interval, whereas acceleration constraints need not be formulated. MEXAX has options for time-continuous solution representation (useful for graphics) and for the location of events such as impacts.

3. Following software packages are used for simulation of multibody systems:

   (a) MBSSIM(Table 2.2,17) of von Schwerin and Winckler [47]

   (b) ODASSL(Table 2.2,15) of Führer [48].

   (c) GEOMS (Table 2.2,14) of Steinbrecher [49].

   (d) MBSpack (Table 2.2,18) of Simeon [50]. Steinbrecher [49].

4. Software package COLDAE [51] is the collocation software for DAE with BVP. COLDAE can solve boundary value problems for nonlinear systems of semi-explicit differential-algebraic equations (DAEs) of index at most 2. Fully implicit index-1 boundary value DAE problems can be handled as well.

5. Software Packages maintained by E.Hairer(Table 2.2,10) are:

   (a) RADAU5: solves DAE by 2-stage Radau IIA method.

   (b) SDIRK4: solves DAE by a diagonally-implicit Runge–Kutta method of order 4.

   (c) RODAS: solves DAE by a Rosenbrock method of order 4.

6. Software Packages maintained by SUNDIAls(Table 2.2,10) are:

   (a) DASSL [52]: DASSL uses the backward differentiation formulas of orders 1 through 5 to solve a DAE system.

   (b) DASPK [52]: *daspk* provides an interface to the FORTRAN DAE solver and uses BDF method to solve DAE.

7. Software Packages maintained by Iavernaro and Mazzia [53]:

   (a) GAMD : The code GAM numerically solves solves first order ordinary differential equations, either stiff or nonstiff in the form $y' = f(x, y)$, with a

given initial condition.The code GAMD is a generalization of GAM for the solution of Differential Algebraic Equations of index less than or equal to 3 in the form $My' = f(x, y)$, with a given initial condition. The methods used in both codes are in the class of Boundary Value Methods (BVMs), namely the Generalized Adams Methods (GAMs) of order 3,5,7,9 with step size control.

8. Special codes for multibody systems are available in commercial packages like Simpack (19), ADAMS (20), or DYMOLA (21).

9. Following algorithms are used by J.R.Cash(Table 2.2, 12):

   (a) MEBDFV: This solver is based on the modified extended backward differentiation formulae of Cash [54]. It is especially suited for the solution of differential-algebraic equations with time- and state-dependent mass matrix.

   (b) MEBDFDAE: The code MEBDFDAE uses the Modified Extended Backward Differentiation Formulas of Cash, that increase the absolute stability regions of the classical BDFs.These methods are A-stable up to the order 4 . This algorithm implements three-stage linear methods with the same Jacobian to be used in the Newton iteration for all the stages.The current versions of this solver for the solutions of ODEs are MEBDF and MEBDFSO,which is designed to solve stiff IVP for very large sparse systems of ODEs.

10. GNU Octave(Table 2.2, 7) [55] is a high-level language, primarily intended for numerical computations. Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via

user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

①	www.netlib.org	NETLIB
②	pitagora.dm.uniba.it/~testset/	IVP Testset
③	www.nag.co.uk/	NAG
④	http://absoft.com/	IMSL
⑤	www.mathworks.com/	MATLAB
⑥	scilabsoft.inria.fr/	SCILAB
⑦	www.octave.org/	OCTAVE
⑧	www.maplesoft.com/	MAPLE
⑨	www.wolfram.com/	MATHEMATICA
⑩	www.unige.ch/math/folks/hairer/	E. Hairer
⑪	www.zib.de/Software/	Zuse Inst. Berlin
⑫	www.ma.ic.ac.uk/~jcash/	J. R. Cash
⑬	www.cwi.nl/cwi/projects/PSIDE	PSIDE
⑭	www.math.tu-berlin.de/numerik/mt/NumMat/	TU Berlin
⑮	synmath.synoptio.de/	SynOptio
⑯	www.llnl.gov/CASC/sundials/	SUNDIALS
⑰	www1.iwr.uni-heidelberg.de/	IWR Heidelberg
⑱	www-m2.ma.tum.de/~simeon/numsoft.html	B. Simeon
⑲	www.simpack.de/	SIMPACK
⑳	www.mscsoftware.com/	ADAMS
㉑	www.dynasim.com/	DYMOLA

Table 2.1: Web paths for DAE Software

Remark 2.1. Many of the above mentioned algorithms are in the state of flux. The iterative methods arise at every iteration step as well as with the use of differentiation packages.The use of these iterative methods for the solution of the linear systems is to

determine the necessary Jacobians. In the standard design of modern software packages [56], the implementation of most of the above mentioned codes is based on existing basic linear algebra subroutines BLAS, see [57], which are usually provided with the computer architecture. BLAS uses high quality linear algebra packages such as LAPACK (1) or SCALAPACK (1), see [58] and [59], respectively, for the solution of problems like the singular value decomposition, linear system solution or the solution of least squares problems. A possible solver for nonlinear problems is NLSCON (11), see [60]. Furthermore, in the solution of the linear and nonlinear systems that arise in the time-stepping procedures, most of the methods need appropriate scaling routines and the design of such routines is essential in getting the codes to work.

Remark 2.2. Almost all of the above mentioned codes contain order and step size control mechanisms. For BDF codes, order and step-size control is described in detail in [6] and for Runge–Kutta methods are discussed in [61], [15].

2.3 Software Methods

2.3.1 MapleSoft: Differential-Algebraic Equations in Maple

MapleSoft uses the function *dsolve* to solve DAE's. dsolve will be discussed further in this section. Maple's differential equation solvers employ advanced techniques to solve [62].

- Ordinary differential equations (ODEs): *dsolve* solves linear and nonlinear ODEs, initial value problems (IVPs), and boundary value problems (BVPs). The ODE Analyzer Assistant provides an interactive way to solve an ODE and plot the solution.
- Partial differential equations (PDEs): The function *pdsolve* finds exact solutions to PDEs and uses Maple's PDE tools to perform structural analysis and order reduction

for PDE systems. pdsolve handles systems of time-dependent PDEs in one spatial dimension with boundary conditions. *pdsolve* includes 11 standard methods for numerically solving PDEs.

- Differential-Algebraic Equations (DAEs): *dsolve* uses order reduction and enable Maple to solve high-index DAE problems:
  Maple has four solver methods for handling DAEs:

  1. Modified Runge-Kutta Fehlberg method:$rk45\_dae$,
  2. Cash-Karp Pair Runge-Kutta method:$ck45$,
  3. Rosenbrock method,
  4. Modified Extended Backward-Differentiation Implicit method:$mebdfi$.

dsolve [63]:

- Calling Sequence:
  $dsolve(DAEsys, numeric, method, vars, options)$

- Parameters
  DAEsys - differential algebraic equations
  numeric - literal name; instruct dsolve to find a numerical solution
  method - type of numerical method/algorithm to use. Default method is $rk45\_dae$
  vars - (optional) any indeterminate function of one variable, or a set or list of them, representing the unknowns of the ODE problem
  options - (optional) equations of the form keyword = value
  procopts - options used to specify the ODE system using a procedure (procedure, initial, start, number, and procvars).

- Description:
  - The dsolve command with the options *numeric* and $method = rkf45$ finds a numerical solution using a Fehlberg fourth-fifth order Runge-Kutta method with degree four. This is the default method of the type=numeric solution for initial value problems when the stiff argument is not used. The other non-stiff method is a Runge-Kutta method with the Cash-Karp coefficients, ck45.
  - Modes of Operation:
    * The $rkf45$ method has two distinct modes of operation (for procedure-type outputs).
    * With the range option : When used with the range option, the method computes the solution for the IVP over the specified range, storing the solution information internally, and uses that information to rapidly interpolate the desired solution value for any call to the returned procedure. Though possible, it is not recommended that the returned procedure be called for points outside the specified range.

      The storage of the interpolant solution used by this method can be disabled by using the $interpolation = false$ option. This is recommended for high accuracy solutions where storage of the interpolant (in addition to the discrete solution) requires too much memory.
    * Without the range option: When used without the range option, the IVP solution values are not stored, but rather computed when requested. Since, not all solution values are stored, computation must restart at the initial values whenever a point is requested between the initial point and the most recently computed point (to avoid reversal of the integration direction), so it

is advisable to collect solution values moving away from the initial value.

– Options

The following options are available for the $rkf45$ method.

* 'output'=keyword or array
* 'known'=name or list of names
* 'abserr'=numeric
* 'relerr'=numeric
* 'initstep'=numeric
* 'interr'=boolean
* 'maxfun'=integer
* 'number'=integer
* 'procedure'=procedure
* 'start'=numeric
* 'initial'=array
* 'procvars'=list
* 'startinit'=boolean
* 'implicit'=boolean
* 'optimize'=boolean
* 'compile'=boolean or auto
* 'range'=numeric..numeric
* 'events'=list
* $'event\_pre'$=keyword
* $'event\_maxiter'$=integer

  - $'event\_iterate'$=keyword
  - $'event\_initial'$=boolean
  - $'complex'$=boolean
- output
  - Specifies the desired output from dsolve. The keywords *procedurelist*, *listprocedure*, or operator provide procedure-type output. The keyword *piecewise* provides output in the form of piecewise functions over a specified range of independent variable values.
- known
  - Specifies user-defined known functions [63].
- abserr, relerr, and initstep
  - Specifies the desired accuracy of the solution and the starting step size for the method [63].The default values for $rkf45$ are $abserr = 1e-7$ and $relerr = 1e-6$. The value for *initstep*, if not specified, is determined by taking the local behavior of the ODE system into account.
- interr
  - By default, this is set to true. It controls the interpolant solution error to integrate into error control. When set to false, areas where the solution is varying rapidly (e.g. a discontinuity in a derivative due to a piecewise) may have a much larger solution error than dictated by the specified error tolerances. When set to true, the step size is reduced to minimize error in regions where the solution is varying rapidly, but for problems where there is a jump discontinuity in the variables, the integration may fail with an error indicating that a singularity is present.

- maxfun
  - Specifies a maximum number of evaluations of the right-hand side of the first order ODE system. This option is disabled by specifying $maxfun = 0$. The default value for $rkf45$ is 30000.
- number, procedure, start, initial, and procvars
  - These options are used to specify the IVP using procedures.
- startinit, implicit, and optimize
  - These options control the method and behavior of the computation [63].
- compile
  - This option specifies that the internally generated procedures compiled for efficiency are used to evaluate the numeric solution. Note that this option will only work if $Digits$ is set within the hardware precision range and the input function contains only $evalhf$ capable functions (e.g. only elementary mathematical functions like exp, sin, and ln). By default, this value is set to false. If set to true and compiling the numeric solution is not possible, an error will be shown. If set to auto and compiling is not possible, the uncompiled procedures will be used directly.
- range
  - Determines the range of values of the independent variable for which solution values are required. Use of this option significantly changes the behavior of the method for the $procedure - style$ output types.
- complex
  - Accepts a boolean value indicating if the problem is (or will become) complex valued. By default, this is detected based on the input system and initial data,

but in cases where the input system is procedure defined, or the system is initially real, it may be necessary to specify $complex = true$ to obtain the solution. It is assumed that for an initially real system that becomes complex, the point at which this transition occurs is considered to be a singularity, so if $complex = true$ is not specified, the integration will halt at that point.

In most cases *dsolve* is able to detect if a given problem is a DAE system, as opposed to an ODE system, by identifying whether a pure algebraic equation in the dependent variables is present. If the input is a DAE system containing no purely algebraic equations, the method must be included to specify that the system is a DAE system.

dsolve has four methods for finding numeric solutions for DAEs. The default DAE IVP method is a modified Runge-Kutta Fehlberg method ($rkf45\_dae$). The other methods, $ck45\_dae$ (an extension of the $ck45$ method, a Runge-Kutta method with the Cash-Karp coefficients), *rosenbrock_dae* (an extension of the rosenbrock method, an Implicit Rosenbrock third-fourth order Runge-Kutta method with degree three), and mebdfi (Modified Extended Backward-Differentiation Formula Implicit method) can be specified using the *method* option. If we specify that the problem is stiff with the option $stiff = true$ without selecting a method, then the method *rosenbrock_dae* will be used.

Maple's core numeric ODE and DAE IVP solvers ($rkf45$, $ck45$, rosenbrock, and the DAE versions of these) can handle user-defined events.

DAE plotting in Maple (using function *odeplot*) [64]

- Calling Sequence
  - odeplot(dsn, vars, range, options)
- Parameters

- dsn-output of dsolve(... , numeric)
- vars-(optional list) axes and functions to plot
- range-(optional) range of the independent variable
- options-(optional) equations that specify plot options

- Description:
  - The *odeplot* function plots one or more solution curves (either 2-D or 3-D) obtained from the output (*dsn*) of *dsolve*.
  - The ordering of the coordinates is given by *vars*. If no coordinates are given, then it is assumed that a plot of the first dependent variable as a function of the independent variable is obtained (that is, the first two coordinates of the solution). Significantly more flexibility is available in the specification of the coordinates. The coordinates can be functions of the independent variable, or any dependent variable and derivative values that are part of the *dsolve* solution.

    For example, for a second-order problem in $y(x)$, we could specify a plot of $y(x)$ versus $y(x)^2 + (\dot{y}(x))^2$ with $[y(x), y(x)^2 + (\dot{y}(x))^2]$.

    Multiple curves can be plotted by specifying a nested list format. For example, $[[x, y(x)], [x, ddxy(x)]]$ displays the dependent variable and its derivative as a function of x on the same plot.

    Curve-specific options can be specified for each curve (for plots with multiple curves) by including them in the desired variable list after the plot variables. Allowed options are color, linestyle, style, symbol, symbolsize, and thickness. For example, the plot described by the nested list above can be displayed with $y(x)$ in blue dots, and $\frac{d}{dx}y(x)$ as a red line of thickness 2 with the argument

$[[x, y(x), color = blue, style = point], [x, diff(y(x), x), color = red, thickness = 2]]$.

- The range argument defines the range of the independent variable to produce the plot, and must evaluate to real numbers. If not specified, the range is determined as follows:
  - If the *dsolve* output is not of a procedure type (that is, it is in the form of matrix), then the values present in the matrix are used directly for the plot.
  - If the problem is a boundary value problem (BVP), then the plot is produced for the entire solution region.
  - If the problem is a initial value problem (IVP), and was created with the *range* option ($rkf45$ and rosenbrock only), then that range is used for the plot.
  - If the problem is a IVP, _Env_smart_dsolve_numeric is set to true, and a prior call was made for the numeric solution, the plot is produced from the initial point to the point used in the prior call.
  - Finally, if none of the above conditions are met, then the plot is produced for the range $x_0 - 10..x_0 + 10$, where x_0 is the initial point for the IVP.
- If the default numerical IVP solvers $rkf45$ and rosenbrock were used with the range argument to obtain *dsn*, then an additional option is available to control the number of points used to produce the plot. The $refine = v$ option tells *odeplot* to use v times the number of stored points for the plot, where v must be a non-negative integer.

  For example, specification of $refine = 1$ tells *odeplot* to use all points in the stored solution, while $refine = 2$ requests twice the computed points, and $refine = 1/3$ requests one-third the computed points.

  Note: Use of above option with a range solution of *dsolve* produces an adaptive plot,

where more points are plotted in more rapidly changing solution regions (that is, regions where a greater number of steps were required by the numerical method).
Note: This option cannot be used with the *numpoints* option.

- If not specified, the labels of the plot are obtained from the *vars* argument (or from the *dsolve* solution if *vars* is not specified). Since these are displayed as text, the derivative of $y(x)$ with respect to x is displayed as $\dot{y}$, and the function $y(x)$ is displayed as y. Variables specified in vars using operator notation, such as $D(y)(x)$ are left unchanged. For 2-D plots, if the length of the text of an automatically-generated axis label exceeds 10 characters, then it is not displayed. To display long labels as a legend (instead of omitting them), specify the $labels = legend$ option.
  Labels can be disabled by specifying them as empty strings or identifiers (that is, $labels = [\text{"", ""}]$ or $labels = [\text{“, “}]$).

- If the $frames = n$ option is given, then odeplot produces an animation of the solution over the independent variable range for the plot, from lowest to highest value. The final frame of the animation is the same as the plot created without the frames option.
  Note: The refine option cannot be used with animations, as *odeplot* must choose the points for the animated plot.

- Remaining arguments must be equations of the form $option = value$. These options are the same as found for plot (in the case of 2-D solution curves) or those found for *plot3d* (in the case of 3-D solution curves).

- The result of a call to *odeplot* is a PLOT or PLOT3D data structure which can be rendered by the plotting device. You can assign the value to a variable, save it in a file, then read it back in for redisplay.

Example: One of the simplest examples is the pendulum system, where the m*g (mass times gravitational constant) term has been replaced by π^2 [64] .

$$dsys := [\frac{d^2}{dt^2}x(t) = -2\lambda(t)x(t), \frac{d^2}{dt^2}y(t) = -2\lambda(t)y(t) - \pi^2, x(t)^2 + y(t)^2 = 1]$$

To display derivatives and functions using a simpler notation, use the PDEtools[declare] command: PDEtools[declare]$(x(t), y(t), \theta(t), r(t), \lambda(t), prime = t, quiet)$
The typical solution approach for this problem is to apply a change of variables to polar co-ordinates, which would proceed as follows:

$$tr := x(t) = r(t)sin(\theta(t)), y(t) = -r(t)cos(\theta(t));$$
$$nsys := PDEtools[dchange](tr, dsys, [r(t), \theta(t)])$$

The last equation implies $r = 1$.
Evaluate and remove $\lambda(t)$ as follows, giving an ODE in $\theta(t)$.
$nsys := nsys||r(t) = 1;$
$isolate(nsys[1]\lambda(t));$
$eval(nsys[2],);$
$pode := simplify(isolate(, d\frac{d^2}{d\theta^2}(t)), trig)$
The above equation is more commonly known as pendulum equation. As an approximation for small displacements, we can approximate $\sin(\theta) \sim \theta$ giving us the simple harmonic oscillator equation: $\frac{d^2}{dt^2}\theta(t) + \pi^2\theta(t) = 0$. This is appropriate for the pendulum equation, but, in general, such a coordinate transform may not exist. Hence, there is a need for numerical solution methods for DAEs [64].

Structure of DAE [65]

Consider the system ($'dsys'$) from the above example.

Consider the constraint: $c1 := dsys[-1]$

Differentiate: $c2 := \frac{\frac{\partial d}{\partial t}c1}{2}$

Differentiate, eliminate, and isolate:

$c3 := eval(\frac{\partial}{\partial t}c2, dsys[1..2]); c3 := factor(isolate(c3, \lambda(t)))$

One more differentiation and elimination yields an ODE for $\lambda(t)$.

d4:=$eval(\frac{\partial}{\partial t}c3, dsys[1..2])$

The above process of successive differentiation until we obtain an ODE for each variable is called index reduction. The number of required differentiations is called the gear index. It sometimes corresponds to the difficulty in obtaining a numerical solution of the DAE problem. (A greater index corresponds to a more difficult problem.)

Issues: We obtained 3 algebraic relations for λ, $x, \dot{x}, y, \dot{y} and c1, c2, c3$ as initial conditions must satisfy these algebraic relations as a result, we only have 2 degrees of freedom in the initial conditions for a problem that would normally have 5. This restricts the initial conditions. They must be checked for consistency.

Note: Constraints are typically nonlinear, and cannot be solved in terms of the free initial conditions. The algebraic constraints must satisfy along the entire solution [65].

Solution Methods [65]

1. Method of Index Reduction: As shown above, the method involves successive differentiation of system until the DAE becomes an ODE system.

   - Advantages:

     -Allows us to obtain all constraints directly

-Allows the use of a standard ODE solver to obtain the solution

- Disadvantages:

  -Does not enforce the constraints, other than at the initial point (resulting in constraint drift)

Example: IRdsys:= {dsys[1],dsys[2],d4} where $g = \phi^2$

$IRcons := [c1, c2, c3]$ with $g = \phi^2$ [65]

- Find suitable initial conditions:

  $ICs := y(0) = -1, D(x)(0) = \frac{1}{10}$

- From the first constraint, we have:

  $IRcons[1]$ where $y(t) = -1$

  $ICs := ICs \cup x(0) = 0 :$

- From the second constraint, we have:

  $eval(convert(IRcons[2], D)att = 0, ICs)$

  $ICs := ICs \cup D(y)(0) = 0 :$

- And finally from the third constraint, we have:

  $eval(convert(IRcons[3], D)att = 0, ICs)$

- $ICs := ICs \cup \{\}$

- Now, we can obtain a numerical solution for the index reduced system ignoring the constraints.

  (Note: We set $maxfun = 0$ to remove the limit on the number of function evaluations.)

- $dsn := dsolve(IRdsys \cup ICs, numeric, maxfun = 0)$

- Consider the value at $t = 1000$:

  $t1000a := dsn(1000)$

- Now consider the constraints at $t = 1000$:

  $evalf[15](eval(map(a \rightarrow (rhs-lhs)(a), convert(IRcons, D)), convert(t1000a[2..-1], D)))$

- This indicates that the constraints are not satisfied. This is one of the disadvantages of this approach.

  Note: Problems arise when small changes to the problem dictate large changes in the solution at a later time (instability).

- The Maple DAE extension solvers ($rkf45\_dae$ and $rosenbrock\_dae$) are modifications to ODE solvers, and can operate in this mode:

  $dsn := dsolve(op(dsys) \cup ICs, numeric, differential = true, projection = false, maxfun = 0)$

- $t1000a2 := dsn(1000)$

- $evalf[15](eval(map(a \rightarrow (rhs-lhs)(a), convert(IRcons, D)), convert(t1000a2[2..-1], D)))$

2. Direct Solution Approaches: These methods work directly with the original DAE system, and do not require differentiation of the constraint. They are typically limited to handling DAEs of index 1, 2, or 3.

   - Advantages:

     -Does not require symbolic manipulation of the system

   - Disadvantages:

     -Does not indicate constraints on the initial data, and simply fails if these are not

satisfied.

-Limited to problems with a maximum index.

-The core solver is usually more complex than for the other approaches.

Maple has the *mebdfi* solver (Modified Extended Backward Difference Formula Implicit). This solver has the ability to fully solve implicit DAE systems. For the above mentioned pendulum example, we get:

- $dsn := dsolve(op(dsys) \cup ICs, numeric, method = mebdfi, maxfun = 0)$:
- t1000e:=dsn(1000)
- $evalf[15](eval(map(a \rightarrow (rhs-lhs)(a), convert(IRcons, D)), convert(t1000e[2..-1], D)))$

  Important: The constraints are all satisfied within tolerance even though they are not explicitly used in the solution process [65] [66].

2.3.2 Wolfram Mathematica

Wolfram Mathematica is a software system with built-in libraries for several areas of technical computing that allow machine learning, statistics, symbolic computation, manipulating matrices, plotting functions and various types of data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other programming languages. It was conceived by Stephen Wolfram, and is developed by Wolfram Research of Champaign, Illinois. The Wolfram Language is the programming language used in Mathematica [67].

Study of Mathematica for Numerical Solution of ODEs and DAEs [68]

In general, a system of ordinary differential equations (ODEs) can be expressed in the normal form,

$$\dot{x}(t) = f(t, x), \quad x \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n \tag{2.15}$$

The derivatives of the dependent variables in (2.23) are expressed explicitly in terms of the independent transient variable and the dependent variables . As long as the function has sufficient continuity, a unique solution can always be found for an initial value problem where the values of the dependent variables are given at a specific value of the independent variable. The derivatives of some of the dependent variables in DAEs typically do not appear in the equations and are not expressed explicitly. For example, the following equation

$$\ddot{x}(t) + y(t) = cos(t) \tag{2.16a}$$

$$x(t) = cos(t) \tag{2.16b}$$

does not explicitly contain any derivatives of y. Such variables are often referred to as algebraic variables [68].

The general form of a system of DAEs is

$$F(t, x, \dot{x}) = 0 \tag{2.17}$$

Solving systems of DAEs often involves many steps. The flow chart shown above indicates the general process associated with solving DAEs in Mathematica with function $NDSolve$. The $NDSolve$ has been discussed further in this chapter.

Generally, a system of DAEs can be converted to a system of ODEs by differentiating it with respect to the independent variable. The index of a DAE is the number of times needed to

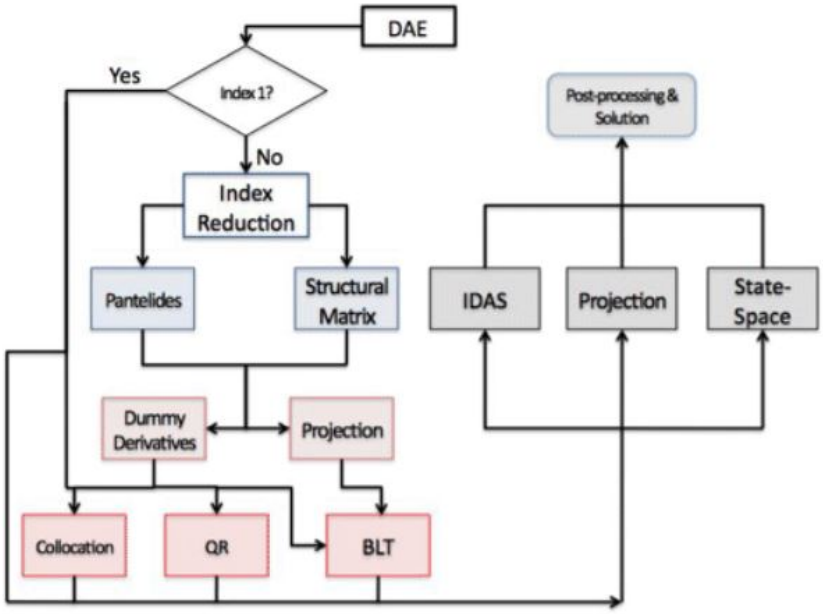


Figure 2.1: Flow chart of steps involved in solving DAE systems in NDSolve.

differentiate the DAEs to get a system of ODEs.

The DAE solver methods built into *NDSolve* works with index-1 systems, so for higher-index systems an index reduction may be necessary to get a solution. *NDSolve* can be instructed to perform that index reduction. When a system is found to have an index greater than 1, *NDSolve* generates a message and number of steps need to be taken in order to solve the DAE [68].

As a first step for high-index DAEs, the index of the system needs to be reduced. The process of differentiating to reduce the index, referred to as index reduction, can be solved with *NDSolve*.

The process of index reduction leads to a new equivalent system. This new system is restructured by substituting new (dummy) variables in place of the differentiated variables. This leads to an expanded system that then can be uniquely solved. Another approach for

restructuring the system involves differentiating the original system n number of times until the differentials for all the variables are accounted for. The preceding $n-1$ differentiated systems are treated as invariants.

In order to solve the new index-reduced system, a consistent set of initial conditions must be found. A system of ODEs in the form, $\dot{x} = f(t, x)$, can always be initialized by giving values for x at the starting time. However for DAEs, it may be difficult to find initial conditions that satisfy the residual equation $F(t, x, \dot{x}) = 0$, this amounts to solving a nonlinear algebraic system where only some variables may be independently specified. Furthermore, the initialization needs to be consistent. This means that the derivatives of the residual equations $\frac{d^n}{dt^n}F(t, x, \dot{x}) = 0$ also need to be satisfied. In general, higher-index systems are harder to initialize. *NDSolve* cannot analyze the interaction of x and hence, not able do an automatic index reduction. However, *NDSolve* has a number of different methods accessible via options to perform index reduction and find a consistent initialization of DAEs [68].

DAE Solver and Solution Methods used in Mathematica [68]

Mathematica has various solution methods built into *NDSolve* for solving DAEs.

Two such methods which work with the general residual form $F(t, x, \dot{x}) = 0$ of index-1 DAEs are,

- IDA—(Implicit Differential-Algebraic solver from the SUNDIALS package) based on backward differentiation formulas (BDF)
- StateSpace—implicitly solves for derivatives to use an underlying ODE solver

If the system has index higher than 1, both of the above mentioned solvers typically fail. To accurately solve such systems, index reduction is needed. Note that index-1 systems can be

reduced to ODEs, but it is often more efficient to use one of the solvers above.

NDSolve [69]

NDSolve is a solver option available in Mathematica that solves for ordinary differencial equations and differencial algebraic equations. *NDSolve* function has the calling sequence

```
NDSolve [eqns, u[x], {x, x_{min}, x_{max}}]
```

which finds a numerical solution to the ODEs and DAEs *eqns* for the function u with the independent variable x in the range x_{min} to x_{max}. *NDSolve* also has some solvers that work with DAEs that can be reduced to special forms, such as:

- MassMatrix—for DAEs of the form $M.\dot{x}(t) = f(t, x(t))$
- Projection—for ODEs with invariants $g(t, x(t)) = 0$

Some of the key features of NDSolve is as follows [69]:

- *NDSolve* gives results in terms of Interpolating Function objects.
- *NDSolve* $[eqns, u[x], x, x_{min}, x_{max}]$ gives solutions for $u[x]$ rather than for the function u itself.
- Differential equations must be stated in terms of derivatives such as $\dot{u}[x]$.
- *NDSolve* solves a wide range of ordinary differential equations as well as partial differential equations.
- *NDSolve* can also solve delay differential equations.
- In ordinary differential equations, the functions u_i must depend only on the single variable t. In partial differential equations, they may depend on more than one variable.

- Initial and boundary conditions are typically stated in the form $u[x_0] == c_0, \dot{u}[x_0] == dc_0$, etc., but may consist of more complicated equations.
- The c_0, dc_0, etc. can be lists, specifying that $u[x]$ is a function with vector or general list values.
- Periodic boundary conditions can be specified using $u[x_0] == u[x_1]$.
- The point x_0 that appears in the initial or boundary conditions need not lie in the range x_{min} to x_{max} over which the solution is solved.
- Boundary values may also be specified using DirichletCondition and NeumannValue.
- The differential equations in $NDSolve$ can involve complex numbers.
- The u_i can be functions of the dependent variables.
- NDSolve adapts its step size so that the estimated error in the solution is just within the tolerances specified by PrecisionGoal and AccuracyGoal.
- AccuracyGoal effectively specifies the absolute local error allowed at each step in finding a solution, while PrecisionGoal specifies the relative local error.
- The option $NormFunction\ f$ specifies that the estimated errors for each of the u_i should be combined using $f[\{e1, e2, \dots\}]$.
- If solutions must be followed accurately when their values are close to zero, AccuracyGoal should be set larger, or to Infinity.
- The default setting of Automatic for AccuracyGoal and PrecisionGoal is equivalent to WorkingPrecision/2.

- The default setting of Automatic for MaxSteps estimates the maximum number of steps to be taken by $NDSolve$, depending on start and stop time and an estimate of the step size. If this is not possible, a fixed number of steps is taken
- The setting for $MaxStepFraction$ specifies the maximum step to be taken by $NDSolve$ as a fraction of the range of values for each independent variable.
- With $DependentVariables = Automatic$, $NDSolve$ attempts to determine the dependent variables by analyzing the equations given.
- $NDSolve$ typically solves differential equations by going through several different stages, depending on the type of equations. With Method-$\{s1- > m1, s2- > m2, \dots\}$, stage s_i is handled by method m_i. The actual stages used and their order are determined by NDSolve, based on the problem to solve.
- Possible solution stages include TimeIntegration solution, BoundaryValue sulutions for ordinary differencial equations and IndexReduction, Initialization for differencial algebraic equations
- With Method m_1 or Method $\{m1, s2- > m2, \dots\}$, the method m_1 is assumed to be for time integration, so Method m_1 is equivalent to Method $\{"TimeIntegration", m_1\}$.
- Possible explicit time integration settings for the Method option include: Adams (predictor-corrector) method with orders 1 through 12, backward differencial formulas (BDF) method with order 1 through 5, ExplicitRungeKutta methods with order 1 through 8, implicit backward differenciation formulas for DAEs, and implicitRungeKutta methods for arbitrary orders.
- With Method function, possible controller methods include composition, DoubleStep, eventLocator, Extrapolation, FixedStep, OrthogonalProjection.

- The setting $InterpolationOrder = All$ specifies that $NDSolve$ should generate solutions that use interpolation of the same order as the above mentioned methods.

IDA Method for NDSolve [70]

The IDA package is part of the SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) developed at the Center for Applied Scientific Computing of Lawrence Livermore National Laboratory [71]. As described in the IDA user guide [72], "IDA is a general purpose solver for the initial value problem for systems of differential-algebraic equations (DAEs). The name IDA stands for Implicit Differential-Algebraic solver. IDA is based on DASPK ...". DASPK [73] is a Fortran code for solving large-scale differential-algebraic systems.

In the Wolfram Language, an interface has been provided to the IDA package so that rather than needing to write a function in C for evaluating the residual and compiling the program, the Wolfram Language generates the function automatically from the equations you input to $NDSolve$.

IDA solves index-1 DAE systems by differentiating the DAEs to get a system of ODEs. IDA solves the system with Backward Differentiation Formula (BDF) methods of orders 1 through 5, implemented using a variable-step form. The BDF of order k at time $t_n = t_{n-1} + h_n$ is given by the formula

$$\sum_{i=0}^{k} a_{n,j} x_{n-i} = h_n \dot{x}_n. \tag{2.18}$$

The coefficients $a_{n,j}$ depend on the order and past step sizes. Applying the BDF to the DAE gives a system of nonlinear equations to solve:

$$F\left(t_n, x_n, \frac{1}{h_n} a_{n,j} x_{n-i}\right) = 0 \tag{2.19}$$

The solution of the system is achieved by Newton-type methods [72], typically using an approximation to the Jacobian

$$J = \frac{\partial F}{\partial x} + c_n \frac{\partial F}{\partial x}, \text{where} \quad c_n = \frac{\alpha_{n,0}}{h_n} \tag{2.20}$$

The IDA's most notable feature is that, in the solution of the underlying nonlinear system (2.27) at each time step, it offers a choice of Newton methods [72]. In the Wolfram Language, you can access the Newton method solvers [72] using *method* options.

IDA computes an estimate E_n at each step (n) of the solution. The local truncation error, step size and order are chosen such that the weighted norm $Norm[E_n/w_n]$ is less than 1. The j^{th} component, $w_{n,j}$, of w_n is given by

$$w_{n,j} = \frac{1}{10^{-prec}|x_{n.j}| + 10^{-acc}} \tag{2.21}$$

The values *prec* and *acc* are taken from the NDSolve settings for the PrecisionGoal (*prec*) and AccuracyGoal (*acc*).

As IDA provides a great deal of flexibility, especially in the way nonlinear equations are solved, there are a number of method options which allow us to get the desired solutions. We can use the method options to IDA by giving $NDSolve$ the option $Method - \{IDA, ida\ method\ options\}$.

When strict accuracy of intermediate values computed with the $InterpolatingFunction$ object is returned from $NDSolve$, we use the NDSolve method option setting $InterpolationOrder = All$ that uses interpolation based on the order of the method. It is sometimes called as dense output and it represents the solution between time steps. By default, $NDSolve$ stores a minimal amount of data to represent the solution well enough for graphical purposes. Keeping the amount of data small saves on both memory and time

for more complicated solutions. When the quantity we want to derive from the solution is complicated, we can ensure that it is locally kept within tolerances by giving it as an algebraic quantity, forcing the solver to keep its error in control. The DAE solution takes far more steps to control the error for the quantity of interest but still uses far less memory than using dense output.

There are two possible settings for the "ImplicitSolver" option, which can be "Newton" or "GMRES". With "Newton", the Jacobian or an approximation to it is computed and the linear system is solved to find the Newton step. On the other hand, "GMRES" is an iterative nonlinear solver method, and rather than computing the entire Jacobian, a directional derivative is computed for each iterative step. The "Newton" method has one method option, "LinearSolveMethod", which we can use to tell the Wolfram Language how to solve the linear equations required to find the Newton step. There are several possible values for this option. The "GMRES" method may be substantially faster, but is typically quite a bit trickier to use because to be really effective, it typically requires a preconditioner which can be supplied via a method option [72]. Using the "GMRES" method without a preconditioner leads to larger computation times and more steps when compared to the default method. It is for this reason that this method is not recommended without a preconditioner. Finding a good preconditioner however is not usually trivial. The setting of the "Preconditioner" option should be a function f, which accepts four arguments that will be given to it by $NDSolve$ such that $f[t, x, \dot{x}, c]$ returns another function that will apply the preconditioner to the residual vector. (See IDA user guide [72] for details on how the preconditioner is used.) The arguments $t, x, \dot{x}, c$ are the current time, solution vector, solution derivative vector and the constant c. For example, if we can determine a procedure that would generate an appropriate preconditioner matrix as a function of these arguments, then we could use

$"Preconditioner" - Function[t, x, xp, c, LinearSolve[P[t, x, xp, c]]]$

to produce a LinearSolveFunction object which will effectively invert the preconditioner matrix P. The LinearSolve Function has been discussed further in this section. Each time the preconditioner function is set up, it is applied to the residual vector several times using some sort of factorization as in LinearSolveFunction. For the diagonal case, the inverse can be affected simply by using the reciprocal. The most difficult part of setting up a diagonal preconditioner is keeping in mind that the values on the boundary should not be affected by it. Thus, even with a crude preconditioner, the "GMRES" method computes the solution faster than using the direct sparse solvers.

LinearSolve [74]

Consider a matrix equation $m.x = b$ *LinearSolve* solves x for the matrix equation. Calling sequence for the matrix equation Linear Solve $[m, b]$. LinearSolve $[m]$ generates a LinearSolveFunction [...] that can be applied repeatedly to different b. Some of the key features of LinearSolve [74] is as follows:

- LinearSolve works on both numerical and symbolic matrices, as well as SparseArray objects.
- The argument b can be either a vector or a matrix.
- The matrix m can be square or rectangular.
- LinearSolve $[m]$ and LinearSolveFunction $[\dots]$ provide an efficient way to solve the same approximate numerical linear system many times.
- LinearSolve$[m, b]$ is equivalent to LinearSolve$[m][b]$.

- For underdetermined systems, LinearSolve will return one of the possible solutions; Solve will return a general solution.
- With Method -Automatic, the method is automatically selected depending upon input.
- Explicit Method settings for approximate numeric matrices include banded matrix solver, Cholesky method for positive definite Hermitian matrices, iterative Krylov sparse solver, direct sparse LU decomposition and parallel direct sparse solver [74].

State-Space Method for DAEs [75]

Consider a DAE system of the following form:

$$f(t, x, \dot{x}, y) = 0 \tag{2.22a}$$

$$g(t, x, y) = 0 \tag{2.22b}$$

If the Jacobian matrices $\frac{\partial f}{\partial \dot{x}}$ and $\frac{\partial g}{\partial y}$ are both invertible, then by the implicit function theorem, the system can effectively be expressed in the state-space form as

$$\dot{x} = F(t, x, G(x)). \tag{2.23}$$

The invertibility requirement is effectively equivalent to the system being of index 1, so a method based on the state-space form is appropriate for index 1 DAE system. The most common form is where $f(t, x, \dot{x}, y) = \dot{x} - f_r(t, x, y)$ and $\frac{\partial f}{\partial \dot{x}}$ is the identity matrix. The "StateSpace" time integration method is based on this representation. Given values of x, Newton iterations are used to find $y = G(x)$ and $\dot{x} = F(t, x, G(x))$ The values $\dot{x}$ are returned for an ODE method that just requires evaluation of the right-hand side of function. In between evaluations, previously computed values of the derivatives and algebraic variables

are saved to initialize iterations for the next evaluation.

The ODEs resulting from the state-space form are often quite stiff and so require a Jacobian. The right-hand side Jacobian can be computed by differentiating the partitioned system with respect to x, as shown below

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} = 0 \tag{2.24a}$$

$$\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}\frac{\partial y}{\partial x} = 0, \tag{2.24b}$$

solving for the right-hand side Jacobian gives

$$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial x} = -\left(\frac{\partial f}{\partial \dot{x}}\right)^{-1}\left(\frac{\partial f}{\partial x} - \frac{\partial f}{\partial y}\left(\frac{\partial g}{\partial y}\right)^{-1}\frac{\partial g}{\partial x}\right). \tag{2.25}$$

Matrix decompositions for $\frac{\partial f}{\partial \dot{x}}$ and $\frac{\partial g}{\partial y}$ are typically saved from the Newton iterations so the application of the inverses can be computed efficiently [75].

The "StateSpace" method is typically slower than the default "IDA" multistep method because it has to solve for each evaluation as opposed to for each time step. Some possible advantage is that a stable integration method could be used to improve overall stability. The StateSpace method is effectively a one-step method and is used to implement arbitrary-precision solutions [75]. The Method option of the "StateSpace" method allows us to choose the integration method.

Block Lower Triangular (BLT) Form [75]

Consider a linear equation $s.x = b$ where s is matrix. When s is a nonsingular matrix, the variables in each successive block along the diagonal depend only on the ones in previous blocks. This means that the above linear equation can be solved by working sequentially down the diagonal, solving each block subsystem in order, forming a block lower triangular

form.

Depending on the structure of the system, it may be advantageous to order the system into block lower triangular (BLT) form. If the largest block size can be greatly reduced using BLT, then it can reduce the computational complexity of the Newton iterations significantly. The BLT form is used in a number of ways for solving DAEs, including consistent initialization, setting up dummy derivatives for index reduction, and solving sparse linear systems. The description in this section is focused on its use in the state-space method, but the applicability extends to other areas as well.

In its simplest form, the BLT ordering is a matrix algorithm, and that is the best general way to access it in the Wolfram Language [75].

Projection Method for NDSolve [76]

When a differential system has a certain structure, it is advantageous if a numerical integration method preserves the structure. In certain situations, it is useful to solve differential equations in which solutions are constrained. Projection methods work by taking a time step with a numerical integration method and then projecting the approximate solution onto the manifold on which the true solution evolves [76].

NDSolve includes a differential algebraic solver which may be appropriate and is described in more detail within Numerical Solution of Differential-Algebraic Equations [68].

Sometimes the form of the equations may not be reduced to the form required by a DAE solver. Furthermore, so-called index reduction techniques can destroy certain structural properties that the differential system may possess (see [77] and [78]). An example that illustrates this can be found in the documentation for DAEs [68].

In such cases it is often possible to solve a differential system and then use a projective procedure to ensure that the constraints are conserved. This is the idea behind the method

"Projection".

If the differential system is ρ-reversible [79], then a symmetric projection process can be advantageous. Symmetric projection is generally more costly than projection and has not yet been implemented in *NDSolve*.

ODEs with invariants: Consider a DAE of the form

$$\dot{x}(t) = f(t, x(t)), \quad x \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n \tag{2.26a}$$

$$g(t, x(t)) = 0 \tag{2.26b}$$

where g is an invariant that is consistent with the differential equations.

When a system of DAEs is converted to ODEs via index reduction, the equations that were differentiated to get the ODE form are consistent with the ODEs, and it is typically important to make sure those equations are well satisfied as the solution is integrated.

Such systems have more equations than unknowns and are overdetermined unless the constraints are consistent with the ODEs. *NDSolve* does not handle such DAEs directly because it expects a system with the same number of equations and dependent variables. In order to solve such systems, the "Projection" method built into *NDSolve* handles the invariants by projecting the computed solution onto after each time step. This ensures that the algebraic equations are satisfied for solution [68].

Example 1. Consider a system of DAE with three equations, but only one differential term given by

$$\dot{x_1}(t) = x_3(t) \tag{2.27a}$$

$$x_2(t)(1 - x_2(t)) = 0 \tag{2.27b}$$

$$x_1(t)x_2(t) + x_3(t)(1 - x_2(t) = t \tag{2.27c}$$

With initial conditions $x_1(0) = -1, x_2(0) = 0$

From the the second equation (2.35b), x_2 requires to be either 0 or 1.

If $x_2(t) = 0$, then the two remaining equations make an index-1 system, since differentiation of the third equation gives a system of two ODEs:

$$\dot{x_1}(t) = x_3(t) \tag{2.28a}$$

$$\dot{x_3}(t) = 1 \tag{2.28b}$$

On solving the system of DAEs with specified initial condition for $x_1 = -1$ and $x_2 = 0$ and $t \in [0, 1]$ in Mathematica gives the solution plot of x_1, x_2 and x_3 as shown in figure 2.2

If $x_2(t) = 1$, then the two remaining equations (2.35a and 2.35c) make an index-2 system. Differentiating the third equation (2.35c) gives $\dot{x}_1(t) = 1$, and substituting into the first equation (2.35a) gives $x_3(t) = 1$, which needs to be differentiated to get an ODE (2.37)

$$\dot{x_1}(t) = x_3(t) \tag{2.29a}$$

$$x_3(t) = 1 \tag{2.29b}$$

On solving the system of DAEs starting with $x_2 = 1$ gives the solution plot

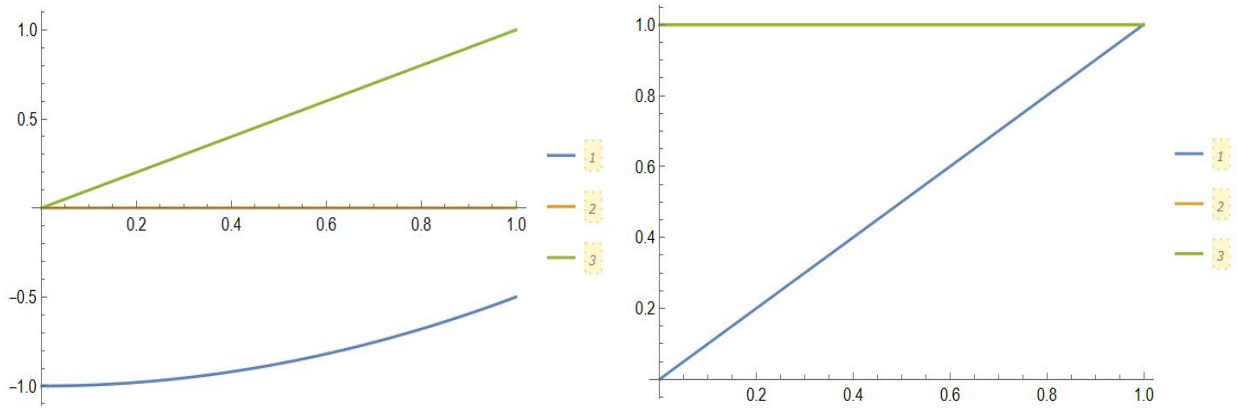


Figure 2.2: Plot for the solution of x_1, x_2, x_3 for x_2 equals to 0 and 1

Both the number of initial conditions needed and the index of a DAE may influence the actual solution being found.

Index Reduction for DAE

The built-in solvers for DAEs in *NDSolve* handles index-1 DAE systems automatically. For higher-index systems, an index reduction is necessary to get to a solution. This index reduction can be performed by giving appropriate options to *NDSolve*.

NDSolve uses symbolic techniques to do index reduction. This means that if our DAE system is expressed in the form $F(t, x, \dot{x}) = 0$, where *NDSolve* cannot see how the components of x interact and compute symbolic derivatives, the index reduction cannot be done. Hence, for this reason NDSolve does not do index reduction, by default.

For example, consider the classic DAE describing the motion of a pendulum:

$$m\ddot{x}(t) + \lambda(t)x(t) = 0, \tag{2.30a}$$

$$m\ddot{y}(t) + \lambda(t)y(t) = -g, \tag{2.30b}$$

$$x^2(t) + y^2(t) = L, \tag{2.30c}$$

with initial conditions $x(0) = 1, \dot{x}(0) = 0, \ddot{x}(0) = 0, y(0) = 0, \dot{y}(0) = 0, \ddot{y}(0) = -1$,where there is a mass m at the point $\{x(t), y(t)\}$ constrained by a string of length L. λ is a Lagrange multiplier that is effectively the tension in the string. For simplicity in the description of index reduction, take $m = L = g = 1$. The software detects that the index is 3, indicating that index reduction is necessary to solve the system. The option setting in $NDSolve$ uses the index reduction method *irmeth* with general index reduction options *iropts*

For all index reduction methods, other options such as "ConstraintMethod" to handle constraints and "IndexGoal" to reduce index to (1 or 0)

Index reduction is done by differentiating equations in the DAE system. Suppose that an equation (*eqn*) is differentiated during the process of index reduction so that $deqn = \frac{d}{dt} eqn$ is included in the system in place of *eqn*. Once the differentiation has been done, the differentiated equations comprise a fully determined system and can be solved. However, it is typically important to incorporate the original equations in the system as constraints. This is controlled by the "ConstraintMethod" index reduction option, using the index reduction and solving the equation with constraints.

In the numerical solution with just the differentiated equations, the length of the pendulum is drifting away. Incorporating the original equations into the solution as constraints is an important aspect of index reduction.

The purpose of the dummy derivative method is to introduce algebraic variables so that the combined system of original equations and equations that have been differentiated for index reduction is not overdetermined. $NDSolve$ can solve the system with index reduction algorithms built on dummy derivatives.

Index Reduction Algorithms

NDSolve has two algorithms for achieving index reduction, the Pantelides and the structural matrix methods. Both of these methods use the symbolic form of the equations to determine what equations to differentiate and then use symbolic differentiation to get the differentiated system.

Both methods make use of the concept of structural incidence in the equations; in particular, if variable v_j appears explicitly in equation e_j, then v_j is termed incident in e_j. The structural incidence matrix is the matrix with elements $m_{ij} = 1$ if v_j is incident in e_j and otherwise $m_{ij} = 0$. A DAE system has structural index 1 if the structural incidence matrix can be reordered so that there are 1s along the diagonal. The existence of such a reordering means that there is a matching between equations and variables such that there is one equation for every variable.

Pantelides Method [80]

The method proposed by Pantelides [80] is a graph theoretical method that was originally proposed for finding consistent initialization of DAEs. It works with a bipartite graph of dependent variables and equations, and when the algorithm can find a traversal of the graph, then the system has been reduced to index of atmost 1. A traversal in this sense effectively means an ordering of variables and equations so that the graph's incidence matrix has no zeros on the diagonal.

When the bipartite graph does not have a complete traversal, the algorithm effectively augments the path by differentiating equations, introducing new variables (derivatives of previous variables) in the process. Unless the original system is structurally singular, the algorithm will terminate with a traversal. Generally, the algorithm does only the differentiations needed to get the traversal, but since it is a greedy algorithm, it is not

always the minimal number.

The implementation built into the Wolfram Language follows the algorithm outlined in [81]. It uses Graph to efficiently implement the graph computations and symbolically differentiates the equations with D when differentiation is called for.

When there is a traversal of the system, it is then possible to find an ordering for the variables and equations such that the incidence matrix is in block lower triangular (BLT) form. The BLT form is used in setting up the dummy derivative method for maintaining constraints and also in the "StateSpace" time integration method. A description of the BLT ordering is included in "State Space Method for DAEs".

The Pantelides algorithm is efficient because it can use graph algorithms with well-controlled complexity even for very large problems. However, since it is based solely on incidence, there can be issues with systems that lead to Jacobians that are singular. The structural matrix method may be able to resolve such cases, but may have larger computational complexity for large systems.

Structural Matrix Method

The structural matrix algorithm is an alternative to the Pantelides algorithm. The structural matrix method follows from the work of Unger [UKM95] and Chowdhary et al. [CKL04]. The graph-based algorithms such as the Pantelides algorithm rely on traversals to perform index reduction. However, they do not account for the fact that sometimes there may be variable cancellations in a particular system. This leads to the algorithm underestimating the index of the system. If a DAE has not been correctly reduced to an index-0 or index-1 system, then the numerical integration may fail or may produce an incorrect result.

Note that the structural index may be smaller than the actual index of the system. In the real system the Jacobian matrix may be singular. The structural matrix method tries to

take into account the possibility of singularities in the Jacobian and will do a better job of index reduction for some systems, but at the cost of greater computational expense.

Constraint Methods

The purpose of the dummy derivative method is to introduce algebraic variables so that the combined system of original equations and equations that have been differentiated for index reduction is not overdetermined. The main idea behind the dummy derivative method of Mattson and Söderlind [82] is to introduce new variables that represent derivatives, thus making it possible to reintroduce constraint equations without getting an overdetermined system.

The alternative to using dummy derivatives is to reduce the index to 0 such that there is a system of ODEs. We use projection to ensure that the original constraints are satisfied. With this method, the original constraints are typically satisfied up to the local tolerances for *NDSolve* specified by the PrecisionGoal and AccuracyGoal options.

2.3.3 MATLAB Toolbox for the Numerical Solution of DAEs

SolveDAE is a MATLAB toolbox [83] for the numerical solution of systems of linear or nonlinear differential-algebraic equations (DAEs) of arbitrary index given in the form

$$F(x(t), \dot{x}(t), t) = 0, \quad x(t_0) = x_0 \tag{2.31}$$

or

$$E(t)\dot{x}(t) = A(t)x(t) + f(t), \quad x(t_0) = x_0 \tag{2.32}$$

in the domian $[t_0, t_f]$

The toolbox employs the Fortran77 subroutines GENDA (GEneral Nonlinear Differential-

Algebraic equation solver) [83] and GELDA (GEneral Linear Differential-Algebraic equation solver) [84] to solve these kind of problems. Both routines provide solutions for under- and overdetermined systems. The toolbox features symbolic differentiation (the Symbolic Math Toolbox is required). In this way the user only has to supply the functions defining the given DAE and does not need to provide the derivatives or Jacobians of these functions. Other features of the toolbox include the computation of characteristic values and the computation of consistent initial values in the least square sense (the Optimization Toolbox is required). In order to make the utilization of the solvers GELDA and GENDA as easy and comfortable as possible, a graphical user interface (GUI) provides the possibility to adjust a variety of parameters that enable the user to customize the solvers to certain problems [83] [83].

GEneral Nonlinear Differential Algebraic Equation Solver (GENDA)

GENDA is a Fortran77 sofware package for the numerical solution of nonlinear differential-algebraic equations (DAEs) of arbitrary index In the domian $[t_0, t_f]$ as shown below: [85]

$$F(x, \dot{x}, t) = 0, \quad x(t_0) = x_0 \tag{2.33}$$

An important invariant in the analysis of DAEs is the so called strangeness index, which generalizes the differentiation index [85], [86], [87] for systems with undetermined components [88]. Many of the integration methods for general DAEs require the system to have differentiation index not higher than one. If this condition is not valid or if the DAE has undetermined components, then the methods as implemented in codes like DASSL of Petzold [52] or LIMEX of Deuflhard/Hairer/Zugck [89] may fail. The implementation of GENDA is based on the construction of the discretization scheme introduced in [1], which transforms the system into a strangeness-free DAE with the same local solution set. The

resulting strangeness-free system is allowed to have nonuniqueness in the solution set or inconsistency in the initial values or inhomogeneities. But this information is now available to the user and systems with such properties can be treated in a least squares sense. When DAE is found to be uniquely solvable, GENDA is able to compute a consistent initial value and apply an integration scheme for DAEs. In GENDA Runge-Kutta scheme of type RADAU IIa of order 5 [90] is implemented [91].

GEneral Linear Differential Algebraic Equation Solver (GELDA)

GELDA is a Fortran77 sofware package for the numerical solution of linear differential-algebraic equations (DAEs) with variable coefficients of arbitrary index

$$E(t)\dot{x}(t) = A(t)x(t) + f(t), \quad t \in [t_0, t_f] \tag{2.34}$$

together with initial condition $x(t_0) = x_0, \quad t_0 \in [t_0, t_f]$

An important invariant in the analysis of linear DAEs is the so called strangeness index, which generalizes the differentiation index ([85], [86], [87]) for systems with undetermined components which occur, for example, in the solution of linear quadratic optimal control problems and differential-algebraic Riccati equations, see e.g. [92], [93], [94]. Many of the integration methods for general DAEs require the system to have differentiation index not higher than one, which corresponds to a vanishing strangeness index, see [95]. If this condition is not valid or if the DAE has undetermined components, then the methods as implemented in codes like DASSL of Petzold [96], LIMEX of Deuflhard/Hairer/Zugck [89], or RADAU5 of Hairer/Wanner [90], [91] may fail. The implementation of GELDA is based on the construction of the discretization scheme introduced in [95], which first determines all the local invariants and then transforms the system into an equivalent strangeness-free

DAE with the same solution set. The resulting strangeness-free system is allowed to have nonuniqueness in the solution set or inconsistency in the initial values or inhomogeneities. But this information is now available to the user and systems with such properties can be treated in a least squares sense. When DAE is found to be uniquely solvable, GELDA is able to compute a consistent initial value and apply the well-known integration schemes for DAEs. In GELDA the BDF methods [97] and Runge-Kutta schemes [90], [91] are implemented.

2.3.4 Matlab Solver ode 15s and 23t [97]

$ode15s$ solves stiff differential equations and DAEs using variable order method. The function $[t, y] = ode15s(odefun, tspan, y_0)$, where $tspan = [t_0, t_f]$, integrates the system of differential equations $\dot{y} = f(t, y)$ from t_0 to t_f with initial conditions y_0. Each row in the solution array y corresponds to a value returned in column vector t.

The $ode15s$ and $ode23t$ solvers can solve index-1 linearly implicit problems with a singular mass matrix $M(t, y)\dot{y} = f(t, y)$, including semi-explicit DAEs of the form

$$\dot{y} = f(t, y, z), \tag{2.35a}$$

$$g(t, y, z) = 0. \tag{2.35b}$$

In this form, the presence of algebraic variables leads to a singular mass matrix, since there are one or more zeros on the main diagonal.

$$M\dot{y} = \begin{pmatrix} \dot{y}_1 & 0 & \dots & 0 \\ 0 & \dot{y}_2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

By default, solvers automatically test the singularity of the mass matrix to detect DAE systems. If we know about singularity ahead of time, then we can set the $MassSingular$ option of *odeset* to 'yes'. With DAEs, we can also provide the solver with a guess of the initial conditions for $\dot{y}_0$ using the $InitialSlope$ property of *odeset*. This is in addition to specifying the usual initial conditions for y_0 in the call to the solver.

The $ode15i$ solver can solve more general DAEs in the fully implicit form

$$f(t, y, \dot{y}) = 0, \quad y \in \mathbb{R}^n, \quad f : \mathbb{R}^n \to \mathbb{R}^n \tag{2.36}$$

In the fully implicit form, the presence of algebraic variables leads to a singular Jacobian matrix. This is because at least one of the columns in the matrix is guaranteed to contain all zeros as the derivative of that variable does not appear in the equations.

$$J = \frac{\partial f}{\partial \dot{y}} = \begin{pmatrix} \frac{\partial f_1}{\partial \dot{y}_1} & \cdots & \frac{\partial f_1}{\partial \dot{y}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \dot{y}_1} & \cdots & \frac{\partial f_m}{\partial \dot{y}_n} \end{pmatrix}$$

The $ode15i$ solver requires that we specify initial conditions for both $\dot{y}_0$ and y_0, Also, unlike the other ODE solvers, $ode15i$ requires the function encoding the equations to accept an extra input: $odefun(t, y, yp)$.

DAEs arise in a wide variety of systems because physical conservation laws often have forms like $x + y + z = 0$. If x, $\dot{x}$, y, and $\dot{y}$ are defined explicitly in the equations, then this conservation equation is sufficient to solve for z without having an expression for $\dot{z}$.

Consistent Initial Conditions [98]

When we are solving a DAE, we can specify initial conditions for both $\dot{y}_0$ and y_0. The *ode*15*i* solver requires both initial conditions to be specified as input arguments. For the *ode*15*s* and *ode*23*t* solvers, the initial condition for $\dot{y}_0$ is optional (but can be specified using the *InitialSlope* option of *odeset*). In both cases, it is possible that the initial conditions we specify do not agree with the equations you are trying to solve. Initial conditions that conflict with one another are called inconsistent [98]. The treatment of the initial conditions varies by solver:

- ode15s and ode23t — If we do not specify an initial condition for $\dot{y}_0$, then the solver automatically computes consistent initial conditions based on the initial condition we provide for y_0. If we specify an inconsistent initial condition for $\dot{y}_0$, then the solver guess the value by attempting to compute consistent values close to the guess, and continues to solve the problem.

- *ode*15*i* — The initial conditions we supply to the solver must be consistent, and *ode*15*i* does not check the supplied values for consistency. The helper function *decic* computes consistent initial conditions for this purpose.

Differential Index

DAEs are characterized by their differential index, which is a measure of their singularity. By differentiating equations, we can eliminate algebraic variables and if we do this enough times, then the equations take the form of a system of explicit ODEs. The differential index of a system of DAEs is the number of derivatives we must take to express the system as an equivalent system of explicit ODEs. Thus, ODEs have a differential index of 0.
The *ode*15*s* and *ode*23*t* solvers only solve DAEs of index 1. If the index of the equations is 2

or higher, then we need to rewrite the equations as an equivalent system of index-1 DAEs. It is always possible to take derivatives and rewrite a DAE system as an equivalent system of index-1 DAEs. Note that if we replace algebraic equations with their derivatives, then we might have removed some constraints. If the equations no longer include the original constraints, then the numerical solution can drift [98].

Imposing Nonnegativity

Most of the options in *odeset* works in the same manner as the DAE solvers $ode15s$, $ode23t$, and $ode15i$. However, one notable exception is with the use of the NonNegative option. The $NonNegative$ option does not support implicit solvers ($ode15s$, $ode23t$, $ode23tb$) applied to problems with a mass matrix. Therefore, you cannot use this option to impose nonnegativity constraints on a DAE problem, which necessarily has a singular mass matrix [98].

Chapter 3

Comparision of DAE system Simulation Software

In this chapter, we are analyzing various systems using integration methods used to solve complex DAE's in professional DAE software. The comparison of solutions have been done at specific time in each software and the solutions has been plotted.

3.1 The Double Pendulum

The double pendulum is a pendulum with an additional weight attached to the first pendulum bob by a string. It is a system with four 2-order ODEs and two index-3 DAEs.

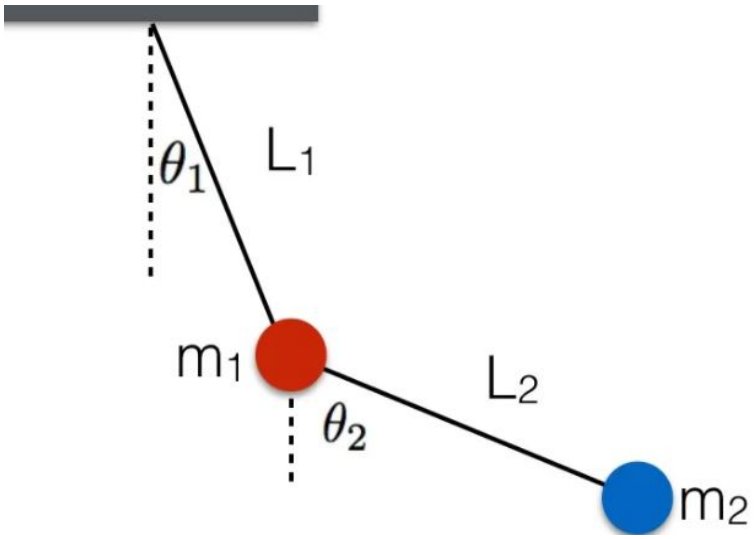


Figure 3.1: Double Pendulum

$(x_1(t), y_1(t))$ are the coordinates of the first pendulum bob, and $(x_2(t), y_2(t))$ are the coordinates of the second, which is attached to the first.

$$m_1\ddot{x_1} + 2\lambda_1 x_1(t) + 2\lambda_2(x_1 - x_2) = 0 \tag{3.1a}$$

$$m_1\ddot{y_1} + m_1 g + 2\lambda_1 y_1 + 2\lambda_2(y_1 - y_2) = 0 \tag{3.1b}$$

$$m_2\ddot{x_2} - 2\lambda_2(x_1 - x_2) = 0, \tag{3.1c}$$

$$m_2\ddot{y_2} + m_2 g - 2\lambda_2(y_1 - y_2) = 0 \tag{3.1d}$$

Given the DAE (3.1) with constraints eqn,

$$x_1^2 + y_1^2 = l_1^2,$$

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 = l_2^2$$

,constants

$$m_1 = 1$$

$$l_1 = 1$$

$$m_2 = 1$$

$$l_2 = 0.5$$

$$g = 9.8$$

and Initial Conditions:

$$x_1(0) = 0,$$

$$\dot{x_1}(0) = -52,$$

$$y_1(0) = -l1,$$

$$\dot{y_1}(0) = 0,$$

$$x_2(0) = 0,$$

$$\dot{x_2}(0) = 3,$$

$$y_2(0) = -l1 - l2,$$

$$\dot{y_2}(0) = 0$$

Results

At t=3s:

Software	MapleSoft	Maplesoft	Mathematica
Method	rkf45	mebdfi	bdf
$y_1(t)$	-0.992989982189998455	-0.992989982189998455	-0.999581
$y_2(t)$	-1.16620471672429	-1.16550276522482821	-1.28058
$x_1(t)$	-0.238568182081203	0.118198571580415718	-0.0289592
$x_2(t)$	-1.16620471672429	0.587495225070804317	-0.442528
$\lambda_1(t)$	0.118771407459081	8.74100178650109960	6.94487
$\lambda_2(t)$	0.587784168432389	8.22823789229041580	5.17561
$\dot{y_1}(t)$	8.76953859380733	-0.236982303015614737	-0.0302166
$\dot{y_2}(t)$	8.28461507797740	1.70158342077031000	0.848122
$\dot{x_1}(t)$	-0.238568182081203	-1.99088067513323863	1.04293
$\dot{x_2}(t)$	1.70427327529758	-1.27826095258629624	0.446134

Table 3.1: Double pendulum solution values at $t = 3$

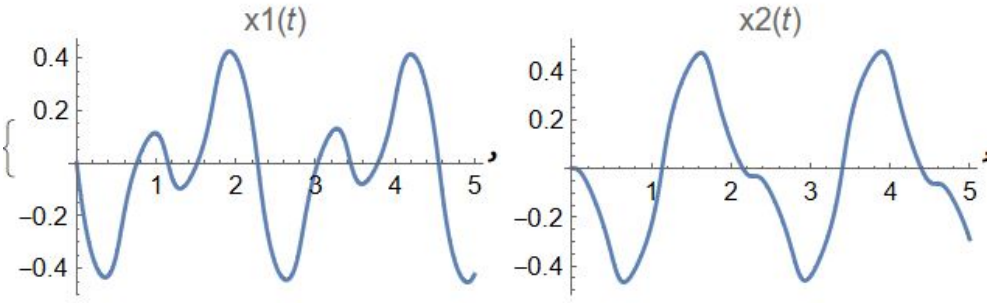


Figure 3.2: Solution plot of $x_1(t)$ and $x_2(t)$

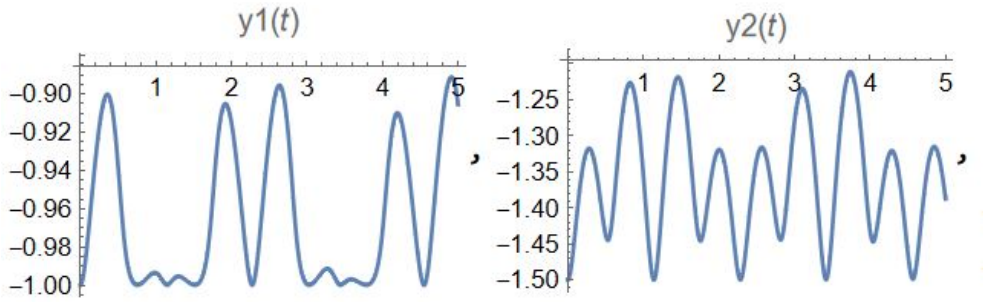


Figure 3.3: Solution plot of $y_1(t)$ and $y_2(t)$

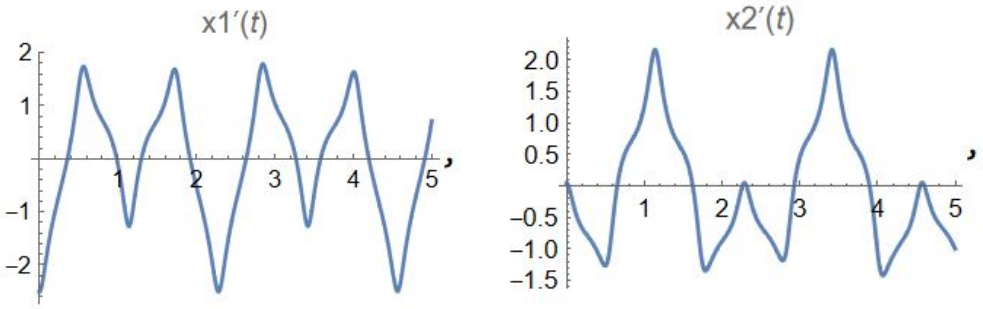


Figure 3.4: Solution plot of $\dot{x}_1(t)$ and $\dot{2}_1(t)$

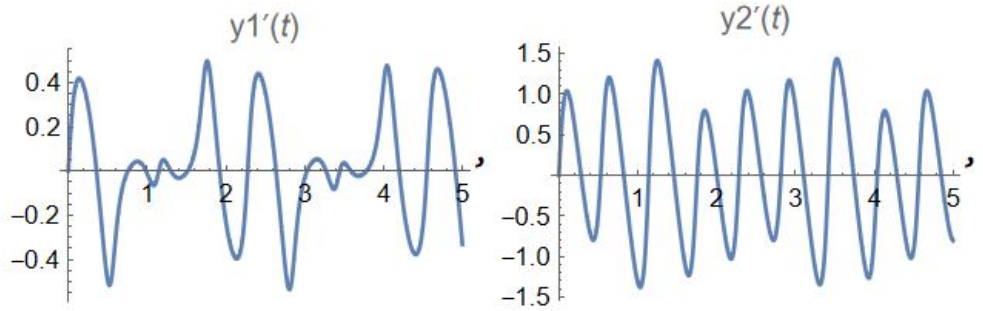


Figure 3.5: Solution plot of $\dot{y}_1(t)$ and $\dot{y}_2(t)$

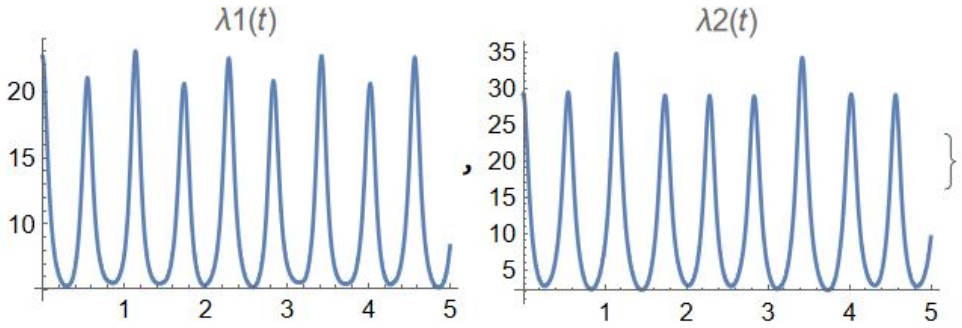


Figure 3.6: Solution plot of $\lambda 1(t)$ and $\lambda 2(t)$

3.2 Linear DAE system

Consider the following linear DAE system of index-1,

$$\dot{x}(t) + \dot{y}(t) = y(t) - sin(t) + 2cos(t) \tag{3.5}$$

Given the DAE (3.5) with constraints eqn,

$$x(t) + y(t) = -sin(t)$$

with Initial Conditions:

$$y(0) = -3$$

Mathematica: The Pantelides algorithm fails in Mathematica software in this case because there is a singular Jacobian matrix. So we solve the system using the structural matrix method for index reduction, which is more accurate.

Results:

At t=3s:

Software	Method	x(t)	y(t)
MapleSoft	rkf45_dae	-3.25221752429386	3.11109748275225
MapleSoft	mebdfi	-3.25221712086898007	3.11109711188525084
Mathematica	bdfi	0.00755848	-0.999126

Table 3.2: Linear DAE system solution using Different Software and Method

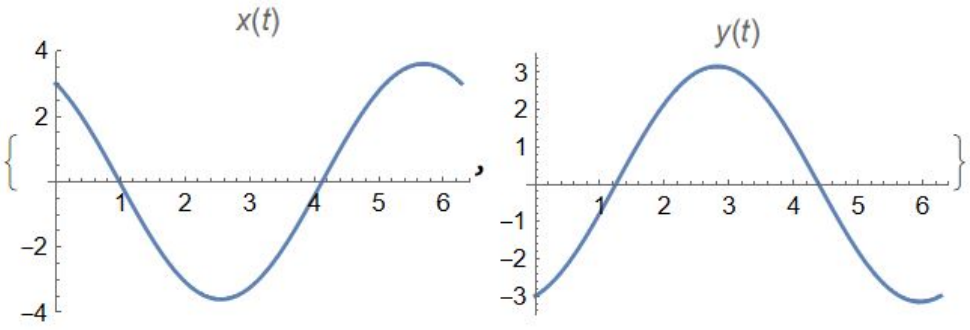


Figure 3.7: Plot of x(t) and y(t) of linear DAE system

3.3 Robertson Problem as Semi-Explicit Differential Algebraic Equations

This example reformulates a system of ODEs as a system of differential algebraic equations (DAEs). The Robertson problem is a classic test problem for programs that solve stiff ODEs.

The system of equations is

$$\dot{y_1}(t) = -0.04y_1 + 10^4 y_2 y_3 \tag{3.8a}$$

$$\dot{y_2} = 0.04y_1 - 10^4 y_2 y_3 - (3 * 10^7) y_2^2 \tag{3.8b}$$

$$\dot{y_3} = (3 * 10^7) y_2^2 \tag{3.8c}$$

Given the DAE (3.8) with constraints eqn,

$$y_1 + y_2 + y_3 = 1$$

and Initial Conditions:

$$y_1(0) = 1$$

$$y_2(0) = 0$$

$$y_3(0) = 0$$

The system of equations can be rewritten as a system of DAEs by using the conservation law to determine the state of y_3. This reformulates the problem as the DAE system.

Results

At t=5s:

Software	MapleSoft	Matlab	Mathematica
Method	rkf45	ode15s	bdf
$y_1(t)$	0.891517815060457	0.8916	0.891518
$y_2(t)$	0.0000208525842127058	0.2086	0.0000208527
$y_3(t)$	0.108461332355330	0.1084	0.108461

Table 3.3: Solution values of semi-explicit DAE system at $t = 5s$

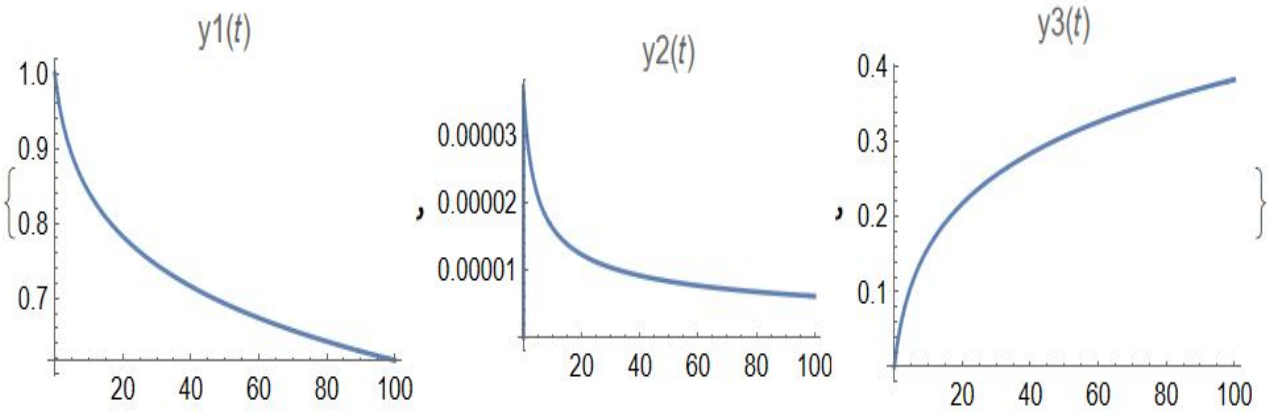


Figure 3.8: Solution plot of $y_1(t)$, $y_2(t)$ and $y_3(t)$ for semi-explicit DAE Vs time

3.4 The Car Axis System

The car-axis model is a simple multi-body system that is designed to simulate the behavior of a car axis on a bumpy road. A schematic of the system is shown below. The model consists of two wheels: the left wheel is assumed to be moving on a flat surface, while the right wheel moves over a bumpy surface. The bumpy surface is modeled as a sinusoid. The

left and right wheels transfer their motion to the chassis of the car through two massless springs as shown following. The chassis of the car is modeled as a bar of mass M. This is an index-3 system.

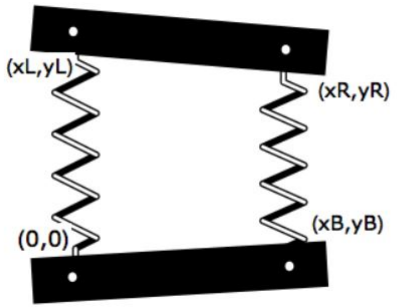


Figure 3.9: Schematic of Car Axis System

The left tire is always on a flat surface, while the right tire periodically goes over a bump.

$$\frac{\ddot{x_l}}{2000} = (\frac{1}{2ll} - 1)x_l + \lambda_1 x_b + 2\lambda_2(x_l - x_r) \tag{3.11a}$$

$$\frac{\ddot{y_l}}{2000} = (\frac{1}{2ll} - 1)y_l + \lambda_1 y_b + 2\lambda_2(y_l - y_r) - \frac{1}{2000} \tag{3.11b}$$

$$\frac{\ddot{x_r}}{2000} = (\frac{1}{2lr} - 1)(x_r - x_b) - 2\lambda_2(x_l - x_r), \tag{3.11c}$$

$$\frac{\ddot{y_r}}{2000} = (\frac{1}{2lr} - 1)(y_r - y_b) - 2\lambda_2(y_l - y_r) - \frac{1}{2000} \tag{3.11d}$$

Given the DAE (3.11) with constraints eqn,

$$x_l x_b + y_l y_b = 0,$$

$$(x_l - x_r)^2 + (y_l - y_r)^2 = 1$$

variables

$$y_b = \frac{sin(10t)}{10}$$
$$x_b = \sqrt{1 - y_b^2}$$
$$l_l = \sqrt{x_l^2 + y_l^2}$$
$$l_r = \sqrt{(x_r - x_b)^2 + (y_r - y_b)^2)}$$

and Initial Conditions:

$$x_l(0) = 0, \quad \dot{x_l}(0) = -12,$$
$$y_l(0) = 12, \quad \dot{y_l}(0) = 0,$$
$$x_r(0) = 1, \quad \dot{x_r}(0) = -12,$$
$$y_r(0) = 12, \quad \dot{y_r}(0) = 0,$$
$$\lambda_1(0) = 0, \quad \lambda_2(0) = 0$$

Results

At t=3s:

Software	MapleSoft	Maplesoft	Mathematica
Method	rkf45	mebdfi	bdf
$y_l(t)$	0.496989454931799	0.496986099958342753	0.496989
$y_r(t)$	0.373910194846413	0.373920932254743710	0.373911
$x_l(t)$	0.0493455779014822	0.0493452447425273694	0.0493456
$x_r(t)$	1.04174242180063	0.0175876121855471045	1.04174
$\lambda_1(t)$	−0.00473687080377494	−0.00473829109687389904	−0.00473689
$\lambda_2(t)$	−0.00110467301491424	−0.00110538962168678020	−0.00110468
$\dot{y_l}(t)$	0.00743988527865740	0.00737709900347955066	0.00744686
$\dot{y_r}(t)$	0.770371407475221	0.770647938480540917	0.770342
$\dot{x_l}(t)$	−0.0770590607488194	−0.0770648014835232098	−0.0770584
$\dot{x_r}(t)$	0.0175614035187924	0.0175876121855471045	0.0175569

Table 3.4: Solution values of multi-body system (Car-axis system) at $t = 3s$

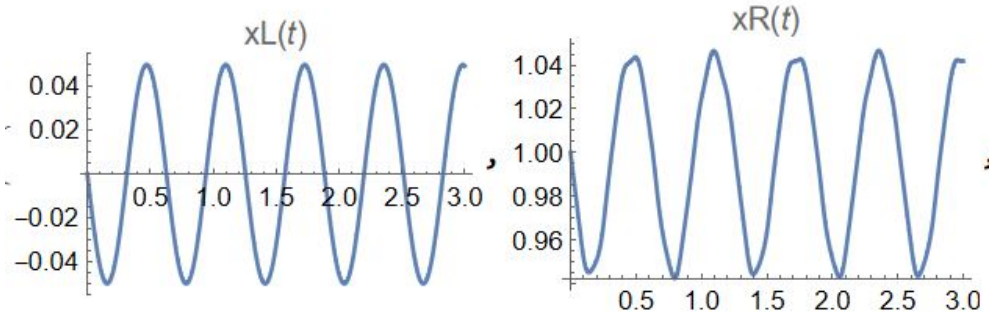


Figure 3.10: Solution plot of $x_L(t)$ and $x_R(t)$

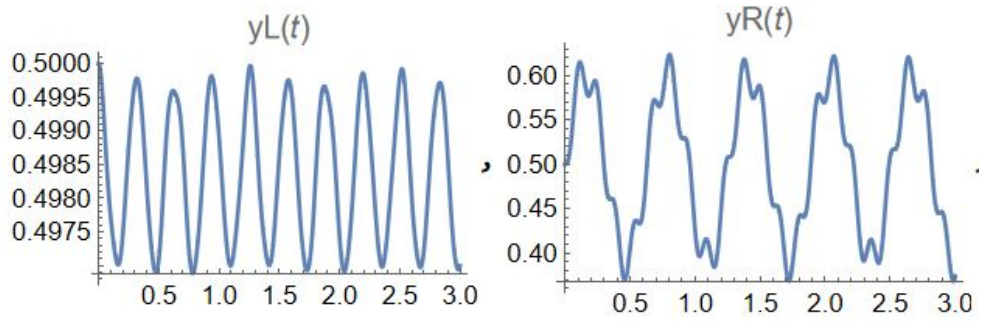


Figure 3.11: Solution plot of $y_L(t)$ and $y_R(t)$

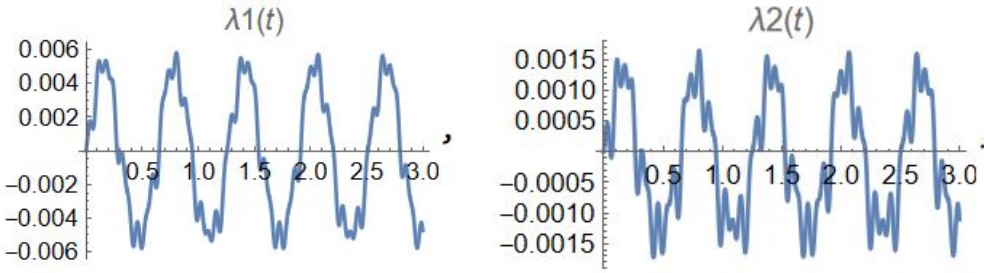


Figure 3.12: Solution plot of $\lambda_1(t)$ and $\lambda_2(t)$

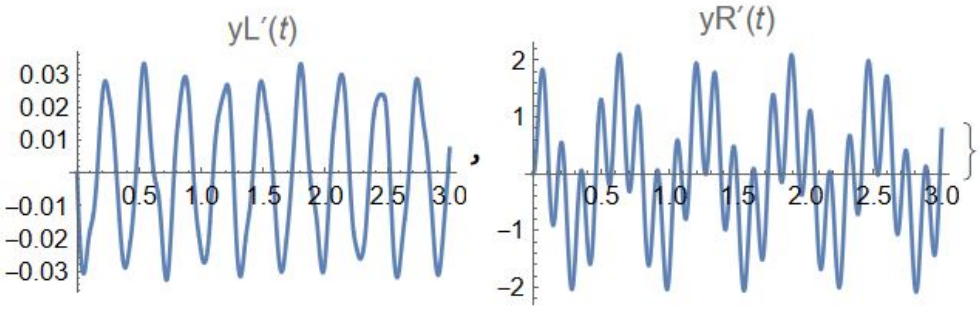


Figure 3.13: Solution plot of $\dot{y}_L(t)$ and $\dot{y}_R(t)$

3.5 Simple Pendulum in Cartesian Coordinates

In this example, we model a simple pendulum in Cartesian coordinates. The equations result in an index-3 DAE, which we solve using *dsolve* with the numeric option in Mathematica.

$$\dot{vx}(t) = F(t)x(t) \tag{3.15a}$$

$$\dot{vy}(t) = F(t)y(t) - g \tag{3.15b}$$

$$\dot{x}(t) = vx(t) \tag{3.15c}$$

$$\dot{y}(t) = vy(t) \tag{3.15d}$$

Given the DAE (3.15) with constraints eqn,

$$x^2 + y^2 = 1$$

Initial Conditions:

$$x(0) = 1,$$

$$y(0) = 0,$$

$$vx(0) = 0,$$

$$vy(0) = 0$$

Results

At t=1.5s:

Software	MapleSoft	Maplesoft	Mathematica
Method	$rkf45\_dae$	mebdfi	bdf
x(t)	−0.885210671853245	−0.885210686075761144	−0.877704
y(t)	−0.465190354526072	−0.465190324459711979	−0.473984
vx(t)	1.40466937778533	1.40466780732217456	1.4137
vy(t)	−2.67294519503751	−2.67294226202269192	−2.67802
F(t)	−13.6765975574988	−13.6766315537367014	−13.9351

Table 3.5: Solution values of simple pendulum in cartesian coordinates at $t = 1.5$

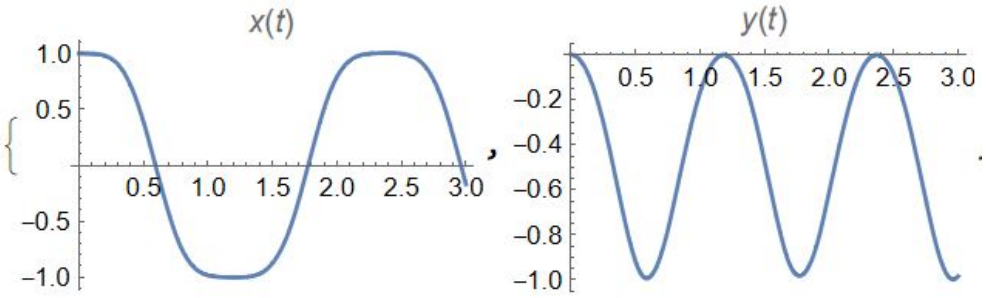


Figure 3.14: Solution plot of $x(t)$ and $y(t)$

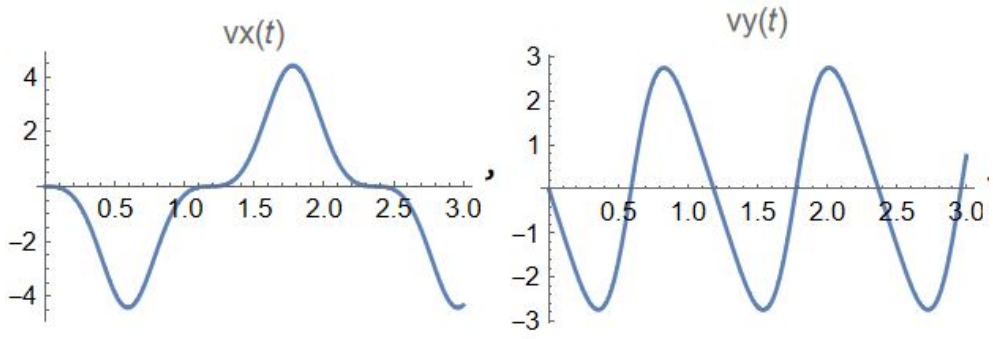


Figure 3.15: Solution plot of $vx(t)$ and $vy(t)$

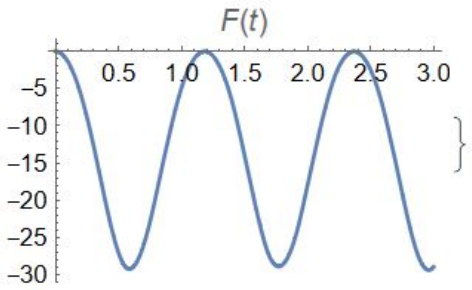


Figure 3.16: Solution plot of $F(t)$

3.6 Akzo-Nobel Chemical Reaction

The following system describes a chemical process in which two species, FLB and ZHU, are mixed and added constantly to the system. The chemical names are fictitious. The reaction equations are given below. Associated with each reaction, there are rates of reaction, which are specified. The algebraic constraint comes from the last reaction, which requires FLB and ZHU and its mixture to maintain a constant relationship.

$$2\,\mathrm{FLB} + \frac{1}{2}\mathrm{CO_2} \xrightarrow{k_1} \mathrm{FLBT} + \mathrm{H_2O}$$

$$\mathrm{ZLA} + \mathrm{FLB} \underset{k_2}{\overset{k_2/K}{\rightleftharpoons}} \mathrm{FLBT} + \mathrm{ZHU}$$

$$\mathrm{FLB} + 2\,\mathrm{ZHU} + \mathrm{CO_2} \xrightarrow{k_3} \mathrm{LB} + \mathrm{nitrate}$$

$$\mathrm{FLB.ZHU} + \frac{1}{2}\mathrm{CO_2} \xrightarrow{k_4} \mathrm{ZLA} + \mathrm{H_2O}$$

$$\mathrm{FLB} + \mathrm{ZHU} \rightleftharpoons \mathrm{FLB.ZHU}$$

$$\dot{FLB}(t) = -2r_1 + r_2 + r_3 + r_4 \tag{3.18a}$$

$$\dot{CO_2}(t) = -0.5r_1 - r_4 - 0.5r_5 + Fin \tag{3.18b}$$

$$\dot{FLBT}(t) = r_1 - r_2 + r_3 \tag{3.18c}$$

$$\dot{ZHU}(t) = -r_2 + r_3 - 2r_4, \tag{3.18d}$$

$$\dot{ZLA}(t) = r_2 - r_3 + r_5 \tag{3.18e}$$

Given the DAE (3.18) with constraints eqn,

$$k_s FLB(t) ZHU(t) = FLBZHU(t)$$

,variables:

$$Fin = klA(pCO_2/H - CO_2(t));$$
$$r_1 = k_1 FLB(t)^2 CO_2(t)^{1/2};$$
$$r_2 = k_2 FLBT(t) \times ZHU(t);$$
$$r_3 = (k_2/KK)FLB(t) \times ZLA(t);$$
$$r_4 = k_3 FLB(t) ZHU(t)^2 CO_2(t);$$
$$r_5 = k_4 FLB(t) \times ZHU(t);$$

,constants :

$$k_1 = 18.7;$$
$$k_2 = 0.58;$$
$$k_3 = 0.09;$$
$$k_4 = 0.42;$$
$$KK = 34.4;$$
$$KlA = 3.3;$$
$$K_s = 115.83;$$
$$pCO_2 = 0.9;$$
$$H = 737;$$

Initial Conditions:

$$FLB(0) = 0.444,$$
$$CO_2(0) = 0.00123,$$
$$FLBT(0) = 0,$$
$$ZHU(0) = 0.007,$$
$$ZLA(0) = 0$$

Results

At t=150s:

Software	MapleSoft	Maplesoft	Mathematica
Method	$rkf45\_dae$	mebdfi	bdf
FLB(t)	0.006127111	0.006126999	0.00601173
ZHU(t)	0.000019876532	0.000019875429	0.0000207108
$CO_2(t)$	0.001198746	0.022644978	0.00121757
ZLA(t)	0.022654983	0.0061267899	0.0237892

Table 3.6: Solution values of Akzo-Nobel chemical reaction $t = 150$

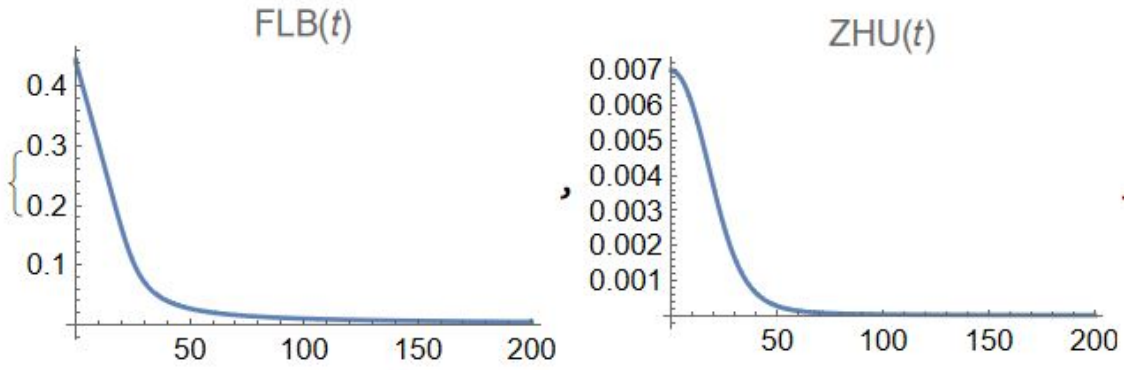


Figure 3.17: Solution plot of $FLB(t)$ and $ZHU(t)$

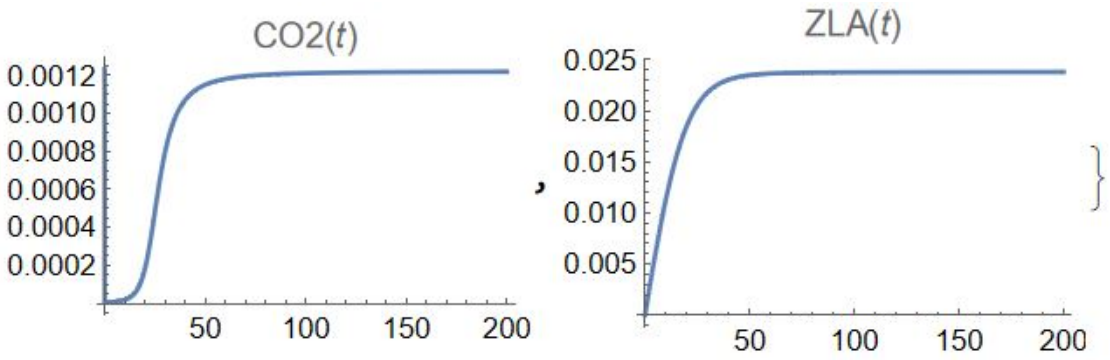


Figure 3.18: Solution plot of $CO_2(t)$ and $ZLA(t)$

Chapter 4

Kalman Filter for index-1 DAE

It is possible to convert a DAE into an ODE and apply a well known state estimation method. There are three problems with this approach [38]:

- Converting a DAE into an ODE and simulating it using an ODE solver can introduce significant errors.
- The updated estimates produced by the state estimator will not necessarily satisfy the algebraic constraints.
- If the DAE is reduced to a model containing only the differential states then measurements which are dependent on algebraic states may not be easily used.

This section explores semi-explicit index-1 DAE-compatible Kalman filter. A reminder of the of definition of semi-explicit index-1 DAE's is presented followed by a discussion on the filters presented with example. A semi-explicit index-1 DAE-compatible EKF was introduced in [99] but this filter is limited in that it can only process systems where no algebraic states are measured. This filter was expanded upon in [100], allowing measurement of algebraic variables. The formulation for semi-explicit index-1 DAE-compatible EKF is focussed of this

section.

Recall: a semi-explicit index-1 DAE has the form

$$\dot{x}_D = f_D(t, x_D, x_A), \tag{4.1a}$$

$$0 = f_A(t, x_D, x_A), \tag{4.1b}$$

where

$$\det\left(\frac{\partial f_A}{\partial x_A}\right) \neq 0$$

Differentiate equation (4.1b) with respect to t

$$0 = \frac{\partial f_A}{\partial t} + \frac{\partial f_A}{\partial x_D}\dot{x}_D + \frac{\partial f_A}{\partial x_A}\dot{x}_A \tag{4.2a}$$

$$\dot{x}_A = -\left(\frac{\partial f_A}{\partial x_A}\right)^{-1}\left(\frac{\partial f_A}{\partial t} + \frac{\partial f_A}{\partial x_D}\dot{x}_D\right) \tag{4.2b}$$

This is the formula for transforming a semi-explicit index-1 DAE into an ODE. This allows numerical methods for ODE's to be applied on a DAE.

Some define the index of a DAE as the number of differential operations needed to transform a DAE into an ODE, specifically this is referred to as the differential index. This is consistent with the index-1 in semi-explicit index-1 DAE as it can be transformed into an ODE with one differential operation as done in equation (4.2).

4.1 EKF for semi-explicit index-1 DAE

The EKF presented here was first introduced in [100] which is a modification of the EKF introduced in [38]. The EKF in [100] is semi-explcit index-1 DAE is compatible with systems where the measurement function is dependent on an algebraic state. This EKF propagates

the state estimate using a DAE solver and propagates the covariance by converting the DAE into an ODE and linearizing about the state estimate, then the matrix exponential of the Jacobian is obtained and the formula used is

$$P_{k+1|k} = AP_{k|k}A^T + Q$$

where P is the covariance, A is the matrix exponential of the Jacobian and Q is the process noise. The covariance has dimensions equal to the amount of the differential states.

The EKF introduced is able the handle semi-explicit index-1 DAE's where the measurement depends on algebraic states. The covariance has dimensions equal to the amount of differential states plus the amount of algebraic states.

Consider a semi-explicit index-1 DAE with noise and measurements taken at discrete intervals

$$\dot{x}_D = f_D(t, x_D, x_A) + w \tag{4.3a}$$

$$0 = f_A(t, x_D, x_A) \tag{4.3b}$$

$$y_k = h(t_k, x_{D,k}, x_{A,k}) + v_k \tag{4.3c}$$

where w and v_k are Gaussian noise with covariance matrices Q and R respectively. Notice that here we assume that noise is injected linearly into the system as opposed to the system used in the EKF.

Linearize (4.3a) and (4.3b) to obtain

$$\dot{x}_D = \begin{bmatrix} A_k & B_k \end{bmatrix} \begin{bmatrix} x_D \\ x_A \end{bmatrix} = A_k x_D + B_k x_A \tag{4.4a}$$

$$0 = \begin{bmatrix} C_k & D_k \end{bmatrix} \begin{bmatrix} x_D \\ x_A \end{bmatrix} = C_k x_D + D_k x_A \tag{4.4b}$$

where

$$\begin{bmatrix} A_k & B_k \\ C_k & D_k \end{bmatrix} = \begin{bmatrix} \frac{\partial f_D}{\partial x_A}(t_k, x_{D,k}, x_{A,k}) & \frac{\partial f_D}{\partial x_A}(t_k, x_{D,k}, x_{A,k}) \\ \frac{\partial f_A}{\partial x_D}(t_k, x_{D,k}, x_{A,k}) & \frac{\partial f_A}{\partial x_A}(t_k, x_{D,k}, x_{A,k}) \end{bmatrix}$$

Differentiate the linearized algebraic constraint (4.4b)

$$0 = C_k \dot{x}_D + D_k \dot{x}_A \tag{4.5a}$$

$$\dot{x}_A = -D_k^{-1} C_k \dot{x}_D \tag{4.5b}$$

$$\dot{x}_A = -D_k^{-1} C_k (A_k x_D + B_k x_A) \tag{4.5c}$$

$$\dot{x}_A = -D_k^{-1} C_k A_k x_D - D_k^{-1} C_k B_k x_A \tag{4.5d}$$

Writing this together with the linearized differential part (4.4a)

$$\begin{bmatrix} \dot{x}_D \\ \dot{x}_A \end{bmatrix} = \begin{bmatrix} A_k & B_k \\ -D_k^{-1} C_k A_k x_D & -D_k^{-1} C_k B_k \end{bmatrix} \begin{bmatrix} x_D \\ x_A \end{bmatrix}$$

Let

$$L_k = \begin{bmatrix} A_k & B_k \\ -D_k^{-1} C_k A_k & -D_k^{-1} C_k B_k \end{bmatrix}$$

and define the transition matrix as

$$\phi = exp(\triangle t_k L_k)$$

where $\triangle t_k$ is the size of a time step from t_{k-1} to t_k, i.e. $\triangle t_k = t_k - t_{k-1}$
Given state estimates and covariance $\hat{x}_{D,k-1|k-1}$, $\hat{x}_{A,k-1|k-1}$, $P_{k-1|k-1} \in \mathbb{R}^{(n_D+n_A)\times(n_D+n_A)}$ we wish to progress the estimate from time t_{k-1} to t_k,At time t_k, measurement y_k is available. Below is the process for progressing according to EKF method introduced.

1. Predict phase:

   (a) Propagate the state estimate from time t_{k-1} to t_k with $\hat{x}_{D,k-1|k-1}$ and $\hat{x}_{A,k-1|k-1}$ as the initial conditions using a DAE solver such as Matlab's ode15s, Mathematica or MapleSoft. Obtain $\hat{x}_{D,k|k-1}$ and $\hat{x}_{A,k|k-1}$

   (b) Obtain new covariance,

$$P_{k|k-1} = \phi_k P_{k-1|k-1} \phi_k^T + \begin{bmatrix} I_{n_D \times n_D} \\ -D_k^{-1} C_k \end{bmatrix} Q \begin{bmatrix} I_{n_D \times n_D} \\ -D_k^{-1} C_k \end{bmatrix}^T$$

   where I is an identity matrix and an element of $\mathbb{R}^{(n_D \times n_D)}$.

2. Update phase:

   (a) Compute the Kalman gain,

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R)^{-1}$$

   where $H_k = \begin{bmatrix} \frac{\partial h}{\partial x_D}(t_k, x_{D,k|k-1}, x_{A,k|k-1}) & \frac{\partial h}{\partial x_A}(t_k, x_{D,k|k-1}, x_{A,k|k-1}) \end{bmatrix}$ is the linearized measurement at time t_k

(b) Update the state estimate,

$$\begin{bmatrix} \hat{x}_{D,k|k} \\ \hat{x}_{A,k|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{D,k|k-1} \\ \hat{x}_{A,k|k-1} \end{bmatrix} + K_k(y_k - h(t_k, \hat{x}_{D,k|k-1}, \hat{x}_{A,k|k-1})).$$

(c) Solve

$$0 = f_A(t_k, \hat{x}_{D,k|k}, \hat{x}_{A,k|k})$$

for $\hat{x}_{A,k|k}$ to obtain a consistent updated algebraic state.

(d) Calculate the updated covariance,

$$P_{k|k} = (I_{n_D \times n_D} - K_k H_k) P_{k|k-1}$$

We now have the next state estimates $x_{D,k|k}$ and $x_{A,k|k}$ and covariance matrix $P_{k|k}$.

4.2 Example: Non linear Vehicle Tracking Problem

Let us consider a simple example to illustrate the efficacy of the constrained Kalman filter. The dynamics of the system is given by the equations as:

$$\dot{x_1} = x_1 + 3x_3 + w_k \tag{4.7a}$$

$$\dot{x_2} = x_2 + 3x_4 + w_k \tag{4.7b}$$

$$\dot{x_3} = x_3 + 2.59 + w_k \tag{4.7c}$$

$$\dot{x_4} = x_4 + 1.5 + w_k \tag{4.7d}$$

$$y = \begin{bmatrix} (x_1 - r_{n1})^2 + x_2 - r_{e1})^2) \\ (x_1 - r_{n2})^2 + x_2 - r_{e2})^2) \end{bmatrix} + e_k$$

With algebraic constraints

$$x_1 - 1.732x_2 = 0 \tag{4.8a}$$

$$x_3 - 1.732x_4 = 0 \tag{4.8b}$$

Substituting the algebraic constraints in equation 4.7 gives

$$\dot{x_1} = x_2 1.732 + 3x_3 \tag{4.9a}$$

$$\dot{x_2} = x_4 1.732 + 3x_4 \tag{4.9b}$$

$$\dot{x_3} = x_3 + 2.59 \tag{4.9c}$$

$$\dot{x_4} = x_4 + 1.5 \tag{4.9d}$$

where w represents process disturbances and e represents measurement errors, and u is the commanded acceleration. T is the sample period of the position estimator. The dynamic equations are linear but the measurement equations are nonlinear, so the extended Kalman filter can be used to estimate the state vector. The navigation reference points are at (0,0) and (173210,100000) meters, while the covariances of the process and measurement noise are

$$Q = Daig(4, 4, 1, 1) \tag{4.10a}$$

$$R = Diag(900, 900) \tag{4.10b}$$

We can use a Kalman filter to estimate the position of the vehicle. During certain times the vehicle may be travelling off-road, or on an unknown road, in which case the problem is unconstrained. At other times it may be known that the vehicle is travelling on a given road, in which case the state estimation problem is constrained. For instance, if it is known that the vehicle is travelling on a road with a constant heading angle. The sample period T is 3s and the heading is set to a constant angle of 60 degee. The commanded acceleration is alternately set to ± 1, as if the vehicle was alternately accelerating and decelerating in traffic. Note that with the 60 deg heading angle, the vehicle and the two reference points form a straight line, which makes the state estimation problem more difficult. The initial conditions are set to

$$x = \hat{x}_0 = [0 \quad 0 \quad 17 \quad 10]^T \tag{4.11a}$$

$$P_0 = Diag[900 \quad 900 \quad 4 \quad 4]^T \tag{4.11b}$$

The unconstrained and constrained Kalman filters were simulated using MATLAB for 300s. The figures show typical simulation results. The Fig. 4.1 shows the true position of the vehicle which in unconstrained.

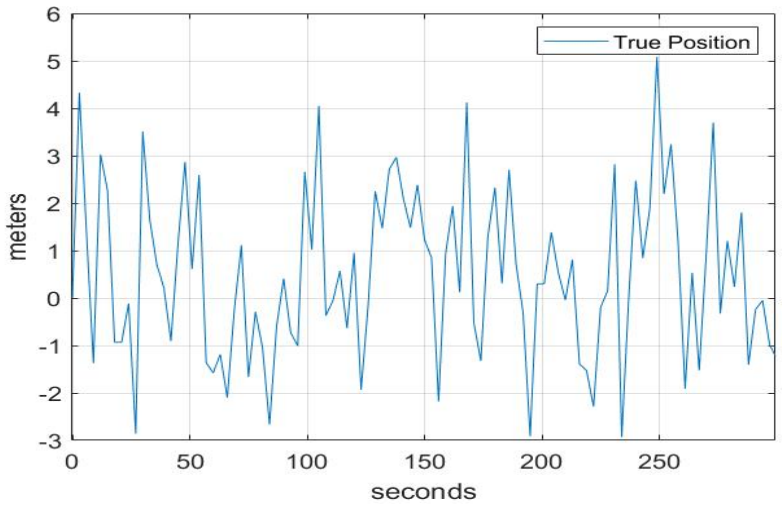


Figure 4.1: True position of the vehicle

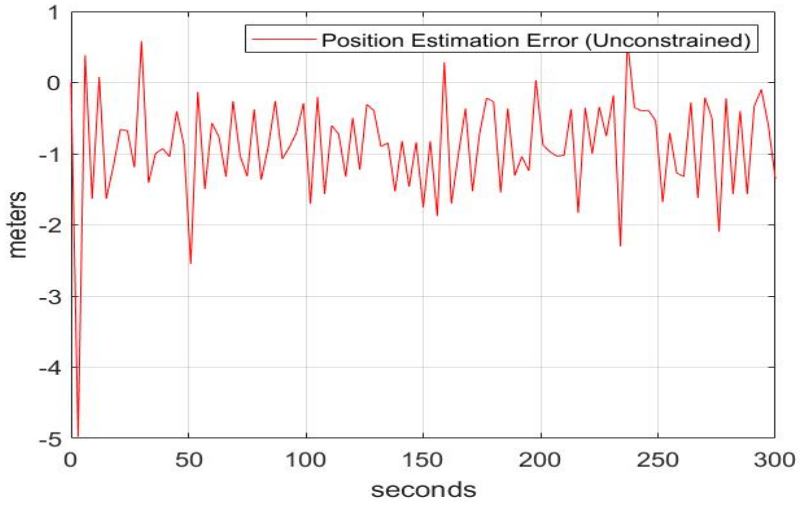


Figure 4.2: Unconstrained filter position error

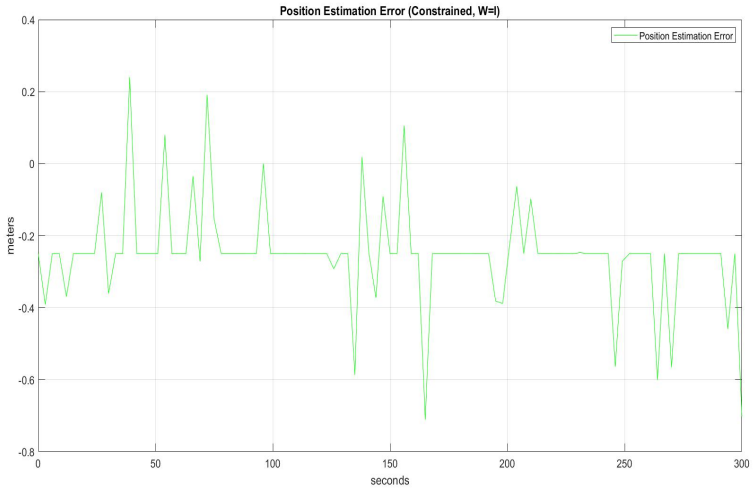


Figure 4.3: Constrained filter position error

Fig. 4.2 shows the position estimation error of the unconstrained Kalman filter, and Fig. 4.3 shows the position estimation error of the constrained Kalman filter. It can be seen that the constrained filter results in much more accurate estimates than the unconstrained filter. The unconstrained filter results in average position errors of about 5.08 m, while the constrained filter results in position errors of about 0.5 m. In addition, for this particular example, the Kalman filter performs identically whether we incorporate state constraints using perfect measurements, or whether we incorporate state constraints using the approach W = I or $W = \sum^1$.

Chapter 5

Conclusions

In this thesis, initially we presented a detailed review of literature on differential algebraic equations with Mathematical structure, index and solvability concepts. Filtering of DAEs have also been explored for linear and non linear DAE system. Some real time applications of DAEs with numerical example has been shown.

Finding numerical solutions of DAEs have always been a challenging task especially with constrains. The use of Professional softwares for solving complex DAEs are investigated along with different integration algorithms that are discussed in this thesis.

Comparison of professional software with various integration algorithms has been done by considering DAE systems of different index. The systems are simulated in specific time domains and its corresponding output has been plotted.

We have observed that IVP and index order reduction has been found challenging in MapleSolf as compared to Mathematica. The Mathematica software has proven to be better in finding the solutions of DAE systems irrespective of initial conditions. Mathematica can also find the initial conditions of the DAE system if needed . The plot commands in Mathematica are more reasonable and easy to plot with the software user interface as

compared to the MapleSoft. In its core algorithms and in most technical application areas, Mathematica offers a deeper, more complete and better integrated collection of features. Also, Mathematica is good at handling numerical work and it is a perfect programming system whereas Matlab is not a perfect programming system. Mathematica is good for handling calculus and differential differential-algebraic equations whereas Matlab is good in design functions. To conclude, if dealing with analytical solution, its better to use Maple, instead if interested in numerical, better to go with Mathematica.

Lastly, Kalman filtering of linear and nonlinear DAEs are studied with real time example. Method of incorporating linear algebraic constraint in kalman filter demonstrates the effectiveness of this method. For nonlinear complex DAE sytem, EKF filtering method gives best filtering estimates of the system. The EKF methods used in the example as described in chapter 4 gives better convergence results.

5.1 Future work

The filtering aspect of the DAE system can be further explored. If the system has nonlinear constraints, even if they are linearized they may result in convergence problem, which can be looked upon. Different filtering methods such as Unscented Kalman filter and Particle filter can also be explored for nonlinear DAE system with linear and nonlinear algebraic constraints. Future work along with the comparison of different filtering methods [101] [102] and adding nonlinear constraints [103] to higher index DAE system could be studied.

Appendix A

Code of 3.1 Pendulum System

A.1 MapleSoft:

dsys := $\{x'' = -2\lambda x, y'' = -2\lambda y - \pi^2, x^2 + y^2 = L^2\}$

ICs:=$\{x(0) = 0, y(0) = -1, x'(0) = 1/10, y'(0) = 0, \lambda(0) = \frac{1}{200} + \frac{\pi^2}{2}\}$

dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0)

(or)

dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0)

A.2 Mathematica:

```
eqa = {x''[t] + \[Lambda][t]*x[t] == 0,
   y''[t] + \[Lambda][t]*y[t] == -1, x[t]^2 + y[t]^2 == 1};
   ic = {x[0] == 0, y[0] == -1};
   NDSolve[{eqa, ic}, {x, y, \[Lambda]}, {t, 0, 3000},
 Method -> {"IndexReduction" -> "Pantelides",
```

```
"EquationSimplification" -> "Residual"}]
sol = NDSolve[{eqa, ic}, {x, y, \[Lambda]}, {t, 0, 3000},
 Method -> {"IndexReduction" -> "Pantelides"}]
 Plot[Evaluate[#[t] /. sol], {t, 0, 50}, PlotRange -> All,
PlotLabel -> #[t]] & /@ {x, y, \[Lambda]}
```

Appendix B

Code of 3.2 The Double Pendulum

B.1 MapleSoft:

dsys := $\{x_1'' + 2\lambda_1 x_1(t) + 2\lambda_2(x_1 - x_2) = 0, y_1'' + 9.8 + 2\lambda_1 y_1 + 2\lambda_2(y_1 - y_2) = 0,$
$x_2'' - 2\lambda_2(x_1 - x_2) = 0, y_2'' + 9.8 - 2\lambda_2(y_1 - y_2) = 0, x_1^2 + y_1^2 = 1, (x_1 - x_2)^2 + (y_1 - y_2)^2 = 0.5^2\}$
ICs:=$\{x_1(0) = 0, x_1'(0) = -52, y_1(0) = -1, y_1'(0) = 0, x_2(0) = 0, x_2'(0) = 3, y_2(0) = -1.5, y_2'(0) = 0\}$
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0)
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0)

B.2 Mathematica:

```
cpdae = {x1''[t] + 2*\[Lambda]1[t]*x1[t] +
      2*\[Lambda]2[t]*(x1[t] - x2[t]) == 0,
   y1''[t] + 9.8 + 2*\[Lambda]1[t]*y1[t] +
      2*\[Lambda]2[t]*(y1[t] - y2[t]) == 0,
```

```
    x2''[t] - 2*\[Lambda]2[t]*(x1[t] - x2[t]) == 0,
    y2''[t] + 9.8 - 2*\[Lambda]2[t]*(y1[t] - y2[t]) == 0,
    x1[t]^2 + y1[t]^2 == 1,
    (x1[t] - x2[t])^2 + (y1[t] - y2[t])^2 ==
     1/4};
cpinit = {x1[0] == 0, x1'[0] == -5/2, y1[0] == -1, y1'[0] == 0,
    x2[0] == 0, x2'[0] == 3, y2[0] == -5/2, y2'[0] == 0};
var = {x1[t], x2[t], y1[t], y2[t], \[Lambda]1[t], \[Lambda]2[t],
    x1'[t], x2'[t], y1'[t], y2'[t] };
    NDSolve[{cpdae, cpinit}, var, {t, 0, 5},
  Method -> {"IndexReduction" -> "Pantelides"}];
  solS2 = NDSolve[{cpdae, cpinit}, var, {t, 0, 5},
  Method -> {"IndexReduction" -> Automatic}]
  solP0 = NDSolve[{cpdae, cpinit}, var, {t, 0, 5},
  Method -> {"IndexReduction" -> {"Pantelides", "IndexGoal" -> 0},
    "EquationSimplification" -> "Residual"}]
    Plot[Evaluate[#[t] /. solP0], {t, 0, 5}, PlotRange -> All,
    PlotLabel -> #[t]] & /@ {x1, x2, y1, y2, x1', x2', y1',
  y2', \[Lambda]1, \[Lambda]2}
```

Appendix C

Code of 3.3 Linear DAE System of Index-4

C.1 MapleSoft:

dsys := $\{\dot{x}(t) + \dot{y}(t) = y(t) - sin(t) + 2cos(t), x(t) + y(t) = -sin(t)\}$

ICs:=$\{y(0) = -3\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

C.2 Mathematica:

```
Clear[x1, x2, x3, x4, x5, x6, ics, vars];
eqns = ({{x1'[t] == Cos[t] - x2[t] + x3[t]}, {x2'[t] ==
       Cos[t] - x3[t] + x4[t]}, {x3'[t] ==
```

```
        Cos[t] - x4[t] + x5[t]}, {x4'[t] ==
        Cos[t] - x5[t] + x6[t]}, {x5'[t] ==
        Cos[t] + Sin[t] - x6[t]}, {0 ==
        x1[t] + x2[t] + x3[t] + x4[t] + x5[t] - 5*Sin[t]}});
ics = {x1[0] == x2[0] == x3[0] == x4[0] == x5[0] == x6[0] == 0};
vars = {x1, x2, x3, x4, x5, x6};
NDSolve[{eqns, ics}, vars, {t, 0, 4 Pi},
     Method -> {"IndexReduction" -> "StructuralMatrix"}] ;
     solSM0 = NDSolve[{eqns, ics}, vars, {t, 0, 4 Pi},
    Method -> {"IndexReduction" -> {"StructuralMatrix",
        "IndexGoal" -> 0}}];
        Row[Plot[Evaluate[(#[t] - Sin[t]) /. solSM0], {t, 0, 4 Pi},
          PlotLabel -> #, PlotRange -> All, ImageSize -> Small] & /@
  vars]
```

Appendix D

Code of 3.4 Linear DAE System

D.1 MapleSoft

dsys := $\{x_1' = cos(t) - x2 + x3, x_2' = cos(t) - x3 + x4, x_3' = cos(t) - x4 + x5, x_4' = cos(t) - x5 + x6,$
$x_5' = cos(t) + sin(t) - x6, x1 + x2 + x3 + x4 + x5 - 5sin(t) = 0\}$
ICs:=$\{x_1(0) = x_2(0) = x_3(0) = x_4(0) = x_5(0) = x_6(0) = 0\}$
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0)
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0)

D.2 Mathematica:

```
eqns = {x'[t] + y'[t] == y[t] - Sin[t] + 2 Cos[t],
    x[t] + y[t] == -Sin[t]};
ic = {y[0] == -3};
NDSolve[{eqns, ic}, {x, y}, {t, 0, 1},
```

```
Method -> {"IndexReduction" -> "Pantelides",
  "EquationSimplification" -> "Residual"}]
  sol = NDSolve[{eqns, ic}, {x, y}, {t, 0, 2 Pi},
   Method -> {"IndexReduction" -> "StructuralMatrix"}]
   Plot[Evaluate[#[t] /. sol], {t, 0, 2 Pi}, PlotRange -> All,
  PlotLabel -> #[t]] & /@ {x, y}
```

Appendix E

Code of 3.5 Robertson Problem

E.1 MapleSoft

dsys := $\{\dot{y_1}(t) = -0.04y_1 + 10^4 y_2 y_3, \dot{y_2} = 0.04y_1 - 10^4 y_2 y_3 - (3 * 10^7)y_2^2, \dot{y_3} = (3 * 10^7)y_2^2,$
$y_1 + y_2 + y_3 = 1\}$
ICs:=$\{y_1(0) = 1, y_2(0) = 0, y_3(0) = 0\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

E.2 Mathematica:

```
cpdae = {y1'[t] == -0.04*y1[t] + 10^4*y2[t]*y3[t],
   y2' [t] == 0.04*y1[t] - 10^4*y2[t]*y3[t] - (3*10^7)*y2[t]^2,
   1 == y1[t] + y2[t] + y3[t]};
cpinit = {y1[0] == 1, y2[0] == 0, y3[0] == 0};
```

```
var = {y1[t], y2[t], y3[t]};
sol = NDSolveValue[{cpdae, cpinit}, var, {t, 0, 100},
    Method -> {"EquationSimplification" -> "MassMatrix"}];
    ucsol = First[
    NDSolve[{cpdae, cpinit}, {y1, y2, y3}, {t, 0, 100},
     Method -> {"IndexReduction" -> {Automatic,
          "ConstraintMethod" -> None}}, MaxSteps -> \[Infinity]]];
```

E.3 Matlab 15s:

```
y0 = [1; 0; 0];
tspan = 0:1:100;
M = [1 0 0; 0 1 0; 0 0 0];
options = odeset('Mass',M);
[t,y] = ode15s(@robertsdae,tspan,y0,options);

y(:,2) = 1e4*y(:,2);
semilogx(t,y);
ylabel('1e4 * y(:,2)');
legend('y1','y2','y3')
title('Robertson DAE problem, solved by ODE15S');

function out = robertsdae(t,y)
out = [-0.04*y(1) + 1e4*y(2).*y(3)
    0.04*y(1) - 1e4*y(2).*y(3) - 3e7*y(2).^2
```

```
    y(1) + y(2) + y(3) - 1 ];
end
```

Appendix F

Code of 3.6 The Car Axis System

F.1 MapleSoft:

dsys := $\{\frac{\ddot{x_l}}{2000} = (\frac{1}{2ll} - 1)x_l + \lambda_1 x_b + 2\lambda_2(x_l - x_r), \frac{\ddot{y_l}}{2000} = (\frac{1}{2ll} - 1)y_l + \lambda_1 y_b + 2\lambda_2(y_l - y_r) - \frac{1}{2000},$
$\frac{\ddot{x_r}}{2000} = (\frac{1}{2lr} - 1)(x_r - x_b) - 2\lambda_2(x_l - x_r), \frac{\ddot{y_r}}{2000} = (\frac{1}{2lr} - 1)(y_r - y_b) - 2\lambda_2(y_l - y_r) - \frac{1}{2000},$
$x_l x_b + y_l y_b = 0, (x_l - x_r)^2 + (y_l - y_r)^2 = 1 y_b = \frac{sin(10t)}{10}, x_b = \sqrt{1 - y_b^2},$
$l_l = \sqrt{x_l^2 + y_l^2}, l_r = \sqrt{(x_r - x_b)^2 + (y_r - y_b)^2)}\}$
ICs:=$\{x_l(0) = 0, \dot{x_l}(0) = -12, y_l(0) = 12, \dot{y_l}(0) = 0, x_r(0) = 1, \dot{x_r}(0) = -12, y_r(0) = 12, \dot{y_r}(0) = 0,$
$\lambda_1(0) = 0, \lambda_2(0) = 0\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

F.2 Mathematica:

```
Clear[yB, xL, yL, xB, L, m, Lleft, Lright, \[Epsilon], eqns,
```

```
  ics, \[Omega], carParams, params, vars, carAxis, M];
  yB[t_] = h Sin[\[Omega] t];
  constraints = {xL[t] xB[t] + yL[t] yB[t] ==
        0, (xL[t] - xR[t])^2 + (yL[t] - yR[t])^2 == L^2};
        m = (\[Epsilon]^2 M)/2;
xB[t_] = Sqrt[L^2 - yB[t]^2];
Lleft = Sqrt[xL[t]^2 + yL[t]^2];
Lright = Sqrt[(xR[t] - xB[t])^2 + ( yR[t] - yB[t])^2];
eqns = ({
        {m xL''[t] == ((L0 - Lleft) xL[t])/
        Lleft + \[Lambda]1[t] xB[t] +
              2 \[Lambda]2[t] (xL[t] - xR[t])},
        {m yL''[t] == ((L0 - Lleft) yL[t])/
        Lleft  + \[Lambda]1[t] yB[t] +
              2 \[Lambda]2[t] (yL[t] - yR[t]) - m},
        {m xR''[t] == (L0 - Lright) (xR[t] - xB[t])/Lright -
              2 \[Lambda]2[t] (xL[t] - xR[t])},
        {m yR''[t] == (L0 - Lright) (yR[t] - yB[t])/Lright -
              2 \[Lambda]2[t] (yL[t] - yR[t]) - m}
       });
ics = {xL[0] == 0, yL[0] == 1/2, yR[0] == 1/2, yL'[0] == 0,
      yR'[0] == 0};
      carParams = {L -> 1, L0 -> 1/2,
      \[Epsilon] -> 10^-2, M -> 10,
      h -> 10^-1, \[Omega] -> 10};
```

```
vars = {xL, yL, xR, yR, \[Lambda]1,
\[Lambda]2, xL', yL', xR', yR'};
carAxis =
    NDSolve[{eqns, constraints, ics} /. carParams, vars,
    {t, 0, 3},
      Method -> {"IndexReduction" -> Automatic},
      AccuracyGoal -> 7] ;
      Plot[Evaluate[#[t] /. carAxis], {t, 0, 3}, PlotRange -> All,
   PlotLabel -> #[t]] & /@ {xL, xR, yL, yR,
   \[Lambda]1, \[Lambda]2,
  xL', xR', yL', yR'}
```

Appendix G

Code of 3.7 Example from Physical Chemistry

G.1 MapleSoft:

dsys := $\{\dot{a} = \frac{-a}{10} + 10^4 b.c, \dot{b} = \frac{a}{10} - 10^4 bc + 10^7 b^2, a + b + c = 1\}$

ICs:=$\{a(0) = 1, b(0) = 0\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

G.2 Mathematica:

```
cpdae = {a'[t] == -a[t]/10 + 10000*b[t]*c[t],
   b'[t] == a[t]/10 - 10000*b[t]*c[t] - 10000000*b[t]^2,
   a[t] + b[t] + c[t] == 1 };
```

```
cpinit = {a[0] == 1, b[0] == 0};
var = {a[t], b[t], c[t]};
NDSolve[{cpdae, cpinit}, var, {t, 0, 4000},
  Method -> {"IndexReduction" -> "Pantelides"}];
  solP0 = NDSolve[{cpdae, cpinit}, var, {t, 0, 4000},
  Method -> {"IndexReduction" -> {"Pantelides", "IndexGoal" -> 0},
    "EquationSimplification" -> "Residual"}]
    Plot[Evaluate[#[t] /. solS1], {t, 0, 4000}, PlotRange -> All,
   PlotLabel -> #[t]] & /@ {a, b, c}
```

Appendix H

Code of 3.8 Simple Pendulum in Cartesian Coordinates

H.1 MapleSoft:

dsys := $\{\dot{vx}(t) = F(t)x(t), \dot{vy}(t) = F(t)y(t) - g, \dot{x}(t) = vx(t), \dot{y}(t) = vy(t), x^2 + y^2 = 1\}$

ICs:=$\{x(0) = 1, y(0) = 0, vx(0) = 0, vy(0) = 0\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

H.2 Mathematica:

```
cpdae = {vx'[t] == F[t]*x[t], vy'[t] == F[t]*y[t] - 9.8,
   x'[t] == vx[t], y'[t] == vy[t], x[t]^2 + y[t]^2 == 1};
cpinit = {x[0] == 1, y[0] == 0, vx[0] == 0, vy[0] == 0};
```

```
var = {x[t], y[t], vx[t], vy[t], F[t]};
NDSolve[{cpdae, cpinit}, var, {t, 0, 3},
  Method -> {"IndexReduction" -> "Pantelides"}];
  solP0 = NDSolve[{cpdae, cpinit}, var, {t, 0, 3},
  Method -> {"IndexReduction" -> {"Pantelides", "IndexGoal" -> 0},
    "EquationSimplification" -> "Residual"}]
    Plot[Evaluate[#[t] /. solP0], {t, 0, 3}, PlotRange -> All,
   PlotLabel -> #[t]] & /@ {x, y, vx, vy, F}
```

Appendix I

Code of 3.9 Akzo-Nobel Chemical Reaction

I.1 MapleSoft:

dsys := $\{\dot{FLB}(t) = -2r_1 + r_2 + r_3 + r_4, \dot{CO_2}(t) = -0.5r_1 - r_4 - 0.5r_5 + Fin,$
$\dot{FLBT}(t) = r_1 - r_2 + r_3, \dot{ZHU}(t) = -r_2 + r_3 - 2r_4, \dot{ZLA}(t) = r_2 - r_3 + r_5,$
$k_s FLB(t)ZHU(t) = FLBZHU(t), Fin = klA(pCO_2/H - CO_2(t)), r_1 = k_1 FLB(t)^2 CO_2(t)^{1/2},$
$r_2 = k_2 FLBT(t) \times ZHU(t), r_3 = (k_2/KK)FLB(t) \times ZLA(t), r_4 = k_3 FLB(t)ZHU(t)^2 CO_2(t),$
$r_5 = k_4 FLB(t) \times ZHU(t), k_1 = 18.7, k_2 = 0.58, k_3 = 0.09, k_4 = 0.42, KK = 34.4, KlA = 3.3,$
$K_s = 115.83, pCO_2 = 0.9, H = 737\}$
ICs:=$\{FLB(0) = 0.444, CO_2(0) = 0.00123, FLBT(0) = 0, ZHU(0) = 0.007, ZLA(0) = 0\}$

```
dsolve(dsys, ICS, numeric,method=mebdfi,maxfun=0 )
(or)
dsolve(dsys, ICS, numeric,method=rkf45_dae,maxfun=0 )
```

I.2 Mathematica:

```
Fin = klA (pCO2/H - CO2[t]);
r1 = k1 FLB[t]^2 CO2[t]^(1/2);
r2 = k2 FLBT[t] ZHU[t];
r3 = (k2/KK) FLB[t] ZLA[t];
r4 = k3 FLB[t] ZHU[t]^2 CO2[t];
r5 = k4 FLB[t] ZHU[t];
params = {k1 -> 18.7, k2 -> 0.58, k3 -> 0.09, k4 -> 0.42,
KK -> 34.4,
    klA -> 3.3, Ks -> 115.83, pCO2 -> 0.9, H -> 737};
    eqns = {FLB'[t] == -2 r1 + r2 - r3 - r4,
    CO2'[t] == -0.5 r1 - r4 - 0.5 r5 + Fin,
    FLBT'[t] == r1 - r2 + r3,
    ZHU'[t] == -r2 + r3 - 2 r4, ZLA'[t] == r2 - r3 + r5};
eqEqn = Ks FLB[t] ZHU[t] == FLBZHU[t];
ic = {FLB[0] == 0.444, CO2[0] == 0.00123, FLBT[0] == 0,
    ZHU[0] == 0.007, ZLA[0] == 0};
    sol = NDSolve[{eqns, eqEqn, ic} /. params,
    {FLB, ZHU, , CO2, ZLA},{t, 0, 200}];
Plot[Evaluate[#[t] /. sol], {t, 0, 200}, PlotRange -> All,
        PlotLabel -> #[t]] & /@ {FLB, ZHU, CO2, ZLA}
```

Bibliography

[1] P. Kunkel and V. Mehrmann., "Differential-algebraic equations - analysis and numerical solution," *EMS Publishing House, Zürich*, 2006.

[2] K. Weierstraß, *Über ein die homogenen Funktionen zweiten Grades betreffendes Theorem, nebst Anwendung desselben auf die Theorie der kleinen Schwingungen.* Monatsh. Akad. derWissensch., Berlin, 1858.

[3] ——, *Zur Theorie der bilinearen quadratischen Formen.* Monatsh. Akad. der Wissensch., Berlin, 1867.

[4] L. Kronecker, *Algebraische Reduction der Schaaren bilinearer Formen.* Sitzungsber. Akad. derWiss., Berlin, 1890.

[5] C. W. Gear, *The simultaneous numerical solution of differential-algebraic equations.* IEEE Trans. Circ. Theor., CT-18:89–95, 1971.

[6] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations.* SIAM Publications, Philadelphia, PA, 2nd edition, 1996.

[7] E. Griepentrog and R. März, *Differential-Algebraic Equations and their Numerical Treatment.* Teubner Verlag, Leipzig, 1986.

[8] E. Hairer, C. Lubich, and M. Roche, *The Numerical Solution ofDifferential-Algebraic Systems by Runge-Kutta Methods.* Springer-Verlag, Berlin, 1989.

[9] A. Elsevier Publications, *Theoretical and Numerical Analysis of Differential Algebraic Equations, volume VIII of Handbook of Numerical Analysis.* SIAM Publications, Philadelphia, PA, 2002.

[10] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential and Differential-Algebraic Equations.* SIAM Publications, Philadelphia, PA, 1998.

[11] S. L. Campbell, *Singular Systems of Differential Equations.* I. Pitman, San Francisco, CA, 1980.

[12] ——, *Singular Systems of Differential Equations II.* Pitman, San Francisco, CA, 1982.

[13] P. Deuflhard and F. Bornemann, *Scientific Computing with Ordinary Differential Equations.* Springer-Verlag, NewYork,NY, 2002.

[14] Teubner Verlag and Stuttgart, *Numerical Methods in Multibody Systems.* ACM Trans. Math. Software, 14:18–32, 1998.

[15] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Springer-Verlag, Berlin, 2nd edition, 1996.

[16] P. J. Rabier and W. C. Rheinboldt, *Non-holonomic Motion of RigidMechanical Systems from a DAE Viewpoint.* SIAM Publications, Philadelphia, PA, 2000.

[17] W. C. Rheinboldt, *Differential-algebraic systems as differential equations on manifolds.* Math. Comp., 43:473–482, 1984.

[18] P. Kunkel and V. Mehrmann, *Canonical forms for linear differential-algebraic equations with variable coefficients.* J. Comput. Appl. Math., 56:225–259, 1994.

[19] ——, *A new look at pencils of matrix valued functions.* Lin. Alg. Appl., 212/213:215–248, 1994.

[20] ——, *Regular solutions of nonlinear differential-algebraic equations and their numerical determination.* R. Numer. Math., 79:581–600, 1998.

[21] ——, *Analysis of over- and underdetermined nonlinear differential-algebraic systems with application to nonlinear control problems.* Math. Control, Signals, Sys., 14:233–256, 2001.

[22] ——, *Characterization ofclasses ofsingular linear differentialalgebraic equations.* Electr. J. Lin. Alg., 13:359–386, 2005.

[23] P. Kunkel, V. Mehrmann, and W. Rath, *Analysis and numerical solution of control problems in descriptor form.* Math. Control, Signals, Sys., 14:29–61, 2001.

[24] U. Ascher and L. Petzold, "Projected implicit runge–kutta methods for differential algebraic equations," *SIAM J. Numer. Anal 28.*, p. 1097–1120, 1991.

[25] F. R. Gantmacher, *The Theory of Matrices II.* Chelsea Publishing Company, New York, NY, 1959.

[26] C. W. Gear and L. R. Petzold, *Differential/algebraic systems and matrix pencils.* In B. Kågström and A. Ruhe, editors, Matrix Pencils, pages . Springer-Verlag, Berlin, 1983.

[27] S. L. Campbell and C. Gear, *The index of general nonlinear DAEs.* Numer. Math., 72:173–196, 1995.

[28] R. März, *The index of linear differential algebraic equations with properly stated leading terms.* Res. in Math., 42:308–338, 2002.

[29] ——, *Solvability of linear differential algebraic equations with properly stated leading terms.* Res. in Math., 45:88–105, 2004.

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

[31] C. W. Gear, *TDifferential-algebraic equation index transformations.* SIAMJ. Sci. Statist. Comput., 9:39–47, 1988.

[32] R. März, *Numerical methods for differential-algebraic equations I: Characterizing DAEs.* Seminarberichte 91-32/I, Fachbereich Mathematik, Humboldt-Universität zu Berlin, Berlin, Germany, 1991.

[33] ——, *Characterizing differential algebraic equations without the use of derivative arrays.* Computers Math. Appl., 50:1141–1156, 2005.

[34] V. Mehrmannand and C. Shi, *Analysis ofhigher order linear differential-algebraic systems.* Preprint 17/2004, Institut für Mathematik, TU Berlin, Berlin, Germany, 2004.

[35] G. Reißig, W. S. Martinson, , and P. I. Barton, *TDifferential-algebraic equations of index 1 may have an arbitrarily high structural index.* SIAM J. Sci. Comput., 21:1987–1990, 2000.

[36] P. Kunkel and V. Mehrmann, *Differential-Algebraic Equations Analysis and Numerical Solution.* European Mathematical Society, 2006.

[37] R. Lamour, R. Marz, and C. Tischendorf, *Differential-Algebraic Equations: A Projector Based Analysis*. Springer, 2013.

[38] B.Sherbak, "Implementation of kalman filtering for differential-algebraic equations," *Deaprtment of Applied Mathematics University of Waterloo, Ontario, Canada,*, pp. 59–61, 2020.

[39] C. Bendtsen and P. G. Thomsen, *Technical Report on Numerical Solution of Differential Algebraic Equations*. Department of Mathematical Modelling, Technical University of Denmark, 1999.

[40] S. A. K. A. Sa', "Thesis on numerical methods for solving differential algebraic equations," *An Najah National University*, 2010.

[41] S. Campbell, A. Ilchmann, V. Mehrmann, and T. Reis, *Applications of Differential-Algebraic Equations: Examples and Benchmarks*. Springer, 2019.

[42] Wikipedia, "Runge–kutta methods," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

[43] ——, "Collocation method," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Collocation_method

[44] ——, "Segregated runge–kutta methods," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Segregated_Runge%E2%80%93Kutta_methods

[45] ——, "Backward differentiation formula," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Backward_differentiation_formula

[46] C. Lubich, U. Nowak, U. Pöhle, and C. Engstler, *MEXX – numerical software for the integration of constrained mechanical multibody systems.* Preprint SC 92-12, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 1992.

[47] R. von Schwerin and M.Winckler, *A guide to the integrator library MBSSIM-version 1.00.* Technical report, IWR, Universität Heidelberg, Heidelberg, Germany, 1997.

[48] C. Führer, *Differential-algebraische Gleichungssysteme in mechanischen Mehrkörpersystemen.* Dissertation, Mathematisches Institut, TU München, München, Germany, 1998.

[49] A. Steinbrecher, *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems.* Dissertation, Institut für Mathematik, TU Berlin, Berlin, Germany„ 2006.

[50] B. Simeon, *MBSPACK—numerical integration software for constrained mechanical motion.* Surv. Math. Ind., 5:169–202, 1995.

[51] U. M. Ascher and R. J. Spiteri, *Collocation software for boundary value differential algebraic equations.* SIAM J. Sci. Comput., 15:938–952, 1994.

[52] L. R. Petzold, "A description of dassl: A differential/algebraic system solver," *in IMACS Trans. Scientific Computing, R. S. Stepleman et al., eds., North-Holland, Amsterdam*, vol. 1, pp. 65–68, 1993.

[53] F. Iavernaro and F. Mazzia, *Block-boundary value methods for the solution ofordinary differential equation.* SIAM J. Sci. Comput.21:323–339, 1998.

[54] J. R. Cash and S. Cosidine, *A MEBDF code for stiff initial value problems.* ACM Trans. Math. Software, 18:142–158, 1992.

[55] W. M. Lioen, J. J. B. de Swart, and W. A. van der Veen, *PSIDE users' guide*. Report MAS-R9834, CWI, Amsterdam, 1998.

[56] S. Hammerling and R. J. Hanson, *Algorithm 656: An extended set of FORTRAN basic linear algebra subprograms*. ACM Trans. Math. Software, 14:18–32, 1988.

[57] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for FORTRAN usage*. CM Trans. Math. Software, 5:308–323, 1979.

[58] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, S. Ostrouchov, and D. Sorenson, *LAPACK Users' Guide*. SIAM Publications, Philadelphia, PA, 2nd edition, 1995.

[59] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D.Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. SIAM Publications, Philadelphia, PA, 1994.

[60] U. Nowak and L. Weimann, *A Family of Newton Codes for Systems of Highly Nonlinear Equations — Algorithm, Implementation, Application*. Report TR 90-11, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 1990.

[61] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin, 2nd edition, 1993.

[62] ., "Differencial algebraic equations in maplesoft," *MapleSoft*, 2021. [Online]. Available: https://www.maplesoft.com/support/help/errors/view.aspx?path=MaplePortal/DAE

[63] ——, "Differencial algebraic equations using maplesoft dsolve/numeric/dae," *MapleSoft*, 2021. [Online]. Available: https://www.maplesoft.com/support/help/Maple/view.aspx?path=dsolve/numeric/DAE

[64] ——, "Odeplot in maplesoft," *MapleSoft*, 2021. [Online]. Available: https://www.maplesoft.com/support/help/maple/view.aspx?path=plots%2Fodeplot

[65] ——, "Introduction to numerical differential-algebraic equation solvers," *MapleSoft*, 2021. [Online]. Available: https://www.maplesoft.com/support/help/Maple/view.aspx?path=examples/numeric_DAE&cid=260

[66] R.RaghuVeer, "Comparison of simulation techniques and filtering methods for linear differential algebraic systems," *McGill University Thesis*, 2021.

[67] Wikipedia, "Wolfram mathematica." [Online]. Available: https://en.wikipedia.org/wiki/Wolfram_Mathematica

[68] .., "Numerical solution of differential-algebraic equations using wolfram language and system." [Online]. Available: https://reference.wolfram.com/language/tutorial/NDSolveDAE.html

[69] W. Research, "Ndsolve," *Mathematica Documentation Center*, 2019. [Online]. Available: https://reference.wolfram.com/language/ref/NDSolve.html

[70] ——, "Ida method for ndsolve," *Mathematica Documentation Center*, 2019. [Online]. Available: https://reference.wolfram.com/language/tutorial/NDSolveIDAMethod.html

[71] L. N. Laboratory, "Sundials: Suite of nonlinear and differential/algebraic equation solvers." [Online]. Available: https://computing.llnl.gov/projects/sundials

[72] A. Hindmarsh and A. Taylor, "User documentation for ida: A differential-algebraic equation solver for sequential and parallel computers," *Lawrence Livermore National Laboratory report, UCRL-MA-136910,*, December 1999.

[73] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, "Using krylov methods in the solution of large-scale differential-algebraic systems," *SIAM J. Sci. Comput.*, no. 15, p. 1467–1488, December 1994.

[74] W. Research, "Linearsolve," *Mathematica Documentation Center*, 2019. [Online]. Available: https://reference.wolfram.com/language/ref/LinearSolve.html

[75] ——, "State-space method for daes," *Mathematica Documentation Center*, 2021. [Online]. Available: https://reference.wolfram.com/language/tutorial/NDSolveStateSpace.html

[76] ——, ""projection" method for ndsolve," *Mathematica Documentation Center*, 2021. [Online]. Available: https://reference.wolfram.com/language/tutorial/NDSolveProjection.html

[77] E. Hairer and G. Wanner, "Solving ordinary differential equations ii: Stiff and differential-algebraic problems." *Springer-Verlag*, vol. 2nd ed., 1996.

[78] E. Hairer, C. Lubich, and G. Wanner, "Geometric numerical integration: Structure-preserving algorithms for ordinary differential equations." *Springer-Verlag*, 2002.

[79] E. Hairer, "Symmetric projection methods for differential equations on manifolds." *BIT 40*, no. 4, p. 726–734, 2000.

[80] C. Pantelides, "The consistent initialization of differential-algebraic equations." *SIAM J Sci. Stat. Comput. 9*, no. 2, p. 213–231, 1991.

[81] C.Pantelides, "The consistent initialization of differential-algebraic equations," *SIAM J Sci. Stat. Comput.*, vol. 9, no. 2, p. 213–231, 200.

[82] S. Mattson and G. Söderlind, "Index reduction in differential-algebraic equations using dummy derivatives." *SIAM J. Sci Comput. 14*, no. 3, p. 677–692, 1991.

[83] P. Kunkel, V. Mehrmann, and I. Seufer, "Genda: A software package for the solution of general nonlinear differential-algebraic equations," *Institut für Mathematik, Technische Universität Berlin*, no. 730-02, 2002.

[84] P. Kunkel, V. Mehrmann, W. Rath, and J. Weickert, "Gelda: A software package for the solution of general linear differential algebraic equations," *SIAM Journal Scientific Computing*, vol. 18, pp. 115–138, 1997.

[85] S. L. Campbell, "Comment on controlling generalized state-space (descriptor) systems," *Internat. J. Control*, vol. 18, no. 46, pp. 2229–2230, 1987.

[86] ——, "Nonregular descriptor systems with delays," *in Proc. Symp. Implicit and Nonlinear Systems, Dallas*, pp. 275–281, 1992.

[87] C. W. Gear, "Differential-algebraic equations index transformations," *SIAM J. Sci. Statist. Comput.*, vol. 9, pp. 39–47, 1988.

[88] P.Kunkel and V.Mehrmann, "Differential-algebraic equations - analysis and numerical solution," *EMS Publishing House, Zürich*, 2006.

[89] P.Deuflhard, E. Hairer, and J. Zugck, "One step and extrapolation methods for differential-algebraic systems," *Numer. Math.*, no. 51, pp. 501–516, 1987.

[90] E. Hairer, C.Lubich, and M. Roche, "The numerical solution of differential-algebraic systems by runge-kutta methods, lecture notes in mathematics„" *Springer-Verlag, Berlin,*, no. 1409, 1989.

[91] E. Hairer and G.Wanner, "Solving ordinary differential equations ii," *Springer-Verlag, Berlin*, 1991.

[92] P.Kunkel and V.Mehrmann, "Numerical solution of differential algebraic riccati equations," *Linear Algebra Appl.*, no. 137/138, pp. 39–66., 1990.

[93] ——, "Smooth factorizations of matrix valued functions and their derivatives," *Numer. Math.*, no. 60, pp. 115–132, 1991.

[94] V.Mehrmann, "The autonomous linear quadratic control problem," *Springer-Verlag, Berlin*, 1991.

[95] P.Kunkel and V.Mehrmann, "A new class of discretization methods for the solution of linear differential-algebraic equations with variable coefficients." *SIAM J. Numer. Anal.*, vol. 33, no. 5, pp. 1941–1961, 1996.

[96] L. R.Petzold, "A description of dassl: A differential/algebraic system solver," *in IMACS Trans. Scientific Computing*, vol. 1, R. S. Stepleman et al., eds., North-Holland, Amsterdam, pp. 65–68, 1993.

[97] K. E.Brenan, S. L.Campbell, and L. R. Petzold, "Numerical solution of initial-value problems in differential algebtraic equations," *Elsevier, North Holland, New York, N.Y.*, 1989.

[98] Matlab, "Solve differencial algebraic equations using matlab," *Matlab Documentation*, 2021. [Online]. Available: https://www.mathworks.com/help/matlab/math/solve-differential-algebraic-equations-daes.html

[99] V.M.Becerra, P. Roberts, and G. Griffiths, "Applying the extended kalman filter to systems described by nonlinear differential-algebraic equations," *Control Engineering Practice*, vol. 3, no. 9, pp. 267–281, mar 2001.

[100] R.K.Mandela, R. Rengaswamy, S. Narasimhan, and L. Sridhar, "Recursive state estimation techniques for nonlinear differential algebraic systems," *Control Engineering Practice*, vol. 65(16), no. 6, pp. 4548–455, aug 2010.

[101] I.Alkov and D. Weidemann, "Unscented kalman filter for higher index nonlinear differential-algebraic equations," *19th International Conference on Methods and Models in Automation and Robotics*, pp. 88–93, 2014.

[102] Y.Puranik, V. A. Bavdekar, S. Patwardhan, and S. Shah, "An ensemble kalman filter for systems governed by differential algebraic equations (daes)," *8th IFAC Symposium on Advanced Control of Chemical Processes*, pp. 531–536, 2012.

[103] D.Simon and T. Chia, "Kalman filter with state equality constraints," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 1, pp. 133–135, 2002.