UNIVERSITY OF TRENTO

Department of Industrial Engineering

Doctoral School in Materials, Mechatronics and Systems Engineering

XXXVI *Cycle*

Symbolic Computation Methods for the Numerical Solution of Dynamic Systems Described by Differential-Algebraic Equations

Supervisors:
Prof. Enrico Bertolazzi
Prof. Francesco Biral

Ph.D. Candidate:
Davide Stocco

Academic Year 2022/2023

Davide Stocco: *Symbolic Computation Methods for the Numerical Solution of Dynamic Systems Described by Differential-Algebraic Equations* © July 2024

Supervisors:
Prof. Enrico Bertolazzi
Prof. Francesco Biral

Members of the Committee:
Prof. Francesca Mazzia
Prof. Matthias Gerdts
Prof. Luca Zaccarian

Date:
August 1, 2024

Location:
Department of Industrial Engineering, University of Trento, Povo, Trento, Italy

COPYRIGHT NOTICE

ABSTRACT

In modern engineering, the accurate and efficient numerical simulation of dynamic systems is crucial, providing valuable insights across various fields such as automotive, aerospace, robotics, and electrical engineering. These simulations help to understand system behaviors, to optimize performance, and to guide design decisions. Nonetheless, systems described by Ordinary Differential Equations (ODEs) and Differential-Algebraic Equations (DAEs) are central to such simulations. While ODEs can be easily solved, they often fall short of modeling systems with constraints or algebraic relationships. DAEs, however, offer a more comprehensive framework, making them suitable for a wider range of dynamic systems. However, the inherent complexity of DAEs poses significant challenges for numerical integration and solution.

In vehicle dynamics, the simulation of systems described by DAEs is particularly relevant. The advances in autonomous and high-performance cars rely heavily on robust simulations that accurately reflect the interactions between mechanical components, control systems, and environmental factors. Achieving accuracy and speed in these simulations is critical for Real-Time (RT) applications, where rapid decision-making and control are essential. The challenges faced in vehicle dynamics simulations, such as equations' stiffness and complexity, are representative of broader issues in dynamic system simulations.

This thesis addresses these challenges by integrating symbolic computation with numerical methods to solve DAEs efficiently and accurately. Specifically, the index reduction approach transforms high-index DAEs into low-index formulations more suitable for numerical integration, enhancing the speed and stability of solvers. Symbolic computation, which handles mathematical expressions in their exact form, aids this process by simplifying the involved expressions, detecting redundancies and symbolic cancellations, and thereby ensuring equations' consistency while keeping complexity at the minimum. Hence, combining symbolic and numerical methods leverages the strengths of both techniques, aiming at improved performance and reliability. Such a hybrid framework is designed to handle the specific requirements of vehicle dynamics and other applications in engineering.

The thesis encompasses several advancements in dynamic system simulation by integrating symbolic computation with numerical methods to reduce computational overhead and improve performance. The research focuses on developing new algorithms for DAEs index reduction, transforming high-index DAEs into more suitable for standard numerical integration methods. Specifically, such an index reduction process is based on symbolic matrix factorization with simultaneous expression swell mitigation. This novel methodology is validated through practical applications, applying the proposed technique to real-world simulation problems to assess its performance, accuracy, and efficiency. Additionally, the research aims to enhance Hard Real-Time (HRT) vehicle dynamic simulation by designing dedicated algorithms and models for simulating tire-road interactions and vehicle structures' deformation, improving both speed and fidelity. Altogether, this thesis introduces several open-source software libraries

made available to the research community with comprehensive documentation and examples.

In summary, this work bridges the gap between symbolic computation and numerical methods for the simulation of dynamic systems described by DAEs. Thanks to mixed symbolic-numeric frameworks, innovative algorithms, and practical tools, it contributes to the advancement of simulation techniques, setting the stage for further investigations and applications in engineering.

CONTENTS

LIST OF ABBREVIATIONS

AABB	Axis-Aligned Bounding Box
AD	Automatic Differentiation
ADAS	Advanced Driver-Assistance Systems
AI	Artificial Intelligence
API	Application Programming Interface
BC	Boundary Condition
BDF	Backward Differentiation Formula
BLAS	Basic Linear Algebra Subprograms
BSD	Berkeley Software Distribution
BVH	Bounding Volume Hierarchy
BVM	Bounding Value Method
BVP	Bounding Volume Primitive
CAS	Computer Algebra System
CERN	Conseil Européen pour la Recherche Nucléaire
CPU	Central Processing Unit
DAE	Differential-Algebraic Equation
DE	Discrete Element
DOF	Degree of Freedom
DIL	Driver In the Loop
DSM	Direct Stiffness Method
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
FFLU	Fraction-Free Lower-Upper
FSAE	Formula SAE
GPU	Graphics Processing Unit
GJ	Gauss-Jordan
GUI	Graphical User Interface
HIL	Hardware In the Loop

HRT	Hard Real-Time
IC	Initial Condition
IVP	Initial Value Problem
KKT	Karush-Kuhn-Tucker
LLNL	Lawrence Livermore National Laboratory
LAPACK	Linear Algebra PACKage
LU	Lower-Upper
MB	Multi-Body
MBD	Multi-Body Dynamics
MIT	Massachusetts Institute of Technology
MBS	Multi-Body System
MPL	Mathematical Pseudo-Language
NASA	National Aeronautics and Space Administration
NLP	nonlinear Programming
OC	Optimal Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
OS	Operative System
PDAE	Partial Differential-Algebraic Equation
PDE	Partial Differential Equation
RAM	Random Access Memory
RDF	Road Data Format
RK	Runge-Kutta
RKF	Runge-Kutta-Fehlberg
RT	Real-Time
RTF	Real-Time Factor
RMS	Root Mean Square
SA	Structural Analysis
SAE	Society of Automotive Engineers
SIL	Software In the Loop
SRT	Soft Real-Time
SUNDIALS	SUite of Nonlinear and DIfferential-ALgebraic equation Solvers
SVD	Singular Value Decomposition
TIRF	Tire Research Facility
TTC	Tire Test Consortium
TPPC	Trajectory Prescribed Path Control
UHD	Ultra High Definition

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In modern engineering practices, the numerical simulation of dynamic systems has become increasingly important, offering invaluable insights into the behavior of complex systems across diverse domains. From automotive engineering to aerospace, robotics, and electrical systems, numerical simulations are indispensable for understanding system behaviors, optimizing performance, and guiding design decisions. Central to many of these simulations are systems described by Ordinary Differential Equations (ODEs) and Differential-Algebraic Equations (DAEs). While ODEs are relatively straightforward to solve, they have limited applicability in modeling systems with constraints or algebraic relationships [9]. DAEs, on the other hand, provide a more elegant framework for modeling complex systems by combining differential equations with algebraic constraints. This versatility makes DAEs a powerful tool for modeling a wide range of dynamic systems. However, their solution poses significant challenges due to their inherent mixed differential and algebraic nature. Partial Differential Equations (PDEs) and Partial Differential-Algebraic Equations (PDAEs) are also used to model dynamic systems. The relationship between PDAEs and PDEs is analogous to the relationship between ODEs and DAEs. Unsurprisingly, the solution of PDAEs is also significantly challenging, requiring special discretization techniques to reduce the system to DAEs, for which more conventional solution methods can be employed [10]. Nevertheless, the focus of this research is on DAEs, which are widely used in engineering applications and present distinctive challenges in terms of numerical integration and solution.

The inherent complexity and possible stiffness of DAEs pose significant obstacles to achieving fast simulations [11–13]. Additionally, stiff systems, characterized by dis-

parate timescales among variables, demand numerical methods capable of handling rapid changes without sacrificing stability or accuracy. Traditional numerical techniques, like explicit solvers, often fail in these scenarios, necessitating the adoption of implicit methods that lead to higher computational costs [9, 14]. Within this context, symbolic computation are promising for overcoming the challenges posed by DAEs in dynamic system simulation. Unlike purely numerical approaches, symbolic computation operates on mathematical expressions in their exact form. Such a capability proves valuable for lowering the structural complexities of DAEs. Indeed, symbolic techniques can systematically reduce DAEs' index, transforming them into forms more appropriate for numerical integration. This process not only unwinds the complexity of the equations but also enhances the speed and stability of numerical solvers, thereby improving the overall performance of simulations [9, 14, 15].

A key advantage of symbolic computation lies in its capacity to derive exact or partial solutions for specific components of the system. These solutions can then be leveraged to enhance the efficiency of numerical solvers by generating optimized code or simplifying the equations before numerical integration. By combining symbolic and numerical methods, researchers can exploit the strengths of each approach, thereby achieving the best performance and reliability in simulation. However, this hybrid computational framework does not come without its challenges. Such integration requires careful consideration of the algorithms, data structures, and computational strategies to ensure an optimal implementation. Indeed, symbolic operations cannot always replace numerical methods, as they may be computationally expensive or impractical for large-scale systems. Vice versa, numerical methods may struggle with overly complex symbolic expressions that could be simplified through symbolic computation, hindering the overall efficiency of the simulation. In other words, symbolic computation and numerical methods are complementary tools that carry different information and must be judiciously combined to achieve the much-desired performance improvements [16, 17]. This research aims to close such a gap between symbolic computation and numerical methods, developing a comprehensive framework that leverages the strengths of both approaches to solve DAEs efficiently and accurately in dynamics simulations applied, but not limited, to vehicle dynamics.

Within the domain of vehicle simulation, the accurate and fast solution of systems described by DAEs holds paramount importance [11–13, 18, 19]. In industries like automotive, the development of Advanced Driver-Assistance Systems (ADAS), autonomous vehicles, and high-performance cars critically depends on robust simulations. These simulations must accurately account for the far from trivial relationships between mechanical components, control systems, and environmental factors such as driver inputs. As a consequence, vehicle dynamics simulations encounter a plethora of challenges, reflecting the complexity of real-world systems. The willingness to push the boundaries of vehicle performance, safety, and autonomous driving capabilities demands simulations that are not only accurate but also fast. This speed is crucial for Real-Time (RT) applications, where rapid decision-making and control are essential. Furthermore, Artificial Intelligence (AI) training and validation require extensive sim-

ulations. Thereby, faster simulations inherently lead to more efficient training and validation processes [20, 21]. Importantly, the challenges faced in vehicle dynamics simulations are not unique to this field but are representative of the broader challenges in dynamic system simulations. The need for effective simulations is a common theme across various engineering disciplines, highlighting the importance of developing advanced computational techniques.
A fast simulation does not come only from the numerical solver but also from the model's structure and the sub-models that compose it. For instance, in vehicle dynamics simulations, the tire-road interaction is a critical aspect that significantly impacts the overall performance and behavior of the vehicle [22]. The tire model's complexity and the computational cost associated with its simulation are crucial factors in determining the overall simulation speed. Another key aspect is the modeling of the vehicle's structure, which involves the simulation of flexible bodies and joints. The interaction between these components and the environment, such as the road surface, further complicates the simulation. Therefore, developing dedicated algorithms and models can significantly enhance the speed and accuracy of vehicle dynamics simulations [19]. In this regard, this research also aims to address these challenges by compounding the solutions of DAEs with tire-road interaction and vehicle structure deformation models suitable for Hard Real-Time (HRT) applications.

1.2 OBJECTIVES AND SIGNIFICANCE

The significance of this research lies in its impact on the simulation and analysis of complex dynamic systems, particularly in vehicle dynamics. By developing a hybrid computational framework that combines symbolic computation with numerical methods, this study addresses critical challenges associated with solving DAEs. The contributions have implications for academic and industrial applications, advancing the state of the art in dynamic system simulation. Specifically, the primary objectives of this thesis and their significance are the following.

- **Enhance existing numerical methods with symbolic computation methods** – This involves creating a robust framework combining numerical and symbolic methods to solve stiff high-index DAEs effectively. The framework aims to handle the complexities and specific requirements of simulations across various engineering disciplines. Indeed, traditional numerical methods often struggle with the complexity and stiffness of DAEs, leading to issues such as numerical instability. By integrating symbolic computation techniques, this study seeks to improve the simulation performances by preprocessing and simplifying equations, reducing computational overhead, providing exact solutions for sub-problems, and boosting numerical solvers' overall stability and performance. This results in more reliable and faster simulations, so crucial for advanced engineering systems' design and optimization.
- **Develop and implement new algorithms for DAEs index reduction** – The index

of a DAE serves as a measure of its complexity, with higher-index equations posing numerical challenges. The novel symbolic algorithms for index reduction presented in this thesis provide a systematic approach to transforming high-index DAEs into a form more suitable for standard numerical methods used with ODEs. The proposed algorithms are specifically designed to mitigate the expression swell phenomenon, a common issue in symbolic computation that can hinder the efficiency and scalability of the index reduction process. Besides expanding the applicability of numerical solvers to more complex systems, this also preserves the integrity and physical relevance of the original models. The ability to handle high-index DAEs effectively is particularly relevant in vehicle dynamics, robotics, and control systems, where such equations frequently arise.

- **Validate the proposed methods through real-world simulation scenarios** – Practical validation is crucial for demonstrating the effectiveness of the proposed methods. This objective involves applying the developed framework and algorithms to real-world simulation problems, such as the dynamics of rigid and flexible Multi-Body (MB) systems, Trajectory Prescribed Path Control (TPPC) problems, and electrical systems. The proposed methods' performance, accuracy, and computational efficiency are assessed in these contexts.
- **Develop sub-models and techniques for HRT vehicle dynamic simulations** – In addition to the core research objectives, this research aims to develop dedicated algorithms and models for simulating the tire-road interaction and the vehicle's structural elements. These sub-models and techniques are integrated into the overall simulation framework, enhancing the speed and accuracy of vehicle dynamics simulations. The goal is to provide a comprehensive solution for simulating complex vehicle systems in HRT applications.

In summary, this research aims to bridge the gap between symbolic computation and numerical methods for solving DAEs in vehicle simulations. Through the development of a hybrid framework, innovative algorithms, and practical tools, it contributes to the advancement of simulation techniques for complex dynamic systems. The validation through real-world applications underscores the practical significance and potential impact of the research. This thesis sets the stage for further exploration and application of symbolic computation in dynamic system simulation, offering new possibilities for modeling, analysis, and control in engineering.

1.3 CONTRIBUTIONS

Through the development of novel methodologies, algorithms, and software tools, this research makes significant contributions to the dynamic system simulation field. The primary contributions are listed below, citing for each one of them the related publications and open-source software libraries that have resulted from this study.

- **Symbolic matrix factorization with expression swell mitigation** [23, 24] – This

research has developed advanced symbolic matrix factorization techniques to simultaneously address the expression swell phenomenon, a common challenge in symbolic computation. Such techniques also mitigate expression swell by introducing improved symbolic pivoting strategies and hierarchical representation methods, enhancing computational efficiency and scalability. These symbolic techniques are implemented within the MAPLE® Computer Algebra System (CAS) on the LEM and LAST software libraries, providing a robust tool for preprocessing and simplifying complex equations before the numerical solution computation. This integration results in a framework apt to the symbolic index reduction algorithm and solution of linear systems.

- **A novel algorithm for DAEs index reduction** [2, 3, 25] – A new symbolic algorithm for reducing the index of DAEs is developed, transforming high-index DAEs into a form more suitable for numerical integration methods used with ODEs. This algorithm addresses the complexities associated with high-index DAEs, facilitating their numerical solution while preserving the physical relevance of the models. Such an algorithm is implemented in the INDIGO software library, available to the research community. This library provides tools for systematically transforming high-index DAEs, enhancing the applicability and efficiency of numerical solvers in various engineering applications.

- **Models for HRT tire-road interaction** [1, 4, 26, 27] – Specialized algorithms and models for simulating tire-road interactions are developed, focusing on the dynamics and physical modeling of tires. These models account for complex factors such as tire deformation and contact mechanics, providing accurate and RT simulations essential for vehicle dynamics studies. The tire-road interaction models are integrated into the overall simulation framework, improving the fidelity and performance of vehicle dynamics simulations. Such an integration supports advanced analyses and design optimizations in automotive engineering. The software libraries supporting the tire-road interaction modeling are ACME, for 3D geometry operations, and ENVE, for tire-ground enveloping. Another software library is also developed for tire physical modeling, but it is not yet available to the public.

- **Symbolic-numerical analysis and solution of structures through Direct Stiffness Method (DSM)** [6, 8, 28] – The research extends the DSM to the mixed symbolic-numerical analysis of structures. By leveraging symbolic computation, the DSM approach reproduced in this research enables efficient assembly and solution of large-scale structural systems, particularly useful in design optimization and parametric studies. A comprehensive software package supporting the symbolic-numerical analysis and solution of structures is developed. This package, named TRUSSME-FEM, integrates symbolic preprocessing with numerical solution techniques and facilitates the efficient simulation and optimization of complex structural systems.

In essence, the research has resulted in the development of several software libraries, including tools for symbolic matrix factorization, DAEs' index reduction, tire-road interaction modeling, and structural analysis using DSM. These libraries are designed to be user-friendly and accessible, promoting wider adoption and further development of advanced computational methods. The software libraries are made available to the community under the Berkeley Software Distribution (BSD) 3-Clause License, with detailed documentation and examples to facilitate their use in various applications. This open-access approach encourages collaboration and innovation in the field of dynamic system simulation.

1.4 THESIS OUTLINE

The main matter of this thesis is organized into several chapters, each dedicated to different aspects of the research objectives and contributions. The structure is designed to provide a comprehensive understanding of the challenges, methodologies, and results related to the numerical simulation of dynamic systems, particularly focusing on the integration of symbolic computation with numerical methods for the solution of DAEs. The software libraries supporting the index reduction and solution of implicit DAE system of the form $\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 0$ are available in [23–25]. The author also contributed to the development of other software libraries for the solution of ODEs and ODEs on manifolds of the form $\mathbf{M}(\mathbf{x}, t)\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$, with $\mathbf{M}(\mathbf{x}, t)$ a non-singular matrix. These are respectively available in [29, 30], but they are not covered in this thesis.

1. **Introduction** sets the stage for the thesis, introducing the research objectives, the significance of the problem, and the contributions of the study. It outlines the motivation behind exploring symbolic computation techniques in the context of dynamic system simulations, particularly but not limited to the case of vehicle dynamics. The introduction provides a high-level overview of the challenges associated with the efficient and accurate solution of dynamic systems in engineering applications. Furthermore, it outlines the challenges associated with solving DAEs and the potential benefits of the proposed hybrid computational framework. Notably, these chapters are based on the previous work presented in [2, 3], as well as in the forthcoming [7, 8].
2. **Brief on Differential-Algebraic Equations** introduces the fundamental concepts of DAEs, their significance, the different types of indices that are associated with them, and explores the Hessenberg forms. It also discusses various solution methods for DAEs, including numerical direct discretization methods, like the Radau collocation. All these concepts will be later used in the development of the symbolic index reduction algorithm. The chapter concludes with a presentation of the index reduction methods and software tools for DAEs that are currently available.
3. **Essential Concepts and Techniques of Symbolic Computation** provides a brief introduction to computer algebra, as well as its applications in solving mathe-

matical problems and engineering challenges. It begins with elementary concepts and progresses to commercial CASs, focusing on the MAPLE® language. This chapter also discusses the phenomenon of expression swell and strategies to mitigate it through the hierarchical representations of expressions. It further explores expression complexity metrics, and symbolic matrix factorization techniques, including full-pivoting Lower-Upper (LU) and Fraction-Free Lower-Upper (FFLU) factorizations. An improved symbolic pivoting strategy is introduced to mitigate expression swell and improve computational efficiency. The chapter concludes with a discussion on the implementation of symbolic linear algebra techniques in the MAPLE® environment.

4. **Solution of Dynamic Systems Described by Differential-Algebraic Equations** delves into the practical application of the theoretical and computational concepts discussed in previous chapters. Particularly, the chapter presents a novel symbolic index reduction algorithm for DAEs based on previously introduced symbolic matrix factorization techniques. The outlined algorithm is designed to reduce the index of generic first-order DAEs that are linear in the states' derivatives. Moreover, the chapter introduces a numerical integration scheme for the reduced-index DAEs that specifically leverages the hierarchical representation for the expression swell mitigation during the symbolic matrix factorization process. It concludes with a proof of the symbolic-numeric scheme's effectiveness.
5. **Application Fields and Examples** presents various practical applications of the developed techniques. It covers Multi-Body Dynamics (MBD), TPPC, and electrical circuits, providing specific examples for each field. The chapter demonstrates the effectiveness of the symbolic index reduction algorithm in reducing the index of DAEs arising in these applications, highlighting the robustness and efficiency of the proposed methods. The results are compared with existing symbolic and numerical approaches, showcasing the capabilities of the developed algorithm.
6. **Conclusions and Future Work** summarizes the key findings and suggests directions for future research. The chapter also speculates on potential future applications of the proposed techniques, such as Optimal Control (OC) and nonlinear optimization.

The appendices provide in-depth technical insights and detailed explanations on complementary topics that are relevant to the Chapter 5 examples, as well as in the overall context of efficient vehicle dynamics simulations. Specifically, they focus on areas that are essential for a broad understanding of HRT tire-road interaction simulations, as well as the symbolic-numerical analysis and solution of structures. Each appendix is designed to stand alone, allowing readers to refer to the sections most relevant to their interests and needs. However, the first three appendices are interconnected, constituting a thorough guide for simulating the tire-ground interaction in vehicle dynamics simulations.

A. **Acme: A Small 3D Geometry Library** explores the implementation and features of a novel 3D geometry library and is based on the work presented in [1] and open-source software library available in [26]. It begins with a discussion on the role of computational geometry in HRT simulations, followed by an introduction to the new library, detailing the design choices. The appendix then explores the data types supported by the library, including points, lines, rays, planes, segments, triangles, disks, and balls. It continues with a section on mesh tools, covering space partitioning and bounding volume hierarchies for fast intersection. The appendix concludes with a step-by-step example to demonstrate the practical application of the library.

B. **Tire-Ground Enveloping Modeling** provides a comprehensive guide to tire-ground enveloping modeling. It presents a novel tire-ground enveloping model for HRT vehicle dynamics simulations, followed by the implementation of the algorithm, including its workflow and software architecture. The specific data types are discussed, highlighting their roles in the algorithm workflow, from tire envelope and mesh initialization to local contact parameters calculation. The appendix also includes simulations and discussions on quasi-static simulations on uneven road surfaces, full-vehicle model simulations, and RT performance. Notably, this appendix is based on the previous work presented in [4], with an open-source software library available in [27].

C. **Tire Physical Modeling** presents a novel tire physical model for HRT vehicle dynamics simulations, previously introduced in [5], with no software library distributed yet. It begins with an overview of the geometric representation of tire and road, the mechanics of tire-road vertical contact, and the associated contact patch length and pressure distribution. The discussion extends to tangential contact mechanics, including carcass deformation models, tire-road kinematic equations, tangential contact mechanics equations, and constitutive relations. The appendix then presents a numerical solution approach, addressing tire-road contact discretization, transition coordinates computation, and the solution method for the nonlinear system arising from the model equations. The final sections discuss the RT performance and validation of the tire physical model.

D. **Symbolic-Numerical Analysis and Solution of Structures** begins with an introduction to symbolic structural analysis and then explains the DSM, covering the elements' stiffness matrices, the global stiffness matrix assemblage, and the partitioning and solution of the linear system arising from the DSM. The appendix provides a detailed description of the developed software, highlighting symbolic computation in Maple® and numerical computation in Matlab®, along with examples of their usage. It concludes with example applications, including design optimization and efficient simulation of parametric mechanisms using model reduction. It must be pointed out that this appendix is based on the works [6, 8], which are currently submitted for publication. The software libraries supporting the symbolic-numerical analysis and solution of structures are available in [23,

24, 28].

A flowchart of the thesis structure is provided in Figure 1.1, illustrating the logical progression of the chapters and appendices.

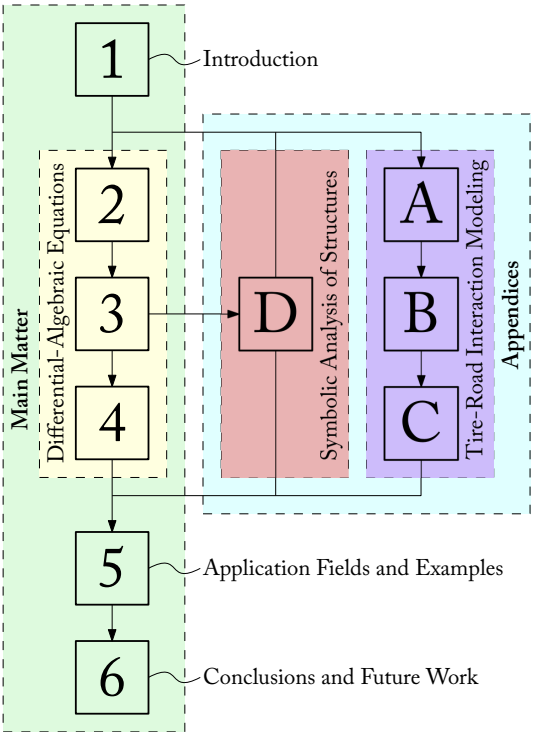


Figure 1.1: Flowchart of the thesis structure, illustrating the logical progression of the chapters and appendices.

CHAPTER 2

BRIEF ON DIFFERENTIAL-ALGEBRAIC EQUATIONS

The solution of dynamical systems represents a fundamental challenge in the fields of applied mathematics, engineering, and physical sciences. These systems, serving as mathematical representations, describe how a system evolves, offering a glimpse into phenomena ranging from chemical reactions to electrical circuit dynamics. Thereby, their solution involves determining the system's state at each time point, considering its Initial Conditions (ICs) and governing equations. This process is critical for comprehending complex system behavior and predicting its evolution. However, due to their inherent complexity, solving such dynamical systems often requires sophisticated algorithms and high-performance computing resources.

While the solution of ODEs has been extensively studied and is well understood, the solution of DAE systems is more challenging due to the presence of algebraic constraints. Specifically, DAEs is a generalization of ODEs that involves both differential and algebraic equations. They first appeared in the last century, and they gained popularity in various fields over these decades. Indeed, due to their simplicity- and straightforwardness-of-use, DAEs is a powerful tool for modeling systems with constraints [11], like mechanisms, electrical circuits, and chemical processes. However, on the contrary to ODEs, their solution requires specialized algorithms to obtain accurate and efficient results. In this chapter, we offer an overview of DAE system and their state-of-the-art solution methods. Afterward, we clarify the motivation behind this research and the contributions of this thesis: The development of a novel and robust algorithm for the index reduction of generic first-order DAEs using symbolic computation methods.

2.1 OVERVIEW ON DIFFERENTIAL-ALGEBRAIC EQUATIONS

A system of equations involving one or more unknown functions and their derivatives is referred to as a DAE system (or DAEs). In its general form, a first-order DAE system is expressed as

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \mathbf{0}\,, \tag{2.1}$$

where $\mathbf{x}(t) = \mathbf{x}$ denotes the vector of unknown functions, and $\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \mathbf{F}$ indicates the vector of functions consisting of n components. Notice the components of $\mathbf{x}$ and $\mathbf{F}$ are denoted as x_i and F_i, respectively, for $i = 1, 2, \ldots, n$. Notably, the term "DAE" is typically used when the highest derivative $\mathbf{x}'$ can not be explicitly solved in terms of the other variables $\mathbf{x}$ and t within the algebraic relationship represented by (2.1). Indeed, the Jacobian $\mathbf{F}_{\mathbf{x}'}$ along a specific solution of the DAE might become singular. Systems of equations like (2.1) are also known as *implicit* systems. Depending on the context, a DAE may be employed to represent either an Initial Value Problem (IVP), if $\mathbf{x}$ is specified at the initial time (*e.g.*, $\mathbf{x}(t_0) = \mathbf{x}_0$), or a boundary value problem, if the solution adheres to n two-point Boundary Conditions (BCs) (*e.g.*, $\mathbf{g}(\mathbf{x}(t_0), \mathbf{x}(t_f)) = \mathbf{0}$). Regardless of which kind of problem the specific DAEs represents, a prevalent category encountered in practical applications is the so-called *semi-explicit* DAE, represented as

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{z}, t) & \text{(2.2a)} \\ \mathbf{0} = \mathbf{g}(\mathbf{x}, \mathbf{z}, t) & \text{(2.2b)} \end{cases},$$

where (2.2a) and (2.2b) denote the *differential equations* and the *algebraic constraints*, respectively. An example of a DAE system of the form (2.2) is the simple pendulum in redundant coordinates, where the motion of a mass at the end of a rod is constrained to a fixed length. In the Cartesian coordinates, the system is described by a set of DAEs that can be restated into the classical ODE for a pendulum.

Example 1 ***(The Simple Pendulum).*** Consider a straightforward example involving the motion of a pendulum in Cartesian coordinates. Let the position and velocity coordinates of the mass m at the end of the rod be (x, y) and $(x', y') = (u, v)$ respectively, with the angle between the rod and the vertical axis denoted as θ. The position of the mass is given by the coordinates $x = \ell \sin\theta$ and $y = \ell \cos\theta$, where ℓ represents the pendulum length. The Euler-Lagrange equations yield the following first-order semi-explicit DAE system

$$\begin{cases} x' = u & \text{(2.3a)} \\ y' = v & \text{(2.3b)} \\ u' = -\dfrac{2\lambda x}{m} & \text{(2.3c)} \\ v' = -\dfrac{2\lambda y}{m} - g & \text{(2.3d)} \\ 0 = x^2 + y^2 - \ell^2 & \text{(2.3e)} \end{cases},$$

which corresponds to the form (2.2). Here, g denotes gravity, and λ is the Lagrange multiplier. The terms $2\lambda x$ and $2\lambda y$ represent the constraint force maintaining the constraint (2.3e) and thereby ensuring the rod's fixed length. In this simple case, transforming variables $x = \ell \sin\theta$ and $y = \ell \cos\theta$, followed by algebraic manipulation, leads to the classical ODE for a pendulum $\theta'' = -g \sin\theta$. ■

It is worth noting that, while the simple pendulum's motion can be described by a simple ODE, the DAE system (2.3) provides a more natural representation of the system's constraints. However, in more complex scenarios, such direct transformations may not be possible, and the DAE system must be solved as is.
DAEs, in either general or specific formulations, are commonly encountered in mathematical models across diverse engineering and scientific disciplines. Further real-life examples of DAE systems, spanning from MB mechanical systems to electrical circuits and TPPCs, are detailed in [9]. It is important to note that while constraints in mechanical systems like the pendulum physically represent the system's limitations, those in other scenarios, such as TPPCs, may also serve as performance specifications or control objectives.

2.1.1 THE SIGNIFICANCE OF DIFFERENTIAL-ALGEBRAIC EQUATIONS

As demonstrated in the example above, working with implicit DAE models often provides a more natural representation compared to explicit formulations, particularly when dealing with complex systems. Indeed, DAE systems represent a generalization of ODEs. However, contrary to DAEs, ODE systems have a well-established literature in mathematical theory and numerical solution technique. They can be typically expressed as $\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$ or $\mathbf{M}(\mathbf{x}, t)\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$ with $\mathbf{M}(\mathbf{x}, t)$ a non-singular matrix. Nonetheless, the broader scope of DAEs encompasses problems with distinct mathematical properties and presents unique challenges for numerical resolution.
To better grasp the distinction between DAEs and ODEs, let us consider this simple example

$$\begin{cases} x' = y & \text{(2.4a)} \\ 0 = x - q & \text{(2.4b)} \end{cases},$$

where $q = q(t)$ is a suitably smooth function. The unique solution is $x = q$ and $y = q'$, without the need for ICs or BCs, which implies that imposing arbitrary ICs could render the DAE inconsistent. Additionally, unlike ODEs, the solution's dependency on the derivative of the inhomogeneous part introduces a distinctive characteristic and, even with consistent ICs, the existence and uniqueness theory for DAEs involve more intricate technical assumptions beyond mere smoothness, as seen in the ODE case [31, 32]. The requirement to differentiate x in (2.4b), thereby involving differentiation of the input function q to determine y, constitutes a fundamental distinction. Hence,

conversely to ODEs, DAEs may necessitate both integration and differentiation to be solved.

2.1.2 THE INDEX... OR BETTER YET, THE INDICES

In the realm of DAEs, the concept of index serves as a metric for measuring the deviation of a DAE from its *underlying* ODE with *invariants*. This index is a non-negative integer providing valuable insights into the mathematical structure and potential challenges associated with analyzing and solving numerically the DAE. Generally, a higher index indicates a more complicated system, which may pose difficulties in its resolution. Various index definitions exist, these include, but are not limited to, the *Kronecker* index (for linear constant coefficient DAEs), *differentiation* index, *structural* index, *tractability* index, *strangeness* index, *geometric* index, and *perturbation* index [33]. While these indices may coincide in simple scenarios, they can differ in more intricate nonlinear and fully implicit systems [34]. Furthermore, the index may exhibit local variability, assuming different values across distinct regions, and could even remain undefined at singular points [34]. Notice that the differentiation index is the most common index used in practice and is typically referred to without any further epithet, *i.e.*, "the" index. In the following sections, we delve into the differentiation, tractability, and structural indices, which are the most used in practice. Together with their qualitative definition, this discussion aims to provide a brief yet comprehensive understanding of what properties we should expect from the DAEs we are dealing with. For this purpose, several considerations that arise from the index definitions are discussed. However, it is still a partially open problem to characterize the exact correspondence between the existing indices. The work on this topic is still ongoing [33] and will not be covered in this thesis.

DIFFERENTIAL INDEX Given that a DAE encompasses both differential equations and constraints, a potential strategy involves repeatedly differentiating the constraint equations (in the semi-explicit DAE systems of the form (2.2)) and substituting them into the differential ones. This process is called *index reduction*, and it aims to transform the DAE into an explicit ODE system in all unknowns. Indeed, the solution of the DAE system corresponds to the solution of its underlying ODE with invariants [35]. The minimum number of differentiations required for this transformation is termed the differential index of the DAE, with the underlying ODEs possessing an index of 0 [33].

Example 2 ***(Differential Index)***. Consider the simple examples involving a given smooth function $y = y(t)$ and the unknown x. Then, the scalar equation

$$x = y \tag{2.5}$$

constitutes a DAE with a differential index of 1, as it requires one differentiation to

yield the ODE $x' = y'$. Similarly, for the system

$$\begin{cases} x_1 = y \\ x_2 = x_1' \end{cases}, \tag{2.6}$$

differentiating the first equation leads to $x_2 = x_1' = y'$, and subsequent differentiation yields $x_2' = x_1'' = y''$. Hence, this system possesses a differential index of 2, necessitating two differentiations to obtain the explicit underlying ODE system. In both cases, the invariants are the equations that are replaced by their derivatives, *i.e.*, $x = y$ for the first example, and $x_1 = y$ and $x_1' = y'$ for the second example. ■

It is crucial to understand that, while n ICs or BCs are required to define the solution of a first-order ODEs of size n, the solution of the simple DAEs in the previous example is solely determined by the right-hand side, necessitating only one consistent IC. General DAE systems often include ODE subsystems, resulting in $k \in [0, n]$ Degree of Freedoms (DOFs) for the DAE solution. However, determining which k pieces of information are necessary to determine the solution can be challenging. ICs or BCs specified for the DAE must be consistent, satisfying both the constraints and the *hidden* constraints of the system, which represent the *solution manifold* [35]. For instance, in the index-1 system (2.5), an IC must adhere to $x_1(0) = y(0)$. In the case of the index-2 system (2.6), meeting the additional constraint $x_2(0) = y'(0)$ is necessary, along with $x_1(0) = y(0)$.

As demonstrated in the example above, the differential index indicates how far is a DAE from an ODE system. However, computing the differential index is not a trivial task, as it involves intensive manipulation of expressions. The differentiation process can lead to large expressions, which are inevitably computationally expensive to handle. Such limitations are particularly evident in high-index DAEs. To address this issue, further indices are introduced to provide additional understandings of the mathematical structure of DAEs without the need for extensive symbolic analysis of the system.

Although the concept of the differentiation index is widely used, it has a limitation: it is unsuitable for over- and underdetermined systems. Indeed, the differentiation index is based on a solvability concept that requires a unique solution for the DAE system. For this reason, the *strangeness index* has been introduced, which extends the differentiation index to over- and underdetermined systems. The difference between the differentiation index and the strangeness index is that the former aims at reformulating the given problem as an ODE for uniquely solvable systems, whereas the latter aims at reformulating it as a DAE with two parts, one part that states all constraints and another part that describes the dynamical behavior [33]. Despite the strangeness index's advantages, it is not widely used in practice, and the work on this index is still ongoing. Nonetheless, we limit our study to well-determined DAE systems.

STRUCTURAL INDEX The structural index was initially introduced in the linear constant coefficient scenario for combinatorial analysis. Given the linear DAEs $\mathbf{E}\mathbf{x}' =$

$\mathbf{A}\mathbf{x} + \mathbf{f}(t)$, the parameter-dependent pencil $(\mathbf{E}(\mathbf{p}), \mathbf{A}(\mathbf{p}))$ is constructed by substituting the nonzero elements of $\mathbf{E}$ and $\mathbf{A}$ with independent parameters $\mathbf{p}$. The structural index, as discussed in [36, 37] and expanded upon in subsequent works [38], refers to the unique integer value matching the *Kronecker* index of $(\mathbf{E}(\mathbf{p}), \mathbf{A}(\mathbf{p}))$ across an open and dense subset of the parameter set. In the context of nonlinear systems, local linearization techniques are commonly applied. While it has been demonstrated in [39] that the differentiation index and the structural index may differ, Pantelides' algorithm, detailed in [36], remains widely used in practical applications. Notably, in studies like [40], combinatorial insights are leveraged to determine which equations should be differentiated, as well as to introduce additional variables for index reduction [41]. However, a comprehensive assessment of the validity of this approach across various scenarios has been provided only in a few specific cases, as discussed in [33].

TRACTABILITY INDEX Relevant studies, such as [42], reveal that the index may vary depending on a specific solution. An example is the following DAE system.

Example 3 ***(Differential Index Dependency on Solution).*** Consider the DAE system in the semi-explicit form

$$\begin{cases} x_1' = x_3 \\ 0 = x_2(1 - x_2) \\ 0 = x_1 x_2 + x_3(1 - x_2) - t \end{cases}.$$

The second equation admits two solutions: $x_2 = 0$ and $x_2 = 1$. If x_2 is continuous, the system does not transition between these values. For $x_2 = 0$, the system is semi-explicit with an index of 1, while for $x_2 = 1$, the index becomes 2. Unlike the index-1 scenario, no IC on x_1 is necessary. Replacing the algebraic equation involving x_2 with $x_2' = 0$, leads the index of the modified DAE system to depend on the IC. Specifically, if $x_2(0) = 1$, the index is 2; otherwise, it remains 1 [42, Section 3.3]. ■

In the definition (2.1), the function $\mathbf{F}$ is inherently assumed to be smooth enough to calculate the derivatives. However, smoothness is not granted in practical applications. The tractability index is a concept introduced in Griepentrog and März [15] and März [43] to overcome this issue. The idea is to replace the smoothness requirements for the coefficients with the requirement for certain subspaces to be smooth. Such a concept leverages DAE systems with *properly stated leading terms* [42], *i.e.*, a vector $\mathbf{d}(\mathbf{x}, t)$ is used when formulating the DAEs as

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \mathbf{A}(\mathbf{x}, t)(\mathbf{d}(\mathbf{x}, t))' + \mathbf{b}(\mathbf{x}, t) = \mathbf{0}\,.$$

In this manner, the leading term $\mathbf{d}(\mathbf{x}, t) = \mathbf{D}(t)\mathbf{x}$ precisely figures out which derivatives are involved. Importantly, all admissible linearizations along the solution have the same tractability index [44].

In essence, the tractability index concept is designed to tackle the challenges posed by DAEs with multiple regular regions. It explicitly suggests that the domain of definition of the DAE divides into maximal smooth subspaces, each delimited by singular points. Within each regular region, the DAE exhibits a consistent structure, which can be revealed using matrix sequences formed with permissible projector functions. This construction is governed by constant rank conditions. Singular points arise when this construction process encounters difficulties. A smooth flow and reliable treatment are expected as long as the solution remains within a regular region. However, crossing or touching a boundary between regions may lead to singularities. Essentially, monitoring the structure involves computing an admissible matrix function sequence and monitoring the rank conditions to ensure the stability and reliability of the solution [45]. Thereby, the fundamental idea behind the tractability index concept involves the utilization of derivatives of projectors instead of derivative arrays. However, if the numerical computation of the projectors is adopted, challenges may arise in obtaining their derivatives [33].

2.1.3 THE HESSENBERG FORMS

As will be better detailed in the forthcoming section, the implicit DAE system (2.1) may represent mathematically ill-defined problems, as well as scenarios where numerical methods fail. Thankfully, numerous high-index problems encountered in practical applications are expressed as a more restrictive formulation composed of ODEs with algebraic constraints. Within such systems, both the algebraic and differential variables are explicitly separated and identified, even for higher-index DAEs, allowing for the elimination of all algebraic variables using either index reduction, numerical direct discretization, or a combination of both. These formulations, known as *Hessenberg forms* of the DAEs [9], are detailed here below and will serve as the basis for further exploration into DAE solutions in subsequent sections and chapters.

- The *Hessenberg index-1* DAE system is represented as

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{z}, t) \\ \mathbf{0} = \mathbf{g}(\mathbf{x}, \mathbf{z}, t) \end{cases},$$

  with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, and where the Jacobian $\mathbf{g}_z$ is assumed non-singular for all t. This configuration closely resembles the semi-explicit index-1 DAE system (2.2) discussed earlier. Semi-explicit index-1 DAEs shares similarities with implicit ODEs. While it is theoretically feasible to solve for $\mathbf{z}$ in the algebraic equation (using the implicit function theorem) and then substitute it into the differential equation to derive the underlying ODE in terms of $\mathbf{x}$ (though uniqueness is not guaranteed), this approach is not universally recommended for numerical solutions due to potential ill-conditioning and stability issues. An example of a Hessenberg index-1 DAE system is in the formulation of TPPCs problems [9].

- The *Hessenberg index-2* DAE system is expressed as

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{z}, t) \\ \mathbf{0} = \mathbf{g}(\mathbf{x}, t) \end{cases},$$

with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, and where the product of Jacobians $\mathbf{g}_\mathbf{y}\, \mathbf{f}_\mathbf{z}$ is assumed non-singular for all t, $\mathbf{x}$, and $\mathbf{z}$. It is important to observe that the algebraic variable $\mathbf{z}$ does not appear in the second equation. This arrangement characterizes a pure index-2 DAE, where all algebraic variables act as index-2 variables. An example of Hessenberg index-2 DAEs arises from the modeling of incompressible fluid flow through discretized Navier-Stokes equations [46].

- The *Hessenberg index-3* DAE system is formulated as

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{z}, t) \\ \mathbf{y}' = \mathbf{g}(\mathbf{x}, \mathbf{y}, t) \\ \mathbf{0} = \mathbf{h}(\mathbf{y}, t) \end{cases},$$

with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, and where the product $\mathbf{h}_\mathbf{y}\, \mathbf{g}_\mathbf{x}\, \mathbf{f}_\mathbf{z}$ is non-singular for all t, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$. Determining the index of a Hessenberg DAEs follows a similar differentiation process to the general case. However, only algebraic constraints need to be differentiated [47]. These index-3 DAE systems are frequently encountered in practical scenarios, notably in the fields of MBD, TPPCs, and various engineering applications [9, 46].

- The *Hessenberg index-$(r+2)$* DAE system is formulated as

$$\begin{cases} \mathbf{x}_r' = \mathbf{f}_r(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r, \mathbf{y}, \mathbf{z}, t) \\ \quad \vdots \\ \mathbf{x}_2' = \mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{y}, t) \\ \mathbf{x}_1' = \mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, t) \\ \mathbf{y}' = \mathbf{g}(\mathbf{x}_1, \mathbf{y}, t) \\ \mathbf{0} = \mathbf{h}(\mathbf{y}, t) \end{cases} \qquad \text{for} \quad r = 1, 2, \ldots, r,$$

with $\mathbf{x}_i \in \mathbb{R}^{n_i}$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{f}_i : \mathbb{R}^{n_i} \times \mathbb{R}^{n_{i+1}} \times \cdots \times \mathbb{R}^{n_r} \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^{n_i}$, $\mathbf{g} : \mathbb{R}^{n_1} \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^m$, and where the product $\mathbf{h}_\mathbf{y}\, \mathbf{g}_{\mathbf{x}_1}\, \mathbf{f}_{1\mathbf{x}_2} \ldots \mathbf{f}_{r-1\mathbf{x}_r}\, \mathbf{f}_{r\mathbf{z}}$ is non-singular for all t [42, Section 3.5].

2.2 SOLUTION METHODS FOR DIFFERENTIAL-ALGEBRAIC EQUATIONS

The literature on DAE solution methods is vast and varied, with numerous approaches available for tackling these complex systems [9, 15, 31, 32]. Methods for solving DAEs

are broadly categorized into two classes: (*i*) direct discretization of the given system and (*ii*) methods involving index reduction, which involves reformulating the system's equations so to ease the numerical solution process. While direct discretization is favored for its simplicity and straightforwardness, particularly with relatively low-index DAEs (typically less or equal than 3), index reduction approaches are essential to tackle higher-index DAEs. It is important to note that these two methods can be used alongside, by applying first index reduction to come up with an index-1 or index-2 DAE system, and secondly by applying a numerical direct discretization. As we will see, this approach keeps expressions' complexity low while exploiting the straightforwardness of direct discretization. Nonetheless, there exist some classes of Radau and Bounding Value Methods (BVMs) that allow for the direct discretization of very high-index DAEs without the need for index reduction [48–51].

2.2.1 NUMERICAL DIRECT DISCRETIZATION

Direct discretization is preferred due to its simplicity and computational efficiency, as index reduction methods can be time-consuming and computationally expensive, often requiring more user inputs and interventions. However, direct discretization is most effective for index-1, index-2, and index-3 Hessenberg DAE systems. Usually, many practical DAEs fall into these categories or can be transformed into simple combinations of Hessenberg systems. Despite this, challenges may still arise, and even for these restricted classes of problems, direct application of numerical ODE methods may lead to insufficient results and instabilities. For DAEs with an index greater than 2, employing index reduction techniques to solve the problem in a lower-index form is often considered the most effective approach. Similar considerations arise in cases resembling singularly perturbed ODE systems, such as

$$\begin{cases} \mathbf{y}' = \mathbf{f}(\mathbf{y}, \mathbf{z}, t) \\ \varepsilon \mathbf{z}' = \mathbf{g}(\mathbf{y}, \mathbf{z}, t) \end{cases}, \tag{2.7}$$

where ε is a small perturbation parameter. Setting $\varepsilon = 0$ reduces (2.7) to the DAEs (2.2). In general, due to the system's stiffness for small ε, methods designed for such ODE system, like Backward Differentiation Formula (BDF) and Radau collocation methods, are natural choices for directly discretizing implicit DAEs of the form (2.1) [48].

BACKWARD EULER METHOD The concept of direct discretization is relatively straightforward: it involves approximating $\mathbf{x}$ and $\mathbf{x}'$ using a discretization technique, such as multistep methods or Runge-Kutta (RK) methods. As a first example, let us take the Backward Euler method, which is the simplest method exhibiting a stiff decay property. Applying the Backward Euler formula to $\mathbf{x}'$ in (2.1), we obtain a system of N nonlinear equations

$$\mathbf{F}\left(\mathbf{x}_n, \frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{h_n}, t_n\right) = \mathbf{0} \quad \text{for} \quad n = 1, 2, \ldots, N. \tag{2.8}$$

Here, $\mathbf{x}_n$ represents the approximation of $\mathbf{x}(t_n)$, $h_n = t_n - t_{n-1}$ denotes the time step, t_n are the time points, and N is the total number of time points. Solving this nonlinear equation system recursively provides a numerical solution for (2.1).
This method is effective for index-1 DAEs and particularly suitable for stiff index-1 DAEs and stiff ODEs. However, for higher-index DAEs this straightforward approach may not suffice. In some cases, even seemingly simple higher-index DAE systems with well-defined and stable solutions can pose challenges, making methods like the Backward Euler method, as well as other multistep and RK methods, unstable or inapplicable [46]. Practical difficulties may arise during the resolution of the nonlinear system (2.8) for $\mathbf{x}_n$ given $\mathbf{x}_{n-1}$, where iterative numerical methods like the Newton method are employed. Due to these implementation challenges, direct discretization of fully implicit DAEs with an index greater than 1 is generally discouraged. Despite this, for fully implicit index-1 and semi-explicit index-2 DAEs, the Backward Euler method is stable, convergent, and first-order accurate [9, 52].

BACKWARD DIFFERENTIATION FORMULA AND LINEAR MULTISTEP METHODS Although Euler is a first-order method, achieving higher accuracy without reducing step size requires the use of higher-order methods. One such method is the constant step-size BDF, which is applied to a general nonlinear DAE system of the form (2.1) and is given by

$$\mathbf{F}\left(\mathbf{x}_n, \frac{1}{\beta_0 h}\sum_{j=0}^{s}\alpha_j \mathbf{x}_{n-j}, t_n\right) = \mathbf{0} \quad \text{for} \quad n = 1, 2, \ldots, N,$$

where β_0 and α_j for $j = 0, 1, \ldots, s$ are the coefficients of the BDF method. The s-step BDF method of fixed step size h is convergent of order $\mathcal{O}(h^s)$ under certain conditions. Similar convergence results have been established for general linear multistep methods, provided their coefficients satisfy a set of order conditions, including those unique to DAEs [9]. BDF methods meet these additional requirements (refer to [9] for additional information).

RADAU COLLOCATION AND IMPLICIT RUNGE-KUTTA METHODS The s-stage implicit RK method, when applied to a general nonlinear DAE of the form (2.1), is expressed as

$$\mathbf{F}\left(\mathbf{x}_k + h_k \sum_{j=1}^{s} a_{ij}\mathbf{K}_j, \mathbf{K}_i, t_k + c_i h_k\right) = \mathbf{0}\,,$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + h_k \sum_{i=1}^{s} b_i \mathbf{K}_i\,,$$

where c_i, a_{ij}, b_i for $i = 1, 2, \ldots, s$ and $j = 1, 2, \ldots, s$ are the coefficients of the RK method, with the additional assumption that the matrix $\mathbf{A} = (a_{ij})$ is non-singular.

Similarly, the s-stage implicit RK method for semi-explicit DAEs of the form (2.2) is given by

$$\begin{aligned}
\mathbf{K}_i &= \mathbf{f}\left(\mathbf{x}_k + h_k \sum_{j=1}^{s} a_{ij}\mathbf{K}_j, \mathbf{z}_k, t_k + c_i h_k\right) = \mathbf{0}\,, \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + h_k \sum_{i=1}^{s} b_i \mathbf{K}_i\,, \\
\mathbf{0} &= \mathbf{g}\left(\mathbf{x}_k, \mathbf{z}_k, t_k + c_i h_k\right)\,,
\end{aligned}$$

for $i = 1, 2, \ldots, s$. Notice that it is possible to avoid the quadrature step for the algebraic variables $\mathbf{z}$ by employing stiffly accurate methods, *i.e.*, RK methods satisfying $b_i = a_{si}$ for $i = 1, 2, \ldots, s$, like the Radau collocation methods [9, 48].

Implementing direct discretization methods for DAEs faces practical challenges. These challenges encompass obtaining a consistent set of ICs, addressing the ill-conditioning of the iteration Jacobian matrix, and managing error estimation for step-size control. However, for specific classes of DAEs, such as semi-explicit ones in Hessenberg form and ODEs with hidden constraints typically found in MB mechanics, highly efficient and robust numerical methods called *stabilized* or *projected methods* exist. The core approach involves discretizing the differential equations using a suitable numerical ODE method with a stabilization technique or coordinate projection step to align the numerical solution with the constraints [53]. For a thorough understanding of numerical aspects concerning DAEs, refer to [9, 46, 52].

2.2.1.1 SOFTWARE FOR DIFFERENTIAL-ALGEBRAIC EQUATIONS NUMERICAL SOLUTION

Several software packages are available for solving DAEs, including both general-purpose and specialized tools. Some of the most widely used software packages for IVP and boundary value problem DAEs's numerical solutions are listed here below.

- **DASSL**, developed by Linda Petzold, utilizes BDF formulas to solve generic index-1 DAEs. Variants tailored for large-scale problems and sensitivity analysis are also available. Specifically, DASPK, a later iteration of DASSL, can handle large Hessenberg index-2 DAEs; DASPKADJOINT, on the other hand, implements the adjoint method for sensitivity analysis of DAE systems [9].
- **RADAU5** is a software by Ernst Hairer and Gerhard Wanner based on the 3-stage Radau collocation method [52]. It tackles problems expressed as $\mathbf{M}\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$, where $\mathbf{M}$ is a constant, possibly singular, square matrix. RADAU5 solves DAEs up to index 3, with the identification of higher-index variables being required from the user. Higher-order Radau methods are available, but they require more intricate and custom implementations [48].

- **Ida** is a submodule of the SUite of Nonlinear and DIfferential-ALgebraic equation Solvers (SUNDIALS) software package developed by Radu Serban and Alan Hindmarsh at Lawrence Livermore National Laboratory (LLNL) [54, 55]. Ida is coded in C and tackles nonlinear DAEs. It is derived from the aforementioned Fortran package Daspk. Additionally, a related code for solving ODEs with invariants is available under the name of Cpodes.
- **Daepack**, is a software library created by Paul Barton and his team at Massachusetts Institute of Technology (MIT) [56]. While its name stands for DAE Package, Daepack's capabilities extend beyond DAE analysis, encompassing both symbolic and numerical components for modeling and general numerical computations.
- **Coldae** was firstly presented in [57]. It employs projected Gauss collocation to solve boundary value problems for semi-explicit DAEs with an index of at most 2.

2.2.2 INDEX REDUCTION METHODS

For higher-index DAEs, direct discretization methods might not offer the most efficient solution. In such scenarios, index reduction methods are preferred. Such methods involve reformulating the DAE into a lower-index form, which can be tackled using direct discretization methods or standard techniques for ODE integration, aided by projection or stabilization techniques. Specifically, the index is reduced by differentiating the algebraic constraints and substituting them into the differential equations. This process repeats until the DAE transforms into an explicit ODE system. As mentioned earlier, the number of iterations needed for this transformation is termed the index of the DAE, with ODEs having an index of 0.
We have seen that there exist multiple definitions of the index of a DAE system, and each of them is tailored to a particular index reduction technique. Consequently, there are various approaches to index reduction, whose characteristics also depend on factors like the capabilities of the available package, the user's expertise, and the specific problem at hand. Each method has advantages and disadvantages, being more or less suitable for different types of DAEs and varying in computational cost, implementation requirements, and theoretical foundations.

DIFFERENTIAL INDEX REDUCTION As expert readers may notice, the differential index reduction method bears close relation to symbolic computation and symbolic computation software such as Maple®, Mathematica®, and Python's library SymPy. These prove invaluable in executing the differentiation and substitution steps essential for reducing the differential index of a DAE system. These tools also facilitate the derivation of the DAEs' Jacobian, crucial for numerical solution approaches. However, in some problems (very few), symbolic computation tools can also furnish the DAEs' solution, aiding in the validation and assessment of numerical solutions

obtained through numerical methods. Nevertheless, it is worth noting that symbolic computation tools are not always the optimal choice for DAEs resolution, particularly for large-scale or intricate systems. Indeed, the computational overhead of symbolic computation tools may become prohibitive in such cases, prompting a preference for a mixed approach involving both symbolic computation and numerical methods.
As previously mentioned, the basic idea behind the differential index reduction method is to differentiate the algebraic constraints and substitute them into the system. This process repeats until the DAE system transforms into an explicit ODE system. To do so, one can indiscriminately differentiate all equations in the DAE system. However, this approach is not the most efficient. In some cases, differentiating only a subset of the equations leads to a more straightforward and less computationally expensive solution, especially for large-scale DAEs. Indeed, algebraic equations should be only differentiated. To this end, the system should be restated into its Hessenberg form, where the algebraic constraints are explicitly separated from the differential equations [58]. Thereby, the isolation process of algebraic equations is crucial for the successful application of the differential index reduction method.

STRUCTURAL INDEX REDUCTION The most common index reduction methods are those based on the *Pantelides algorithm* [36], which fall under the umbrella of the Structural Analysis (SA) methods. However, recent advances have led to the development of more effective algorithms, such as S. Campbell and W. Gear's method of *differential arrays* [59] and J. Pryce's Σ*-method* [60]. This latter method provides a systematic approach to index reduction and the solution of DAEs using the Taylor series. Nonetheless, it has been proven to be a generalization of the Pantelides algorithm [37]. The SA examines the regularity or singularity of DAE in a generic sense. This analysis is typically conducted through a simplified representation of the system's structure using tools such as the bipartite graph, which characterizes the system's incidence matrix non-zeros pattern. Structural regularity or singularity, as well as related properties, are then assessed based on this graph. This approach is effective for sparse systems, where the incidence matrices have a small proportion of non-zero entries. SA techniques also extend to numerical regularity, which is determined by the system's Jacobian matrix. Understanding the link between numerical regularity and structural regularity is crucial for correctly applying SA methods. Additionally, the SA of systems of equations can be extended to include systems with existential quantifiers [38].
In essence, the SA of DAE systems aims to address the following problem. Considering a DAE system represented by equation (2.1) as a set of algebraic equations with the leading variables as unknowns, the goal is to determine if this system is structurally non-singular. If it is, then given values for all the derivatives x_k' for $i = 1, 2, \ldots, n$ and $k = 0, 1, \ldots, d_{i-1}$, a unique leading value for the i variables can be computed structurally and, in other words, the system behaves like a ODE system. However, if the system is not structurally non-singular, additional latent equations can be derived by suitably differentiating selected equations. This process transforms the system into a ODE-like form without altering its solution set [38].

The structural index introduced earlier serves as a metric for quantifying the number of times the system needs to be differentiated to attain a structurally non-singular form. It is important to emphasize that the structural index can be significantly higher than the differential index. In particular, Reißig, Martinson, and Barton [39] highlights that the structural index of constant coefficients linear DAEs, with a differential index of 1, can be arbitrarily high, which contrasts many previous findings in the literature. This reveals that applying Pantelides' algorithm to index-1 DAEs may lead to an indefinite number of iterations and differentiations.

TRACTABILITY INDEX REDUCTION Another approach to index reduction is the so-called *projector–based analysis*, thoroughly detailed in Lamour, März, and Tischendorf [42] and März [61]. The projector-based analysis, founded on the tractability index, explores the concept that complex DAEs can consist of distinct regularity regions bordered by critical points. These regions reveal a consistent structure that can be uncovered using matrix function sequences formed with admissible projector functions and guided by constant rank conditions. Smooth solutions are expected within these regions, but crossing borders may lead to singularities. In essence, this method provides a way of tracking the system's structure by computing an admissible matrix function sequence and monitoring rank conditions [45].
More specifically, the projector-based analysis involves finding a projector that separates the differential and algebraic components of a DAE system with properly stated leading term [45]. Hence, the projector transforms the DAE system into a lower-index form, which is then integrated with improved numerical stability (thanks to the properly stated leading term) [62]. Notice that in the case of nonlinear DAEs, the projector-based analysis is typically performed pointwise, which means that the algorithmic steps of the computation of admissible null-space projectors are performed at each time step, ensuring that the changes in the DAEs structure are accounted for at each time step. Special continuous projector-valued functions can be reutilized for multiple time steps, thus reducing the computational cost of the method [34].

2.2.3 SOFTWARE FOR INDEX REDUCTION

Following this brief introduction to the most widely recognized index reduction algorithms, we provide an overview of the specific index reduction algorithms found in current state-of-the-art software. The following list is limited to the most prominent solutions dedicated to dynamic system modeling and simulation that are adopted in both academia and industry.

- **MATLAB®** uses the Pantelides algorithm [36] to reduce the DAEs to index-1. Alternatively, or if the latter fails, the more reliable but slower Gaussian elimination algorithm is employed to obtain an index-0 DAE system [63].
- **MODELICA** [64, 65], MODELICA-based software and **MODELINGTOOLKIT** [66] employ the Pantelides algorithm [36] along with the dummy derivatives

method [41] to automatically perform the reduction to index-1 DAEs.

- MATHEMATICA® offers a comprehensive suite of index reduction algorithms [67]. It can reduce the index of DAEs using the Pantelides [36] or structural matrix [40, 68] methods. Additionally, it implements dummy derivatives [41] and projection methods for taking hidden constraints into account during numerical integration.
- MAPLE® performs symbolic index reduction within the `dsolve` function [69]. However, the implemented algorithms are not documented or referenced. They are likely to be based on the projection method outlined in [58]. Notably, the patents by the same authors [70–72] discuss techniques for eliminating isolated parameters, extracting parameter sub-expressions from DAEs, and establishing minimal disconnected clusters of parameter sub-expressions. We would like to point out that this information is to be taken with caution, as MAPLE® does not provide specific references on this topic.
- **DAESA**, developed by Guangning Tan and Ned Nedialkov in collaboration with John Pryce, is a MATLAB® toolbox designed for conducting SA of DAEs. DAESA can analyze fully nonlinear systems, irrespective of their order or index, making it capable of determining crucial attributes such as the structural index, DOFs, constraints, and variables requiring initialization, while also proposing a suitable solution strategy [73, 74]. Additionally, it can generate a block-triangular form of the DAEs, enabling efficient block-wise solution strategies.
- **DAETS**, short for DAEs by Taylor Series, is a C++ package developed by Guangning Tan and Ned Nedialkov in collaboration with John Pryce [75–77]. It is designed for tackling IVPs associated with DAE systems. DAETS leverages Pryce's method for the SA of DAEs, providing a powerful means to determine the system's index, DOFs, and to precisely identify which components necessitate ICs.
- **INITDAE** is a PYTHON prototype designed to calculate consistent ICs for DAEs, determining their index, as well as a condition number that aids in identifying singularities. The initialization algorithm utilizes a constrained optimization approach based on projectors, with Automatic Differentiation (AD). Local structural characteristics of the DAE system are examined through the Singular Value Decomposition (SVD) [78]
- **GENDA** is a software package written in FORTRAN for the numerical solution of generic nonlinear DAEs of the form (2.1), with arbitrary high-index [79]. Its implementation is based on the construction of the discretization scheme introduced in [80], which transforms the system into a strangeness-free DAEs with the same local solution set. Specifically, GENDA exploits the definition of *strangeness index*, which generalizes the differentiation index for systems with undetermined components [81].

All the showcased software solutions offer integrators for index-0 and index-1 DAEs. Depending on the system's stiffness, users can select the most appropriate algorithm for numerically integrating the reduced-index system.

2.3 ADVANCING INDEX REDUCTION

Notably, in findings presented by the group of Roswitha März, which are detailed in [42], there is little mention of symbolic computation tools in the derivation of projector functions and matrix function sequences. Instead, the authors rely on numerical factorization methods with a coupled automatic differentiation method to compute the projector functions and the matrix function sequences [45, 82]. However, it is noteworthy that in Lamour, März, and Tischendorf [42], authors show that the latest state-of-the-art CAS allows the derivation of smooth symbolic projector functions, especially for simple or small-scale DAEs. This can significantly reduce the computational cost of the projector-based analysis method as it eliminates the need for numerical factorization methods and automatic differentiation.
Nevertheless, the most relevant consideration arising from what we stated in Section 2.2.2, as well as in Mehrmann [33], Bojarincev [83], Gear and Petzold [84], and Griepentrog and März [85] (despite part of these works focused on linear DAEs), is that a neat separation between the differential and algebraic equations of the DAE system is crucial for the successful application of the differential index reduction method. This separation is achieved by transforming the DAE system into its Hessenberg form. However, this process is not always straightforward. Particularly, for generic nonlinear DAEs, the separation process can be computationally expensive and time-consuming.

A NOVEL SYMBOLIC ALGORITHM FOR INDEX REDUCTION Starting from these observations, we recall and further specify the primary objective of this thesis: The development of an index reduction algorithm for DAEs based on symbolic matrix factorization techniques. It is worth noting that the algorithm is neither directly based on the projector-based analysis nor in index concepts other than the differential index. The algorithm is limited to generic well-determined DAEs of the form (2.1), linear in the states' derivatives. Such a reduction algorithm is implemented as an open-source Maple® package. On the other hand, the numerical integration is performed by an open-source Matlab® toolbox. Both software constitutes the Indigo toolbox [25], whose dependencies are the symbolic linear algebra package LAST [24] and the large expression management package LEM [23]. The proposed symbolic algorithm and its dedicated numerical scheme are validated through a set of benchmark DAEs from the literature arising in various fields, including problems from MBD, electrical circuit simulation, and TPPC. In the forthcoming chapters, we will build up the necessary theoretical background and tools to understand the algorithm's inner workings.

CHAPTER 3

ESSENTIAL CONCEPTS AND TECHNIQUES OF SYMBOLIC COMPUTATION

Symbolic linear algebra plays a crucial role in the index reduction of DAEs. Nonetheless, it is a powerful tool to handle complex mathematical tasks involving extensive algebraic and calculus operations. However, symbolic computation brings its own challenges. In particular, expression swelling is a common issue when intensively working with symbolic expressions. In this chapter, we discuss the expression swell phenomenon and discuss strategies for its mitigation through hierarchical representation techniques. Additionally, we tackle the solution of linear systems of equations using novel matrix factorization methods. Using the newly developed LEM and LAST MAPLE® packages, practical implementation details are demonstrated, showcasing improved symbolic linear algebra capabilities with expression swell mitigation. Nevertheless, before we delve into the details of these packages, let us first introduce the concept of computer algebra and its applications in scientific computing.

3.1 INTRODUCTION TO COMPUTER ALGEBRA

Mathematical scientists employ methodical approaches to model natural phenomena. This involves translating empirical observations and theoretical constructs into mathematical expressions consisting of numerical values, variables, functions, and operators. These expressions are then subjected to established methods of mathematical reasoning, where they are carefully manipulated or transformed to unveil novel insights into the studied phenomenon. This mathematical methodology has been integral to the scientific method in the physical sciences since the era of Galileo Galilei and René Descartes. Following their legacy, Isaac Newton applied this approach to develop a systematic, quantitative framework for describing the motion of objects. Through

mathematical reasoning, he uncovered the universal law of gravitation and formulated additional principles governing phenomena such as tidal motion and planetary orbits. Thus, the discipline of mechanics emerged, solidifying the practice of manipulating and transforming mathematical expressions as a fundamental tool for advancing our understanding of the physical universe.

Over the past five decades, computers have evolved into indispensable tools for mathematical exploration, greatly expanding our capacity to tackle complex problems. Mathematicians frequently employ computers to generate numerical and graphical solutions for challenges that surpass manual capabilities. However, computers go beyond simple arithmetic; fundamentally, they operate by manipulating symbols, represented as binary digits (0s and 1s), following precise rules. Given this capability, it is natural to wonder about the possibility of automating other aspects of mathematical reasoning. While it is unrealistic to expect machines to create foundational axioms like Newton's or develop groundbreaking theories from scratch, there is a significant area of mathematical reasoning that lends itself well to algorithmic treatment, and more specifically to the mechanical manipulation and analysis of mathematical expressions. Currently, computer programs routinely handle tasks such as simplifying algebraic expressions, integrating complex functions, accurately solving differential equations, and performing many other operations crucial in applied mathematics, scientific research, and engineering. The interdisciplinary field at the intersection of mathematics and computer science dedicated to this pursuit is commonly referred to as *computer algebra* or *symbolic computation*.

3.1.1 ELEMENTARY CONCEPTS OF COMPUTER ALGEBRA

A CAS is specialized software designed to execute symbolic mathematical operations, allowing for precise computations across various mathematical domains. Its capabilities encompass a wide range of functionalities, including arithmetic operations such as unlimited precision rational number arithmetic, complex arithmetic, and combinatorial functions. Furthermore, it enables algebraic manipulation through simplification, expansion, factorization, and substitution operations, as well as polynomial operations like structural operations, polynomial division, and finding the greatest common divisors. Additionally, it can handle equation solving for polynomial equations, systems of linear equations, and recurrence relations, while also performing trigonometric operations, calculus computations, and solving differential equations, including ODEs and PDEs. Moreover, it supports advanced algebraic manipulations, linear algebra operations, and code generation for conventional programming languages and mathematical word processing.

3.1.2 COMMERCIAL COMPUTER ALGEBRA SYSTEMS

Over the past 15 years, we have witnessed the development and widespread dissemination of several large-scale (yet user-friendly) CASs. Among the most notable are:

- Axiom: A comprehensive CAS developed during the 1970s at IBM under the name of Scratchpad, later sold to the Numerical Algorithms Group in England and became the Axiom commercial system. It was withdrawn from the market in 2001 and released as free software. Further details about Axiom can be found in [86].
- Derive: A compact CAS originally crafted by Soft Warehouse Inc. for personal computer use. Derive has also been integrated into Texas Instruments Inc.'s TI-89 and TI-92 handheld calculators. More information about Derive is available online, yet the software is no longer actively maintained.
- Macsyma: A robust CAS initially conceived at the Massachusetts Institute of Technology during the late 1960s and 1970s. Various versions of the original Macsyma system are currently in circulation. Additional insights into Macsyma can be gleaned from [87].
- Maple®: A highly sophisticated CAS initially developed by the Symbolic Computation Group at the University of Waterloo (Canada) and presently distributed by Waterloo Maple® Inc. For more information about Maple®, refer to [88] or visit the website [69].
- Mathematica®: An advanced CAS created by Wolfram Research Inc. Further details about Mathematica® are provided in [89] or on the website [67].
- MuPAD: A sizable CAS developed by the University of Paderborn (Germany) and SciFace Software GmbH & Co. KG. Refer to [90]. Later versions of MuPAD are distributed by MathWorks Inc. and currently constitute the Matlab® CAS.
- Reduce: One of the earliest CASs, developed in the late 60s and 70s. Further information about Reduce can be found in [91].
- SymPy: A Python library for symbolic mathematics that aims to become a full-featured CAS. SymPy is open-source and freely available.

Specialized CASs are also available for specific mathematical domains, such as the system developed for physics applications from the 1970s onward. These include, but are not limited to:

- Trigman [92] and Trip [93], developed in 1972 for manipulates Poisson series (trigonometric series with polynomial coefficients);
- Schoonschip [94], introduced in 1971 for quantum physics applications at Conseil Européen pour la Recherche Nucléaire (CERN);
- Camal [95] presented in 1972 for celestial mechanics and the general theory of relativity
- Sheep [96] first released in 1977 for the general theory of relativity.

Each of these software packages constitutes an integrated mathematics problem-solving system, featuring capabilities for exact symbolic computations, as well as some capacity for approximate numerical solutions of mathematical problems and high-quality graphics.

3.1.3 THE MAPLE LANGUAGE

As we have just shown, there exist several commercial CASs that provide a wide range of symbolic computation capabilities. Among these, MAPLE® is a powerful system developed by MAPLESOFT® that offers a comprehensive suite of mathematical tools. The MAPLE® system is equipped with a high-level programming language that allows users to interact with the system intuitively. The MAPLE® language is designed to facilitate the execution of symbolic computations, enabling users to perform complex mathematical operations with ease. The language is equipped with a wide range of built-in functions and operators that can be used to manipulate mathematical expressions, solve equations, and perform various other mathematical tasks. In the following examples, we illustrate an interactive session with the MAPLE® CAS. User inputs, denoted by the prompt (>) at the line beginning, are entered at a computer workstation. Commands such as `factor`, `convert`, `compoly`, and `simplify` are among the mathematical operators available in the MAPLE® system. Upon receiving these commands, the program carries out the corresponding mathematical operations and displays the results using notation similar to conventional mathematical expressions.

Example 4 *(Symbolic operations from algebra and trigonometry in* MAPLE®*).*

```
> p1 := x^5 - 4*x^4 - 7*x^3 - 41*x^2 - 4*x + 35;
```

$$p1 := x^5 - 4x^4 - 7x^3 - 41x^2 - 4x + 35$$

```
> factor(p1);
```

$$(x+1)(x^2 - 7x + 5)(x^2 + 2x + 7)$$

```
> p2 := (x^4 + 6*x^2 + 3)/(x^5 + x^3 + x^2 + 1);
```

$$p2 := \frac{x^4 + 6x^2 + 3}{x^5 + x^3 + x^2 + 1}$$

```
> convert(p2, parfrac, x);
```

$$\frac{x+7}{3(x^2 - x + 1)} - \frac{x+1}{x^2+1} + \frac{5}{3(x+1)}$$

```
> p3 := x^6+9*x^5+30*x^4+5*x^3+35*x^2+4*x+10;
```

$$p3 := x^6 + 9x^5 + 30x^4 + 5x^3 + 35x^2 + 4x + 10$$

```
> p4 := 1/(1/a+c/(a*b))+(a*b*c+a*c^2)/(b+c)^2;
```

$$\frac{1}{\frac{1}{a}+\frac{c}{ab}}+\frac{abc+ac^2}{(b+c)^2}$$

```
> simplify(p4);
```

$$a$$

```
> p5 := (sin(x)+sin(3*x)+sin(5*x)+sin(7*x))/
       (cos(x)+cos(3*x)+cos(5*x)+cos(7*x))-tan(4*x);
```

$$\frac{\sin(x)+\sin(3x)+\sin(5x)+\sin(7x)}{\cos(x)+\cos(3x)+\cos(5x)+\cos(7x)}-\tan(4x)$$

```
> simplify(p5);
```

$$0$$

■

In Example 4, the initial two prompts involve assigning a polynomial to a variable, `p1`, using the "`:=`" operator, followed by factoring it into irreducible factors over the rational numbers. Subsequently, the third and fourth prompts involve inputting a rational polynomial and determining its partial fraction decomposition.
In the example below, we demonstrate differentiation operation using the prompt command `diff`, followed by integration using the `int` command. Notably, the output of the `int` operator lacks the arbitrary constant of integration. In the fifth prompt, we assign a first-order differential equation to the variable `p7`. In this prompt result, it can be seen that MAPLE® denotes the derivative of an unknown function $y(x)$ using the total derivative symbol "d". Subsequently, we request MAPLE® to solve the differential equation at the sixth prompt. The presence of the symbol `C1` in the solution indicates MAPLE®'s inclusion of an arbitrary constant. This means that MAPLE® includes an arbitrary constant in the solution of a differential equation but does not include the arbitrary constant for an anti-differentiation.

Example 5 *(Symbolic operations from calculus and differential equations in* MAPLE®*).*

```
> p6 := cos(4*x + 3)/(x^2 + 1);
```

$$p6 := \frac{\cos(4x+3)}{x^2+1}$$

```
> diff(p6, x);
```

$$-\frac{4\sin(4x+3)}{x^2+1}-\frac{2x\cos(4x+3)}{(x^2+1)^2}$$

```
> p7 := cos(x)/(sin(x)^2 + 6*sin(x) + 4);
```

$$p7 := \frac{\cos(x)}{\sin(x)^2 + 6\sin(x) + 4}$$

```
> int(p7, x);
```

$$-\frac{\sqrt{5}}{5}\operatorname{arctanh}\left(\frac{(2\sin(x)+6)\sqrt{5}}{10}\right)$$

```
> p8 := diff(y(x), x) + 3*y(x) = x^2 + sin(x);
```

$$\frac{\mathrm{d}}{\mathrm{d}x}y(x) + 3y(x) = x^2 + \sin(x)$$

```
> dsolve(p8, y(x));
```

$$y(x) = \frac{1}{3}x^2 - \frac{2}{9}x + \frac{2}{27} - \frac{\cos(x)}{10} + \frac{3\sin(x)}{10} + \mathrm{e}^{-3x}\_C1$$

■

The term computer algebra language or symbolic programming language refers to the programming language used to interact with a CAS, as illustrated in Examples 4 and 5. Most CASs offer both programming and interactive modes. In programming mode, mathematical operators like `factor` and `simplify` are combined with standard programming *functions* or *procedures* to create programs capable of solving complex mathematical problems. To illustrate this concept, let us examine the task of determining the equation for the tangent line to the curve

$$y = f(x) = x^2 + 5x + 6$$

at the point $x = 2$. Initially, we derive a general formula for the slope through differentiation. Subsequently, we substitute $x = 2$ into this expression to obtain the slope at that specific point,

$$m = \frac{dy}{dx}(2) = 9\,.$$

Using the point-slope form for a line, the equation for the tangent line is derived as

$$\begin{aligned} y = m(x-2) + f(2) &= 9(x-2) + f(2) \\ &= 9x + 8\,. \end{aligned}$$

The final formula is obtained by expanding the right side of this last equation. This process can be automated using a CAS programming language, as demonstrated in the following examples.

Example 6 *(Implementation of the* **`TangentLine`** *procedure in* MAPLE®*).*

```
TangentLine := proc(f, x, a)
 local m, l;
 m := subs(x = a, diff(f, x));
 l := expand(m*(x - a) + subs(x = a, f));
 return l;
end proc:
```

■

Example 7 *(Execution of the* **`TangentLine`** *procedure in* MAPLE®*).*

```
> TangentLine(x^2 + 5*x + 6, x, 2);
```

$$9x + 2$$

■

In Example 6, we provide a general procedure written in the MAPLE® computer algebra language, which performs computations for the tangent line. This procedure calculates the tangent line formula for any function f at the point $x = a$. It utilizes the `diff` operator for differentiation and the `subs` operator for substitution. Additionally, the `expand` operator is used to simplify the output. Once this procedure is entered into the MAPLE® system, it can be accessed from the system's interactive mode, as shown in Example 7.

3.2 PURPOSES AND APPLICATIONS OF COMPUTER ALGEBRA

In their influential work "Mathematics Applied to Deterministic Problems in the Natural Sciences" ([97], SIAM, 1988, pages 5-7), Lin and Segel delineate the objectives of applied mathematics: applied mathematics aims to clarify scientific concepts and depict scientific phenomena using mathematical tools, fostering the advancement of mathematics through such endeavors. They discuss three fundamental aspects of this process concerning the resolution of scientific challenges.

1. Formulating scientific problems in mathematical terms, which involves translating real-world scientific problems into mathematical language, enabling the application of mathematical tools for analysis and solution.
2. Solving the mathematical problems thus formulated using appropriate mathematical techniques, ranging from algebraic manipulation to differential equations and numerical methods.
3. Interpreting the solutions and empirically verifying them in the context of the original scientific problem and validated through empirical observation or experimentation.

The authors of [97], also emphasize the interconnected nature of this process.

> 4 Generating scientifically relevant new mathematics by engaging in the formulation, generalization, abstraction, and axiomatic formulation of mathematical concepts and methods, applied mathematics contributes to the development of new mathematical theories and techniques that are pertinent to scientific inquiry.

While CASs theoretically have the potential to facilitate steps (1), (2), and (4) of this process, in practice, they primarily focus on step (2), *i.e.*, the actual solving of mathematical problems. Their role in steps (1) and (4) is comparatively limited, serving more as tools for computation rather than for the conceptualization or creation of new mathematics. Specifically, CASs are invaluable tools in scientific research for performing complex calculations, solving equations, and verifying mathematical identities, especially in cases where the calculations are too complex or tedious to be performed by hand. In the subsequent part of this section, we present some examples showcasing the application of computer algebra software in the problem-solving process. All of these are simple examples that are only meant to illustrate the main capabilities of CASs, and are not intended to be exhaustive.

Example 8 ***(Solution of a Linear System of Equations)***. Suppose we have the following system of equations

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

We want to find the symbolic solution for x and y. We can use MAPLE® CAS to solve this system symbolically.

```
# Define the system of equations
> eq1 := 3*x + 2*y = 10;
> eq2 := 4*x - 5*y = 20;

# Solve the system symbolically
> sol := solve({eq1, eq2}, {x, y});
```

In this code, we first define the system of equations `eq1` and `eq2`. We then use the `solve` function to find the symbolic solution for x and y and store the symbolic solution in the variable `sol`. ■

Example 9 ***(Optimization of a Multivariable Function)***. Let us consider a more complex example involving a multivariable function. We minimize the Rosenbrock function, which is a commonly used test function for optimization algorithms.

```
# Define the Rosenbrock function
```

```
> rosenbrock := (x,y) -> (1-x)^2+100*(y-x^2)^2;

# Use optimization package to minimize the Rosenbrock function
> sol := Optimization:-Minimize(rosenbrock(x,y), {x=-2..2, y=-1..3});
```

In this example, we define the Rosenbrock function $f(x,y) = (1-x)^2 + 100(y - x^2)^2$, and then use MAPLE®'s optimization function `Optimization:-Minimize` to find the minimum value and the corresponding point of (x,y). The result, stored in the variable `sol`, gives us the minimum value of the function and its corresponding point at which it occurs (the minimum point is $f(x,y) = 0$ at $(x,y) = (1,1)$). ■

Example 10 ***(Solution of Ordinary Differential Equations).*** CASs can also be used to obtain closed-form solutions of ODEs. For example, consider the following second-order ODE representing a ball bouncing on a flat surface with a coefficient of restitution of 7/10. The ODE is given by

$$\begin{cases} y''(t) = -\dfrac{981}{100} \\ y(0) = 5, \quad y'(0) = 0 \\ \text{WhenEvent}(y(t) = 0, y'(t) = -\dfrac{7}{10}y'(t)) \end{cases} \qquad \text{with} \quad t \in [0,5]\,.$$

We can use MATHEMATICA® to solve this ODE symbolically and obtain the closed-form solution for $y(t)$

$$y(t) = \begin{cases} 5 - \dfrac{981}{200}t^2 & 0 \le t \le \dfrac{10\sqrt{\frac{10}{109}}}{3} \\ \dfrac{3}{200}\left(-327t^2 + 34\sqrt{1090}t - 800\right) & \dfrac{10\sqrt{\frac{10}{109}}}{3} < t \le 8\sqrt{\dfrac{10}{109}} \\ \dfrac{3\left(-1635t^2 + 289\sqrt{1090}t - 13520\right)}{1000} & 8\sqrt{\dfrac{10}{109}} < t \le \dfrac{169\sqrt{\frac{2}{545}}}{3} \\ \dfrac{-49050t^2 + 11169\sqrt{1090}t - 687154}{10000} & \dfrac{169\sqrt{\frac{2}{545}}}{3} < t \le \dfrac{2033}{15\sqrt{1090}} \\ \dfrac{3\left(-817500t^2 + 215305\sqrt{1090}t - 15404041\right)}{500000} & \dfrac{2033}{15\sqrt{1090}} < t \le \dfrac{7577}{50\sqrt{1090}} \\ \dfrac{3\left(-81750000t^2 + 23571350\sqrt{1090}t - 1849674509\right)}{50000000} & \dfrac{7577}{50\sqrt{1090}} < t \le \dfrac{244117}{1500\sqrt{1090}} \\ \dfrac{-24525000000t^2 + 7499983500\sqrt{1090}t - 624651217823}{5000000000} & \dfrac{244117}{1500\sqrt{1090}} < t \le 5 \end{cases}.$$

Even more complex ODEs can be solved using CASs for design optimization, control systems, and other engineering applications. For example, MATHEMATICA® can find the symbolic solution for a proportional-derivative controller keeping the position of

a moving mass $x(t)$ constant through the control input $u(t)$, given by

$$\begin{cases} x''(t) = u(t) \\ x(0) = x'(0) = 0 \\ u(0) = 1 \\ \text{WhenEvent}(\text{mod}(t,\tau) = 0, u(t) = (1 - x(t) - x'(t))) \end{cases} \quad \text{with} \quad t \in [0, 15]\,.$$

The solution for $x(t)$ and $u(t)$ is respectively given by

$$x(t) = \begin{cases} \frac{t^2}{2} & 0 \le t \le 1 \\ \frac{1}{4}\left(-t^2 + 6t - 3\right) & 1 < t \le 2 \\ -\frac{3t^2}{8} + 2t - \frac{5}{4} & 2 < t \le 3 \\ \frac{1}{16}\left(-t^2 + 2t + 25\right) & 3 < t \le 4 \\ \frac{1}{32}\left(5t^2 - 52t + 162\right) & 4 < t \le 5 \\ \frac{1}{64}\left(7t^2 - 74t + 249\right) & 5 < t \le 6 \\ \frac{1}{128}\left(-3t^2 + 56t - 114\right) & 6 < t \le 7 \\ \frac{1}{256}\left(-17t^2 + 266t - 767\right) & 7 < t \le 8 \\ \frac{1}{512}\left(-11t^2 + 164t - 62\right) & 8 < t \le 9 \\ \frac{23t^2 - 482t + 3521}{1024} & 9 < t \le 10 \\ \frac{45t^2 - 944t + 6942}{2048} & 10 < t \le 11 \\ \frac{-t^2 + 114t + 2873}{4096} & 11 < t \le 12 \\ \frac{-91t^2 + 2364t - 7070}{8192} & 12 < t \le 13 \\ \frac{-89t^2 + 2310t + 1577}{16384} & 13 < t \le 14 \\ \frac{93t^2 - 2968t + 56270}{32768} & 14 < t \le 15 \end{cases}, \quad \text{and} \quad u(t) = \begin{cases} 1 & 0 \le t \le 1 \\ -\frac{1}{2} & 1 < t \le 2 \\ -\frac{3}{4} & 2 < t \le 3 \\ -\frac{1}{8} & 3 < t \le 4 \\ \frac{5}{16} & 4 < t \le 5 \\ \frac{7}{32} & 5 < t \le 6 \\ -\frac{3}{64} & 6 < t \le 7 \\ -\frac{17}{128} & 7 < t \le 8 \\ -\frac{11}{256} & 8 < t \le 9 \\ \frac{23}{512} & 9 < t \le 10 \\ \frac{45}{1024} & 10 < t \le 11 \\ -\frac{1}{2048} & 11 < t \le 12 \\ -\frac{91}{4096} & 12 < t \le 13 \\ -\frac{89}{8192} & 13 < t \le 14 \\ \frac{93}{16384} & 14 < t \le 15 \end{cases}.$$

■

3.3 EXPRESSION SWELL

Over the past four decades, CAS have increasingly found applications in both teaching and research. The advantages of incorporating CAS into teaching methodologies have been extensively documented. For instance, in Stoutemyer [98], general benefits of CAS are outlined, while Pavelle [99] provides numerous examples where CAS quickly solves problems that were previously considered complicated or time-consuming to tackle by hand. However, Mitic and Thomas [100] highlight an essential precondition for the effective and successful CAS utilization: users must be aware of potential pitfalls

and limitations, across all mathematical proficiency levels. Besides software bugs, the primary source of performance deterioration of CASs is the *expression swelling*.
The expression swell is a common phenomenon of exact computations in which the size of numbers and expressions involved in a calculation grows dramatically as the calculation progresses, thereby slowing down the execution. While symbolic calculation software like MAPLE® and MATHEMATICA® can handle substantial symbolic expressions, the growth of expression size during manipulation severely degrades the CASs performance, resulting in prohibitively long Central Processing Unit (CPU) times. During symbolic manipulation, it is not uncommon to find computations that are executed just once; once a result is obtained, recalculating it becomes unnecessary. However, the time required to obtain a result is often unknown. Predicting the memory and CPU time prerequisites for a given calculation poses challenges since the size of expressions generated during computation is known only after the computation is completed. For these reasons, expression swell consistently poses a challenge, emerging as a major source of CAS faults. It manifests in two forms: *inherent* expression swell and *intermediate* expression swell, each of which is discussed in the following sections.

3.3.1 INHERENT EXPRESSION SWELL

Inherent expression swell occurs when a calculation generates large expressions as a result of the problem itself. For example, the solution of a large system of linear equations can lead to large expressions. In this case, the problem is the large number of variables and equations, and the large expressions are a natural consequence of the problem. This type of expression swell is difficult both to mitigate and to spot, as it can arise from any problem, without any specific pattern.

Example 11 ***(Inherent expression swell).*** As an example, consider the following Hankel matrix

$$\begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix},$$

which has the following characteristic polynomial

$$\begin{aligned} c(\lambda) &= \lambda^9 + \lambda^8 - 40\lambda^7 - 24\lambda^6 + 240\lambda^5 + 144\lambda^4 \\ &= \lambda^4(\lambda + 6)(\lambda^4 - 5\lambda^3 - 10\lambda^2 + 36\lambda + 24)\,. \end{aligned}$$

The Hankel matrix has four zero eigenvalues $\lambda_{1,2,3,4} = 0$, one eigenvalue is $\lambda_5 = -6$, and the other four eigenvalues are roots of the seemingly simple-looking quartic

polynomial

$$\lambda^4 - 5\lambda^3 - 10\lambda^2 + 36\lambda + 24\,,$$

which are

$$\lambda_{6,7} = \frac{5}{4} \pm \frac{\sqrt{3}}{12}\sqrt{\frac{8a^2+155a+1856}{a}} + \frac{\sqrt{6}}{12}\sqrt{\frac{155a-4a^2-928}{a} \pm \frac{111\sqrt{3}}{\sqrt{\frac{8a^2+155a+1856}{a}}}}\,,$$

$$\lambda_{8,9} = \frac{5}{4} \pm \frac{\sqrt{3}}{12}\sqrt{\frac{8a^2+155a+1856}{a}} + \frac{\sqrt{6}}{12}\sqrt{\frac{155a-4a^2-928}{a} \pm \frac{666\sqrt{3}}{\sqrt{\frac{8a^2+155a+1856}{a}}}}\,,$$

where $a = \sqrt[3]{3142 + 18i\sqrt{8071}}$. ■

3.3.2 INTERMEDIATE EXPRESSION SWELL

Intermediate expression swell is an important special case of expression swell in which, during the middle stages of a calculation, intermediate expressions' size can grow substantially, along the way to a possibly and comparatively simple final results of the calculation.

Example 12 ***(Intermediate expression swell).*** Let us verify the Bianchi identity for a symmetric connection

$$K^{\ell}_{j\,bk;p} + K^{\ell}_{j\,kp;j} + K^{\ell}_{j\,ph;k} = 0\,,$$

where K is the Riemann curvature tensor. Expanding the left-hand side in terms of Christoffel symbols of the second kind, one obtains a sum of 72 terms, each of which is a product of 2 or 3 Christoffel symbols, for a total of 180 Christoffel symbols. However, upon simplifying this expression by consistently renaming the dummy indices, the simple result of zero is obtained, which verifies the identity. ■

3.3.3 MITIGATION STRATEGIES

Although memory space, rather than time, is the main factor limiting the use of computer algebra, symbolic operations will take considerably longer time than their numerical counterparts. It should be borne in mind that, in the case of symbolic manipulations, the execution time is strongly dependent, once again, on the degree of complexity and size of the input. As mentioned earlier, the production of large expressions during the computation in the form of inherent but especially as intermediate expression swell is, as stated in Noor and Andersen [101] a serious problem in symbolic computation and maybe its ultimate limitation. There, it was anticipated that future symbolic manipulation systems would automatically carry out remedial actions to alleviate this problem such as

- recognition of common sub-expressions in an expression and renaming them by a single parameter;
- handling more expressions in a factored form rather than in an expanded form;
- deferred expansion of a function or variable in an expression.

On the other hand, it is interesting that while Korncoff and Fenves [102] also recognizes this problem by stating that special problem formulation techniques will have to be adopted in light of symbolic manipulation Korncoff and Fenves place the onus of its solution (or, at least, alleviation) on the user rather than the system, saying that as the use of symbolic processors increases, users will have to acquire the skills and insight required to formulate problems to best optimize the function of a particular processor. Up to now, the problem of expression swell has not yet been solved consistently. However, as we will see in the next section, some strategies have been introduced recently to mitigate the problem.

3.4 HIERARCHICAL REPRESENTATION

To mitigate the expression swell, one may use hierarchical representation techniques [103, 104].

Definition 1 ***(Hierarchical Representation [105]).*** *A hierarchical representation over a generic domain $\mathbb{K}$ and a set of n independent variables $\{x_1, \ldots, x_n\}$ is an ordered list $[v_1, v_2, \ldots, v_m]$ of m symbols, together with an associated list $[d_1, d_2, \ldots, d_m]$ of definitions of the symbols. For each v_i, with $i \geq 1$, the definition d_i is of the form $d_i = f_i(\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,k_i})$ where $f_i \in \mathbb{K}[\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,k_i}]$, and each $\sigma_{i,j}$ is either a symbol in $[v_1, v_2, \ldots, v_m]$ or an expression in the independent variables.* ■

Hence, the main idea behind this hierarchical representation tool is to *veil* complicated expressions from the user by using auxiliary variables called *veil variables* or *veilings* $[v_1, v_2, \ldots, v_m]$, and to *unveil* them by reapply their definitions, *i.e.*, $v_1 = d_1$, $v_2 = d_2$, …, $v_m = d_m$ only when it is strictly necessary. In other words, the veil variables are used to represent the complicated expressions in a compact form, while the actual size of the expressions is hidden in the veil variables. Algorithm 1 describes the veiling procedure, which is the core of the hierarchical representation technique.

Algorithm 1 Veil an expression.

```
1: Require: An expression e, and the optional expression dependencies x
2: procedure VEIL(e)                                   ▷ Veil an expression
3:     c ← Normalizer(e)                  ▷ Transform e into factored normal form
4:     if ExpressionCost(e) > m then      ▷ Check if c complexity is above the threshold m
5:         return c                       ▷ Return the expression in factored normal form
6:     end if
7:     i ← IntegerContent(e)              ▷ Retrieve the integer content without sign
8:     s ← Sign(c)                                ▷ Store the symbolic sign of c
```

```
 9:    if si = c then                          ▷ Check if the expression is a constant
10:        return c                            ▷ Return the expression in factored normal form
11:    else
12:        v ← StoreVeil(sc/i)                 ▷ Store the veiled expression and return the veiling symbol
13:        return siv(x)                       ▷ Return the veiled expression with its dependencies
14:    end if
15: end procedure
```

Despite the simplicity of the idea, the implementation of the hierarchical representation is not straightforward. The main difficulty is to choose the right moment to veil and unveil the veil variables. Indeed, the user must decide when to do it by experience, and this decision has neither theoretical background nor specific rules to follow. As a rule of thumb, the expression veiling should occur only when they become just too large for further calculations. As a result, a metric of expression complexity and size must be introduced to choose the right moment to veil and unveil the veil variables.

3.4.1 EXPRESSION COMPLEXITY METRIC

Choosing the complexity and size boundaries between intermediate expressions appropriately is of utmost importance in order not to produce too much expression swelling, but also not to have too many veiling levels. Therefore, a metric for measuring expression proneness to swell must be introduced first. It is important to note that the concepts of expression complexity and size are not synonymous, yet, they are both closely related to the expression swell phenomenon. For these reasons, more than one measure to quantify the complexity and size of an expression could be given.
There exist two main metrics to measure the complexity and size of an expression, each with its own advantages and disadvantages.

- The *length* of an expression, in terms of the number of characters used to internally represent the expression. A possible measure that exploits the length of an expression is the following

$$N_\beta = \begin{cases} \lfloor \log_\beta(|n|) \rfloor + 1 & n > 0 \\ 0 & n = 0 \end{cases},$$

  where $n \in \mathbb{Z}$ is a non-negative integer representing the expression length, and $\beta \in \mathbb{N}$ is the base in which the length is measured. Notably used in Carette, Zhou, Jeffrey, and Monagan [103] and Zhou, Carette, Jeffrey, and Monagan [104], this metric is calculated through the `length` MAPLE® function. This metric is very helpful in understanding the amount of memory space required to store expressions, as well as the CPU effort required to write, process, and read them. However, it does not provide any information about the operands and operations involved in the expression.

- The *computational cost* of an expression, calculated through the `cost` function of `codegen` package. This metric is somehow complementary to the previous one,

and it provides insights into the computational cost of the expressions. Conversely, it does not provide any information about the amount of memory space required to store an expression.

It is evident that it is not possible to use both metrics at the same time, as they are neither directly comparable nor convertible to each other. Hence, the choice of the metric to use is strictly dependent on the specific problem to solve.
Previous works [103, 104] on symbolic linear algebra have shown that the hierarchical representation of expressions is applied to matrix factorization tasks, and it has been proven to be effective in mitigating the expression swell problem. Nonetheless, the LULEM package [103], which implements large expression management strategies in LU decomposition, significantly outperforms the MAPLE®'s built-in matrix factorization routines. Instead, throughout this thesis we use the computational cost. As demonstrated in the following example, the latter metric is insensitive to the number of characters used to represent the expression itself and guarantees better control of the final expression size, regardless of the variables' names. Nonetheless, the computational cost also provides us with a better prediction of both the computational effort and growth of the expression size during a given symbolic operation.

Example 13 ***(Expression size and complexity calculation).*** Let us consider two algebraically equivalent expressions stored in `expr_1` and `expr_2` variables.

```
> expr_1 := (x^2+y^2)^2/g(x)-z/f(x):
> expr_2 := (x_tmp^2+y_tmp^2)^2/g_tmp(x)-z_tmp/f_tmp(x):
```

If we respectively calculate the expression complexity calculation through the `length` and `codegen:-cost` functions we obtain the following results.

```
> map(length, <expr_1, expr_2>);
```

$$\begin{bmatrix} 53 \\ 73 \end{bmatrix}$$

```
> map(codegen:-cost, <expr_1, expr_2>);
```

$$\begin{bmatrix} 3\text{multiplications} + 2\text{additions} + 2\text{divisions} + 2\text{functions} \\ 3\text{multiplications} + 2\text{additions} + 2\text{divisions} + 2\text{functions} \end{bmatrix}$$

As it can be seen, the `length` function is sensitive to the characters that MAPLE® internally uses to represent the expression. Conversely, the `codegen`'s `cost` calculates the actual computational complexity of the two expressions and returns the same result. ■

3.4.2 LARGE EXPRESSION MANAGEMENT

There exist specific modules to perform large expression management tasks and help the user handle hierarchical representations [103, 105]. The MAPLE® module `LargeExpressions` already does this job. However, from the authors' perspective, it

has some minor limitations given by the chosen user interface rather than the underlying idea or the adopted programming technique. For this reason, the authors have reinterpreted it to a new object-oriented LEM package [23]. The new version does not differ much from its original version, but it allows more effective control and straightforward use of the veiling variables. The object-oriented feature also allows for the creation of multiple instances of LEM objects, giving the ability to sharply separate veiling variables that could lead to conflicts if improperly used. Here an example is given to briefly illustrate the capabilities of the large expression management technique and, particularly, of the LEM module.

Example 14 ***(Large expression management with LEM [23]).*** Let us consider a random polynomial `p` generated by the `randpoly` function.

```
> p := randpoly([x,y,z], degree=3, dense):
```

Then, we create a LEM object instance and set the veiling label to `X`.

```
> LEM_obj := Object(LEM):
> LEM_obj:-SetVeilingLabel('X'):
```

The expressions are veiled to get a more compact hierarchical representation.

```
> p_X := collect(p, x, i -> LEM_obj:-Veil(i));
```

$$p\_X := -7x^4 + (2, y - 55z - 94)x^3 + X[1]x^2 - X[2]x$$

The veiling variables are stored in the LEM object and can be used to unveil the expression whenever necessary.

```
> <LEM_obj:-VeilList()>;
```

$$\begin{bmatrix} X[1] = 87y^2 - 56yz - 62z^2 + 97z - 73 \\ X[2] = 4y^3 + 83y^2z - 62yz^2 + 44z^3 + 10y^2 + 82yz - 71z^2 - 80y + 17z + 75 \end{bmatrix}$$

In addition to the few functions just presented, LEM allows to customize the strategy and parameters used to control the veiling process. For further information on the LEM package refer to the documentation in [23]. ■

3.4.3 SIGNATURE OF A HIERARCHICAL REPRESENTATION

In order to *zero-test* for an expression in hierarchical representations, we use the probabilistic approach of *signatures* [106], which relies on the *DeMillo-Lipton-Schwarz-Zippel lemma* [107–109]. Signature functions verify the presence of equivalent expressions within thousands of sub-expressions through hashing techniques [110–113]. In MAPLE®, each expression is stored in the simplification table using its signature as a key. The signature of an expression is itself a hashing function, with one very important feature: equivalent expressions have identical signatures. In other words, the signature of an expression is a unique identifier that is computed from the expression itself.
On the contrary of [103, 105], we do not implement the signature function as a separate module. Instead, we employ the MAPLE®'s `signature` function to exploit the latest

improvements in the symbolic computation engine. Given for granted that the signature function is a hashing function, it is possible to compute the define the signature of a hierarchical representation as follows.

Definition 2 ***(Signature of a Hierarchical Representation [105]).*** *Let H denote a hierarchical representation with n independent variables $\{x_1, \ldots, x_n\}$, expressed as lists of symbols $[v_1, v_2, \ldots, v_m]$ and definitions $[d_1, d_2, \ldots, d_m]$, where $v_i = f_i(\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,k_i})$ and $\sigma_{i,j}$ represents an expression in the independent or hierarchical variables. Take p as a prime number. The signature of v_i is recursively defined as follows.*

- *We define $s(v_i) = f_i(\delta_{i,j}, \ldots, \delta_{i,k_i}) \bmod p$.*
- *If $\sigma_{i,j}$ is an expression in the independent variables only, then $\delta_{i,j} = s(\sigma_{i,j}, p)$.*
- *If $\sigma_{i,j}$ is an expression in $[v_1, v_2, \ldots, v_i - 1]$, then we necessarily have $i > 1$. Let $h_{i,j} \in [1, \ldots, i-1]$ such that $\sigma_{i,j} = v_{h_{i,j}}$, then $\delta_{i,j} = s(\sigma_{i,j}, p) = s(v_{h_{i,j}}, p)$, which is known by induction assumption.*

Thereby, the signature of H is $s(H) = [s(v_1), s(v_2), \ldots, s(v_m)]$. ■

The signature of the expression is computed before veiling an expression in hierarchical representation. This value then becomes the signature of the veiling symbol. When that symbol itself appears in an expression to be veiled, the signature of the symbol is used in the calculation of the new signature. In particular, it is not necessary to unveil any symbol in order to compute its signature. The main advantages of using such a technique are that it is fast, flexible, and can be adapted to different applications. Moreover, hierarchical representations can often lead to a more compact and elegant output, and the code can solve a wider class of problems and often "reduces" intermediate expression swell.

3.4.3.1 EXTENDING SIGNATURES CALCULATION

Calculating the signature hashing function of any expression in polynomial time is not always possible. In particular, trigonometric functions can present an obstacle to the signature computation. However, it is possible to use ad hoc *coordinate changes* to transform trigonometric expressions into polynomials of which the signature computation can be computed with standard techniques. Nonetheless, the transformation into polynomials may facilitate the expression simplification, hence leading to a more compact and less swelling-prone output. In particular, for many multi-body applications, the only independent variable is time. Depending on the modeling choices, such DAE systems may be made of trigonometric polynomials of the angles between different components. As shown in [114], one common method to convert such systems to rational form is the transformation

$$\cos(\theta) = x, \quad \sin(\theta) = y, \quad \text{where} \quad x^2 + y^2 = 1. \tag{3.1}$$

which has the disadvantage that an additional constraint is introduced. Another useful change of coordinates is the Weierstraß transformation [115]

$$\cos(\theta) = \frac{1-u^2}{1+u^2}, \quad \sin(\theta) = \frac{2u}{1+u^2}, \quad \text{where} \quad u = \tan\left(\frac{\theta}{2}\right). \tag{3.2}$$

If one solves for u, then the usual problem regarding the choice of an appropriate branch for the inverse arises. Still, this transformation has the advantage that the number of variables remains the same, with no additional introduced constraint. Please notice that the signature computation used in this work is implemented within the SIG sub-package of LEM [23]. The SIG sub-package is an improved object-oriented version of the signature computation package present in LULEM [103].

3.5 SYMBOLIC MATRIX FACTORIZATION

As previously mentioned, matrix factorization is a widely employed technique for addressing linear systems. There are several types of decompositions, each with distinct properties and characteristics. In the context of purely numerical matrices, the practice aligns well with the theoretical foundations of the algorithm. However, when dealing with matrices consisting of either symbolic or mixed symbolic-numeric entries, the situation becomes more intricate [105]. In exact symbolic linear algebra scenarios, the cost of each operation during factorization can vary due to uncontrolled expression swell [104]. Furthermore, the presence of symbolic values hinders the guarantee of numerical stability. Consequently, a key objective is to derive an output format that retains the symbolic structure of the input matrix and ensures numerical stability. Nonetheless, in symbolic linear algebra, much effort has been devoted to controlling the growth of expression size by developing. Very little work has been done on the guarantee of numerical stability. The main reason is that the techniques that are helpful for the numeric case are often unstable or impractical for the purely symbolic case, ending in the pivoting on small quantities and resulting instability.
In this section, we will focus on the full-pivoting LU and FFLU decompositions, which are the fundamental techniques that will be used in the next chapters, both for the large linear system solution and DAEs index reduction. It is important to note that, the LU and FFLU are preferred in symbolic linear algebra over the QR and Gauss-Jordan (GJ), as LU decomposition involves simpler operations, thereby mitigating the issue of expression swell. Additionally, as we will see, employing the LU factorization with a minimum degree pivoting strategy proves superior in reducing fill-in and reducing the subsequent numeric and symbolic computational cost.

3.5.1 THE FULL-PIVOTING LOWER-UPPER FACTORIZATION

The full-pivoting LU and FFLU decompositions are widely used algorithms for solving linear systems with minimal computational effort. They are defined as follows.

Definition 3 ***(Full-Pivoting LU Decomposition).*** *Given a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$*, with* $m \geq n$*, the full-pivoting LU decomposition is defined as the process of decomposing* $\mathbf{A}$ *into the product of*

- *a* $\mathbf{L} \in \mathbb{R}^{m \times m}$ *lower-triangular matrix with all diagonal entries equal to* 1*;*
- *a* $\mathbf{U} \in \mathbb{R}^{m \times n}$ *upper-triangular matrix;*
- *a* $\mathbf{P} \in \mathbb{R}^{m \times m}$ *and a* $\mathbf{Q} \in \mathbb{R}^{n \times n}$ *matrices for rows and columns permutation, respectively;*

such that $\mathbf{PAQ} = \mathbf{LU}$. ■

Definition 4 ***(Full-Pivoting FFLU Decomposition).*** *Given a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$*, with* $m \geq n$*, the full-pivoting FFLU decomposition is defined as the process of decomposing* $\mathbf{A}$ *into the product of*

- *a lower-triangular matrix* $\mathbf{L} \in \mathbb{R}^{m \times m}$ *with all diagonal entries equal to* 1*;*
- *a diagonal matrix* $\mathbf{D} \in \mathbb{R}^{m \times m}$*;*
- *an upper-triangular matrix* $\mathbf{U} \in \mathbb{R}^{m \times n}$*;*
- *a* $\mathbf{P} \in \mathbb{R}^{m \times m}$ *and a* $\mathbf{Q} \in \mathbb{R}^{n \times n}$ *matrices for rows and columns permutation, respectively;*

such that $\mathbf{PDAQ} = \mathbf{LU}$. ■

Pivoting for LU factorization is the process of systematically selecting pivots for Gaussian elimination during the LU factorization of a matrix. The LU factorization is closely related to Gaussian elimination, which is unstable in its pure form. To guarantee the elimination process goes to completion, we must ensure that there is a non-zero pivot at every step of the elimination process. This is the reason we need to pivot when computing LU factorization. But we can do more with pivoting than just making sure Gaussian elimination is completed. To ensure the numerical stability of the LU, we need to consider that the domain of the entries of the matrix $\mathbf{A}$ is not only the real number set but also the more generic symbolic domain. Hence, we need to consider the symbolic stability of the LU factorization from two different perspectives: the numerical one and the symbolic one.

3.5.1.1 THE LOWER-UPPER FACTORIZATION FROM A NUMERICAL COMPUTATION POINT OF VIEW

From a numerical perspective, we can reduce round-off errors during computation and improve the algorithm *backward stability* by implementing the right pivoting strategy. Depending on the matrix $\mathbf{A}$, some LU decompositions may become numerically unstable if either numerically small pivots or symbolically zero pivots are used. Relatively small pivots cause instability because they operate very similar to zeros during Gaussian elimination. Through the process of pivoting, we can greatly reduce this instability

by ensuring that we use the largest available entry as our pivot elements. This prevents large factors from appearing in the computed $\mathbf{L}$ and $\mathbf{U}$, which reduces round-off errors during computation. The following example illustrates the importance of pivoting in the LU factorization.

Example 15 ***(Backward stability of LU factorization).*** When a calculation is undefined because of a division by zero, the same calculation will suffer numerical difficulties when there is a division by a non-zero number that is relatively small. Given the following matrix $\mathbf{A} \in \mathbb{R}^{2\times 2}$,

$$\mathbf{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 2 \end{bmatrix},$$

when computing the factors $\mathbf{L}$ and $\mathbf{U}$, the process does not fail in this case because there is no division by zero.

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 2 - 10^{20} \end{bmatrix}.$$

When these computations are performed in floating point arithmetic, the number $2 - 10^{20}$ is not represented exactly but will be rounded to the nearest floating point number which we will say is -10^{20}. This means that our matrices are now floating point matrices $\mathbf{L}'$ and $\mathbf{U}'$ where

$$\mathbf{L}' = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \quad \mathbf{U}' = \begin{bmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{bmatrix}.$$

The small change we made in $\mathbf{U}$ to get $\mathbf{U}'$ shows its significance when we compute $\mathbf{L}'\mathbf{U}'$

$$\mathbf{L}'\mathbf{U}' = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix} \neq \mathbf{A},$$

thus when trying to solve a system such as $\mathbf{Ax} = \mathbf{b}$ using the LU factorization as the factors $\mathbf{L}'\mathbf{U}'$ suffer from a large error. This is a clear example of how the LU factorization can be numerically unstable when small pivots are used. ■

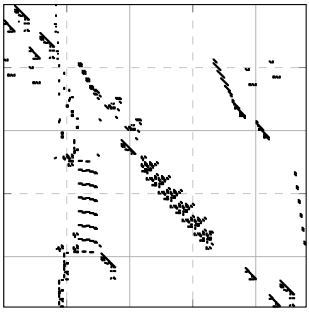
After the LU factorization of a sparse matrix $\mathbf{A}$, it is common to observe that the joint non-zeroes pattern of $\mathbf{L}$ and $\mathbf{U}$ exhibit either equal or lower sparsity compared to the original non-zero pattern of $\mathbf{A}$. The additional elements in $\mathbf{L}$ and $\mathbf{U}$ are known as the *fill-in*. This phenomenon diminishes performance as the number of non-zero elements in the $\mathbf{L}$ and $\mathbf{U}$ factors is directly related to the number of operations required to solve a linear system using the LU factorization. In other words, the more non-zero elements in the $\mathbf{L}$ and $\mathbf{U}$ factors, the more operations are required to solve a linear system. This is especially important when dealing with sparse matrices, where the number of non-zero elements in the $\mathbf{L}$ and $\mathbf{U}$ factors can be significantly reduced by using the right pivoting strategy. Indeed, specific reordering algorithms can be embedded in the pivoting strategy to minimize the fill-in of the factored matrix. These algorithms mainly

include *nested dissection* [116, 117] and *minimum degree* [118, 119] techniques. The following example illustrates the importance of pivoting in reducing fill-in during the LU factorization.

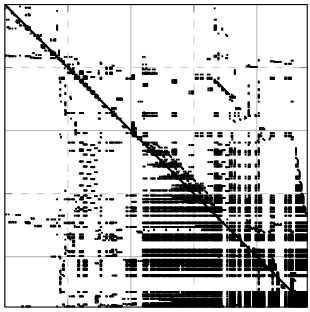
Example 16 ***(Fill-in reduction in LU factorization).*** Consider the `west0479` sparse unsymmetric matrix $\mathbf{A} \in \mathbb{R}^{479\times479}$, having non-zero 1887 entries [63]. The sparsity pattern of the original matrix $\mathbf{A}$ and the $\mathbf{L}$ and $\mathbf{U}$ factors with and without pivoting, using the minimum degree and nested dissection pivoting strategies, are shown in Figure 3.1. As illustrated, the LU factorization of $\mathbf{A}$ with pivoting has significantly less fill-in than the LU factorization of $\mathbf{A}$ without pivoting, thus highlighting the relevance of pivoting in reducing fill-in during the LU factorization. ■

3.5.1.2 THE LOWER-UPPER FACTORIZATION FROM A SYMBOLIC COMPUTATION STANDPOINT

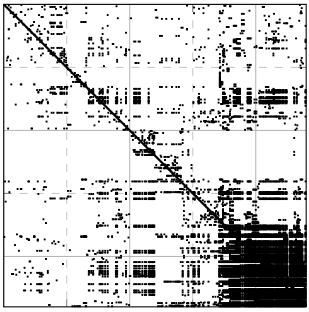
From a symbolic computation standpoint, *i.e.*, when the entries of the matrix $\mathbf{A}$ are not only real numbers but also symbolic expressions, the above considerations are not sufficient anymore. Indeed, in exact linear algebra, the cost of each operation during factorization can differ either for the fact that the expression size is not known a priori or for the uncontrolled expression swell [104]. Moreover, numerical stability is not guaranteed due to the presence of undefined values. Therefore, an important goal is to obtain an output format that both maintains the symbolic structure of the input matrix and is also stable when numerically evaluated.
Designing a pivoting strategy that is both numerically stable and symbolically viable in terms of expression growth is a challenging task. In this context, the main issue is that the choice of the pivot is not only related to the degrees of the matrix entries but also to the actual complexity of the expressions. Starting from the fill-in reduction, the minimum degree algorithm is preferred due to its ease of implementation. Conversely, the nested dissection is not yet considered as it involves working on the system's graph to identify graph separators, which is no easy task in the symbolic case. Secondly, the expression swell must be addressed both in terms of preventing the growth of the expression size and in terms of ensuring that the output symbolic code generated by the factorization is also numerically stable. The prevention of expression swell is achieved by the utilization of hierarchical representations with the aid of the LEM package, which employs the computational cost metric to control the expression size. The numerical stability of the symbolic code is ensured by choosing pivots that are both good in terms of their degrees and their actual computation complexity.
It is important to note that the expression may inevitably grow during the factorization process, thereby hindering the simplification of the expressions. This issue is mitigated by the *zero-testing* capabilities of the previously presented signature functions, which are used to verify the presence of null expressions without the need for simplification. The zero-testing is a crucial detail in symbolic pivoting as it strongly improves numerical stability allowing for the detection of null expressions in a very efficient way. Other



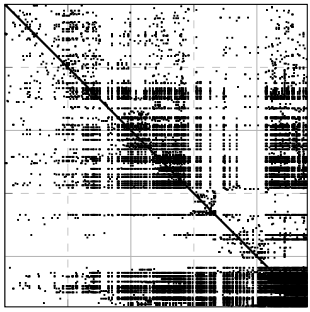
(a) Original sparsity pattern of matrix $\mathbf{A}$ (1887 non-zero elements).



(b) Sparsity pattern of $\mathbf{L}$ and $\mathbf{U}$ factors using standard pivoting (15918 non-zero elements).



(c) Sparsity pattern of $\mathbf{L}$ and $\mathbf{U}$ factors using minimum degree pivoting (13316 non-zero elements).



(d) Sparsity pattern of $\mathbf{L}$ and $\mathbf{U}$ factors using nested dissection pivoting (12216 non-zero elements).

Figure 3.1: Sparsity patterns of the original `west0479` matrix $\mathbf{A} \in \mathbb{R}^{479\times479}$ and the $\mathbf{L}$ and $\mathbf{U}$ factors with and without pivoting, using the minimum degree and nested dissection pivoting strategies.

than the zero-testing, the *hybrid symbolic-numerical* or *static pivoting approach* is also employed to validate the stability of symbolic code through random numerical evaluations. This approach may be computationally less efficient compared to the former method, but it yields satisfactory results. In other words, the "choice of pivots is numerically good at most numerical specializations" as emphasized in [120]. However, signature-based zero-testing is preferred as it is a proven and effective technique in previous successful symbolic linear algebra works [103, 105]. It is worth noting that, to the authors' knowledge, there is still no well-established rule for determining whether a *generic* expression is "likely" null. For this reason, this topic is still an open question.

Algorithm 2 Symbolic LU Factorization.

Require: A $m \times n$ matrix $\mathbf{A}$.
procedure LU($\mathbf{A}$, k) ▷ Symbolic full-pivoting LU procedure
  $\mathbf{M} \leftarrow \mathbf{A}$ ▷ Initialize the matrix $\mathbf{M}$
  $rnk \leftarrow \min(m, n)$ ▷ Initialize the rank of $\mathbf{M}$
  for k **from** 1 **to** rnk **do** ▷ Perform Gaussian elimination
    $p, q, l \leftarrow$ SymbolicPivoting($\mathbf{M}$, k) ▷ Find the best pivot for the k-th step
    if $p = 0$ **then** ▷ Check for null pivot
      $rnk \leftarrow k - 1$ ▷ The rank of $\mathbf{M}$ is $k - 1$
      break ▷ The matrix is singular
    end if
    $\mathbf{r}_k, \mathbf{c}_k \leftarrow q, l$ ▷ Store the pivot row and column indices
    $\mathbf{M} \leftarrow$ SwapRows($\mathbf{M}$, k, q) ▷ Swap the k-th and q-th rows
    $\mathbf{M} \leftarrow$ SwapColumns($\mathbf{M}$, k, l) ▷ Swap the k-th and l-th columns
    for i **from** $k + 1$ **to** m **do** ▷ Compute the k-th column of $\mathbf{L}$
      $M_{kk} \leftarrow$ Veil(M_{kk}) ▷ Veil the k-th pivot
      $M_{ik} \leftarrow$ Veil(Normalizer(M_{ik}/M_{kk})) ▷ Normalize the k-th pivot
      for j **from** $k + 1$ **to** n **do** ▷ Compute the k-th row of $\mathbf{U}$
        $M_{ij} \leftarrow$ Veil(Normalizer($M_{ij} - M_{ik}M_{kj}$)) ▷ Finalize the Schur complement
      end for
    end for
  end for
  $\mathbf{P}, \mathbf{Q} \leftarrow$ PermutationMatrices($\mathbf{r}$, $\mathbf{c}$) ▷ Compute the permutation matrices
  $\mathbf{L} \leftarrow$ LowerTriangular($\mathbf{M}$) ▷ Extract the lower-triangular part of $\mathbf{M}$
  $\mathbf{U} \leftarrow$ UpperTriangular($\mathbf{M}$) ▷ Extract the upper-triangular part of $\mathbf{M}$
  return $\mathbf{L}$, $\mathbf{U}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{r}$, $\mathbf{c}$, rnk ▷ Return the factors and the rank of $\mathbf{A}$
end procedure

Algorithm 3 Solve a square linear system $\mathbf{Ax} = \mathbf{b}$ using the LU factorization.

Require: The LU factors $\mathbf{L}$, $\mathbf{U}$, $\mathbf{P}$, $\mathbf{Q}$, and a vector $\mathbf{b}$.
procedure SolveLU($\mathbf{L}$, $\mathbf{U}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$) ▷ Solve the linear system $\mathbf{Ax} = \mathbf{b}$
  $\mathbf{y} \leftarrow \mathbf{Pb}$ ▷ Apply the permutation matrix $\mathbf{P}$ to the vector $\mathbf{b}$
  $m, n \leftarrow$ Size($\mathbf{L}$) ▷ Get the size of $\mathbf{L}$
  for i **from** 2 **to** m **do** ▷ Solve $\mathbf{Ly} = \mathbf{Pb}$
    $y_i \leftarrow \text{Veil}\left(y_i - \sum_{j=1}^{i-1} L_{ij}y_j\right)$ ▷ Perform forward substitution
  end for
  $x_n \leftarrow$ Veil(y_n/U_{nn}) ▷ Perform the first backward substitution
  for i **from** $n - 1$ **to** 1 **do** ▷ Solve $\mathbf{Ux} = \mathbf{y}$
    $x_i \leftarrow \text{Veil}\left(y_i - \sum_{j=i+1}^{n} U_{ij}x_j\right)$ ▷ Perform backward substitution
    $x_i \leftarrow$ Veil(x_i/U_{ii})
  end for
  $\mathbf{x} \leftarrow \mathbf{Q}^\top\mathbf{x}$ ▷ Apply the permutation matrix $\mathbf{Q}^\top$ to the solution $\mathbf{x}$
end procedure

Algorithm 4 Symbolic FFLU Factorization.

Require: A $m \times n$ matrix $\mathbf{A}$.
procedure FFLU($\mathbf{A}$, k) ▷ Symbolic full-pivoting FFLU procedure
  $\mathbf{M} \leftarrow \mathbf{A}$ ▷ Initialize the matrix $\mathbf{M}$
  $rnk \leftarrow \min(m, n)$ ▷ Initialize the rank of $\mathbf{M}$
  for k **from** 1 **to** rnk **do** ▷ Perform Gaussian elimination
    $p, q, l \leftarrow$ SymbolicPivoting($\mathbf{M}$, k) ▷ Find the best pivot for the k-th step
    if $p = 0$ **then** ▷ Check for null pivot
      $rnk \leftarrow k - 1$ ▷ The rank of $\mathbf{M}$ is $k - 1$
      break ▷ The matrix is singular
    end if
    $\mathbf{r}_k, \mathbf{c}_k \leftarrow q, l$ ▷ Store the pivot row and column indices
    $\mathbf{M} \leftarrow$ SwapRows($\mathbf{M}$, k, q) ▷ Swap the k-th and q-th rows
    $\mathbf{M} \leftarrow$ SwapColumns($\mathbf{M}$, k, l) ▷ Swap the k-th and l-th columns
    $\mathbf{D}_{kk} \leftarrow M_{kk}$ ▷ Veil the k-th pivot
    for i **from** $k + 1$ **to** m **do** ▷ Compute the k-th column of $\mathbf{L}$
      for j **from** $k + 1$ **to** n **do** ▷ Compute the k-th row of $\mathbf{U}$
        $M_{ij} \leftarrow M_{kk}M_{ij} - M_{ik}M_{kj}$ ▷ Pertorm the "division-free" elimination
        $M_{ij} \leftarrow$ Veil(Simplify(M_{ij})) ▷ Veil the simplified expression
      end for
    end for
  end for
  $\mathbf{P}, \mathbf{Q} \leftarrow$ PermutationMatrices($\mathbf{r}$, $\mathbf{c}$) ▷ Compute the permutation matrices
  $\mathbf{L} \leftarrow$ LowerTriangular($\mathbf{M}$) ▷ Extract the lower-triangular part of $\mathbf{M}$
  $\mathbf{U} \leftarrow$ UpperTriangular($\mathbf{M}$) ▷ Extract the upper-triangular part of $\mathbf{M}$
  return $\mathbf{L}$, $\mathbf{U}$, $\mathbf{D}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{r}$, $\mathbf{c}$, rnk ▷ Return the factors and the rank of $\mathbf{A}$
end procedure

Algorithm 5 Solve a square linear system $\mathbf{Ax} = \mathbf{b}$ using the FFLU factorization.

Require: The FFLU factors $\mathbf{L}$, $\mathbf{U}$, $\mathbf{D}$, $\mathbf{P}$, $\mathbf{Q}$, and a vector $\mathbf{b}$.
procedure SolveFFLU($\mathbf{L}$, $\mathbf{U}$, $\mathbf{D}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$) ▷ Solve the linear system $\mathbf{Ax} = \mathbf{b}$
  $\mathbf{y} \leftarrow \mathbf{Pb}$ ▷ Apply the permutation matrix $\mathbf{P}$ to the vector $\mathbf{b}$
  $m, n \leftarrow$ Size($\mathbf{L}$) ▷ Get the size of $\mathbf{L}$
  for i **from** 1 **to** $m - 1$ **do** ▷ Solve $\mathbf{Ly} = \mathbf{Pb}$
    $y_i \leftarrow \text{Veil}\left(D_{ii}y_i - \sum_{j=i+1}^{m} L_{ij}y_j\right)$ ▷ Perform forward substitution
  end for
  $x_n \leftarrow \text{Veil}(y_n / U_{nn})$ ▷ Perform the first backward substitution
  for i **from** $n - 1$ **to** 1 **do** ▷ Solve $\mathbf{Ux} = \mathbf{y}$
    $x_i \leftarrow \text{Veil}\left(y_i - \sum_{j=i+1}^{n} U_{ij}x_j\right)$ ▷ Perform backward substitution
    $x_i \leftarrow \text{Veil}(x_i / U_{ii})$
  end for
  $\mathbf{x} \leftarrow \mathbf{Q}^\top \mathbf{x}$ ▷ Apply the permutation matrix $\mathbf{Q}^\top$ to the solution $\mathbf{x}$
end procedure

Algorithm 6 Compute permutation matrices **P** and **Q**.

```
1:  Require: The pivot row and column indices r and c.
2:  procedure PermutationMatrices(r, c)          ▷ Compute the permutation matrices
3:      m, n ← Size(r), Size(c)                  ▷ Retrieve the size of the permutation matrices
4:      P, Q ← Identity(m), Identity(n)          ▷ Initialize the permutation matrices
5:      for i from 1 to m do                     ▷ Build the rows permutation matrix
6:          P ← SwapRows(P, i, r_i)              ▷ Swap the i-th and r_i-th rows
7:      end for
8:      for i from 1 to n do                     ▷ Build the columns permutation matrix
9:          Q ← SwapColumns(Q, i, c_i)           ▷ Swap the i-th and c_i-th columns
10:     end for
11:     return P, Q                              ▷ Return permutation matrices
12: end procedure
```

3.5.2 AN IMPROVED SYMBOLIC PIVOTING STRATEGY

We have shown that a crucial detail of both LU and FFLU decomposition, either numerical or symbolic, is the pivoting strategy. This strategy hinges on two main considerations: the degrees of the elements within the matrix and the actual complexity of the expressions. From an operational standpoint, the pivoting process starts by arranging the elements in the matrix in descending order of their degrees. Then, the pivot of the least complexity is chosen. Sometimes these two features are conflicting. In such cases, the prioritization of the pivot with the lowest degree is preferred. It is important to notice that pivots consisting of numerical values take precedence over those with symbolic values, primarily due to their inherent minimum expression complexity. Throughout the pivoting process, the utilization of signatures is also employed whenever possible to confirm the presence of null expressions without the need for simplification. To summarize, the main steps of the pivoting strategy are the following.

1. The degree for each of the system matrix's entries is calculated.
2. The pivots are sorted by degree and a permutation is generated.
3. The pivots are iterated in the order of the permutation and a candidate pivot is selected at each step.
4. The candidate pivot is checked for null expressions with the aid of signatures.
5. If the candidate pivot signature is not null, the expression is simplified and their complexity is calculated.
6. If the candidate pivot is numeric, its numerical value is calculated, otherwise, it is set to infinity.
7. The candidate pivot with the lowest complexity or largest absolute numeric value is selected as the best pivot and returned.

A detailed description of the developed symbolic pivoting strategy is presented in Algorithm 7.

Algorithm 7 Symbolic Full-Pivoting Strategy.

Require: A $m \times n$ matrix $\mathbf{A}$.
The k-th pivoting stage.
procedure SymbolicPivoting($\mathbf{A}$, k) ▷ Symbolic pivoting procedure for the k-th pivot
  $\mathbf{d}^r$, $\mathbf{d}^c \leftarrow$ ComputeDegrees($\mathbf{A}$) ▷ Calculate the row and column degrees of $\mathbf{A}$
  for i **from** k **to** m **do** ▷ Iterate over the rows
    for j **from** k **to** n **do** ▷ Iterate over the columns
      $D_{ij} \leftarrow \infty$ ▷ Set the combined degree matrix to infinity
      if $A_{ij} \neq 0$ **then** $D_{ij} \leftarrow d_i^r \max(0,\ d_j^k - 1) + d_j^k \max(0,\ d_i^r - 1)$ ▷ The degree
    end for
  end for
  $\mathcal{P} \leftarrow$ Sort($\mathbf{D}$) ▷ Find the permutation that sorts the pivots list by degree cost
  $q,\ l \leftarrow 0,\ 0$ ▷ Initialize the temporary pivot row and column indices
  $p,\ p_c,\ p_n \leftarrow \infty,\ \infty,\ \infty$ ▷ Initialize the pivot value, complexity and numerical value
  for all (i, j) **in** $\mathcal{P}$ **do** ▷ Iterate on the permutation set
    if $p_c \neq \infty$ **and** $D_{ij} > D_{ql}$ **then break** ▷ No more good pivots to check
    $t \leftarrow A_{ij}$ ▷ Get the pivot value
    if Signature(t) $= 0$ **then continue** ▷ Skip the next pivot
    $t \leftarrow$ Simplify(t) ▷ Try to simplify the pivot expression
    $t_c \leftarrow$ ExpressionComplexity(t) ▷ Calculate the computational cost of the pivot
    $t_n \leftarrow \infty$ ▷ Set the default numerical value of the pivot to infinity
    if t is numeric **then** $t_n \leftarrow \max(1,\ \mathrm{abs}(t))$ ▷ Set the numerical value of the pivot
    if $t_c < p_c$ **or** $(t_c = p_c$ **and** $t_n > p_n)$ **then** ▷ If the pivot is better than the current one
      $q,\ l \leftarrow i, j$ ▷ Update the best pivot row and column indices
      $p,\ p_c,\ p_n \leftarrow t,\ t_c,\ t_n$ ▷ Save the best pivot value, complexity and numerical value
    end if
  end for
return $p,\ q,\ l$ ▷ The k-th pivot and its position
end procedure

Algorithm 8 Matrix Degrees Computation.

Require: A $m \times n$ matrix $\mathbf{A}$.
procedure ComputeDegrees($\mathbf{A}$) ▷ Compute the row and column degrees of $\mathbf{A}$
  $\mathbf{d}^r$, $\mathbf{d}^c \leftarrow$ ZerosVector(m), ZerosVector(n) ▷ Initialize degree vectors
  for i **from** 1 **to** m **do** ▷ Iterate over the rows
    for j **from** 1 **to** n **do** ▷ Iterate over the columns
      if $A_{ij} \neq 0$ **then** $d_i^r \leftarrow d_i^r + 1$ ▷ Increment the row degree
    end for
  end for
  for j **from** 1 **to** n **do** ▷ Iterate over the columns
    for i **from** 1 **to** m **do** ▷ Iterate over the rows
      if $A_{ij} \neq 0$ **then** $d_j^c \leftarrow d_j^c + 1$ ▷ Increment the column degree
    end for
  end for
return $\mathbf{d}^r$, $\mathbf{d}^c$ ▷ Return the row and column degrees of $\mathbf{A}$
end procedure

3.5.3 LINEAR ALGEBRA SYMBOLIC TOOLBOX

The considerations just made are the basis of the LAST package [24]. This package is a MAPLE® toolbox for symbolic linear algebra. It is based on the original works in [103, 121] and offers a set of routines for symbolic full-pivoting LU, a FFLU, QR decomposition, and GJ factorization. The package LAST is designed to be used in conjunction with the LEM package [23] to limit the expression swell.
An important aspect of LU decomposition is the pivoting strategy. In the LAST package, the pivoting process is developed to take into account the aforementioned aspects of expression swell and numerical stability. In particular, the pivots are chosen based on the degree of the elements of the system matrix and the actual complexity of the expressions. The elements of the system matrix are sorted in descending order of their degree and the least complex element is chosen as the pivot. Pivots with numeric values are preferred over pivots with symbolic values due to their inherent numerical stability. During the pivoting procedure, whenever possible, signatures are also exploited to verify the presence of null expressions. Here a simple example is given to briefly illustrate the usage of the LAST package.

Example 17 ***(Symbolic linear system solving with LAST [24]).*** Let us consider a simple linear system of equations in the form $\mathbf{Ax} = \mathbf{b}$ and initialize them as follows.

```
> A := Matrix(3, symbol='a'):
> B := Vector(3, symbol='b'):
```

Then we create a LAST object and initialize the built-in LEM object with label `V`.

```
> LAST_obj := Object(LAST):
> LAST_obj:-InitLEM('V');
```

To solve the linear system with LAST it is first necessary to perform one of the available decompositions, *i.e.*, LU, FFLU, QR, or GJ. The intermediate results of the decomposition are internally stored in the LAST object and are available on demand. Once the decomposition is performed, the solution of the linear system can be obtained by calling the `GetResults` routine.

```
> LAST_obj:-LU(A):
> 'L' = LAST_obj:-GetResults("L"), 'U' = LAST_obj:-GetResults("U");
  'r' = LAST_obj:-GetResults("r")^%T, 'c' = LAST_obj:-GetResults("c")^%T;
```

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 \\ -\frac{a_{2,1}}{a_{1,1}} & \frac{V_2}{V_1} & 1 \end{bmatrix}, U = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & V_1 & V_3 \\ 0 & 0 & V_5 \end{bmatrix}$$

$$r = \begin{bmatrix} 1, & 2, & 3 \end{bmatrix}, c = \begin{bmatrix} 1, & 2, & 3 \end{bmatrix}$$

where `r` and `c` are the row and column permutation vectors, respectively. Pivots chosen during the decomposition are also stored in the LAST object and are available on demand

```
> LAST_obj:-GetResults("pivots")
```

$$\left[a_{1,1}, \quad V_1, \quad V_5\right]$$

The LAST package also provides a routine to solve the linear system directly without the need to call the `GetResults` routine. This routine is called `SolveLinearSystem` and it is used as follows.

```
> LAST_obj:-SolveLinearSystem(B)^%T;
```

$$\left[-\frac{V_9}{a_{1,1}}, \quad -\frac{V_8}{V_1}, \quad \frac{V_7}{V_5}\right]$$

```
> LEM_obj := LAST_obj:-GetLEM(LAST_obj);
> <LEM_obj:-VeilList(LEM_obj)>;
```

$$\left[\begin{array}{c}
V_1 = \dfrac{a_{2,2}a_{1,1} - a_{2,1}a_{1,2}}{a_{1,1}} \\
V_2 = \dfrac{a_{3,2}a_{1,1} - a_{3,1}a_{1,2}}{a_{1,1}} \\
V_3 = \dfrac{a_{2,3}a_{1,1} - a_{2,1}a_{1,3}}{a_{1,1}} \\
V_4 = \dfrac{a_{3,3}a_{1,1} - a_{3,1}a_{1,3}}{a_{1,1}} \\
V_5 = \dfrac{V_4V_1 - V_2V_3}{V_1} \\
V_6 = \dfrac{b_2a_{1,1} - a_{2,1}b_1}{a_{1,1}} \\
V_7 = \dfrac{b_3a_{1,1}V_1 - a_{3,1}b_1V_1 - V_2V_6a_{1,1}}{a_{1,1}V_1} \\
V_8 = \dfrac{V_3V_7 - V_6V_5}{V_5} \\
V_9 = \dfrac{b_1V_1V_5 - a_{1,3}V_7V_1 + a_{1,2}V_8V_5}{V_1V_5}
\end{array}\right]$$

It must be noticed that in this example the size of the system is chosen to be small for the sake of simplicity. In practice, the LAST package is designed to be used with large systems of equations. For further information on the LAST package refer to the documentation in [24]. ■

CHAPTER 4

SOLUTION OF DYNAMIC SYSTEMS DESCRIBED BY DIFFERENTIAL-ALGEBRAIC EQUATIONS

In this chapter, we present an algorithm for the index reduction of first-order DAEs. The proposed approach can be applied to generic DAEs and exploits neither a priori knowledge nor ad hoc techniques to leverage the specific formulation of the system. The index reduction is performed only by using symbolic manipulation and linear algebra techniques. It is based on the successive separation of the differential and algebraic equations of the system and the subsequent differentiation of the algebraic part. Improved symbolic matrix factorization is used to perform the DAEs partitioning, ensure numerical stability, and limit the expression swell of the reduced-index system. The effectiveness of the algorithm will be validated in the next chapter through a set of examples on a wide range of systems, including physical systems, engineering applications, and "artificial" DAEs with specific properties. The proposed symbolic index reduction algorithm is implemented in MAPLE® as part of an open-source library.

4.1 DIFFERENTIAL-ALGEBRAIC EQUATIONS INDEX REDUCTION AND SOLUTION

As we already mentioned in the previous chapters, DAEs are extensively used in dynamic system modeling. The challenge in numerically solving a DAE system is assessed through its differentiation index, commonly referred to as "the" index [59, 122]. Achieving accurate simulations necessitates converting high-index DAEs into low-index counterparts, posing a well-recognized challenge [14]. This process involves transforming a DAE system into an equivalent system with a lower index through

successive differentiation of the system equations. For this reason, the differentiation index is roughly defined as the number of times algebraic equations are differentiated to obtain an equivalent system of ODEs with invariants. Index reduction is a crucial step prior to the numerical integration of DAEs, as integrating high-index systems can be impractical. The primary obstacle lies in the necessity to solve nonlinear systems of equations at each integration step, which can be computationally expensive and, in certain cases, numerically unstable. Specific numerical techniques, introduced in [14, 123, 124], have been developed to address this challenge. However, these methods are not universally effective and may be inapplicable to some high-index DAEs. Consequently, index reduction becomes an indispensable preliminary stage before numerical integration [42].

Given its complicated nature, index reduction is the subject of extensive research and is often carried out by leveraging the specific formulation of the DAE system, like in the multi-body modeling [105, 114, 125–127]. When the specific formulation of the DAE system is not known a priori, or the system is not in a specific form, the index reduction process becomes more challenging. Many current simulation software packages for dynamic systems use index-reduction algorithms based on the SA of the system, such as the Pantelides algorithm [36] and the dummy derivatives method [41], which are subcases of the Pryce's Σ-method [37, 60, 73–77, 128]. These algorithms are effective in reducing the index of most of the systems, but they can fail either for numerical cancellations [129] or underestimation of the differentiation index [36, 40], like in the case of Reißig's DAEs family [39]. Symbolic manipulation is proven to be successful in restating a DAEs on which the Σ-method fails to a DAEs on which the SA may succeed [74].

Index reduction based on symbolic-numeric aided SA has been successful in handling failures of the Pryce's Σ-method [74] as well as in performing symbolically-informed SA [68]. This latter SA approach, which is named σv-method, uses symbolic-numeric LU factorization for variable substitution and rank determination on linear constant coefficient DAEs. A valuable attempt to use pure symbolic manipulation in DAEs index reduction is presented in [105, 114, 125], where implicit involutive form and LU decomposition are successfully used to reduce the index of a simple constrained multi-body system. It is clear that index reduction algorithms based on pure symbolic matrix factorization represent a viable alternative to classic SA techniques. However, symbolic matrix factorization feasibility is strongly tied to the performance of the symbolic computation kernel and its capabilities [121]. Large expressions can lead to strong performance degradation of the kernel. Techniques aimed at limiting this decrease of performance while performing symbolic linear algebra operations are presented in [104, 105, 125]. In these works, the hierarchical representation of expressions is applied to matrix factorization tasks. Nonetheless, the LULEM package [103], which implements large expression management strategies in LU decomposition, significantly outperforms the Maple®'s built-in matrix factorization routines.

The absence of user-invocable standalone functions within the Maple® environment that allows for the automatic index reduction of DAEs, and the inability to extract

both the reduced-index system and invariants from the `dsolve` function, are the primary motivations of this research. Furthermore, recent advances in symbolic matrix factorization, combined with expressions hierarchical representation techniques, provide valuable tools that enable us to further investigate the applicability of a novel algorithm for the index reduction of DAE systems, firstly presented in [2] as a preliminary work. The proposed methodology is similar to the previous work of Chowdhry, Krendl, and Linninger [68] and extends it to generic first-order DAEs, linear in the states' derivatives. Nonetheless, the proposed algorithm does not work on the structural matrix of the system but on the DAE system symbolic expressions. Specifically, the idea of using matrix factorization for variable substitution and rank determination is adopted to iteratively separate the differential and algebraic equations of the system. The algebraic equations are then differentiated to obtain an equivalent system with a lower differentiation index. Not less important, new libraries for symbolic matrix factorization (LAST), large expression management (LEM), and signature computation (SIG) are presented. These libraries are based on the LULEM package and extend the work of Carette, Zhou, Jeffrey, and Monagan [103] and Zhou, Carette, Jeffrey, and Monagan [104] and Zhou [105]. The newly presented libraries are designed to ensure the numerical stability of the numerically evaluated expressions, limit the expression swell, and provide an updated object-oriented interface. The effectiveness of the presented index reduction algorithm is validated through symbolic-numerical examples on a wide range of systems, including physical systems, engineering applications, as well as "artificial" systems with specific properties. The proposed algorithm is implemented in the MAPLE® environment and is available as a collection of open-source packages [23, 24]. Furthermore, the insights and the techniques presented in [103–105] are used to improve the presented algorithm to embed the hierarchical representation of expressions in a future implementation of the algorithm.

4.2 A NEW INDEX REDUCTION ALGORITHM

In this section, we explore the theoretical aspect of reducing the differential index of DAE systems. Specifically, to systematically reduce the DAE system's index, a novel iterative algorithm is presented. This algorithm comprises two main phases: initially, the separation of the differential equations from the algebraic equations inherent in DAEs; and subsequently, the differentiation of the algebraic ones to obtain an equivalent system with a reduced index. The algorithm that is presented in this section is implemented in the MAPLE® environment and is available as an open-source package [25].

4.2.1 DIFFERENTIAL AND ALGEBRAIC EQUATIONS SEPARATION

The initial phase of the index reduction procedure involves the partitioning of the DAE system into its differential and algebraic equations. While in the case of small systems this can be accomplished through manual identification and isolation of the algebraic

equations, this approach is inconvenient when dealing with large systems. An alternative method for automating the separation process leverages the cokernel, or left null space, of the DAE system matrix, which is computed through matrix factorization techniques. The usage of the cokernel offers a more efficient and reliable means of accomplishing the separation task, variable substitution and not less importantly rank determination.
Consider a first-order system of DAEs $\mathbf{F}(\mathbf{x}, \mathbf{x}', t)$ of the form

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \mathbf{A}(\mathbf{x}, t)\,\mathbf{x}' - \mathbf{b}(\mathbf{x}, t) = \mathbf{0}\,. \tag{4.1}$$

We denote the cokernel and its orthogonal complement of $\mathbf{A}(\mathbf{x}, t)$ with $\mathbf{K}(\mathbf{x}, t)$ and $\mathbf{N}(\mathbf{x}, t)$ respectively (see 4.2.1.1 for details on the cokernel computation with symbolic matrix factorization). Notice that the cokernel is the subspace obtained by the span of $\mathbf{K}(\mathbf{x}, t)$'s columns. For this reason, hereafter, we refer to the cokernel as the matrix $\mathbf{K}(\mathbf{x}, t)$ whose columns span the cokernel. Using $\mathbf{K}(\mathbf{x}, t)$ and $\mathbf{N}(\mathbf{x}, t)$ it is possible to separate the algebraic part of the DAEs (4.1) as

$$\mathbf{A}(\mathbf{x}, t)\,\mathbf{x}' = \mathbf{b}(\mathbf{x}, t)\,, \quad \text{and thus} \quad \begin{cases} \mathbf{E}(\mathbf{x}, t)\,\mathbf{x}' = \mathbf{g}(\mathbf{x}, t) \\ \mathbf{0} = \mathbf{a}(\mathbf{x}, t) \end{cases}, \tag{4.2}$$

$$\text{where} \quad \mathbf{A}(\mathbf{x}, t) = \begin{bmatrix} \mathbf{E}(\mathbf{x}, t) \\ \mathbf{0} \end{bmatrix}, \quad \text{and} \quad \mathbf{b}(\mathbf{x}, t) = \begin{bmatrix} \mathbf{g}(\mathbf{x}, t) \\ \mathbf{a}(\mathbf{x}, t) \end{bmatrix}.$$

The separated equations of the DAEs are obtained by the left product of $\mathbf{K}(\mathbf{x}, t)$ and $\mathbf{N}(\mathbf{x}, t)$ as

$$\begin{aligned} \mathbf{E}(\mathbf{x}, t) &= \mathbf{N}(\mathbf{x}, t)\,\mathbf{A}(\mathbf{x}, t)\,, \\ \mathbf{g}(\mathbf{x}, t) &= \mathbf{N}(\mathbf{x}, t)\,\mathbf{b}(\mathbf{x}, t)\,, \\ \mathbf{a}(\mathbf{x}, t) &= \mathbf{K}(\mathbf{x}, t)\,\mathbf{b}(\mathbf{x}, t)\,. \end{aligned}$$

This results in an equivalent DAE system, where the algebraic equations $\mathbf{a}(\mathbf{x}, t)$ are now explicit. The details of the cokernel and its orthogonal complement computation are discussed in 4.2.1.1.

4.2.1.1 COKERNEL COMPUTATION WITH MATRIX FACTORIZATION

The cokernel of a generic matrix $\mathbf{A}$ is denoted as $\mathbf{K}$ and satisfy $\mathbf{KA} = \mathbf{0}$. As mentioned before, the cokernel is the subspace obtained by the span of $\mathbf{K}$'s columns. The orthogonal complement of the cokernel is denoted as $\mathbf{N}$, moreover, the matrices $\mathbf{N}$ and $\mathbf{K}$ stacked compose a square non-singular matrix. It is common knowledge that matrix factorization techniques can be employed to compute the cokernel and its orthogonal complement. Among these techniques, the LU factorization stands out as one of the most frequently used methods.

LOWER-UPPER DECOMPOSITION The full-pivoting LU decomposition of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ (with $m \geq n$) is represented as the product of matrices $\mathbf{L}$ and $\mathbf{U}$ with the permutation matrices $\mathbf{P}$ and $\mathbf{Q}$. It is characterized by the following properties:

- $\mathbf{PAQ} = \mathbf{LU}$;
- $\mathbf{L} \in \mathbb{R}^{m\times m}$ is a lower-triangular matrix with all diagonal entries equal to 1;
- $\mathbf{U} \in \mathbb{R}^{m\times n}$ is an upper-triangular matrix;
- $\mathbf{P} \in \mathbb{R}^{m\times m}$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are the rows and columns permutation matrices, respectively.

If $\mathbf{M}$ is defined such that $\mathbf{M} = \mathbf{L}^{-1}\mathbf{P}$, then the following relation holds

$$\mathbf{MA} = \mathbf{L}^{-1}\mathbf{PA} = \mathbf{L}^{-1}\mathbf{PAQQ}^\top = \mathbf{L}^{-1}\mathbf{LUQ}^\top = \mathbf{UQ}^\top = \begin{bmatrix}\mathbf{U}_1\\ \mathbf{0}\end{bmatrix}\mathbf{Q}^\top.$$

If the identity matrix $\mathbf{I}$ is partitioned as

$$\mathbf{I} = \begin{bmatrix}\mathbf{I}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_2\end{bmatrix}, \quad \text{where} \quad \mathbf{I}_1 \in \mathbb{R}^{m\times m} \quad \text{and} \quad \mathbf{I}_2 \in \mathbb{R}^{(m-n)\times(m-n)},$$

we can write

$$\begin{bmatrix}\mathbf{I}_1 & \mathbf{0}\end{bmatrix}\mathbf{MA} = \begin{bmatrix}\mathbf{I}_1 & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{U}_1\\ \mathbf{0}\end{bmatrix}\mathbf{Q}^\top = \mathbf{U}_1\mathbf{Q}^\top,$$

and

$$\begin{bmatrix}\mathbf{0} & \mathbf{I}_2\end{bmatrix}\mathbf{MA} = \begin{bmatrix}\mathbf{0} & \mathbf{I}_2\end{bmatrix}\begin{bmatrix}\mathbf{U}_2\\ \mathbf{0}\end{bmatrix}\mathbf{Q}^\top = \mathbf{0}.$$

Eventually, the matrices $\mathbf{N}$ and $\mathbf{K}$ have the following form

$$\mathbf{N} = \begin{bmatrix}\mathbf{I}_1 & \mathbf{0}\end{bmatrix}\mathbf{M} \quad \text{and} \quad \mathbf{K} = \begin{bmatrix}\mathbf{0} & \mathbf{I}_2\end{bmatrix}\mathbf{M},$$

where

$$\begin{matrix}\mathbf{KA} = \mathbf{0}\\ \mathbf{NA} = \mathbf{U}_1\mathbf{Q}^\top \text{ is full-rank}\end{matrix}, \quad \text{and} \quad \begin{bmatrix}\mathbf{N}\\ \mathbf{K}\end{bmatrix} \quad \text{is non-singular.}$$

FRACTION-FREE LOWER-UPPER DECOMPOSITION The FFLU factorization is a variant of the LU decomposition. It is based on the same principles as the standard LU decomposition, but it is designed to avoid the appearance of fractions in the intermediate results. Similarly to the LU case, the FFLU decomposition of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ (with $m \geq n$) is represented as the quintet of matrices $\mathbf{L}$, $\mathbf{U}$, $\mathbf{D}$, $\mathbf{P}$, and $\mathbf{Q}$, and is characterized by the properties:

- $\mathbf{PDAQ} = \mathbf{LU}$;
- $\mathbf{L} \in \mathbb{R}^{m\times m}$ is a lower-triangular matrix with all diagonal entries equal to 1;

- $\mathbf{D} \in \mathbb{R}^{m \times m}$ is a diagonal matrix;
- $\mathbf{U} \in \mathbb{R}^{m \times n}$ is an upper-triangular matrix;
- $\mathbf{P} \in \mathbb{R}^{m \times m}$ and $\mathbf{Q} \in \mathbb{R}^{n \times n}$ are the rows and columns permutation matrices, respectively.

The procedure for computing the cokernel of a matrix $\mathbf{A}$ using the FFLU is similar to the case of the LU decomposition. The only difference is in the computation of the matrix product $\mathbf{MA}$. If $\mathbf{M} = \mathbf{L}^{-1}\mathbf{P}\mathbf{D}$, then $\mathbf{MA}$ are written as

$$\mathbf{MA} = \mathbf{L}^{-1}\mathbf{PDA} = \mathbf{L}^{-1}\mathbf{PDAQQ}^{\top} = \mathbf{L}^{-1}\mathbf{LUQ}^{\top} = \mathbf{UQ}^{\top} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{0} \end{bmatrix} \mathbf{Q}^{\top} .$$

Then, the subspaces $\mathbf{N}$ and $\mathbf{K}$ are computed as in the LU case.

4.2.2 ALGEBRAIC EQUATIONS DIFFERENTIATION

The algebraic equations in the DAE system (4.2) are now differentiated as

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{a}(\mathbf{x}, t) = \mathbf{E}_{\mathrm{a}}(\mathbf{x}, t)\,\mathbf{x}' - \mathbf{g}_{\mathrm{a}}(\mathbf{x}, t) .$$

The DAEs (4.2) after differentiation of algebraic equations now take a form similar to that of (4.1), where

$$\mathbf{A}(\mathbf{x}, t) = \begin{bmatrix} \mathbf{E}(\mathbf{x}, t) \\ \mathbf{E}_{\mathrm{a}}(\mathbf{x}, t) \end{bmatrix} , \quad \text{and} \quad \mathbf{b}(\mathbf{x}, t) = \begin{bmatrix} \mathbf{g}(\mathbf{x}, t) \\ \mathbf{g}_{\mathrm{a}}(\mathbf{x}, t) \end{bmatrix} .$$

The set of invariants, which are collected in $\mathbf{h}(\mathbf{x}, t)$, is updated adding the algebraic equations $\mathbf{a}(\mathbf{x}, t)$

$$\mathbf{h}(\mathbf{x}, t) \quad \xleftarrow[\text{update}]{\text{Invariants}} \quad \begin{bmatrix} \overbrace{\mathbf{h}(\mathbf{x}, t)}^{\text{The old } \mathbf{h}(\mathbf{x}, t)} \\ \mathbf{a}(\mathbf{x}, t) \end{bmatrix} .$$

The iterative procedure, involving the sequential separation and differentiation of the algebraic segment of the system, is iterated until $\mathbf{A}(\mathbf{x}, t)$ is non-singular. When $\mathbf{A}(\mathbf{x}, t)$ is non-singular the DAE corresponds to a system of ODEs compounded by the invariants $\mathbf{h}(\mathbf{x}, t)$, which are the collection of the hidden constraints produced in the index reduction process. The invariants $\mathbf{h}(\mathbf{x}, t)$ can be initialized empty or with user-defined algebraic equations aimed at preserving crucial system properties, such as energy conservation and/or momentum conservation. A pseudocode of the index reduction algorithm can be found in Algorithm 9. Notice that in the algorithm, the choice of permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ computed during the matrix factorization are dependent on the state variables $\mathbf{x}$ and free variable t. However, during the numerical integration of the reduced-index system, the pivoting choice is assumed not to change, therefore the permutation matrices are fixed and their dependencies are dropped.

Algorithm 9 Index reduction algorithm (without large expression management) [2].

Require: A DAE system of the form $\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \mathbf{A}(\mathbf{x}, t)\,\mathbf{x}' - \mathbf{b}(\mathbf{x}, t) = \mathbf{0}$.
procedure REDUCEINDEX($\mathbf{F}(\mathbf{x}, \mathbf{x}', t)$) ▷ Index reduction procedure
  $\mathbf{h}(\mathbf{x}, t) \leftarrow \varnothing$ ▷ The set of invariants
  $\mathbf{A}(\mathbf{x}, t),\ \mathbf{b}(\mathbf{x}, t) \leftarrow$ GenerateMatrix($\mathbf{F}(\mathbf{x}, \mathbf{x}', t),\ \mathbf{x}'$) ▷ The DAE system matrix
  $m \leftarrow$ Size($\mathbf{x}$) ▷ The size of $\mathbf{x}$
  while $\mathbf{A}(\mathbf{x}, t)$ is singular **do**
    ▷ Differential and algebraic equations separation (Section 4.2.1)
    $\mathbf{L}(\mathbf{x}, t),\ \mathbf{U}(\mathbf{x}, t),\ \mathbf{P},\ \mathbf{Q} \leftarrow$ MatrixFactorization($\mathbf{A}(\mathbf{x}, t)$) ▷ Factorization of $\mathbf{A}(\mathbf{x}, t)$
    $r \leftarrow$ Rank($\mathbf{U}(\mathbf{x}, t)$) ▷ The rank of $\mathbf{U}(\mathbf{x}, t)$ is equal to the rank of $\mathbf{A}(\mathbf{x}, t)$
    $\mathbf{I}_1 \leftarrow$ IdentityMatrix($r,\ r$) ▷ The upper identity matrix
    $\mathbf{I}_2 \leftarrow$ IdentityMatrix($m - r,\ m - r$) ▷ The lower identity matrix
    $\mathbf{E}(\mathbf{x}, t) \leftarrow [\mathbf{I}_1,\ \mathbf{0}]\,\mathbf{U}(\mathbf{x}, t)\,\mathbf{Q}^\top$ ▷ The reordered part of $\mathbf{A}(\mathbf{x}, t)$
    $\mathbf{g}(\mathbf{x}, t) \leftarrow [\mathbf{I}_1,\ \mathbf{0}]\,\mathbf{L}(\mathbf{x}, t)^{-1}\,\mathbf{P}\,\mathbf{b}(\mathbf{x}, t)$ ▷ The differential part of $\mathbf{b}(\mathbf{x}, t)$
    $\mathbf{a}(\mathbf{x}, t) \leftarrow [\mathbf{0},\ \mathbf{I}_2]\,\mathbf{L}(\mathbf{x}, t)^{-1}\,\mathbf{P}\,\mathbf{b}(\mathbf{x}, t)$ ▷ The algebraic part of $\mathbf{b}(\mathbf{x}, t)$
    ▷ Algebraic equations differentiation (Section 4.2.2)
    $\mathbf{E}_\mathbf{a}(\mathbf{x}, t),\ \mathbf{g}_\mathbf{a}(\mathbf{x}, t) \leftarrow$ GenerateMatrix(Diff($\mathbf{a}(\mathbf{x}, t),\ t$), $\mathbf{x}'$) ▷ Differentiate $\mathbf{a}(\mathbf{x}, t)$
    $\mathbf{A}(\mathbf{x}, t) \leftarrow \begin{bmatrix} \mathbf{E}(\mathbf{x}, t) \\ \mathbf{E}_\mathbf{a}(\mathbf{x}, t) \end{bmatrix}$ and $\mathbf{b}(\mathbf{x}, t) \leftarrow \begin{bmatrix} \mathbf{g}(\mathbf{x}, t) \\ \mathbf{g}_\mathbf{a}(\mathbf{x}, t) \end{bmatrix}$ ▷ The new $\mathbf{A}(\mathbf{x}, t)$ and $\mathbf{b}(\mathbf{x}, t)$
    $\mathbf{h}(\mathbf{x}, t) \leftarrow \mathbf{h}(\mathbf{x}, t) \cup \mathbf{a}(\mathbf{x}, t)$ ▷ Add the algebraic equations to the set of invariants
  end while
return $\mathbf{A}(\mathbf{x}, t),\ \mathbf{b}(\mathbf{x}, t),\ \mathbf{h}(\mathbf{x}, t)$ ▷ The DAEs reduced to an ODE system with invariants
end procedure

4.2.3 A STEP-BY-STEP EXAMPLE

Within this Section, we present the step-by-step results of the index reduction algorithm. To do so we exploit a simple non-stiff index-3 problem found in WOLFRAM MATHEMATICA® documentation [67]. The initial value problem is defined as follows

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \begin{bmatrix} x_2' + x_1 - \sin(t) \\ x_3' + x_2 - \sin(t) \\ x_3 - \cos(t) \end{bmatrix}, \tag{4.3}$$

with states $\mathbf{x} = [x_1,\ x_2,\ x_3]^\top$ and ICs $\mathbf{x}_0 = [-1,\ 0,\ 1]^\top$. Notice that the analytical solution of this problem is $\mathbf{x}_{\text{exact}} = [\sin(t) - 2\cos(t),\ 2\sin(t),\ \cos(t)]^\top$. The index reduction algorithm is applied to the DAE system (4.3) and the step-by-step results for the matrices $\mathbf{E}(\mathbf{x}, t)$, $\mathbf{g}(\mathbf{x}, t)$, and $\mathbf{a}(\mathbf{x}, t)$ are reported here below.

$$\text{Index-3 DAEs:}\quad \mathbf{E}(\mathbf{x}, t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\quad \mathbf{g}(\mathbf{x}, t) = \begin{bmatrix} \sin(t) - x_1 \\ \sin(t) - x_2 \end{bmatrix},$$
$$\mathbf{a}(\mathbf{x}, t) = \begin{bmatrix} \cos(t) - x_3 \end{bmatrix}.$$

$$\text{Index-2 DAEs:} \quad \mathbf{E}(\mathbf{x},t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{g}(\mathbf{x},t) = \begin{bmatrix} \sin(t) - x_1 \\ \sin(t) - x_2 \end{bmatrix}, \quad \mathbf{a}(\mathbf{x},t) = \begin{bmatrix} 2\sin(t) - x_2 \end{bmatrix}.$$

$$\text{Index-1 DAEs:} \quad \mathbf{E}(\mathbf{x},t) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{g}(\mathbf{x},t) = \begin{bmatrix} \sin(t) - x_2 \\ \sin(t) - x_1 \end{bmatrix}, \quad \mathbf{a}(\mathbf{x},t) = \begin{bmatrix} \sin(t) - 2\cos(t) - x_1 \end{bmatrix}.$$

$$\text{Index-0 DAEs:} \quad \mathbf{E}(\mathbf{x},t) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{g}(\mathbf{x},t) = \begin{bmatrix} \sin(t) - x_2 \\ \sin(t) - x_1 \\ 2\sin(t) + \cos(t) \end{bmatrix}, \quad \mathbf{a}(\mathbf{x},t) = \varnothing.$$

The final form of the system is an index-0 DAEs system is then

$$\mathbf{F}(\mathbf{x},\mathbf{x}',t) = \begin{bmatrix} x_2' - x_1 - \cos(t) \\ x_3' - x_2 - \sin(t) \\ x_1' - \cos(t) - 2\sin(t) \end{bmatrix},$$

with invariants

$$\mathbf{h}(\mathbf{x},t) = \begin{bmatrix} \cos(t) - x_3 \\ 2\sin(t) - x_2 \\ \sin(t) - 2\cos(t) - x_1 \end{bmatrix}.$$

Although the just presented example is not so complex and relevant for a real validation of the algorithm, it is useful to demonstrate the step-by-step results of the index reduction algorithm. Furthermore, the expression complexities encountered throughout the index reduction algorithm applied to the Index-3 problem are reported below in Table 4.1.

4.2.4 ACKNOWLEDGING SOME ALGORITHM LIMITATIONS

While the algorithm just presented is relatively straightforward to implement, it does have two major sources of potential issues that are both determined by the technology used and the fundamental theory.

- *Expression complexity*. Symbolic manipulation often leads to a growth in expression complexity. For this reason, expression simplification may not always be feasible due to software limitations or excessive CPU time demands. Making the algorithm insensitive to expression swell is thus crucial to its effectiveness.
- *Numerical stability of symbolic matrix factorization*. The description of the algorithm involves the manipulation of matrices and vectors with either symbolic or mixed symbolic-numeric entries. Ensuring that symbolic matrix factorization maintains numerical stability is a critical requirement of the algorithm. In the case of LU decomposition, inadequate pivoting strategies can lead to the generation of singular matrices, which in turn can cause the algorithm to fail [105, 114, 120].

Table 4.1: Expression complexity encountered throughout the index reduction of the index-3 step-by-step example problem [67] DAE system index reduction with both the LU and FFLU factorization techniques. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Index-3 DAEs (LU Factorization) [67]

Original DAEs		$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 10\,\mathrm{f} + 5\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$2\,\mathrm{f} + 1\,\mathrm{a}$
Index-2 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$2\,\mathrm{f} + 1\,\mathrm{m} + 1\,\mathrm{a}$
Index-1 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$3\,\mathrm{f} + 1\,\mathrm{m} + 1\,\mathrm{a}$
Index-0 DAEs	0	$6\,\mathrm{f} + 1\,\mathrm{m} + 3\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 12\,\mathrm{f} + 1\,\mathrm{m} + 6\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 7\,\mathrm{f} + 2\,\mathrm{m} + 4\,\mathrm{a}$

Index-3 DAEs (FFLU Factorization) [67]

Original DAEs		$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 10\,\mathrm{f} + 5\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$2\,\mathrm{f} + 1\,\mathrm{a}$
Index-2 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$2\,\mathrm{f} + 1\,\mathrm{m} + 2\,\mathrm{a}$
Index-1 DAEs	0	$4\,\mathrm{f} + 2\,\mathrm{a}$	$3\,\mathrm{f} + 1\,\mathrm{m} + 2\,\mathrm{a}$
Index-0 DAEs	0	$6\,\mathrm{f} + 1\,\mathrm{m} + 3\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 12\,\mathrm{f} + 1\,\mathrm{m} + 6\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 7\,\mathrm{f} + 2\,\mathrm{m} + 4\,\mathrm{a}$

These are the two main points that we acknowledge in the implementation of the algorithm. In the forthcoming sections, each of these matters is discussed in detail, with recommendations on techniques and open-source software solutions that are used to address them.

4.3 INDEX REDUCTION WITH EXPRESSION SWELL MITIGATION

The most interesting and relevant aspect to be explored is the connection between the hierarchical representation of expressions (see Section 3.4) and the DAE index reduction. The use of veiling variables may be used to hide some parts of the expressions by the collection of common sub-expressions. Even if this may appear to be a substantial improvement, it is not so frequent to encounter expressions that are common to all the equations of the DAE system. Still, this concept can be extended to mitigate the expression swell during matrix factorization (see Section 3.5). The veiling variables $\mathbf{v}(\mathbf{x}, t)$ would then include the states $\mathbf{x}$ of the DAE system. In this manner, the hierarchical representation of the expression serves as a system augmentation technique as well as a means to limit expression swell during the index reduction procedure. The augmented

DAE system would then be expressed as

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = \mathbf{A}(\mathbf{x}, \mathbf{v}, t)\,\mathbf{x}' - \mathbf{b}(\mathbf{x}, \mathbf{v}, t) = \mathbf{0}\,, \tag{4.4a}$$

$$\text{where} \quad \mathbf{v}(\mathbf{x}, t) = \begin{bmatrix} v_1(\mathbf{x}, t) \\ v_2(v_1, \mathbf{x}, t) \\ \vdots \\ v_n(v_1, \ldots, v_{n-1}, \mathbf{x}, t) \end{bmatrix}. \tag{4.4b}$$

Notice that if the matrix $\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$ is non-singular, the augmented DAE system (4.4) has index-1. This is a crucial aspect to be taken into consideration since, as demonstrated in Section 4.4.3.1, the final reduction to index-0 DAEs is costly. Furthermore, the vector $\mathbf{v}(\mathbf{x}, t)$ and its Jacobian with respect to the states $\mathbf{x}$ can be sequentially evaluated for additional reduction of the computational burden. Nonetheless, the augmented formulation (4.4) allows for the full exploitation of the signature technique to detect null expressions without the need for symbolic simplification [113]. Eventually, this will be the subject of future research and implementations that will exploit index-1 DAEs integrators similarly to the other state-of-the-art DAEs solver presented in the Introduction. A pseudocode and a flowchart of the index reduction algorithm with expression swell mitigation can be found in Algorithm 10 and Figure 4.1, respectively. Similarly to Algorithm 9, the choice of permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ computed during the matrix factorization are dependent on the state variables $\mathbf{x}$ and free variable t. However, their dependencies are dropped in the algorithm since the pivoting choice is assumed not to change during the numerical integration of the reduced-index system.

Algorithm 10 Index reduction algorithm with expression swell mitigation.

Require: A DAE system of the form $\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = \mathbf{A}(\mathbf{x}, \mathbf{v}, t)\,\mathbf{x}' - \mathbf{b}(\mathbf{x}, \mathbf{v}, t) = \mathbf{0}$.
procedure ReduceIndex($\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t)$) ▷ Index reduction procedure
  $\mathbf{h}_i(\mathbf{x}, \mathbf{v}, t) \leftarrow \varnothing$ ▷ The set of invariants
  $\mathbf{v}(\mathbf{x}, t) \leftarrow \varnothing$ ▷ The veiling variables vector
  $\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$, $\mathbf{b}(\mathbf{x}, \mathbf{v}, t) \leftarrow$ GenerateMatrix($\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t)$, $\mathbf{x}'$) ▷ The DAE system matrix
  $m \leftarrow$ Size($\mathbf{x}$) ▷ The size of $\mathbf{x}$
  while $\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$ is singular **do**
    ▷ Differential and algebraic equations separation (Section 4.2.1)
    $\mathbf{L}(\mathbf{x}, \mathbf{v}, t)$, $\mathbf{U}(\mathbf{x}, \mathbf{v}, t)$, $\mathbf{P}$, $\mathbf{Q} \leftarrow$ MatrixFactorization($\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$) ▷ Factorization step
    $\mathbf{v}_d(\mathbf{x}, \mathbf{v}, t) \leftarrow$ NewVeilings($\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$, $\mathbf{v}(\mathbf{x}, t)$) ▷ New veils from factorization
    $\mathbf{v}(\mathbf{x}, t) \leftarrow \mathbf{v}(\mathbf{x}, t) \cup \mathbf{v}_d(\mathbf{x}, \mathbf{v}, t)$ ▷ Update the veiling variables vector
    $r \leftarrow$ Rank($\mathbf{U}(\mathbf{x}, \mathbf{v}, t)$) ▷ The rank of $\mathbf{U}(\mathbf{x}, \mathbf{v}, t)$ is equal to the rank of $\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$
    $\mathbf{I}_1 \leftarrow$ IdentityMatrix(r, r) ▷ The upper identity matrix
    $\mathbf{I}_2 \leftarrow$ IdentityMatrix($m - r$, $m - r$) ▷ The lower identity matrix
    $\mathbf{E}(\mathbf{x}, \mathbf{v}, t) \leftarrow [\mathbf{I}_1, \mathbf{0}]\,\mathbf{U}(\mathbf{x}, \mathbf{v}, t)\,\mathbf{Q}^\top$ ▷ The reordered part of $\mathbf{A}(\mathbf{x}, \mathbf{v}, t)$
    $\mathbf{g}(\mathbf{x}, \mathbf{v}, t) \leftarrow [\mathbf{I}_1, \mathbf{0}]\,\mathbf{L}(\mathbf{x}, \mathbf{v}, t)^{-1}\,\mathbf{P}\,\mathbf{b}(\mathbf{x}, \mathbf{v}, t)$ ▷ The differential part of $\mathbf{b}(\mathbf{x}, \mathbf{v}, t)$
    $\mathbf{a}(\mathbf{x}, \mathbf{v}, t) \leftarrow [\mathbf{0}, \mathbf{I}_2]\,\mathbf{L}(\mathbf{x}, \mathbf{v}, t)^{-1}\,\mathbf{P}\,\mathbf{b}(\mathbf{x}, \mathbf{v}, t)$ ▷ The algebraic part of $\mathbf{b}(\mathbf{x}, \mathbf{v}, t)$
    ▷ Algebraic equations differentiation (Section 4.2.2)
    $\mathbf{v}_\mathbf{x}(\mathbf{x}, \mathbf{v}, t) \leftarrow$ Jacobian($\mathbf{v}(\mathbf{x}, t)$, $\mathbf{x}$) ▷ The Jacobian of $\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$ with respect to $\mathbf{x}$
    $\mathbf{a}(\mathbf{x}, \mathbf{v}, t) \leftarrow$ Diff($\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$, t) ▷ Differentiate $\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$
    $\mathbf{a}(\mathbf{x}, \mathbf{v}, t) \leftarrow$ Substitute($\mathbf{v}_\mathbf{x}(\mathbf{x}, \mathbf{v}, t)$, $\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$) ▷ Remove derivatives from $\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$

22: $\mathbf{E}_a(\mathbf{x},\mathbf{v},t),\ \mathbf{g}_a(\mathbf{x},\mathbf{v},t) \leftarrow \text{GenerateMatrix}(\mathbf{a}(\mathbf{x},\mathbf{v},t),\ \mathbf{x}')$ ▷ Get new system blocks
23: $\mathbf{A}(\mathbf{x},\mathbf{v},t) \leftarrow \begin{bmatrix} \mathbf{E}(\mathbf{x},\mathbf{v},t) \\ \mathbf{E}_a(\mathbf{x},\mathbf{v},t) \end{bmatrix}$ ▷ The new matrix $\mathbf{A}(\mathbf{x},\mathbf{v},t)$
24: $\mathbf{b}(\mathbf{x},\mathbf{v},t) \leftarrow \begin{bmatrix} \mathbf{g}(\mathbf{x},\mathbf{v},t) \\ \mathbf{g}_a(\mathbf{x},\mathbf{v},t) \end{bmatrix}$ ▷ The new vector $\mathbf{b}(\mathbf{x},\mathbf{v},t)$
25: $\mathbf{h}_i(\mathbf{x},\mathbf{v},t) \leftarrow \mathbf{h}_i(\mathbf{x},\mathbf{v},t) \cup \mathbf{a}(\mathbf{x},\mathbf{v},t)$ ▷ Add the $\mathbf{a}(\mathbf{x},\mathbf{v},t)$ to the set of invariants
26: **end while**
27: **return** $\mathbf{A}(\mathbf{x},\mathbf{v},t),\ \mathbf{b}(\mathbf{x},\mathbf{v},t),\ \mathbf{h}_i(\mathbf{x},\mathbf{v},t),\ \mathbf{v}(\mathbf{x},t)$ ▷ The reduced DAEs
28: **end procedure**

4.4 THE SYMBOLIC-NUMERIC SOLUTION SCHEME

After this fairly long discussion on the actual implementation aspects of the index reduction algorithm, we can now present the INDIGO index reduction and integration toolbox [25]. This toolbox consists of two main components: a MAPLE® package to carry out symbolic index reduction of DAE systems, and a MATLAB® toolbox to perform numerical integration of the reduced-index system. INDIGO is designed to be used in conjunction with the LEM package, to limit the expression swell, and the LAST package, to conveniently factorize matrices. In the following paragraphs, we briefly discuss the usage of the INDIGO package.

4.4.1 INDEX REDUCTION

The index reduction algorithm implemented in the INDIGO MAPLE® package is the one presented in Section 4.2. To reduce the index of a DAE system in the MAPLE® environment we need to first create an INDIGO object instance.

```
> Indigo_obj := Object(Indigo);
> Indigo_obj:-InitLAST();
```

Then, the system of DAEs `eqns` with coordinates `vars` is loaded.

```
> Indigo_obj:-LoadEquations('Generic', eqns, vars);
```

The `Generic` symbol is used to specify the type of the system. Notice that in this example, we assume that the equations of the system are already available in the MAPLE® session. The automatic index reduction process can be performed by calling the `ReduceIndex` method, which iterates the separation and differentiation steps until an index-0 DAE system is obtained.

```
> Indigo_obj:-ReduceIndex();
```

Intermediate results of the process are stored internally in the INDIGO object and are available on demand. Once the index reduction process is completed, the user can generate the `name` MATLAB® class file to perform numerical integration of the reduced-index system.

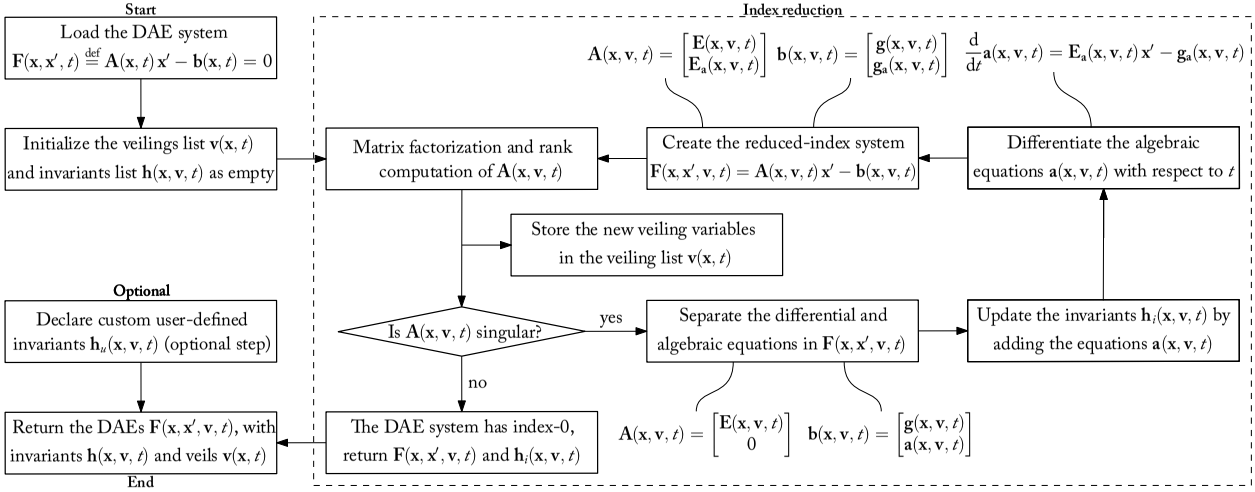


Figure 4.1: Flowchart of the index reduction algorithm with expression swell mitigation.

```
> Indigo_obj:-GenerateMatlabCode(name, type, data=pars);
```

A file `name.m` is generated in the current directory. As it is explained in the next section, The string parameter `type` can be either `Implicit`, `SemiExplicit` or `Explicit` depending on the desired numerical integration scheme. The optional parameter `data` introduces default internal object data.

4.4.2 NUMERICAL INTEGRATION SCHEME

The INDIGO MATLAB® toolbox is an object-oriented library that allows the user to exploit the automatically generated code of the reduced-index system. It is capable of integrating systems of ODEs and DAEs using a variety of RK numerical integration schemes. In particular, the system of equations which is integrated is composed of the following elements.

- A differential part, which can be expressed by one of the following classes:

$$\begin{aligned} \mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) &= \mathbf{0} && \texttt{Implicit} \text{ system class,} \\ \mathbf{A}(\mathbf{x}, \mathbf{v}, t)\, \mathbf{x}' &= \mathbf{b}(\mathbf{x}, \mathbf{v}, t) && \texttt{SemiExplicit} \text{ system class,} \\ \mathbf{x}' &= \mathbf{f}(\mathbf{x}, \mathbf{v}, t) && \texttt{Explicit} \text{ system class.} \end{aligned}$$

- The invariants, composed of the hidden constraints obtained from the index reduction process $\mathbf{h}_i(\mathbf{x}, \mathbf{v}, t)$, and optional user-defined invariants $\mathbf{h}_u(\mathbf{x}, \mathbf{v}, t)$, namely

$$\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = \begin{bmatrix} \mathbf{h}_i(\mathbf{x}, \mathbf{v}, t) \\ \mathbf{h}_u(\mathbf{x}, \mathbf{v}, t) \end{bmatrix} = \mathbf{0}\,.$$

- The veils, which are the set of the expression hierarchical representation variables used in LEM to limit the expression swell

$$\mathbf{v}(\mathbf{x}, t) = \begin{bmatrix} v_1(\mathbf{x}, t) \\ v_2(v_1, \mathbf{x}, t) \\ \vdots \\ v_n(v_1, \ldots, v_{n-1}, \mathbf{x}, t) \end{bmatrix}.$$

To respect the invariants during the integration the *standard projection* method is applied [130]. This method consists of projecting the solution $\mathbf{x}$ of the numerically integrated reduced-index system onto the invariants manifold $\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = \mathbf{0}$, which is equivalent to the following constrained minimization

$$\underset{\mathbf{x}}{\text{minimize}} \quad \frac{1}{2}\,(\mathbf{x} - \bar{\mathbf{x}})^2 \quad \text{subject to} \quad \mathbf{h}(\mathbf{x}, \mathbf{v}, t) = \mathbf{0}.$$

To integrate the system generated through the MATLAB® package or a custom system `sys`, the user must first instantiate a INDIGO RK solver.

```
>>> solver = IndigoSolver('solver_name');
>>> solver.set_system(sys);
```

Once the solver is instantiated, it is only necessary to specify the ICs and the integration time vector.

```
>>> [x, t, v, h] = solver.solve(t_ini:d_t:t_end, ics);
```

The solver returns the solution of the system in the form of multiple outputs: `x` contains the integrated solution, `x_dot` the states' time derivative, `t` the time vector, `v` the veiling variables, and finally `h` the values of the invariants over the specified time mesh `t_ini:d_t:t_end`.

4.4.3 PROVING THE ALGORITHM IMPLEMENTATION

In this section, we showcase an example of a high-index DAE system: an index-3 problem with an analytical solution, which describes the motion of a particle on a 3D torus surface [131]. This example is employed to validate the numerical stability of the reduced-index system, as well as the good conditioning of both the symbolic matrix factorization and the numerical integration scheme. To ease the understanding of the results, we purposely avoid the use of the veiling variables in this example. However, the veiling variables just add an evaluation layer to the expressions, and they do not affect the numerical properties of the integrator.

4.4.3.1 EXPRESSION COMPLEXITY OF THE REDUCED-INDEX SYSTEMS

As a first demonstration of the proposed index reduction algorithm capabilities, we first consider the given example from a purely symbolic perspective. In particular, we consider the computational cost of the expressions generated during the presented procedure, both with LU and FFLU factorization of the system matrix. The compactness of the expressions generated during the index reduction algorithm is a crucial aspect, as it ensures that limited computational overhead is introduced in the numerical integration of the reduced-index system, as well as in the projection of the solution on the hidden constraints. Specifically, the index reduction algorithm is applied to the DAE system and reduced to index-0. For each reduction stage of the example considered, the computational cost is reported in Table 4.2. Notably, the results show that the expression complexity is similar for both LU and FFLU factorization, with a slight increase in the number of multiplications and divisions for the latter.

4.4.3.2 NUMERICAL INTEGRATION OF THE REDUCED-INDEX SYSTEM

The numerical stability and consistency of the reduced-index system are demonstrated by exploiting the analytical solution of the problem in [131]. Specifically, the DAE system consists of three position variables $[x_1,\, x_2,\, x_3]^\top$, three velocity variables $[u_1,\, u_2,\, u_3]^\top$, and one constraint with Lagrange multiplier λ. The solution manifold

Table 4.2: Expression complexity encountered throughout the index reduction of the particle motion DAE system with both the LU and FFLU factorization techniques. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Particle Motion (LU Factorization) [131]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 47\,\mathrm{f} + 30\,\mathrm{m} + 23\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$7\,\mathrm{f} + 10\,\mathrm{m} + 6\,\mathrm{a}$
Index-2 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$22\,\mathrm{f} + 20\,\mathrm{m} + 8\,\mathrm{a}$
Index-1 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$68\,\mathrm{f} + 72\,\mathrm{m} + 33\,\mathrm{a}$
Index-0 DAEs	$388\,\mathrm{f} + 424\,\mathrm{m} + 180\,\mathrm{a}$	$79\,\mathrm{f} + 77\,\mathrm{m} + 26\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 258\,\mathrm{f} + 239\,\mathrm{m} + 109\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 97\,\mathrm{f} + 102\,\mathrm{m} + 47\,\mathrm{a}$

Particle Motion (FFLU Factorization) [131]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 47\,\mathrm{f} + 30\,\mathrm{m} + 23\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$7\,\mathrm{f} + 10\,\mathrm{m} + 6\,\mathrm{a}$
Index-2 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$26\,\mathrm{f} + 23\,\mathrm{m} + 8\,\mathrm{a}$
Index-1 DAEs	0	$39\,\mathrm{f} + 36\,\mathrm{m} + 13\,\mathrm{a}$	$68\,\mathrm{f} + 72\,\mathrm{m} + 33\,\mathrm{a}$
Index-0 DAEs	$388\,\mathrm{f} + 424\,\mathrm{m} + 180\,\mathrm{a}$	$79\,\mathrm{f} + 77\,\mathrm{m} + 26\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 258\,\mathrm{f} + 239\,\mathrm{m} + 109\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 101\,\mathrm{f} + 105\,\mathrm{m} + 47\,\mathrm{a}$

is 4D, and the exact solution is

$$\mathbf{x}_{\text{exact}} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} (\rho\cos(2\pi - t) + r)\cos(t) \\ (\rho\cos(2\pi - t) + r)\sin(t) \\ \rho\sin(2\pi - t) \end{bmatrix}.$$

The initial value problem is defined as follows

$$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \begin{bmatrix} x_1' - u_1 \\ x_2' - u_2 \\ x_3' - u_3 \\ u_1' - u_3\cos(t) + x_3\sin(t) + u_2 - 2cx_1\lambda \\ u_2' - u_3\sin(t) - x_3\cos(t) - u_1 - 2cx_2\lambda \\ u_3' + x_3 - 2x_3\lambda \\ x_1^2 + x_2^2 + x_3^2 - 2r(x_1^2 + x_2^2)^{1/2} + r^2 - \rho^2 \end{bmatrix}, \tag{4.5}$$

with parameters $\rho = 5$ and $r = 10$, $c = 1 - r/(x_1^2 + x_2^2)^{1/2}$, states $\mathbf{x} = [x_1,\ x_2,\ x_3,\ u_1,\ u_2,\ u_3,\ \lambda]^\top$, and ICs $\mathbf{x}_0 = [15,\ 0,\ 0,\ 0,\ 15,\ -5,\ \lambda]^\top$.
The numerical integration of the reduced-index system is performed through Implicit Euler, RadauIIA3, and RadauIIA5 RK methods. To respect the invariants during the integration the *standard projection* method is applied [130]. This method consists of

projecting the solution $\mathbf{x}$ of the numerically integrated system onto the invariants on the hidden constraints $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$, which is equivalent to the constrained minimization

$$\underset{\mathbf{x}}{\text{minimize}} \quad \frac{1}{2}(\mathbf{x} - \tilde{\mathbf{x}})^2 \quad \text{subject to} \quad \mathbf{h}(\mathbf{x}, t) = \mathbf{0}\,.$$

To verify that the projection is performed correctly and does not affect the order of the RK method, numerical integration is performed in the interval $t \in [0,\ 2\pi]$ seconds with different integration time steps Δt. The error of the numerical integration $\varepsilon = \|\mathbf{x} - \mathbf{x}_{\text{exact}}\|_\infty$ is reported in Figure 4.2. As it can be seen, the implemented projection preserves the order of the method for all the integration time steps. It is important to highlight that to obtain such results the absolute error tolerances of the integrator and the projection are both set to $\varepsilon = 10^{-10}$. The same tolerances are used in the numerical integration of the reduced-index system in the interval $t \in [0,\ 400\pi]$ seconds with step $\Delta t = 0.025$ seconds. The results are reported in Figure 4.3, where Implicit Euler, RadauIIA3, and RadauIIA5 RK methods are employed, and the projection on the hidden constraints $\mathbf{h}(\mathbf{x}, t)$ is performed. The effect of the projection is highlighted on the bottom left plot.

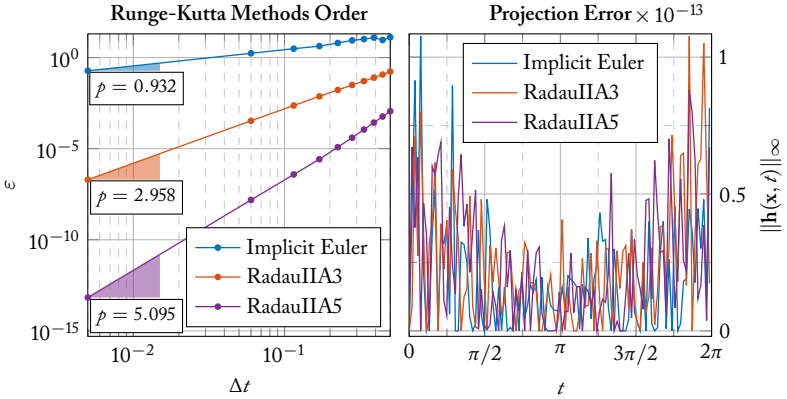


Figure 4.2: Numerical integration error $\varepsilon = \|\mathbf{x} - \mathbf{x}_{\text{exact}}\|_\infty$ of the DAEs (4.5) over different integration time steps Δt, along with the computed order of the method (left). The projection on the hidden constraints is performed and the invariants violation $\|\mathbf{h}(\mathbf{x}, t)\|_\infty$ is reported (right). Notice that the implemented projection preserves the order of the method for all the integration time steps. The tests are performed in $t \in [0,\ 2\pi]$ seconds, using Implicit Euler, RadauIIA3, and RadauIIA5 RK methods.

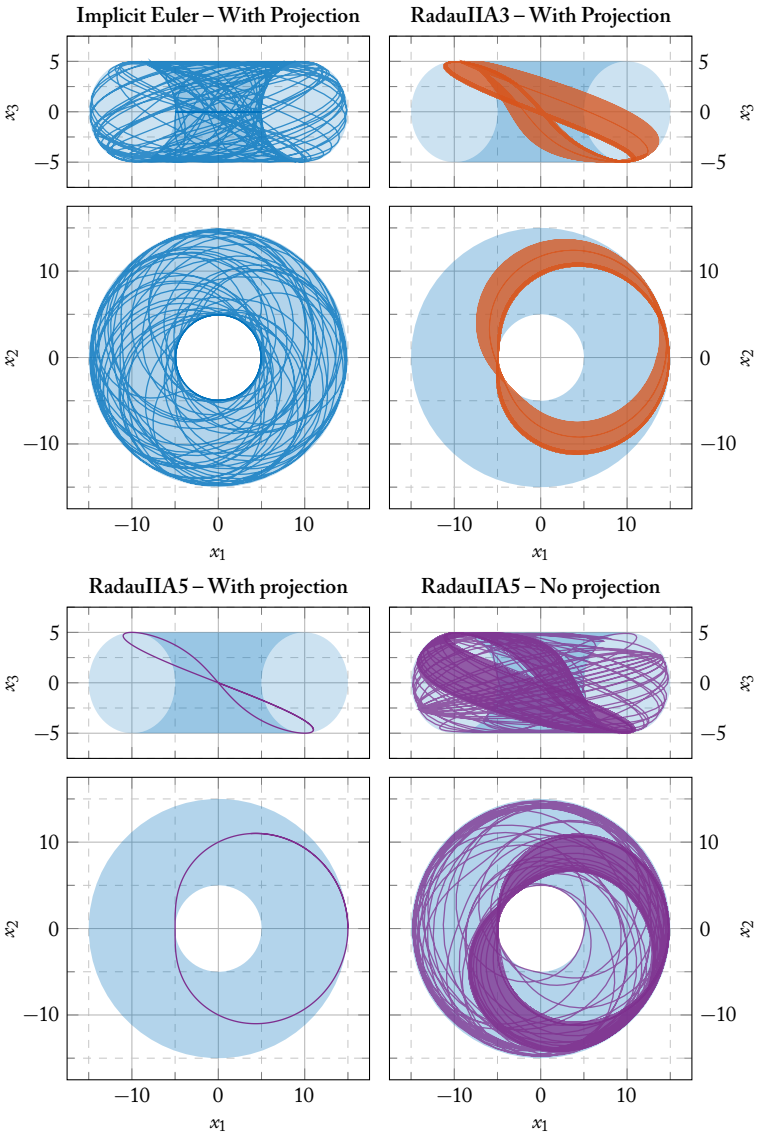


Figure 4.3: Numerically integrated solution of the DAEs (4.5) in the interval $t \in [0, 400\pi]$ seconds, with step $\Delta t = 0.025$ seconds, using Implicit Euler (top left), RadauIIA3 (top right), and RadauIIA5 (bottom left and right) RK methods.

CHAPTER 5

APPLICATION FIELDS AND EXAMPLES

In the previous chapter, we presented a novel methodology for the automatic index reduction of DAE systems. Such an index reduction is based on the separation of the system into differential and algebraic parts with the help of symbolic linear algebra, *i.e.*, LU, or FFLU matrix factorization. No information on the DAE system structure is leveraged during the reduction process. For this reason, this methodology can be applied to generic DAEs linear in the state derivatives. Numerical integration goodness is showcased by assessing the order of the numerical integration scheme and the accuracy of the projection. In this chapter, we take for granted the correctness of the proposed methodology, and we focus on the application and capabilities of the index reduction algorithm.

5.1 BENCHMARK PROBLEMS AND SOFTWARE COMPARISON

The algorithm is applied to a variety of DAE systems, which are chosen to demonstrate the capabilities and the wide range of applicability of the proposed methodology. The examples are divided into three main categories: *multi-body dynamics*, *trajectory prescribed path control* problems, and *electrical circuits*. Each of these categories is characterized by a different structure of the DAE system, which is analyzed in detail and helps us to understand the effectiveness of the proposed methodology. The examples are mainly taken from the test sets [132–135] and are the following.

1. Multi-body dynamics:
   (a) car-axis system [132, 134];
   (b) flexible slider-crank mechanism [132, 134];

(c) the multi-pendula system [77];
(d) double-wishbone suspension system [132, 134].

2. Trajectory prescribed path control problems:

(a) initial stage space shuttle reentry problem [9];
(b) final stage space shuttle reentry problem [9];
(c) the robotic arm system [60].

3. Electrical circuits:

(a) eight-nodes transistor-amplifier [132, 134];
(b) electric ring modulator [132, 134];
(c) cascaded differential amplifier [9].

Notably, in the application example 1d, the sub-models and techniques used for both tire-ground interaction and model reduction are those presented in Appendices A, B, C, and D. Moreover, such an example is also extensively discussed in [6, 8].
A comparison between the built-in joint index reduction algorithm and numerical integration schemes offered by MAPLE®, with those of INDIGO, will demonstrate the effectiveness of the proposed methodology and software implementation. Notice that, to ensure a fair comparison, the same Runge-Kutta-Fehlberg (RKF) 4(5) method with a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-7} is used in both software packages. If such a method fails, the integration is carried out with the RadauIIA5 method in INDIGO and the implicit Rosenbrock 3(4) method in MAPLE®. The computation time limit for both index reduction and code generation is 100 s. Numerical integration is not limited by a maximum computation time, however the results are only reported if the integration is completed thoroughly.
For the sake of completeness, we also compare *some* of our results with those obtained by the MATLAB® symbolic toolbox, which works on the MUPAD CAS. Notably, MATLAB® is capable of performing DAEs index reduction through either the Pantelides algorithm (`reduceDAEToODE` function) [36] or the Gaussian elimination method (`reduceDAEIndex` function). The first is reported to be less robust and to have a higher computational cost. On the other hand, the Gaussian elimination method is claimed to be more robust and, to some extent, might be similar to the approach proposed in this work, although its working principles are not fully disclosed and its success is guaranteed only if semilinear DAE systems are considered [136]. For the same examples solved with MATLAB®, we also include the results obtained by MATHEMATICA®, which performs DAEs index reduction through the Pantelides algorithm too. It is worth noticing that MATHEMATICA® does not provide access to the reduced index DAE system. Moreover, MATLAB® has no built-in function to compute the computational cost of each equation in the reduced-index DAE system. This makes the comparison with the proposed methodology and other software packages more challenging. For this

reason, we will only limit our considerations to overall performance, *i.e.*, the success of both the index reduction process and numerical integration processes. A summary of this comparison will be later reported in Table (5.18).
Before we proceed with the examples, it is also worth mentioning that during the index reduction process, the trigonometric identities provided by Weierstraß (3.2) or (3.1) can be used to reformulate the DAEs expressions, as well as to obtain polynomials expressions and improve the detection of symbolic eliminations (see Section 3.4.3). However, using such trigonometric identities does not typically lead to a significant improvement in the computational cost of the expressions generated during the index reduction procedure as the polynomial expressions obtained are difficult to simplify as well. Furthermore, such transformations impose a change of coordinates in the original DAE system, which may be undesirable in some applications. For this reason, no reformulation is carried out in the examples presented in this chapter.

5.2 MULTI-BODY DYNAMICS

Historically, the MBD problems appeared in the early 60s when the first computer simulations of mechanical systems were performed. Over the years, such systems have been studied extensively, and many numerical or hybrid symbolic-numeric methods have been developed to solve them [13]. As a consequence, the MBD field has become one of the most relevant areas of research in the mechanical engineering domain having a wide range of applications. In this section, we present four examples of MBD problems, which are solved using the proposed index reduction algorithm. But first, we provide a brief overview of the MBD problems, their mathematical formulation, as well as the formal index reduction of the corresponding generic MB DAEs.
The MBD category is characterized by DAE systems that describe the motion of a mechanism composed of interconnected rigid or flexible bodies. Within these systems, the differential equations represent motion equations, while the algebraic equations correspond to kinematic constraints, collectively constituting these MB problems. Typically, such problems are posed as Hessenberg semi-explicit index-3 DAEs of the form

$$\begin{cases} \mathbf{p} = \mathbf{q}' \\ \mathbf{M}(\mathbf{q},\mathbf{p},t)\mathbf{p}' - \mathbf{\Phi}_\mathbf{q}(\mathbf{q},t)^\top \boldsymbol{\lambda} = \mathbf{f}(\mathbf{q},\mathbf{p},t)\,, \\ \mathbf{\Phi}(\mathbf{q},t) = \mathbf{0} \end{cases} \quad \text{with} \quad \mathbf{\Phi}_\mathbf{q}(\mathbf{q},t) = \frac{\partial}{\partial \mathbf{q}}\mathbf{\Phi}(\mathbf{q},t)\,, \tag{5.1}$$

where $\mathbf{q} \in \mathbb{R}^n$ and $\mathbf{p} \in \mathbb{R}^n$ indicate respectively the generalized coordinates and the generalized velocities, $\mathbf{M}(\mathbf{q},\mathbf{p},t) \in \mathbb{R}^{n\times n}$ is the mass matrix, $\mathbf{\Phi}(\mathbf{q},t) \in \mathbb{R}^m$ is the constraint vector, $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the vector of Lagrange multipliers, and lastly $\mathbf{f}(\mathbf{q},\mathbf{p},t) \in \mathbb{R}^n$ collects all the contributions from Coriolis and centrifugal effects, as well as external forces.

INDEX REDUCTION OF MULTI-BODY DYNAMICS EQUATIONS To solve the problem in (5.1), one of the possible approaches is to transform the problem into a

system of ODEs with invariants. Moreover, one may also isolate the Lagrange multipliers $\boldsymbol{\lambda}$ and write them explicitly in terms of the state variables to obtain an index-1 DAEs representation of the MBD problem. In such cases, the index reduction can be performed by exploiting the structure of the problem, as well as the properties of the mass matrix $\mathbf{M}(\mathbf{q},\mathbf{p},t)$ and constraint vector $\boldsymbol{\Phi}(\mathbf{q},t)$. To do so, we first need to differentiate the constraint $\boldsymbol{\Phi}(\mathbf{q},t)$, this yields to

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Phi}(\mathbf{q},t) = \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p} + \boldsymbol{\Phi}_{t}(\mathbf{q},t) = \mathbf{0}\,, \quad \text{with} \quad \boldsymbol{\Phi}_{t}(\mathbf{q},t) = \frac{\partial}{\partial t}\boldsymbol{\Phi}(\mathbf{q},t)\,. \tag{5.2}$$

Notice that when constraint does not depend on time (5.2) expresses the intuitive idea that generalized velocity $\mathbf{p}$ must be orthogonal to constraint's gradient. This provides the opportunity to highlight that constraint $\boldsymbol{\Phi}(\mathbf{q},t)$ imposes relationships also at the velocity and acceleration level, *i.e.*, (5.2) restricts the set of feasible velocities. Finally, it must be noticed that (5.2) is still algebraic in the $\mathbf{p}$ coordinates, thus to obtain a fully differential relationship, another differentiation is needed. This results in the following equation

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{\Phi}(\mathbf{q},t) = \dot{\boldsymbol{\Phi}}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p} + \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p}' + \boldsymbol{\Phi}_{tt}(\mathbf{q},t) = \mathbf{0}\,, \tag{5.3}$$

$$\text{with} \quad \boldsymbol{\Phi}_{tt}(\mathbf{q},t) = \frac{\partial^2}{\partial t^2}\boldsymbol{\Phi}(\mathbf{q},t)\,, \quad \text{and} \quad \dot{\boldsymbol{\Phi}}_{\mathbf{q}}(\mathbf{q},t) = \frac{\mathrm{d}}{\mathrm{d}t}\,\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)\,,$$

which does not impose any algebraic constraint on the mechanical system states and leads to the following index-1 DAE system

$$\begin{cases} \mathbf{q}' = \mathbf{p} \\ \mathbf{M}(\mathbf{q},\mathbf{p},t)\mathbf{p}' - \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}\boldsymbol{\lambda} = \mathbf{f}(\mathbf{q},\mathbf{p},t)\,. \\ -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p}' = \dot{\boldsymbol{\Phi}}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p} + \boldsymbol{\Phi}_{tt}(\mathbf{q},t) \end{cases} \tag{5.4}$$

System (5.4) can be also written in compact matrix notation as

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}(\mathbf{q},\mathbf{p},t) & -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top} \\ \mathbf{0} & -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}' \\ \mathbf{p}' \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{f}(\mathbf{q},\mathbf{p},t) \\ \dot{\boldsymbol{\Phi}}_{\mathbf{q}}(\mathbf{q},t)\mathbf{p} + \boldsymbol{\Phi}_{tt}(\mathbf{q},t) \end{bmatrix}. \tag{5.5}$$

It is easy to notice that if the left-hand side matrix of (5.5) is invertible, then we can write the system in explicit form. In specific, the matrix is invertible if and only if the sub-block

$$\begin{bmatrix} \mathbf{M}(\mathbf{q},\mathbf{p},t) & -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top} \\ -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t) & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$$

is non-singular. To prove its non-singularity the rank additivity formula is applied as follows

$$\begin{aligned} \operatorname{rank}\left(\begin{bmatrix} \mathbf{M}(\mathbf{q},\mathbf{p},t) & -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top} \\ -\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t) & \mathbf{0} \end{bmatrix}\right) &= \operatorname{rank}\left(\mathbf{M}(\mathbf{q},\mathbf{p},t)\right)\ldots \\ &\quad + \operatorname{rank}\left(\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}\right). \end{aligned}$$

Since $\mathbf{M}(\mathbf{q}, \mathbf{p}, t)$ is positive definite (and thus also non-singular), then $\forall \mathbf{q} \in \mathbb{R}^n$, $\text{rank}(\mathbf{M}(\mathbf{q}, \mathbf{p}, t)) = n$. Moreover, the second term in the right-hand side of the equation above is non-negative and it is zero if and only if

$$\text{rank}\left(\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)\mathbf{M}(\mathbf{q}, \mathbf{p}, t)^{-1}\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top\right) = m\,, \tag{5.6}$$

which is not generally true. However, if we assume that the constraints are *locally* linearly independent, then $\forall \mathbf{q} \in \mathbb{R}^n$, $\forall t \in \mathbb{R}$, $\text{rank}(\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)) = m$. Therefore, the matrix in (5.5) is invertible, and the overall system is written explicitly as

$$\begin{bmatrix} \mathbf{q}' \\ \mathbf{p}' \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}(\mathbf{q}, \mathbf{p}, t) & -\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top \\ \mathbf{0} & -\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t) & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} \\ \mathbf{f}(\mathbf{q}, \mathbf{p}, t) \\ \dot{\mathbf{\Phi}}_\mathbf{q}(\mathbf{q}, t)\mathbf{p} + \mathbf{\Phi}_t(\mathbf{q}, t) \end{bmatrix}$$

An explicit expression for the inverse matrix above can be obtained through tedious symbolic calculations. However, regardless of the specific solution, we want to stress an important point: by differentiating the constraint twice, we can now explicitly write an expression for the Lagrange multipliers (index-1 variables). If the accelerations $\mathbf{p}'$ are isolated from (5.5) and substituted into (5.3), we obtain

$$\mathbf{p}' = \mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top\boldsymbol{\lambda} + \mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{f}(\mathbf{q}, \mathbf{p}, t) = \mathbf{0}\,. \tag{5.7}$$

Then, substituting (5.7) into (5.3) we obtain the following expression

$$\begin{aligned} \dot{\mathbf{\Phi}}_\mathbf{q}(\mathbf{q}, t)\mathbf{p} + \mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)\mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top\boldsymbol{\lambda} \ldots \\ + \mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)\mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{f}(\mathbf{q}, \mathbf{p}, t) + \mathbf{\Phi}_t(\mathbf{q}, t) = \mathbf{0}\,. \end{aligned}$$

Here the non-singular matrix in (5.6) is easily recognized, therefore the expression for the Lagrange multipliers is given by

$$\begin{aligned} \boldsymbol{\lambda} = -\left(\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)\mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top\right)^{-1} \ldots \\ \left(\dot{\mathbf{\Phi}}_\mathbf{q}(\mathbf{q}, t)\mathbf{p} + \mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)\mathbf{M}(\mathbf{q}, \mathbf{q}', t)^{-1}\mathbf{f}(\mathbf{q}, \mathbf{p}, t) + \mathbf{\Phi}_t(\mathbf{q}, t)\right), \end{aligned}$$

which can be either substituted into the explicit form of the DAE system (5.4) or properly applied to obtain a numerically efficient representation of the MBD problem, *i.e.*,

$$\text{a set of ODEs} \quad \begin{cases} \mathbf{q}' = \mathbf{p} \\ \mathbf{p}' = \dot{\mathbf{p}} \end{cases},$$

where $\dot{\mathbf{p}}$ is obtained by solving the linear system

$$\begin{bmatrix} \mathbf{M}(\mathbf{q}, \mathbf{p}, t) & -\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t)^\top \\ -\mathbf{\Phi}_\mathbf{q}(\mathbf{q}, t) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{p}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{q}, \mathbf{p}, t) \\ \dot{\mathbf{\Phi}}_\mathbf{q}(\mathbf{q}, t)\mathbf{p} + \mathbf{\Phi}_t(\mathbf{q}, t) \end{bmatrix}.$$

Technically, mechanical systems subject to holonomic constraints are represented by DAEs systems of index 3, *i.e.*, the differential index is one plus the number of differentiations of the constraint that are needed to be able to eliminate the Lagrange multipliers. Otherwise, equivalently, the number of times that we need to differentiate the constraint to obtain a differential equation also for each of the Lagrange multipliers.

EXPLICIT MATRIX INVERSE For the sake of completeness, we provide the explicit expression for the inverse of matrix in (5.5), which is given by

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}(\mathbf{q},\mathbf{p},t) & -\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top} \\ \mathbf{0} & -\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t) & \mathbf{0} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_{11} & \mathbf{X}_{12} \\ \mathbf{0} & \mathbf{X}_{21} & \mathbf{X}_{22} \end{bmatrix}$$

where the quantities $\mathbf{X}_{11} \in \mathbb{R}^{n\times n}$, $\mathbf{X}_{12} \in \mathbb{R}^{n\times m}$, $\mathbf{X}_{21} \in \mathbb{R}^{m\times n}$, and $\mathbf{X}_{22} \in \mathbb{R}^{m\times m}$ are defined as follows

$$\begin{aligned} \mathbf{X}_{11} &= \mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1} + \mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}\mathbf{X}_{22}\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}, \\ \mathbf{X}_{12} &= \mathbf{X}_{21}^{\top} = \mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}\mathbf{X}_{22}, \\ \mathbf{X}_{22} &= -\left(\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}\right)^{-1}. \end{aligned}$$

Notice that the matrix $\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)\mathbf{M}(\mathbf{q},\mathbf{p},t)^{-1}\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q},t)^{\top}$ also appears in (5.6), for which invertibility must be guaranteed.

5.2.1 CAR-AXIS DYNAMICS

After introducing the MBD problems, we now present the first example, which is the car-axis dynamics problem. This example is taken from [132, 134] and is used to model the motion of a car as riding over an uneven road. Here, the system is modeled as two springs connected to two bars. The bottom of the left tire is the origin. The chassis is represented by a bar of mass m, and its position is given by $\mathbf{q} = [x_l, y_l, x_r, y_r]^{\top}$, as also illustrated in Figure 5.1. The left tire is always on a flat surface, while the right tire periodically rides over a sinusoidal road surface of height $y_b(t) = h\sin(\omega t)$. The distance between the wheels is fixed, and the car chassis must always have a fixed length. Hence, the following constraints are imposed

$$\mathbf{\Phi}(\mathbf{q}) = \begin{bmatrix} x_l x_b - y_l y_b \\ (x_l - x_r)^2 + (y_l - y_r)^2 - \ell^2 \end{bmatrix} = \mathbf{0}.$$

The equations of motion for the car are derived using Lagrangian mechanics. The mass matrix $\mathbf{M}$, and the force vector $\mathbf{f}$ are given by

$$\mathbf{M}(\mathbf{q}) = \operatorname{diag}\left((\ell_0 - \ell_l)\frac{x_l}{\ell_l}, (\ell_0 - \ell_l)\frac{x_l}{\ell_l}, (\ell_0 - \ell_r)\frac{x_r}{\ell_r}, (\ell_0 - \ell_r)\frac{x_r}{\ell_r}\right)$$

and

$$\mathbf{f} = \left[0, -\varepsilon^2 \frac{m}{2}, 0, -\varepsilon^2 \frac{m}{2}\right]^\top,$$

where

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

are respectively the length of the left and right springs, ℓ_0 is the relaxed length of both springs, h is the amplitude of the bump, and $1/\varepsilon^2$ is the Hooke's constant of the springs. The parameters of the car are chosen as $\ell = 1\,\text{m}$, $\ell_0 = 1/2\,\text{m}$, $\varepsilon = 0.01$, $m = 10\,\text{kg}$, $h = 0.1\,\text{m}$, and $\omega = 10\,\text{rad/s}$. While the ICs according to [132, 134] are $\mathbf{q}_0 = [-1/2, 0, -1/2, 0]^\top$ and $\mathbf{p}_0 = [0, 1/2, 1, 1/2]^\top$, with an integration time interval of $t \in [0, 3]\,\text{s}$.

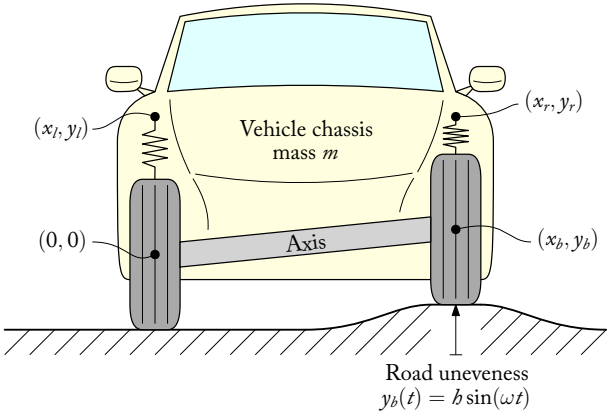


Figure 5.1: Car-axis dynamics problem [132, 134]. The car is modeled as springs connected to two bars. The bottom of the left tire is the origin. The car chassis is represented by a bar of mass m. The position of the chassis is given by $\mathbf{q} = [x_l, y_l, x_r, y_r]^\top$. The left tire is always on a flat surface, while the right tire periodically goes over a sinusoidal bump of the form $y_b(t) = h \sin(\omega t)$. The distance between the wheels must always remain fixed, and the car chassis must always have a fixed length.

The car-axis DAE system is solved using the proposed index reduction algorithm with both LU and FFLU factorization. The results are reported in Table 5.1. As expected, the index reduces to 0, and the system is transformed into a set of ODEs. The complexity of the expressions encountered during the index reduction is given in terms of the number of functions f, additions a, multiplications m, and divisions d; and, as reported in the table, the index reduction is successful and little expression swell is observed only in the last reduction step. Notice that the FFLU factorization produces more complex expressions than the LU factorization and, for this reason, the former is

dropped in the following simulations. The reduced system is then numerically solved using the RKF 4(5) integrator, and the results are shown in Figure 5.2, where the springs' lengths are plotted. Notably, during the simulation, the left spring remains nearly constant at its relaxed length, while the right spring length periodically varies due to road bumps. It is worth mentioning that also MATHEMATICA® and MATLAB® (with both Pantelides and Gaussian elimination techniques) can correctly reduce the index of the car-axis problem and integrate the resulting system.

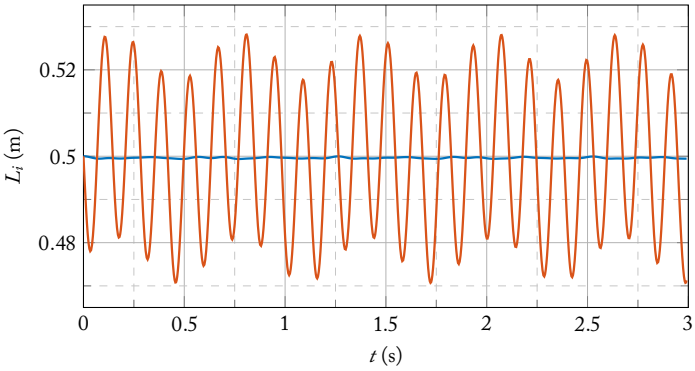


Figure 5.2: Springs length of the car-axis problem [132, 134]. *Legend*: ■ left spring L_l, ■ right spring L_r.

5.2.2 FLEXIBLE SLIDER-CRANK MECHANISM

The slider-crank mechanism presented in Figure 5.3 is a classic problem in mechanical engineering. The mechanism consists of a crank, a connecting rod, and a slider. The crank is driven by a motor at a constant angular velocity Ω, and the slider moves back and forth in a straight line. In this case, the problem is a flexible MB system. Specifically, the rod connecting the crank and the slider is a flexible beam. The flexible MBD problem is posed as a second-order index-3 DAEs of the form

$$\begin{cases} \mathbf{M}(\mathbf{q}_r, \mathbf{q}_f) \begin{bmatrix} \mathbf{q}_r'' \\ \mathbf{q}_f'' \end{bmatrix} - \mathbf{\Phi}_{\mathbf{q}_r,\mathbf{q}_f}(\mathbf{q}_r, \mathbf{q}_f, t)^\top \boldsymbol{\lambda} = \mathbf{f}(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') \\ \mathbf{\Phi}(\mathbf{q}_r) = \mathbf{0} \end{cases},$$

with

$$\mathbf{\Phi}_{\mathbf{q}_r,\mathbf{q}_f}(\mathbf{q}_r, \mathbf{q}_f, t) = \frac{\partial \mathbf{\Phi}(\mathbf{q}_r, \mathbf{q}_f, t)}{\partial(\mathbf{q}_r, \mathbf{q}_f)},$$

Table 5.1: Expression complexity encountered throughout the index reduction of the car-axis problem [132, 134] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Car-Axis (LU Factorization) [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 108\,\mathrm{f} + 131\,\mathrm{m} + 56\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$	
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	12 m	94 f + 145 m + 54 a	14 f + 16 m + 10 a
Index-2 DAEs	12 m	94 f + 145 m + 54 a	26 f + 45 m + 15 a
Index-1 DAEs	12 m	94 f + 145 m + 54 a	136 f + 4 d + 261 m + 95 a
Index-0 DAEs	1060 f + 38 d + 1901 m + 717 a	431 f + 8 d + 842 m + 268 a	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 896\,\mathrm{f} + 4\,\mathrm{d} + 1202\,\mathrm{m} + 546\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 176\,\mathrm{f} + 4\,\mathrm{d} + 322\,\mathrm{m} + 120\,\mathrm{a}$	

Car-Axis (FFLU Factorization) [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 108\,\mathrm{f} + 131\,\mathrm{m} + 56\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$	
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	0	94 f + 8 d + 150 m + 54 a	14 f + 21 m + 10 a
Index-2 DAEs	0	94 f + 8 d + 154 m + 54 a	26 f + 1 d + 44 m + 15 a
Index-1 DAEs	0	94 f + 8 d + 155 m + 54 a	136 f + 6 d + 4 d + 261 m + 95 a
Index-0 DAEs	1066 f + 55 d + 1888 m + 717 a	431 f + 18 d + 851 m + 268 a	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 1549\,\mathrm{f} + 73\,\mathrm{d} + 2765\,\mathrm{m} + 1011\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 176\,\mathrm{f} + 7\,\mathrm{d} + 326\,\mathrm{m} + 120\,\mathrm{a}$	

where the position or gross motion coordinates are

$$\mathbf{q}_r = \begin{bmatrix} \phi_1 \\ \phi_2 \\ x_3 \end{bmatrix} \quad \begin{array}{l} \text{crank angle,} \\ \text{connecting rod angle,} \\ \text{sliding block displacement.} \end{array}$$

The deformation coordinates of the flexible connecting rod are

$$\mathbf{q}_f = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \quad \begin{array}{l} \text{first lateral mode } \sin(\pi x/\ell_2)\,, \\ \text{second lateral mode } \sin(2\pi x/\ell_2)\,, \\ \text{longitudinal displacement midpoint,} \\ \text{longitudinal displacement endpoint.} \end{array}$$

The mass matrix $\mathbf{M}(\mathbf{q}_r, \mathbf{q}_f)$, and the force vector $\mathbf{f}(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f')$ are given by

$$\mathbf{M}(\mathbf{q}_r, \mathbf{q}_f) = \begin{bmatrix} \mathbf{M}_r(\mathbf{q}_r) + \mathbf{M}_e(\mathbf{q}_r, \mathbf{q}_f) & \mathbf{C}(\mathbf{q}_r, \mathbf{q}_f)^\top \\ \mathbf{C}(\mathbf{q}_r, \mathbf{q}_f) & \mathbf{M}_d \end{bmatrix},$$

with rigid motion mass matrix

$$\mathbf{M}_r(\mathbf{q}_r) = \begin{bmatrix} J_1 + m_2\ell_1^2 & \ell_1\ell_2 m_2 \cos(\phi_{12})/2 & 0 \\ \ell_1\ell_2 m_2 \cos(\phi_{12})/2 & J_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix},$$

coupling blocks

$$\mathbf{M}_e(\mathbf{q}_r, \mathbf{q}_f) = \begin{bmatrix} 0 & \rho\ell_1(\cos(\phi_{12})\mathbf{c}_1^\top + \sin(\phi_{12})\mathbf{c}_2^\top)\mathbf{q}_f & 0 \\ \rho\ell_1(\cos(\phi_{12})\mathbf{c}_1^\top + \sin(\phi_{12})\mathbf{c}_2^\top)\mathbf{q}_f & \mathbf{q}_f^\top \mathbf{M}_d \mathbf{q}_f + 2\rho\mathbf{c}_{12}^\top \mathbf{q}_f & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\mathbf{C}(\mathbf{q}_r, \mathbf{q}_f)^\top = \begin{bmatrix} \rho\ell_1(\cos(\phi_{12})\mathbf{c}_2^\top - \sin(\phi_{12})\mathbf{c}_1^\top) \\ \rho\mathbf{c}_{21}^\top + \rho\mathbf{q}_f^\top \mathbf{B} \\ \mathbf{0}^\top \end{bmatrix},$$

as well as elastic body space discretization mass matrix

$$\mathbf{M}_d = \rho d h \ell_2 \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 8 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

The force vector is given by

$$\mathbf{f}(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') = \begin{bmatrix} \mathbf{f}_r(\mathbf{q}_r, \mathbf{q}_r') + \mathbf{f}_e(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') \\ \mathbf{f}_d(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') - \nabla\mathbf{W}_d(\mathbf{q}_f) - \mathbf{D}_d\mathbf{q}_f' \end{bmatrix}.$$

where the rigid motion terms are collected in

$$\mathbf{f}_r(\mathbf{q}_r, \mathbf{q}_r') = \begin{bmatrix} -\ell_1(\gamma(m_1 + 2m_2)\cos(\phi_1)/2 + \ell_2 m_2 \phi_2'^2 \sin(\phi_{12})) \\ -\ell_2 \gamma m_2 \cos(\phi_2)/2 + \ell_1 \ell_2 m_2 \phi_1'^2 \sin(\phi_{12})/2 \\ 0 \end{bmatrix} .$$

and $\phi_{12} = \phi_1 - \phi_2$. For the force term $\mathbf{f}_e(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f')$ the expression is

$$\mathbf{f}_e(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') = \begin{bmatrix} \rho\ell_1\phi_2'^2(\cos(\phi_{12})\mathbf{c}_2^\top - \sin(\phi_{12})\mathbf{c}_1^\top)\mathbf{q}_f - 2\rho\ell_1\phi_2'(\cos(\phi_{12})\mathbf{c}_1^\top + \sin(\phi_{12})\mathbf{c}_2^\top)\mathbf{q}_f' \\ \rho\ell_1\phi_1'^2(\sin(\phi_{12})\mathbf{c}_1^\top - \cos(\phi_{12})\mathbf{c}_2^\top)\mathbf{q}_f - 2\rho\phi_2'\mathbf{c}_{12}^\top\mathbf{q}_f' - 2\phi_2'\mathbf{q}_f'^\top\mathbf{M}_d\mathbf{q}_f \dots \\ -\rho\mathbf{q}_f'^\top\mathbf{B}\mathbf{q}_f' - \rho\gamma(\cos(\phi_2)\mathbf{c}_1^\top\mathbf{q}_f - \sin\phi_2\mathbf{c}_2^\top\mathbf{q}_f) \\ 0 \end{bmatrix} ,$$

and for $f_d(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f')$ the expression is

$$\begin{aligned} f_d(\mathbf{q}_r, \mathbf{q}_f, \mathbf{q}_r', \mathbf{q}_f') = \phi_2'^2\mathbf{M}_d\mathbf{q}_f + \rho(\phi_2'^2\mathbf{c}_{12} + \ell_1\phi_1'^2(\cos(\phi_{12})\mathbf{c}_1 \dots \\ + \sin(\phi_{12})\mathbf{c}_2) + 2\phi_2'\mathbf{B}\mathbf{q}_f') - \rho\gamma\,(\sin\phi_2\mathbf{c}_1 + \cos(\phi_2)\mathbf{c}_2) . \end{aligned}$$

The gradient of the elastic potential $\mathbf{W}_d(\mathbf{q}_f)$ in case of linear elasticity is $\nabla\mathbf{W}_d(\mathbf{q}_f) = \mathbf{K}_d\mathbf{q}_f$ with the stiffness matrix

$$\mathbf{K}_d = \frac{Edh}{\ell_2}\begin{bmatrix} \pi^4/24(h/\ell_2)^2 & 0 & 0 & 0 \\ 0 & \pi^4 2/3(h/\ell_2)^2 & 0 & 0 \\ 0 & 0 & 16/3 & -8/3 \\ 0 & 0 & -8/3 & 7/3 \end{bmatrix} .$$

Alternatively, in the case of the nonlinear beam model, it holds $\nabla\mathbf{W}_d(\mathbf{q}_f) = \mathbf{K}_d\mathbf{q}_f + \mathbf{K}_d(\mathbf{q}_f)$, where

$$\mathbf{K}_d(\mathbf{q}_f) = \frac{\pi^2 Edh}{2\ell_2^2}\begin{bmatrix} q_1q_4 - \beta q_2(-4q_3 + 2q_4) \\ 4q_2q_4 - \beta q_1(-4q_3 + 2q_4) \\ 4\beta q_1q_2 \\ q_1^2/2 + 2q_2^2 - 2\beta q_1q_2 \end{bmatrix} , \quad \text{with} \quad \beta = \frac{80}{9\pi^2} .$$

The damping matrix $\mathbf{D}_d$ is by default zero. The coupling matrices and vectors arising from the space discretization read

$$\mathbf{B} = dh\ell_2\begin{bmatrix} 0 & 0 & -16/\pi^3 & 8/\pi^3 - 1/\pi \\ 0 & 0 & 0 & 1/(2\pi) \\ 16/\pi^3 & 0 & 0 & 0 \\ 1/\pi - 8/\pi^3 & -1/(2\pi) & 0 & 0 \end{bmatrix} ,$$

and

$$\begin{aligned}
\mathbf{c}_1 &= dh\ell_2\ [0,0,2/3,1/6]^\top\ ,\\
\mathbf{c}_2 &= dh\ell_2\ [2/\pi,0,0,0]^\top\ ,\\
\mathbf{c}_{12} &= dh\ell_2^2\ [0,0,1/3,1/6]^\top\ ,\\
\mathbf{c}_{21} &= dh\ell_2^2\ [1/\pi,-1/(2\pi),0,0]^\top\ .
\end{aligned}$$

The constraints of the slider-crank mechanism are given by

$$\mathbf{\Phi}(\mathbf{q}_r) = \begin{bmatrix} \ell_1\sin(\phi_1)+\ell_2\sin(\phi_2)+q_4\sin(\phi_2) \\ x_3-\ell_1\cos(\phi_1)-\ell_2\cos(\phi_2)-q_4\cos(\phi_2) \\ \phi_1-\Omega t \end{bmatrix} = \mathbf{0}\,.$$

For the simulation, the following parameters are used:

$\ell_1 = 0.15\,\mathrm{m}$	body 1 length,
$\ell_2 = 0.30\,\mathrm{m}$	body 2 length,
$m_1 = 0.36\,\mathrm{kg}$	body 1 mass,
$m_2 = 0.151104\,\mathrm{kg}$	body 2 mass,
$m_3 = 0.075552\,\mathrm{kg}$	body 3 mass,
$J_1 = 0.002727\,\mathrm{kg\,m^2}$	body 1 moment of inertia,
$J_2 = 0.0045339259\,\mathrm{kg\,m^2}$	body 2 moment of inertia,
$\rho = 7870\,\mathrm{kg/m^3}$	rod density,
$E = 200\,\mathrm{GN/m^2}$	rod Young's modulus,
$\gamma = 0$	gravity constant,
$\Omega = 150\,\mathrm{rad/s}$	prescribed crank angular velocity.

The ICs according to [132, 134] are

$$\mathbf{q}_{r\,0} = \begin{bmatrix} 0.000000000000000\cdot10^{+0} \\ 0.000000000000000\cdot10^{+0} \\ 4.500169330000000\cdot10^{-1} \end{bmatrix}\ ,\quad \mathbf{q}'_{r\,0} = \begin{bmatrix} 1.500000000000000\cdot10^{+2} \\ -7.499576703969453\cdot10^{+1} \\ -2.689386719979040\cdot10^{-6} \end{bmatrix}\ ,$$

$$\mathbf{q}_{f0} = \begin{bmatrix} 0.000000000000000\cdot10^{+0} \\ 0.000000000000000\cdot10^{+0} \\ 1.033398630000000\cdot10^{-5} \\ 1.693279690000000\cdot10^{-5} \end{bmatrix}\ ,\quad \mathbf{q}'_{f0} = \begin{bmatrix} 4.448961125815990\cdot10^{-1} \\ 4.634339319238670\cdot10^{-3} \\ -1.785910760000550\cdot10^{-6} \\ -2.689386719979040\cdot10^{-6} \end{bmatrix}\ ,$$

$$\boldsymbol{\lambda}_0 = \begin{bmatrix} 6.552727150584648\cdot10^{-8} \\ -3.824589509350831\cdot10^{+2} \\ 4.635908708561371\cdot10^{-9} \end{bmatrix}\ ,$$

with an integration time interval of $t \in [0,\,0.1]$ s.
The slider-crank mechanism DAE system is solved with the aid of the proposed index reduction algorithm. Specifically, both the linear and nonlinear flexible beam models are considered. The results in terms of expression complexity encountered throughout the index reduction are similar for both formulations, and they are summarized

in Table 5.2. As shown in the table, the expression complexity increases significantly during the index reduction process, which is expected due to the inherent complexity of the problem. Hierarchical representation is also employed to limit the expression swelling, and the results are summarized in Table 5.3. The results show that the hierarchical representation reduces the expression complexity by an order of magnitude, which is beneficial for the numerical solution of the DAE system. Nevertheless, it must be highlighted that, despite the mitigation of the expression complexity, the hierarchical representation does not prevent the expression complexity from increasing significantly during the index reduction process, which allows us to conclude that this problem is affected by inherent expression swelling. The results of the numerical simulation are shown in Figure 5.4, where the longitudinal displacements q_3 and q_4 of the flexible beam are illustrated. The flexible beam undergoes significant and fast deformation during the motion of the slider-crank mechanism, confirming the high stiffness of the DAE system. In particular, the nonlinear beam model shows a more pronounced and fast deformation compared to the linear beam model, which makes the numerical solution more challenging. Indeed, the numerical solution of the slider-crank mechanism with the nonlinear beam model exhibits an overflow error after $t = 0.05\,\mathrm{s}$, which is likely due to either the high stiffness of the DAE system or strongly undamped oscillations. The linear beam model, on the other hand, does not exhibit such issues and is successfully solved. Nonetheless, in Figure 5.5 the LU pivots having minimum values are depicted. Neither of them is crossing nor getting close to zero, which is a good indicator of the numerical stability of the LU factorization and thereby of the efficacy of the symbolic pivoting strategy.

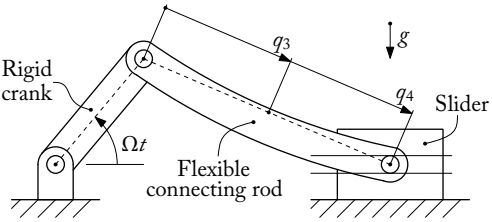


Figure 5.3: Representation of the flexible slider-crank mechanism [132, 134].

5.2.3 THE MULTI-PENDULA SYSTEM

The multi-pendula system consists of a chain of p coupled pendula. The first pendulum is a standard simple pendulum, while the remaining pendula are coupled to the previous one. Specifically, the tension in pendulum $i-1$, which is represented by the Lagrange multiplier λ_{i-1}, has a small effect on the length of pendulum p, for $i = 2, \ldots, p$. The

Table 5.2: Expression complexity encountered throughout the index reduction of the flexible slider-crank mechanism problem [132, 134] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Flexible Slider-Crank Mechanism [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 281\,\mathrm{f} + 10\,\mathrm{d} + 607\,\mathrm{m} + 144\,\mathrm{a} \quad \mathbf{h}(\mathbf{x}, t) = 0$		
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	$64\,\mathrm{f} + 9\,\mathrm{d} + 1158\,\mathrm{m} + 167\,\mathrm{a}$	$395\,\mathrm{f} + 15\,\mathrm{d} + 5552\,\mathrm{m} + 871\,\mathrm{a}$	$12\,\mathrm{f} + 5\,\mathrm{m} + 6\,\mathrm{a}$
Index-2 DAEs	$64\,\mathrm{f} + 9\,\mathrm{d} + 1158\,\mathrm{m} + 167\,\mathrm{a}$	$395\,\mathrm{f} + 15\,\mathrm{d} + 5552\,\mathrm{m} + 871\,\mathrm{a}$	$22\,\mathrm{f} + 10\,\mathrm{m} + 8\,\mathrm{a}$
Index-1 DAEs	$20\,\mathrm{f} + 5\,\mathrm{d} + 110\,\mathrm{m} + 19\,\mathrm{a}$	$277\,\mathrm{f} + 13\,\mathrm{d} + 1907\,\mathrm{m} + 326\,\mathrm{a}$	$\star(0.4\,\mathrm{f} + 2.4\,\mathrm{m} + 0.1\,\mathrm{a})\cdot 10^6 + 15\,\mathrm{d}$
Index-0 DAEs	$\star(0.8\,\mathrm{f} + 4.8\,\mathrm{m} + 0.3\,\mathrm{a})\cdot 10^5 + 77\,\mathrm{d}$	$\star(5.8\,\mathrm{f} + 35.7\,\mathrm{m} + 2.2\,\mathrm{a})\cdot 10^6 + 1259\,\mathrm{d}$	0
Reduced DAEs	$\star\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = (5.8\,\mathrm{f} + 36.2\,\mathrm{m} + 2.2\,\mathrm{a})\cdot 10^6 + 1336\,\mathrm{d} \quad \star\mathbf{h}(\mathbf{x}, t) = (0.5\,\mathrm{f} + 2.4\,\mathrm{m} + 0.1\,\mathrm{a})\cdot 10^6 + 15\,\mathrm{d}$		

Table 5.3: Expression complexity encountered throughout the index reduction with the aid of hierarchical representation of the flexible slider-crank mechanism problem [132, 134] DAE system. *Legend*: f = functions, v = veiling variables, a = additions, m = multiplications, and d = divisions.

Flexible Slider-Crank Mechanism [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = 281\,\mathrm{f} + 10\,\mathrm{d} + 607\,\mathrm{m} + 144\,\mathrm{a} \quad \mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 0$		
Reduction step	$\mathbf{E}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{g}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$
Index-3 DAEs	$28\,\mathrm{f} + 1\,\mathrm{v} + 7\,\mathrm{d} + 160\,\mathrm{m} + 28\,\mathrm{a}$	$91\,\mathrm{f} + 2\,\mathrm{v} + 8\,\mathrm{d} + 291\,\mathrm{m} + 44\,\mathrm{a}$	$12\,\mathrm{f} + 5\,\mathrm{m} + 6\,\mathrm{a}$
Index-2 DAEs	$18\,\mathrm{f} + 1\,\mathrm{v} + 6\,\mathrm{d} + 88\,\mathrm{m} + 15\,\mathrm{a}$	$101\,\mathrm{f} + 5\,\mathrm{v} + 10\,\mathrm{d} + 360\,\mathrm{m} + 58\,\mathrm{a}$	$22\,\mathrm{f} + 10\,\mathrm{m} + 8\,\mathrm{a}$
Index-1 DAEs	$10\,\mathrm{f} + 1\,\mathrm{v} + 4\,\mathrm{d} + 39\,\mathrm{m} + 6\,\mathrm{a}$	$93\,\mathrm{f} + 4\,\mathrm{v} + 9\,\mathrm{d} + 294\,\mathrm{m} + 48\,\mathrm{a}$	$10\,\mathrm{f} + 2\,\mathrm{v} + 9\,\mathrm{d} + 82\,\mathrm{m} + 14\,\mathrm{a}$
Index-0 DAEs	$(0.2\,\mathrm{f} + 1.1\,\mathrm{m} + 0.1\,\mathrm{a}) \cdot 10^6 + 1171\,\mathrm{v} + 1158\,\mathrm{d}$	$(0.7\,\mathrm{f} + 3.6\,\mathrm{m} + 0.4\,\mathrm{a}) \cdot 10^5 + 137\,\mathrm{v} + 1172\,\mathrm{d}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = (0.3\,\mathrm{f} + 1.5\,\mathrm{m} + 0.1\,\mathrm{a}) \cdot 10^6 + 1308\,\mathrm{v} + 2330\,\mathrm{d} \quad \mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 44\,\mathrm{f} + 6\,\mathrm{v} + 2\,\mathrm{d} + 97\,\mathrm{m} + 28\,\mathrm{a}$		

Hierarchical representation details (9 veils)

Original DAEs	$\mathbf{v}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{v}(\mathbf{x}, t)$
Index-3 DAEs	$309\,\mathrm{f} + 1\,\mathrm{v} + 7\,\mathrm{d} + 2946\,\mathrm{m} + 485\,\mathrm{a}$
Index-2 DAEs	$309\,\mathrm{f} + 1\,\mathrm{v} + 7\,\mathrm{d} + 2946\,\mathrm{m} + 485\,\mathrm{a}$
Index-1 DAEs	$422\,\mathrm{f} + 7\,\mathrm{v} + 14\,\mathrm{d} + 3249\,\mathrm{m} + 541\,\mathrm{a}$
Index-0 DAEs	$492919\,\mathrm{f} + 63922\,\mathrm{v} + 138\,\mathrm{d} + 2441245\,\mathrm{m} + 155539\,\mathrm{a}$
Reduced DAEs	$\mathbf{v}(\mathbf{x}, t) = 492919\,\mathrm{f} + 63922\,\mathrm{v} + 138\,\mathrm{d} + 2441245\,\mathrm{m} + 155539\,\mathrm{a}$

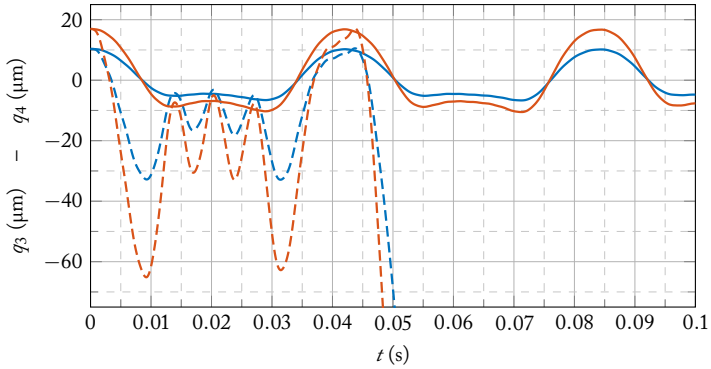


Figure 5.4: Longitudinal displacement endpoint of the in the flexible slider-crank problem [132, 134]. *Color legend*: ■ longitudinal displacement q_3 at the rod midpoint, and ■ longitudinal displacement q_4 at the rod endpoint. *Line legend*: — linear beam model, and – – nonlinear beam model.

first-order equations of motion for the multi-pendula system are the following [77]

$$\begin{cases} u_1' = x_1 \\ v_1' = y_1 \\ x_1' = -\lambda_1 x_1 \\ y_1' = -\lambda_1 y_1 - g \\ 0 = x_1^2 + y_1^2 - \ell^2 \end{cases} \quad \text{and} \quad \begin{cases} u_i' = x_i \\ v_i' = y_i \\ x_i' = -\lambda_p x_i \\ y_i' = -\lambda_p y_i - g \\ 0 = x_i^2 + y_i^2 - (\ell + c\lambda_{i-1})^2 \end{cases} \quad \text{for} \quad i = 2, \ldots, p\,. \tag{5.8}$$

where x_i, y_i and u_i, v_i are the generalized coordinates and velocities of i-th pendulum mass. The parameters of the multi-pendula system are chosen as $\ell = 1\,\mathrm{m}$, $g = 9.81\,\mathrm{m/s^2}$, and $c = 0.1$. The system (5.8) leads to a class of DAEs of size $5p$ and index $2p+1$ [77], with arbitrary $p \in \mathbb{Z}_{\geq 2}$. ICs are chosen randomly, and the time interval is $t \in [0, 60]\,\mathrm{s}$.

Here, we consider p up to 5, leading to a DAE system of 25 variables and index 11. The computational complexities encountered during the index reduction process are summarized in Table 5.4, for the 2-pendula problem, Table 5.5, for the 3-pendula problem, Table 5.6, for the 4-pendula problem, and lastly, Table 5.7, for the 5-pendula problem. The multi-pendula systems with $p \leq 3$ exhibit low computational complexity growth during the index reduction process. However, for $p \geq 4$, there is a substantial expression swell, which makes the index reduction process time-consuming and computationally demanding, yet still feasible. The introduction of veiling variables is avoided on purpose to assess the impact of strong expression swelling on the stability of the numerical solution. It is worth noting that with $p = 4$ the maximum computational complexity that MAPLE® can handle within a reasonable time is reached. For

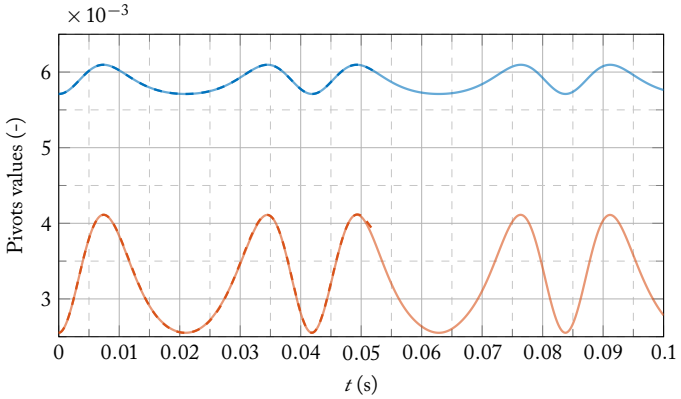


Figure 5.5: Representation of the two pivots of the flexible slider-crank mechanism [132, 134] having minimum values. *Color legend*: ■ pivot 1, and ■ pivot 2. *Line legend*: — linear beam model, and – – nonlinear beam model.

this reason, performing the last reduction steps for $p > 4$ requires a significant amount of time and computational resources. Furthermore, we believe that reducing the index of the DAE system for $p > 5$ is not feasible within an acceptable time frame, even with the aid of hierarchical representation. The numerical integration results of the 5-pendula system are depicted in Figure 5.6, where the pendula lengths are illustrated in the time interval $t \in [0, 60]$ s. Notice that the numerical solution of the 5-pendula system is computed from its index-2 DAE form, as its index-0 form demands a significant amount of computational resources and time to be numerically solved.

5.2.4 DOUBLE-WISHBONE SUSPENSION SYSTEM

As a final example for this application field, we consider a double-wishbone suspension system, which has been extensively discussed in [6, 8].

5.2.4.1 SUSPENSION SYSTEM MODELING

Double A-arm or double-wishbone suspensions are independent suspension systems widely used in the automotive industry, especially in high-performance vehicles, due to their superior handling characteristics. The studied double A-arm suspension system, shown in Figure 5.7 is characterized by two A-shaped arms, one upper and one lower, connected to the chassis at one end and to the wheel carrier at the other end forming, together with the tie rod, the principal kinematic chain of the suspension system. A second kinematic chain is formed by the push rod and the rocker, which transfer the vertical load from the wheel carrier to the shock absorber.

Table 5.4: Expression complexity encountered throughout the index reduction of the 2-pendula problem [77] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

2-Pendula [60]			
Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 33\,\mathrm{f} + 15\,\mathrm{m} + 15\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$	
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-5 DAEs	0	$12\,\mathrm{f} + 8\,\mathrm{m} + 2\,\mathrm{a}$	$6\,\mathrm{f} + 12\,\mathrm{m} + 6\,\mathrm{a}$
Index-4 DAEs	$1\,\mathrm{f} + 5\,\mathrm{m} + 1\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	$4\,\mathrm{f} + 4\,\mathrm{m} + 1\,\mathrm{a}$
Index-3 DAEs	$1\,\mathrm{f} + 5\,\mathrm{m} + 1\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	$7\,\mathrm{f} + 12\,\mathrm{m} + 4\,\mathrm{a}$
Index-2 DAEs	$1\,\mathrm{f} + 5\,\mathrm{m} + 1\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	$18\,\mathrm{f} + 2\,\mathrm{d} + 30\,\mathrm{m} + 9\,\mathrm{a}$
Index-1 DAEs	$1\,\mathrm{f} + 5\,\mathrm{m} + 1\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	$118\,\mathrm{f} + 2\,\mathrm{d} + 283\,\mathrm{m} + 72\,\mathrm{a}$
Index-0 DAEs	$555\,\mathrm{f} + 21\,\mathrm{d} + 1213\,\mathrm{m} + 287\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 482\,\mathrm{f} + 2\,\mathrm{d} + 807\,\mathrm{m} + 229\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 153\,\mathrm{f} + 4\,\mathrm{d} + 341\,\mathrm{m} + 92\,\mathrm{a}$	

Table 5.5: Expression complexity encountered throughout the index reduction of the 3-pendula problem [77] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

3-Pendula [77]			
Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 50\,\mathrm{f} + 23\,\mathrm{m} + 23\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 0$	
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-7 DAEs	0	$18\,\mathrm{f} + 12\,\mathrm{m} + 3\,\mathrm{a}$	$10\,\mathrm{f} + 21\,\mathrm{m} + 10\,\mathrm{a}$
Index-6 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$4\,\mathrm{f} + 4\,\mathrm{m} + 1\,\mathrm{a}$
Index-5 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$7\,\mathrm{f} + 12\,\mathrm{m} + 4\,\mathrm{a}$
Index-4 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$18\,\mathrm{f} + 2\,\mathrm{d} + 30\,\mathrm{m} + 9\,\mathrm{a}$
Index-3 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$118\,\mathrm{f} + 2\,\mathrm{d} + 283\,\mathrm{m} + 72\,\mathrm{a}$
Index-2 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$992\,\mathrm{f} + 3\,\mathrm{d} + 2077\,\mathrm{m} + 479\,\mathrm{a}$
Index-1 DAEs	$2\,\mathrm{f} + 10\,\mathrm{m} + 2\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$6824\,\mathrm{f} + 3\,\mathrm{d} + 17665\,\mathrm{m} + 4030\,\mathrm{a}$
Index-0 DAEs	$54152\,\mathrm{f} + 51\,\mathrm{d} + 136388\,\mathrm{m} + 28945\,\mathrm{a}$	$26\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 28319\,\mathrm{f} + 3\,\mathrm{d} + 64295\,\mathrm{m} + 15806\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, t) = 7973\,\mathrm{f} + 10\,\mathrm{d} + 20092\,\mathrm{m} + 4605\,\mathrm{a}$	

Table 5.6: Expression complexity encountered throughout the index reduction of the 4-pendula problem [77] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

4-Pendula [77]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 67\,\mathrm{f} + 31\,\mathrm{m} + 31\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-9 DAEs	0	$24\,\mathrm{f} + 16\,\mathrm{m} + 4\,\mathrm{a}$	$14\,\mathrm{f} + 30\,\mathrm{m} + 14\,\mathrm{a}$
Index-8 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$4\,\mathrm{f} + 4\,\mathrm{m} + 1\,\mathrm{a}$
Index-7 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$7\,\mathrm{f} + 12\,\mathrm{m} + 4\,\mathrm{a}$
Index-6 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$18\,\mathrm{f} + 2\,\mathrm{d} + 30\,\mathrm{m} + 9\,\mathrm{a}$
Index-5 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$118\,\mathrm{f} + 2\,\mathrm{d} + 283\,\mathrm{m} + 72\,\mathrm{a}$
Index-4 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$992\,\mathrm{f} + 3\,\mathrm{d} + 2077\,\mathrm{m} + 479\,\mathrm{a}$
Index-3 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$6824\,\mathrm{f} + 3\,\mathrm{d} + 17665\,\mathrm{m} + 4030\,\mathrm{a}$
Index-2 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$(4.8\,\mathrm{f} + 4\,\mathrm{d} + 11.9\,\mathrm{m} + 2.7\,\mathrm{a}) \cdot 10^5$
Index-1 DAEs	$3\,\mathrm{f} + 15\,\mathrm{m} + 3\,\mathrm{a}$	$36\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	$\star(3.0\,\mathrm{f} + 14.9\,\mathrm{m} + 0.4\,\mathrm{a}) \cdot 10^6 + 4\,\mathrm{d}$
Index-0 DAEs	$\star(3.0\,\mathrm{f} + 14.7\,\mathrm{m} + 0.4\,\mathrm{a}) \cdot 10^7 + 92\,\mathrm{d}$	$30\,\mathrm{f} + 28\,\mathrm{m} + 7\,\mathrm{a}$	0
Reduced DAEs	$\star\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = (3.0\,\mathrm{f} + 14.7\,\mathrm{m} + 0.4\,\mathrm{a}) \cdot 10^7 + 92\,\mathrm{d}$		$\star\mathbf{h}(\mathbf{x}, t) = (3.1\,\mathrm{f} + 15.1\,\mathrm{m} + 0.5\,\mathrm{a}) \cdot 10^6 + 18\,\mathrm{d}$

Table 5.7: Expression complexity encountered throughout the index reduction of the 5-pendula problem [77] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

5-Pendula [77]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 84\,\mathrm{f} + 39\,\mathrm{m} + 39\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-11 DAEs	0	$30\,\mathrm{f} + 20\,\mathrm{m} + 5\,\mathrm{a}$	$18\,\mathrm{f} + 39\,\mathrm{m} + 18\,\mathrm{a}$
Index-10 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$4\,\mathrm{f} + 4\,\mathrm{m} + 1\,\mathrm{a}$
Index-9 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$7\,\mathrm{f} + 12\,\mathrm{m} + 4\,\mathrm{a}$
Index-8 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$18\,\mathrm{f} + 2\,\mathrm{d} + 30\,\mathrm{m} + 9\,\mathrm{a}$
Index-7 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$118\,\mathrm{f} + 2\,\mathrm{d} + 283\,\mathrm{m} + 72\,\mathrm{a}$
Index-6 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$992\,\mathrm{f} + 3\,\mathrm{d} + 2077\,\mathrm{m} + 479\,\mathrm{a}$
Index-5 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$6824\,\mathrm{f} + 3\,\mathrm{d} + 17665\,\mathrm{m} + 4030\,\mathrm{a}$
Index-4 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$(4.8\,\mathrm{f} + 4\,\mathrm{d} + 11.9\,\mathrm{m} + 2.7\,\mathrm{a}) \cdot 10^5$
Index-3 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$\star(3.0\,\mathrm{f} + 14.9\,\mathrm{m} + 0.4\,\mathrm{a}) \cdot 10^6 + 4\,\mathrm{d}$
Index-2 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$\star(5.1\,\mathrm{f} + 4.0\,\mathrm{m} + 1.3\,\mathrm{a}) \cdot 10^7 + 5\,\mathrm{d}$
Index-1 DAEs	$4\,\mathrm{f} + 20\,\mathrm{m} + 4\,\mathrm{a}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	$\star(9.1\,\mathrm{f} + 11.3\,\mathrm{m} + 5.9\,\mathrm{a}) \cdot 10^7 + 5\,\mathrm{d}$
Index-0 DAEs	$\star(8.8\,\mathrm{f} + 7.2\,\mathrm{m} + 1.0\,\mathrm{a}) \cdot 10^8 + 5\,\mathrm{d}$	$46\,\mathrm{f} + 36\,\mathrm{m} + 9\,\mathrm{a}$	0
Reduced DAEs	$\star\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = \star(8.8\,\mathrm{f} + 7.2\,\mathrm{m} + 1.0\,\mathrm{a}) \cdot 10^8 + 5\,\mathrm{d}$		$\star\mathbf{h}(\mathbf{x}, t) = (9.2\,\mathrm{f} + 6.5\,\mathrm{m} + 1.0\,\mathrm{a}) \cdot 10^7 + 18\,\mathrm{d}$

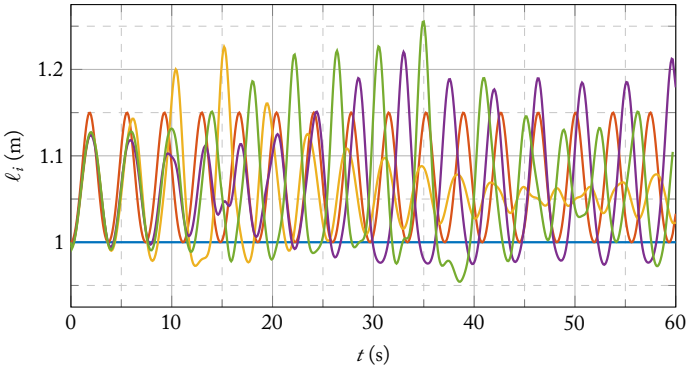


Figure 5.6: Pendula lengths of the multi-pendula problem [77] (up to 5 pendula). *Legend*: 1st pendulum length $\ell_1 = \ell$, 2nd pendulum length $\ell_2 = \ell + c\lambda_1$, 3rd pendulum length $\ell_3 = \ell + c\lambda_2$, 4th pendulum length $\ell_4 = \ell + c\lambda_3$, and 5th pendulum length $\ell_5 = \ell + c\lambda_4$.

In this example, the double A-arm suspension compliance is modeled using macro elements of the TrussMe-Fem package [28], which is a Maple® package for the symbolic modeling of compliant structures. Details on the modeling of the compliant elements are reported in Appendix D. With the approach there presented, two suspension compliance models are generated one does not include the bushings compliance, while the other does. The linear systems have different sizes depending on the number of DOFs considered. In the case of the suspension without bushings, the system is composed of 78 DOFs, and has the following form

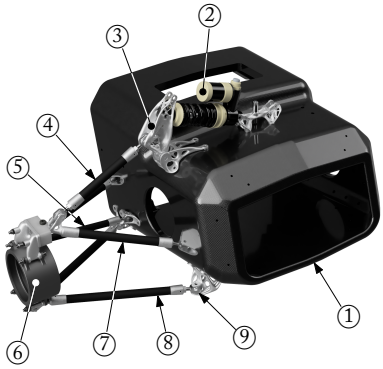
$$\begin{bmatrix} \mathbf{K}_{ff,\,42\times42}(\mathbb{R}) & \mathbf{K}_{fs,\,42\times36}(\mathbb{R}) \\ \mathbf{K}_{sf,\,36\times42}(\mathbb{R}) & \mathbf{K}_{ss,\,36\times36}(\mathbb{R}) \end{bmatrix} \begin{bmatrix} \mathbf{d}_{f,\,42\times1}(\mathbb{R}) \\ \mathbf{d}_{s,\,36\times1}(\mathbb{R}) \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{f,\,42\times1}(\mathbb{R}) \\ \mathbf{f}_{s,\,36\times1}(\mathbb{R}) \end{bmatrix}.$$

On the other hand, the suspension model with the bushings influence is composed of 138 DOFs

$$\begin{bmatrix} \mathbf{K}_{ff,\,72\times72}(\mathbb{R}) & \mathbf{K}_{fs,\,72\times66}(\mathbb{R}) \\ \mathbf{K}_{sf,\,66\times72}(\mathbb{R}) & \mathbf{K}_{ss,\,66\times66}(\mathbb{R}) \end{bmatrix} \begin{bmatrix} \mathbf{d}_{f,\,72\times1}(\mathbb{R}) \\ \mathbf{d}_{s,\,66\times1}(\mathbb{R}) \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{f,\,72\times1}(\mathbb{R}) \\ \mathbf{f}_{s,\,66\times1}(\mathbb{R}) \end{bmatrix},$$

where the additional 60 DOFs are related to the compliance of the bushings. Both systems are solved using the technique presented in Appendix D, which given the generic compliant mechanism described by the linear system of equations

$$\underbrace{\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fs} \\ \mathbf{K}_{sf} & \mathbf{K}_{ss} \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} \mathbf{d}_f \\ \mathbf{d}_s \end{bmatrix}}_{\mathbf{d}} = \underbrace{\begin{bmatrix} \mathbf{f}_f \\ \mathbf{f}_s \end{bmatrix}}_{\mathbf{f}}, \quad \text{is sequentially solved as} \quad \begin{aligned} \mathbf{d}_f &= \mathbf{K}_{ff}^{-1}\left(\mathbf{f}_f - \mathbf{K}_{fs}\mathbf{d}_s\right), \\ \mathbf{f}_s &= \mathbf{K}_{sf}\mathbf{d}_f + \mathbf{K}_{ss}\mathbf{d}_s, \end{aligned} \tag{5.9}$$



#	Component	TrussMe-Fem type
①	Carbon fiber cell	Rigid support
②	Shock absorber	Constrained node
③	Rocker	Generic element
④	Push rod	Rod element
⑤	Tie rod	Rod element
⑥	Wheel carrier	Generic element
⑦	Upper wishbone	Beam elements
⑧	Lower wishbone	Beam elements
⑨	Rod end	Compliant node

Figure 5.7: Rendering and description of the TrussMe-Fem elements [28] used to model the rear left double-wishbone suspension of the Formula SAE *E-Agle Trento Racing Team* vehicle [137].

where subscripts s and f indicate respectively the specified and the free DOFs. Notice that the free and the specified DOFs are those that are and are not constrained by the BCs, respectively.
The dynamic characteristic of the system is modeled through a DAE system of the following type:

$$\begin{cases} \mathbf{y} = \mathbf{q} + \mathbf{d} & \text{(5.10a)} \\ \mathbf{M}(\mathbf{y})\mathbf{y}'' + \mathbf{r}(\mathbf{q}, \mathbf{q}', \mathbf{d}, \mathbf{d}') = \mathbf{f}(t) & \text{(5.10b)} \\ \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})^{\top} \boldsymbol{\lambda} = \mathbf{r}(\mathbf{q}, \mathbf{q}', \mathbf{d}, \mathbf{d}') + \mathbf{b}(\mathbf{q}, \mathbf{q}', t) & \text{(5.10c)} \\ \boldsymbol{\Phi}(\mathbf{q}) = \mathbf{0} & \text{(5.10d)} \end{cases}$$

$$\text{with} \quad \mathbf{r}(\mathbf{q}, \mathbf{q}', \mathbf{d}, \mathbf{d}') = \mathbf{K}_c(\mathbf{q})\mathbf{d} + \mathbf{C}_c(\mathbf{q})\mathbf{d}', \qquad \text{(5.10e)}$$

where $\mathbf{q}$ is the state vector, and t is the time. The quantity $\mathbf{d}$ represents the compliance contribution of the mechanism members' deformation. States and deformations are conveniently condensed in a single variable $\mathbf{y}$ as in (5.10a). Equation (5.10b) represents the compliance contribution to the dynamics of the system, where $\mathbf{K}_c$, and $\mathbf{C}_c$ are the stiffness and damping matrices of the compliant bodies, respectively. The matrix $\mathbf{M}$ represents the masses of the mechanism, while $\mathbf{f}$ is the vector of external forces. For convenience, we collect $\mathbf{r}$ in Equation (5.10e) as the vector of internal forces at the compliant joint. Notice that the product $\mathbf{K}_c(\mathbf{q})\mathbf{d}$ can be computed through the symbolic solution of the linear systems (5.9). Equation (5.10c) represents the equilibrium equations between the rigid and compliant parts of the suspension. Specifically, $\boldsymbol{\Phi}_{\mathbf{q}}$ is the Jacobian matrix of the constraint vector $\boldsymbol{\Phi}$ with respect to the coordinates $\mathbf{q}$, while $\mathbf{b}$ is

the vector of external forces applied to the rigid part of the suspension. Lastly, Equation (5.10d) represents the kinematic constraints of the system. The pick-up points of the modeled suspension are reported in Table 5.8. It is possible to derive the lengths of the various elements from these coordinates and to impose constraints to ensure a proper assembling of the mechanism. Components and materials specifications are specified in Tables 5.9 and 5.10, respectively.
It is important to note that one may consider the compliance contribution as a superposed effect. To this end, we first assume that the influence of the members' deformation $\mathbf{d}$ is small with respect to the dimensions of the mechanism itself, and thus to the state vector $\mathbf{q}$, *i.e.*, $\mathbf{d} \ll \mathbf{q}$. From (5.10a) it follows that $\mathbf{M}(\mathbf{y}) \approx \mathbf{M}(\mathbf{q})$. It is then possible to split the resolution of the DAEs (5.10) into two stages. Firstly, the implicit differential equation (5.10c) and the manifold (5.10d) are integrated. Nonetheless, these equations are of an index-3 MB DAEs system of the type (5.1), which can be reduced to an index-0 or index-1 DAEs system and solved as above explained. Then, the second stage consists of adding the stationary contribution of the members' deformation $\mathbf{d}$ to the state vector $\mathbf{q}$, that is $\mathbf{y} = \mathbf{q} + \mathbf{d} = \mathbf{q} + \mathbf{K}_c(\mathbf{q})^{-1}(\mathbf{f}(t) - \mathbf{M}(\mathbf{q})\mathbf{q}'')$.

Table 5.8: Suspension pick-up points' coordinates in the nominal position.

Pick-up point name	**Constrained elements**		**Coordinates**		
			x (mm)	y (mm)	z (mm)
P_1	Chassis	Upper-front rod	−719	270	240
P_2	Chassis	Upper-rear rod	−1010	265	225
P_3	Chassis	Lower-front rod	−730	265	120
P_4	Chassis	Lower-rear rod	−1010	235	98
P_5	Chassis	Tie rod	−775	265	163
P_6	Chassis	Rocker	−895	243	375
P_7	Chassis	Shock absorber	−895	50	412
P_8	Wheel carrier	Upper-front rod	−895	517	293
P_9	Wheel carrier	Upper-rear rod	−895	517	293
P_{10}	Wheel carrier	Lower-front rod	−885	550	120
P_{11}	Wheel carrier	Lower-rear rod	−885	550	120
P_{12}	Wheel carrier	Tie rod	−790	532	198
P_{13}	Rocker	Shock absorber	−895	236	466
P_{14}	Rocker	Push rod	−895	280	418
P_{15}	Push rod	Wheel carrier	−895	479	312

Table 5.9: Suspension shock absorber and wheel specifications used in ANSYS® and TRUSSME-FEM simulations.

Component	Property	Quantity
Shock absorber	Stiffness	255 kN/m
	Damping	500 N s/m
	Travel	0.06 m
Wheel body	Mass	12.5 kg
	Inertia diam.	0.21 kg m^2
	Inertia axial	0.42 kg m^2

Table 5.10: Properties of suspension materials used in ANSYS® and TRUSSME-FEM simulations, where E is the Young modulus, ν is the Poisson ratio, and ρ is the density.

Material	Properties		
	E (GPa)	ν (–)	ρ (kg m^3)
AISI 1045	210.0	0.30	7800
AISI 316L	196.0	0.25	7990
Ergal 7075-T6	71.7	0.33	2810
Carbon fiber	150.0	0.34	1500
Epoxy glue	1.718	0.33	1440

5.2.4.2 SYSTEM SOLUTION

Prior to any analyses, the DAE system is handled to INDIGO for index reduction. For this purpose, the system is reduced to a system of ODEs, and the computational complexities encountered during the reduction process are reported in Table 5.11. Notice the substantial increase in the expression complexity of the reduced system in the last two reduction steps despite the simplification capabilities of MAPLE®, which indicates a high level of inherent expression swell.

After the reduction process, the reduced system is numerically solved generating appropriate code for the SIMULINK® environment. The system is also reproduced in ANSYS® for comparison and validation. The first test that is carried out aims to verify the accuracy of the reduced system and to predict the impact of compliance on the system dynamics. The results of the frequency response analysis are shown in Figure 5.9. The results show that the SIMULINK® multi-body simulation with full-compliance dynamics contribution and the ANSYS® Finite Element (FE) modal analysis are in good agreement and the first three modal shapes, which are shown in Figure 5.8, are well captured by the reduced system with full-compliance dynamics contribution. Conversely, the reduced system with steady-state compliance dynamics contribution can not capture the first three modal shapes of the suspension. This result, despite expected, also introduces a zero along the suspension moving direction in the system's frequency response, which is not present in the full-compliance dynamics contribution. The following considerations can be made from these results: the compliance of the suspension system has an impact on the system dynamics for high-frequency analyses, and the reduced system with full-compliance dynamics contribution can capture the system dynamics accurately. For computationally efficient simulations, the reduced system with steady-state compliance dynamics can be used to capture the system's behavior assuming the suspension system is driven by low-frequency inputs.

The reduced system with full-compliance dynamics is then used to perform a transient

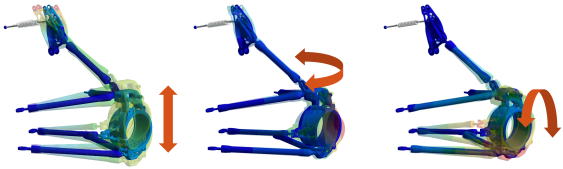
Table 5.11: Expression complexity encountered throughout the index reduction with the aid of hierarchical representation of the double-wishbone suspension system DAE system. *Legend*: f = functions, v = veiling variables, a = additions, m = multiplications, and d = divisions.

Double-Wishbone Suspension [6, 8]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = 845\,\mathrm{f} + 610\,\mathrm{m} + 465\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 0$	
Reduction step	$\mathbf{E}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{g}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$
Index-3 DAEs	$16\,\mathrm{f} + 15\,\mathrm{m} + 4\,\mathrm{a}$	$217\,\mathrm{f} + 3\,\mathrm{v} + 165\,\mathrm{m} + 104\,\mathrm{a}$	$252\,\mathrm{f} + 154\,\mathrm{m} + 128\,\mathrm{a}$
Index-2 DAEs	$16\,\mathrm{f} + 15\,\mathrm{m} + 4\,\mathrm{a}$	$217\,\mathrm{f} + 3\,\mathrm{v} + 165\,\mathrm{m} + 104\,\mathrm{a}$	$161\,\mathrm{f} + 3\,\mathrm{v} + 105\,\mathrm{m} + 49\,\mathrm{a}$
Index-1 DAEs	$14\,\mathrm{f} + 14\,\mathrm{m} + 4\,\mathrm{a}$	$223\,\mathrm{f} + 4\,\mathrm{v} + 1\,\mathrm{d} + 170\,\mathrm{m} + 104\,\mathrm{a}$	$7\,\mathrm{v} + 6\,\mathrm{a}$
Index-0 DAEs	$104\,\mathrm{f} + 155\,\mathrm{v} + 15\,\mathrm{d} + 160\,\mathrm{m} + 81\,\mathrm{a}$	$223\,\mathrm{f} + 11\,\mathrm{v} + 1\,\mathrm{d} + 170\,\mathrm{m} + 105\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = 551\,\mathrm{f} + 144\,\mathrm{v} + 7\,\mathrm{d} + 542\,\mathrm{m} + 269\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 413\,\mathrm{f} + 10\,\mathrm{v} + 259\,\mathrm{m} + 183\,\mathrm{a}$	

Hierarchical representation details (151 veils)

Original DAEs	$\mathbf{v}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{v}(\mathbf{x}, t)$
Index-3 DAEs	$320\,\mathrm{f} + 1\,\mathrm{v} + 264\,\mathrm{m} + 212\,\mathrm{a}$
Index-2 DAEs	$716\,\mathrm{f} + 1\,\mathrm{v} + 582\,\mathrm{m} + 440\,\mathrm{a}$
Index-1 DAEs	$6391\,\mathrm{f} + 28\,\mathrm{v} + 10\,\mathrm{d} + 5385\,\mathrm{m} + 3301\,\mathrm{a}$
Index-0 DAEs	$119901\,\mathrm{f} + 3765\,\mathrm{v} + 192\,\mathrm{d} + 108637\,\mathrm{m} + 64865\,\mathrm{a}$
Reduced DAEs	$\mathbf{v}(\mathbf{x}, t) = 119901\,\mathrm{f} + 3765\,\mathrm{v} + 192\,\mathrm{d} + 108637\,\mathrm{m} + 64865\,\mathrm{a}$



(a) $f_1 = 7.6\,\text{Hz}$ (b) $f_2 = 88.5\,\text{Hz}$ (c) $f_3 = 159.7\,\text{Hz}$

Figure 5.8: First three modal shapes of the suspension.

analysis of the suspension system, coupled with the tire-ground enveloping model and the tire model presented in Appendix B (supported by the Acme C++ library described in Appendix A) and Appendix C, respectively. The transient analysis is conducted to understand the impact of compliance on the vehicle dynamics as well as suspension rods' diameter in the development of tire-ground forces. A side slip angle ramp simulation is thus performed to evaluate the effect of suspension compliance on the lateral force of the tire. For this simulation, the following four cases are analyzed:

- pure kinematic suspension model;
- kinematic and nominal compliance model;
- kinematic and compliance model with -10% suspension rods diameter;
- kinematic and compliance model with -20% suspension rods diameter.

The results of the simulation are shown in Figure 5.10, which compares the four cases. The lateral force F_y is depicted with solid lines, while the difference between the lateral force of the pure kinematic simulation and the other cases is illustrated in dashed lines. The displayed curves show a non-negligible effect of suspension compliance on the lateral force of the tire is observed. As expected, the larger difference in the generated F_y force is observed in the linear region of the tire slip characteristic curve, where the slip cornering stiffness is higher. The difference then decreases as the peak approaches. Further observations could be made on the impact of these findings on the vehicle dynamics and handling. However, this is beyond the scope of this work.
Finally, to estimate what would be the impact of compliance in a real race, the telemetry and inputs of a full-vehicle simulation conducted on the *Pista Azzurra* track in Jesolo (Italy), are applied to the modeled rear left double-wishbone suspension. The simulation is conducted using the previously reduced MB-DAE system, the tire-ground enveloping model, and the tire model presented in Appendices B and C, respectively. The results of the simulation are shown in Figure 5.11, where the suspension travel z is shown in the first plot, while the translations δ and rotation θ components of the compliance contribution at the wheel hub are shown in the second and third plots, respectively. On the other hand, Figure 5.12 shows the wheel hub trajectories during the drive simulation. The pure kinematical trajectory of the wheel hub is shown in blue,

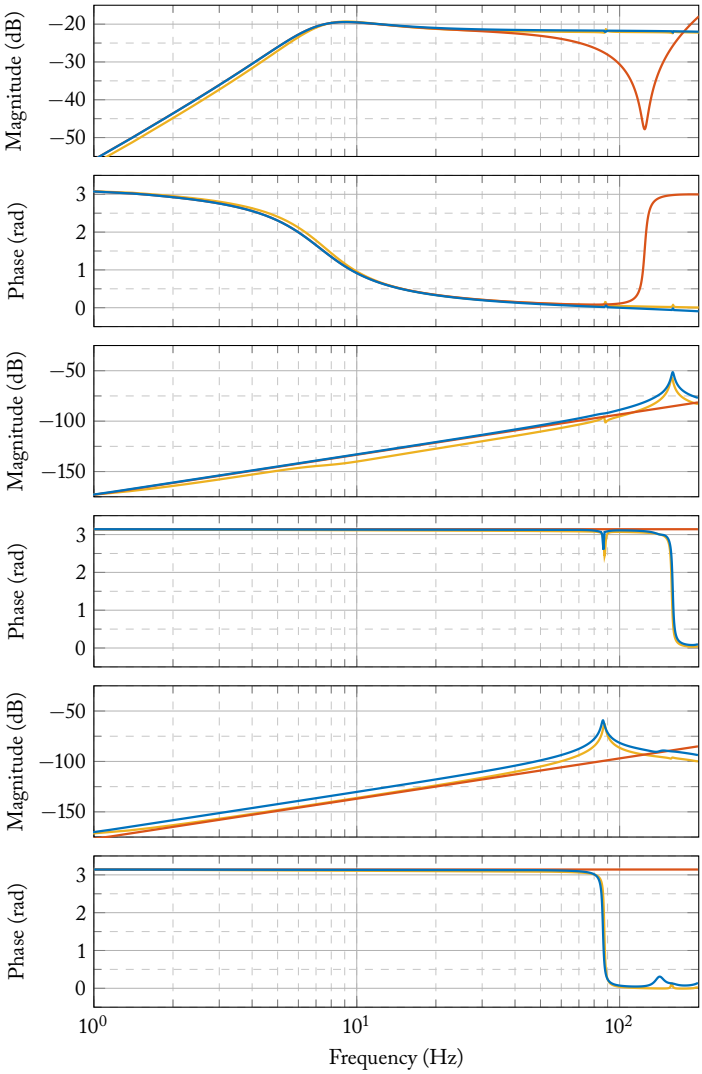


Figure 5.9: Frequency response analysis of the suspension model is conducted, with tests performed under the equilibrium between the suspension system and a vertical force of 500 N applied at the wheel hub. Frequency responses are assessed by applying input forces/torques at the wheel hub in the form of a linear chirp spanning frequencies from 0 Hz – 200 Hz, with a constant amplitude of 5 N/5 N m. *Legend:* ■ Simulink® multi-body simulation with full-compliance dynamics contribution, ■ Simulink® multi-body with steady-state compliance contribution, ■ Ansys® FE modal analysis.

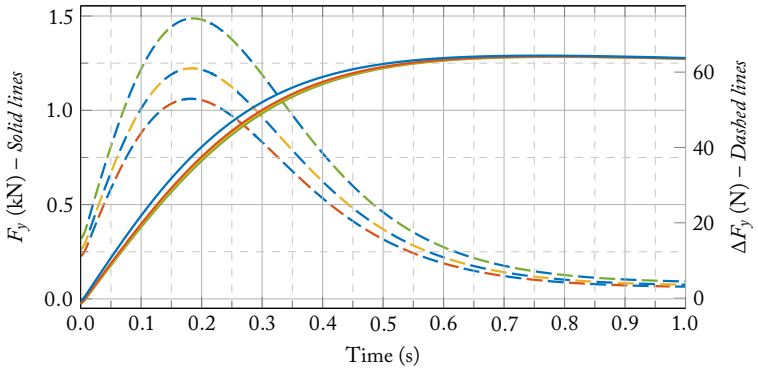


Figure 5.10: Tire lateral force F_y during a side slip angle ramp simulation. Solid lines represent the results obtained with the SIMULINK® model, while dashed lines represent the difference between the various simulations (see legend below). *Solid lines legend:* ■ kinematics, ■ kinematics and compliance, ■ kinematics and compliance (−10% rods diameter), ■ kinematics and compliance (−20% rods diameter). *Dashed lines legend:* ■■ = ■ − ■, ■■ = ■ − ■, ■■ = ■ − ■.

while the overall position given by both the kinematic and compliance contributions is displayed in red. Both figures show that the compliance of the suspension system significantly affects the wheel hub's trajectory, especially in terms of rotations.

5.3 TRAJECTORY PRESCRIBED PATH CONTROL PROBLEMS

The TPPC problems are characterized by DAE systems that describe the motion of a dynamical system whose trajectory or performance is prescribed by adding a set of path constraints to the equations of motion. The model equations evolve into a nonlinear semi-explicit DAEs. Within this system, the differential equations represent motion equations, while the algebraic equations correspond to the system's description and imposed path constraints, collectively constituting TPPC problems. Historically, addressing specific TPPC challenges involved the development of software models that promptly adjust control variables to approximate prescribed path profiles. In this context, we investigate general numerical techniques directly applicable to DAEs. TPPC problems are present in various fields, such as robot control, chemical process management, as well as space vehicle and aircraft guidance.
The DAE systems arising from TPPC simulations has typically the Hessenberg form [138]

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) & \text{differential equations} \\ \mathbf{0} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t) & \text{path constraints} \end{cases} \quad \text{with} \quad \mathbf{g}_\mathbf{x}\,\mathbf{f}_\mathbf{u} \text{ non-singular}$$

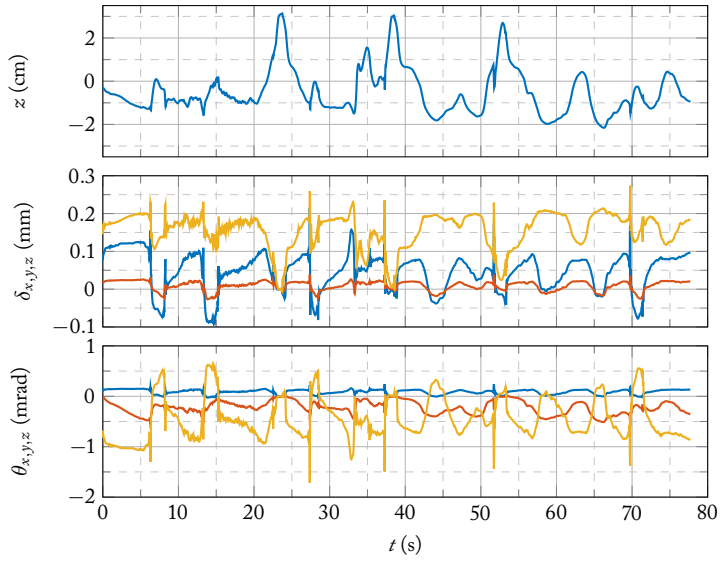


Figure 5.11: Simulation of the rear left double-wishbone suspension of the Formula SAE *E-Agle Trento Racing Team* vehicle [137] on the *Pista Azzurra* track in Jesolo (Italy). The simulation is conducted using the index-reduced DAE system, the tire-ground enveloping model, and the tire model presented in Appendices B and C, respectively. In the first plot, the suspension travel z is shown, while the second and third plots show the translations δ and rotation θ components of the compliance contribution at the wheel hub. *Legend*: ■ x-axis component, ■ y-axis component, ■ z-axis component.

for index-2 problems, and

$$\begin{cases} \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{u}, t) \\ \mathbf{y}' = \mathbf{g}(\mathbf{x}, \mathbf{y}, t) \\ \mathbf{0} = \mathbf{h}(\mathbf{y}, t) \end{cases} \qquad \text{with} \quad \mathbf{h}_\mathbf{y}\, \mathbf{g}_\mathbf{x}\, \mathbf{f}_\mathbf{u} \text{ non-singular}$$

for index-3 problems. The index of such systems is typically higher than the non-controlled counterpart. Indeed, the path constraints control is embedded in the state equations, often increasing the length of the differentiation chain to obtain a set of ODEs. Specifically, when the Jacobian $\mathbf{g}_\mathbf{u}$ is non-singular, the path equations are commonly referred to as control variable constraints and the corresponding DAEs has index-1. It is not uncommon to find that the path constraints in a TPPC problem are functions only of the differential variables so that $\mathbf{g}_\mathbf{u} = \mathbf{0}$ and the DAEs will be of higher index [9]. To showcase the capabilities of the proposed index reduction algo-

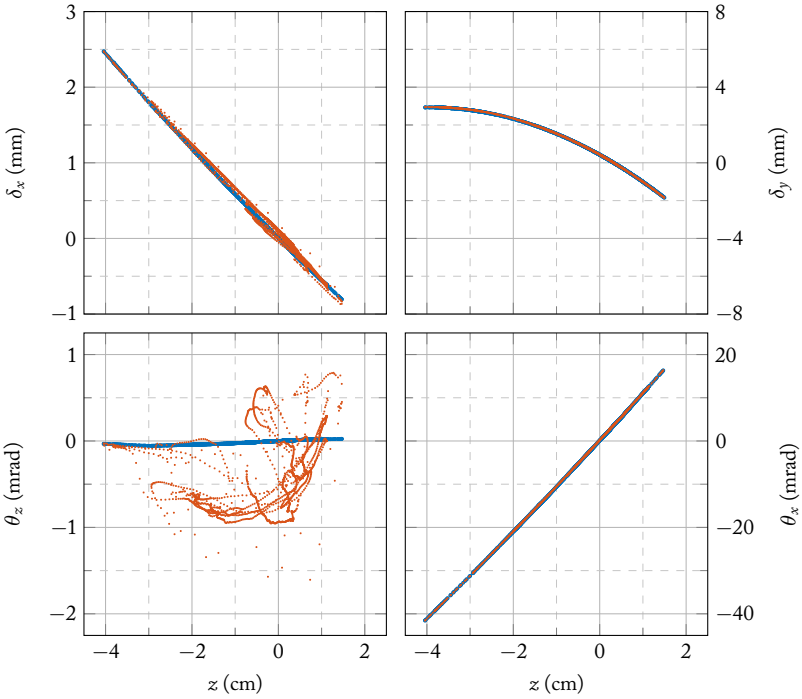


Figure 5.12: Wheel hub trajectories during the track drive simulation on the *Pista Azzurra* track in Jesolo (Italy). The chars show the pure kinematical trajectory of the wheel hub in blue (■) juxtaposed to the overall position given by both the kinematic and compliance contributions displayed in red (■).

rithm in handling such high-index TPPC problems, different examples are presented. Two problems regarding the initial and final phases of the space shuttle reentry, described by index-2 and index-3 DAEs [9]. Lastly, one problem on the control of a robotic arm, which is described as a complicated index-5 system [60]. A brief discussion of each of these examples, together with an introduction to the application field, is presented in the following sections.

5.3.1 SPACE SHUTTLE REENTRY PROBLEMS

In space applications, TPPC problems aid in vehicle performance analysis during design, particularly for lifting reentry vehicles aiming to determine maximum crossrange (or downrange) capability. Trajectory profiles are constrained by skin temperature

limits set by the thermal protection design. OC theory offers a direct approach to addressing maximum crossrange capability with heating constraints, formulated as a two-point boundary value problem involving DAEs and adjoint variables. However, solving such Optimal Control Problems (OCPs) requires starting solutions close to the optimal, especially with heating constraints, due to extreme sensitivity to initial guesses in shooting problems. An indirect TPPC approach or using TPPC to generate initial solutions for OC may offer more success. Typically, maximizing crossrange capability involves holding the angle of attack α near the maximum lift/drag value, often set at around 40 deg in National Aeronautics and Space Administration (NASA) space shuttle reentry simulations, leaving bank angle β adjustments to satisfy remaining functional constraints. In certain scenarios, varying system parameters can effectively optimize vehicle crossrange capability by ensuring trajectory adherence to specific constraints, thereby presenting a semi-explicit nonlinear DAEs TPPC problem [9, 138]. The discussion is confined to a reentry vehicle in the absence of propulsive forces, where we simplify the simulation to model solely spherical geopotential and spherical earth. The equations of motion in relative coordinates are thus expressed as follows

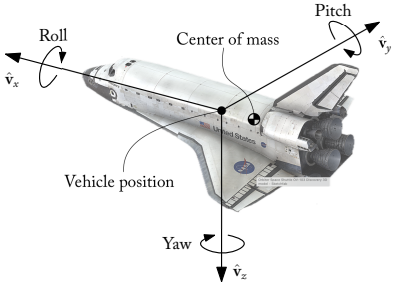
$$\begin{cases} H' = V_r \sin(\gamma) \\ \xi' = \dfrac{V_r \cos(\gamma)\sin(\mathcal{A})}{r\cos(\lambda)} \\ \lambda' = \dfrac{V_r}{r}\cos(\gamma)\cos(\mathcal{A}) \\ V_r' = -\dfrac{D}{m} - g\sin(\gamma) - \Omega_e^2 r\cos(\lambda)(\sin(\lambda)\cos(\mathcal{A})\cos(\gamma) - \cos(\lambda)\sin(\gamma)) \\ \gamma' = \dfrac{L\cos(\beta)}{mV_r} + \dfrac{\cos(\gamma)}{V_r}\left(\dfrac{V_r^2}{r} - g\right) + 2\Omega_e\cos(\lambda)\sin(\mathcal{A})\ldots \\ \qquad + \dfrac{\Omega_e^2 r\cos(\lambda)}{V_r}(\sin(\lambda)\cos(\mathcal{A})\sin(\gamma) + \cos(\lambda)\cos(\gamma)) \\ \mathcal{A}' = \dfrac{L\sin(\beta)}{mV_r\cos(\gamma)} + \dfrac{V_r}{r}\cos(\gamma)\sin(\mathcal{A})\tan(\lambda)\ldots \\ \qquad -2\Omega_e(\cos(\lambda)\cos(\mathcal{A})\tan(\gamma) - \sin(\lambda)) + \dfrac{\Omega_e^2 r\cos(\lambda)\sin(\lambda)\sin(\mathcal{A})}{V_r\cos(\gamma)} \end{cases}, \tag{5.11}$$

where the state variables are $\mathbf{x} = [H, \xi, \lambda, V_r, \gamma, \mathcal{A}]^\top$. The parameters are the following

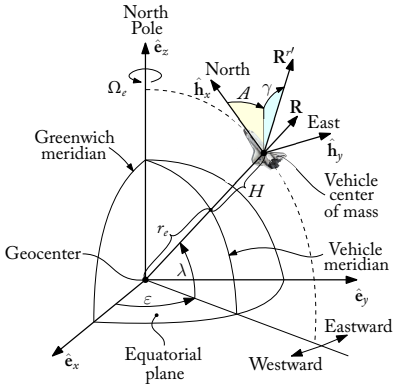
$r = H + r_e$,	distance from the earth center,
$r_e = 20902900$ ft	earth radius,
$g = \mu/r^2$	gravity force,
$\mu = 1.407653916 \times 10^16\,\text{ft}^3/\text{s}^2$	gravitational constant,
$\Omega_e = 360/(24 \cdot 60 \cdot 60)$ deg/s	earth angular speed,
$\rho(H) = 0.002378\exp(-H/23800)$	atmospheric density,
$L(V_r) = 1/2\rho C_L S V_r^2$	aerodynamic lift force,
$D(V_r) = 1/2\rho C_D S V_r^2$	aerodynamic drag force.

The aerodynamic lift and drag coefficients, respectively $C_L(\alpha)$ and $C_D(\alpha)$, as well as the

vehicle cross-sectional area S and mass m, will be later specified on the specific test. The control variables, which dictate both the magnitude and direction of the aerodynamic force applied to the vehicle, are assessed within the body coordinate system (refer to Figure 5.13a). The bank angle β corresponds to a rotation or *roll* about the vehicle's x-axis, while the angle of attack α is measured from the relative velocity vector of the vehicle to the body x-axis, representing a rotation or *pitch* about the body y-axis. For a more detailed explanation of these parameters and the coordinate system on the presented tests, please refer to [139]. Nonetheless, figure 5.13 illustrates the space shuttle coordinate system, as well as the vehicle's position with respect to the earth reference frame.



(a) Space shuttle reentry vehicle coordinate system.



(b) Earth reference frame.

Figure 5.13: Coordinate systems for the space shuttle reentry problem [9, 138].

Once the space shuttle concludes its mission in space, it must return to Earth for landing, subject to various mission constraints, such as heating limitations to prevent vehicle damage. Instead of directly imposing these heating constraints as algebraic limitations, an alternative approach involves prescribing a nominal drag acceleration versus a relative velocity profile. This profile is selected to ensure that temperature constraints are satisfied as long as the vehicle follows a trajectory complying with this drag constraint. During reentry, the standard equations of motion (5.11) are compounded with an algebraic constraint representing the drag acceleration profile, forming a DAE system. Typically, the angle of attack remains constant or is only slightly varied, while the bank angle serves as the control variable. In the following, we examine the initial and final phases of reentry. The first involves maneuvering the vehicle from a given state to one lying on the nominal drag constraint. While the second entails flying the vehicle along the nominal drag constraint.

5.3.1.1 INITIAL STAGE REENTRY PROBLEM

We now examine the initial stage reentry TPPC problem in the same form of [138]. This problem is part of a broader trajectory optimization process aimed at determining a surface of admissible reentry states. Following completion of on-orbit maneuvers, the vehicle must transition to a state vector enabling safe flight to the landing site. This set of allowable states is denoted as a target line. A given initial state qualifies as a target line point if a trajectory can be executed from that state in such a way that the vehicle's drag versus relative velocity profile smoothly aligns with the specified nominal profile, without overshooting and thus without violating any temperature constraints. Consider now the description of the vehicle's drag acceleration versus relative velocity profile, expressed as

$$\frac{D}{m} - (C_0 + C_1(V_r - V_0) + C_2(V_r - V_0)^2 + C_3(V_r - V_0)^3) = 0\,, \tag{5.12}$$

for a time $t \in [t_0, t_1] = [32.868734542, 419.868734542]$ s, and where V_0, is the vehicle's initial velocity at t_0 and $C_0 = 3.974960446019$, $C_1 = -0.01448947694635$, $C_2 = -0.2156171551995 \cdot 10^{-4}$, and $C_3 = -0.1089609507291 \cdot 10^{-7}$ are constants chosen so that the initial state vector satisfies (5.12) and its first derivative at t_0, and so that this transitional phase smoothly joins with the nominal profile. Notice that, posed in this way, the resulting TPPC system is a semi-explicit index-3 DAEs problem. In this test, the set of initial values are $\gamma = -0.749986488$ deg, $A = 62.7883367$ deg, $H = 264039.3280$ ft, $\xi = 177.718047$ deg, $\lambda = 32.0417885$ deg, $V_r = 24317.0798$ ft/s, and $\beta = 41.10071834$ deg. The lift and drag coefficients $C_L = 0.8769230769$ and $C_D = 0.8246153846$, as well as the angle of attack $\alpha = 40$ deg are assumed to be constant throughout the simulation. The vehicle mass is $m = 5964.496499824$ slugs, and its cross-sectional reference area is $S = 2690$ ft^2.

To address this index-3 DAEs, we utilize the proposed index reduction algorithm. The complexity of expressions encountered during the algorithm's execution is detailed in Table 5.12. Notably, the index reduction algorithm effectively reduced the system to index-0 without the aid of hierarchical representations, and the expression growth is inhibited strongly by successful simplification and only limited to the last reduction step. We conduct numerical integration of the reduced DAE system using both the MAPLE® and INDIGO solvers. Specifically, MAPLE® encounters challenges in integrating the original DAE system due to difficulties in projecting initial values into the solution space, where ICs are deemed inconsistent with the algebraic constraints. In contrast, the INDIGO numerical solver successfully integrates the reduced DAE system, yielding results depicted in Figure 5.14.

5.3.1.2 FINAL STAGE REENTRY PROBLEM

We now examine the final stage reentry TPPC problem in the same form of [9]. Let us assume the objective is to navigate a vehicle along a predetermined azimuth A and

Table 5.12: Expression complexity encountered throughout the index reduction of the initial stage space shuttle reentry problem [9] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Initial Stage Space Shuttle Reentry Problem [9]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 153\,\mathrm{f} + 2\,\mathrm{d} + 275\,\mathrm{m} + 59\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-3 DAEs	$11\,\mathrm{f} + 9\,\mathrm{m} + 5\,\mathrm{a}$	$118\,\mathrm{f} + 1\,\mathrm{d} + 220\,\mathrm{m} + 36\,\mathrm{a}$	$6\,\mathrm{f} + 1\,\mathrm{d} + 28\,\mathrm{m} + 10\,\mathrm{a}$
Index-2 DAEs	$11\,\mathrm{f} + 9\,\mathrm{m} + 5\,\mathrm{a}$	$118\,\mathrm{f} + 1\,\mathrm{d} + 220\,\mathrm{m} + 36\,\mathrm{a}$	$81\,\mathrm{f} + 2\,\mathrm{d} + 472\,\mathrm{m} + 87\,\mathrm{a}$
Index-1 DAEs	$11\,\mathrm{f} + 9\,\mathrm{m} + 5\,\mathrm{a}$	$118\,\mathrm{f} + 1\,\mathrm{d} + 220\,\mathrm{m} + 36\,\mathrm{a}$	$567\,\mathrm{f} + 3\,\mathrm{d} + 6198\,\mathrm{m} + 888\,\mathrm{a}$
Index-0 DAEs	$4053\,\mathrm{f} + 24\,\mathrm{d} + 41102\,\mathrm{m} + 5792\,\mathrm{a}$	$118\,\mathrm{f} + 1\,\mathrm{d} + 220\,\mathrm{m} + 36\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 3075\,\mathrm{f} + 4\,\mathrm{d} + 34811\,\mathrm{m} + 4734\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 654\,\mathrm{f} + 6\,\mathrm{d} + 6698\,\mathrm{m} + 985\,\mathrm{a}$

Table 5.13: Expression complexity encountered throughout the index reduction of the final stage space shuttle reentry problem [9] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Final Stage Space Shuttle Reentry Problem [9]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 157\,\mathrm{f} + 1\,\mathrm{d} + 272\,\mathrm{m} + 56\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-2 DAEs	$11\,\mathrm{f} + 9\,\mathrm{m} + 5\,\mathrm{a}$	$126\,\mathrm{f} + 1\,\mathrm{d} + 237\,\mathrm{m} + 39\,\mathrm{a}$	$2\,\mathrm{f} + 4\,\mathrm{m} + 4\,\mathrm{a}$
Index-1 DAEs	$5\,\mathrm{f} + 3\,\mathrm{m} + 3\,\mathrm{a}$	$43\,\mathrm{f} + 89\,\mathrm{m} + 14\,\mathrm{a}$	$92\,\mathrm{f} + 1\,\mathrm{d} + 173\,\mathrm{m} + 30\,\mathrm{a}$
Index-0 DAEs	$428\,\mathrm{f} + 6\,\mathrm{d} + 714\,\mathrm{m} + 119\,\mathrm{a}$	$52\,\mathrm{f} + 1\,\mathrm{d} + 107\,\mathrm{m} + 18\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 425\,\mathrm{f} + 1\,\mathrm{d} + 742\,\mathrm{m} + 138\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 94\,\mathrm{f} + 1\,\mathrm{d} + 177\,\mathrm{m} + 34\,\mathrm{a}$

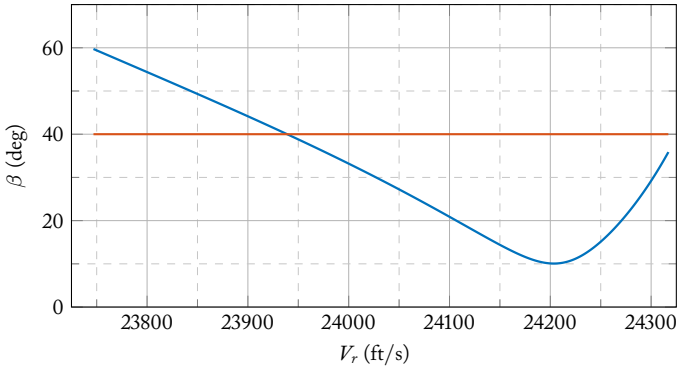


Figure 5.14: Control history of the initial stage space shuttle reentry problem [9]. The vehicle's trajectory is prescribed by the drag acceleration D/m versus a relative velocity V_r profile reported in (5.12). The vehicle's state is controlled by the bank angle β and the angle of attack is kept fixed at $\alpha = 40$ deg. *Legend*: ■ bank angle β, and ■ angle of attack α.

flight path angle trajectory γ, as defined by the constraints on state variables

$$\gamma + 1 + 9\left(\frac{t}{300}\right)^2 = 0\,, \quad \text{and} \quad A - 45 + 90\left(\frac{t}{300}\right)^2 = 0\,. \tag{5.13}$$

for a time $t \in [0, 300]$ s. The flight path angle γ spans from $[-1, -10]$ deg, while the azimuth A ranges from $[45, 135]$ deg. Both the bank angle β and the angle of attack α serve as control variables, *i.e.*, $\mathbf{u} = [\beta, \alpha]^\top$. The DAE system is index-2 for a lifting reentry vehicle as long as $(0.05\rho S V_r/m)^2 C_L(\alpha)/\cos(\gamma) \neq 0$. Notice that to be physically consistent we require that $V_r \neq 0$ deg, $\rho \neq 0$ deg, $\alpha \neq 0$ deg, $\gamma \neq 90$ deg, as well as $\lambda \neq 180$ deg. The related index-1 DAEs can be obtained directly by differentiating the algebraic constraints once and substituting for A' and γ' from the differential equations. Consistent initial values for the DAE system are determined by selecting the initial differential and control variables to satisfy (5.13) the two new hidden constraints in the related index-1 system. The set of initial values used in this experiment are $H = 100000$ ft, $\xi = 0$ deg, $\lambda = 0$ deg, $A = 0$ deg, $V_r = 12000$ ft/s, $\gamma = -1$ deg, $A = 45$ deg, $\beta = -0.05220958616134$ deg, and $\alpha = 2.6728700742$ deg. The lift and drag coefficients are set to $C_L = 0.01\alpha$ and $C_D = 0.04 + 0.1C_L^2$, respectively. The vehicle mass is $m = 2.890532728$ slugs, and its cross-sectional reference area is $S = 1\,\text{ft}^2$.

To solve the index-2 DAE system, we apply the proposed index reduction algorithm. The expression complexity encountered throughout the index reduction is reported in Table 5.13. As we can see, the index reduction algorithm successfully reduces the index of the DAE system to index-0 with minimal expression swelling in the last reduction

step. The numerical integration of the reduced DAE system is performed using both the MAPLE® and INDIGO numerical solvers. In this regard, MAPLE® is again not able to integrate the original DAE system due to the incapacity of projecting the initial values into the solution space (*i.e.*, ICs are not judged to be consistent with the algebraic constraints). Conversely, the numerical integration of the reduced DAE system using the INDIGO numerical solver is successful, and the results are presented in Figure 5.15, where the bank angle β and the angle of attack α controls history are shown.

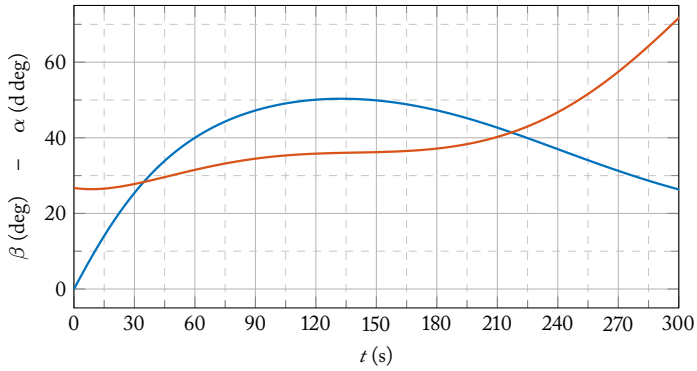


Figure 5.15: Controls history of the space shuttle reentry problem [9] in the time interval $t \in [0, 300]$ s. *Legend*: ■ bank angle β, and ■ angle of attack α.

5.3.2 ROBOT ARM CONTROL

Another example of a TPPC problem is the control of a robot arm, described as an index-5 DAE system [60]. The system describes the path control of a two-link, flexible joint, planar robotic arm from [140]. This system, which is frequently used as a DAEs benchmark test, is characterized by a high index – typical of TPPC-MB problems – as well as by the presence of various singularities [141]. The problem is a semi-explicit DAEs of dimension 8 with 2 path constraints. The system is described by the following

equations

$$
\begin{cases}
x_1' = x_4 \\
x_2' = x_5 \\
x_3' = x_6 \\
x_4' = 2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) - (2x_3 - x_2)(a(x_3) + 2b(x_3)) - a(x_3)(u_1 - u_2) \\
x_5' = 2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) + (2x_3 - x_2)(1 - 3a(x_3) - 2b(x_3)) \dots \\
\qquad -a(x_3)(u_1 - u_2) + u_2 \\
x_6' = 2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) + (2x_3 - x_2)(a(x_3) - 9b(x_3)) \dots \\
\qquad -(a(x_3) + b(x_3))(u_1 - u_2) - d(x_3)(x_4+x_6)^2 - 2x_4^2 c(x_3) \\
0 = \cos(x_1) + \cos(x_1 + x_3) - p_1(t) \\
0 = \sin(x_1) + \sin(x_1 + x_3) - p_2(t)
\end{cases},
$$

with

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

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

Here, the variables $[x_1, x_2, x_3]^\top$ represent the angular coordinates of the end effector, while $[x_4, x_5, x_6]^\top$ are their derivatives. The control variables are $[u_1, u_2]^\top$, which represent the torques applied to the joints. The end effector path constraints are given by

$$
p_1(t) = \cos(\exp(t) - 1) + \cos(t - 1), \quad \text{and} \quad p_2(t) = \sin(1 - \exp(t)) + \sin(1 - t).
$$

Figure 5.16 illustrates the robotic arm problem, as well as its variables and control parameters.

The complexity of expressions encountered throughout the index reduction is detailed in Table 5.14. Notably, the index reduction algorithm effectively reduces the system to index-0 without introducing any veiling variable, however, substantial expression growth is observed. The simplification of the expressions within 100 s of CPU time is not feasible. Hierarchical representation through veiling variables is necessary to simplify the handling of the system expressions. The complexity of the expressions encountered throughout the index reduction with the aid of hierarchical representation is detailed in Table 5.15. Here, it can be noticed that the introduction of veiling variables effectively reduces the overall expression complexity by 3 orders of magnitude. This is attributed to the fact that the chunks of the system are now more efficiently handled by the CAS, and simplification is performed.

The numerical integration of the reduced DAE system is performed using both the MAPLE® and INDIGO numerical solvers. In this regard, MAPLE® can not integrate the original DAE system due to the incapacity of projecting the initial values into the solution space. Conversely, the numerical integration of the reduced DAE system using the RadauIIA5 INDIGO numerical solver is successful in the interval $t \in [0,\ 0.98]$ s.

Table 5.14: Expression complexity encountered throughout the index reduction of the robotic arm problem [9] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Robotic Arm [60]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 125\,\mathrm{f} + 19\,\mathrm{d} + 56\,\mathrm{m} + 64\,\mathrm{a} \quad \mathbf{h}(\mathbf{x}, t) = 0$		
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-5 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{a}$
Index-4 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$24\,\mathrm{f} + 6\,\mathrm{m} + 14\,\mathrm{a}$
Index-3 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$162\,\mathrm{f} + 2\,\mathrm{d} + 138\,\mathrm{m} + 114\,\mathrm{a}$
Index-2 DAEs	$14\,\mathrm{f} + 2\,\mathrm{d} + 6\,\mathrm{m} + 6\,\mathrm{a}$	$372\,\mathrm{f} + 4\,\mathrm{d} + 375\,\mathrm{m} + 253\,\mathrm{a}$	$972\,\mathrm{f} + 1\,\mathrm{d} + 1062\,\mathrm{m} + 770\,\mathrm{a}$
Index-1 DAEs	$14\,\mathrm{f} + 2\,\mathrm{d} + 6\,\mathrm{m} + 6\,\mathrm{a}$	$372\,\mathrm{f} + 4\,\mathrm{d} + 375\,\mathrm{m} + 253\,\mathrm{a}$	$\star(6.5\,\mathrm{f} + 5.6\,\mathrm{m} + 1.8\,\mathrm{a}) \cdot 10^6 + 4\,\mathrm{d}$
Index-0 DAEs	$\star(8.3\,\mathrm{f} + 7.1\,\mathrm{m} + 2.3\,\mathrm{a}) \cdot 10^7 + 58\,\mathrm{d}$	$(2.4\,\mathrm{f} + 2.0\,\mathrm{m} + 0.9\,\mathrm{a}) \cdot 10^6 + 8\,\mathrm{d}$	0
Reduced DAEs	$\star\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = (8.6\,\mathrm{f} + 7.3\,\mathrm{m} + 2.4\,\mathrm{a}) \cdot 10^7 + 66\,\mathrm{d} \quad \star\mathbf{h}(\mathbf{x}, t) = (6.5\,\mathrm{f} + 5.6\,\mathrm{m} + 1.8\,\mathrm{a}) \cdot 10^6 + 7\,\mathrm{d}$		

Table 5.15: Expression complexity encountered throughout the index reduction with the aid of hierarchical representation of the robotic arm problem [9] DAE system. *Legend*: f = functions, v = veiling variables, a = additions, m = multiplications, and d = divisions.

Robotic Arm [60]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = 125\,\mathrm{f} + 19\,\mathrm{d} + 56\,\mathrm{m} + 64\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 0$	$\mathbf{v}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{g}(\mathbf{x}, \mathbf{v}, t)$	$\mathbf{a}(\mathbf{x}, \mathbf{v}, t)$
Index-5 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$16\,\mathrm{f} + 12\,\mathrm{a}$
Index-4 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$24\,\mathrm{f} + 6\,\mathrm{m} + 14\,\mathrm{a}$
Index-3 DAEs	0	$66\,\mathrm{f} + 3\,\mathrm{d} + 50\,\mathrm{m} + 35\,\mathrm{a}$	$162\,\mathrm{f} + 2\,\mathrm{d} + 138\,\mathrm{m} + 114\,\mathrm{a}$
Index-2 DAEs	$14\,\mathrm{f} + 2\,\mathrm{d} + 6\,\mathrm{m} + 6\,\mathrm{a}$	$66\,\mathrm{f} + 1\,\mathrm{v} + 3\,\mathrm{d} + 51\,\mathrm{m} + 35\,\mathrm{a}$	$1\,\mathrm{m} + 1\,\mathrm{v}$
Index-1 DAEs	$2\,\mathrm{v} + 1\,\mathrm{a}$	$66\,\mathrm{f} + 1\,\mathrm{v} + 3\,\mathrm{d} + 51\,\mathrm{m} + 35\,\mathrm{a}$	$9\,\mathrm{f} + 4\,\mathrm{v} + 2\,\mathrm{d} + 8\,\mathrm{m} + 5\,\mathrm{a}$
Index-0 DAEs	$7\,\mathrm{v} + 1\,\mathrm{d} + 2\,\mathrm{m} + 2\,\mathrm{a}$	$66\,\mathrm{f} + 2\,\mathrm{v} + 3\,\mathrm{d} + 52\,\mathrm{m} + 35\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', \mathbf{v}, t) = 90\,\mathrm{f} + 9\,\mathrm{v} + 4\,\mathrm{d} + 63\,\mathrm{m} + 48\,\mathrm{a}$	$\mathbf{h}(\mathbf{x}, \mathbf{v}, t) = 202\,\mathrm{f} + 5\,\mathrm{v} + 4\,\mathrm{d} + 141\,\mathrm{m} + 130\,\mathrm{a}$	

Hierarchical representation details (29 veils)

Original DAEs	$\mathbf{v}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{v}(\mathbf{x}, t)$
Index-5 DAEs	0
Index-4 DAEs	0
Index-3 DAEs	0
Index-2 DAEs	$1278\,\mathrm{f} + 3\,\mathrm{v} + 6\,\mathrm{d} + 1319\,\mathrm{m} + 918\,\mathrm{a}$
Index-1 DAEs	$8401\,\mathrm{f} + 20\,\mathrm{v} + 24\,\mathrm{d} + 9451\,\mathrm{m} + 6095\,\mathrm{a}$
Index-0 DAEs	$37010\,\mathrm{f} + 558\,\mathrm{v} + 56\,\mathrm{d} + 45087\,\mathrm{m} + 28665\,\mathrm{a}$
Reduced DAEs	$\mathbf{v}(\mathbf{x}, t) = 37010\,\mathrm{f} + 558\,\mathrm{v} + 56\,\mathrm{d} + 45087\,\mathrm{m} + 28665\,\mathrm{a}$

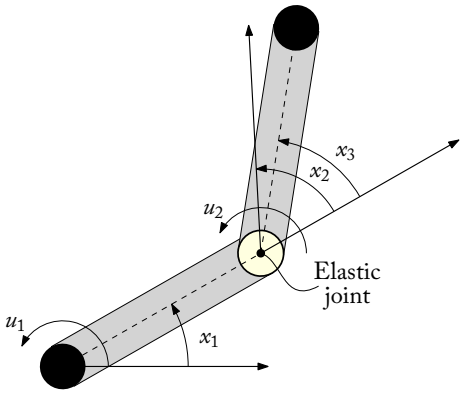


Figure 5.16: Robot arm control problem.

Notice that this system presents many singularities that hinder a flawless integration (refer to [141] for a detailed analysis). For what concerns MATHEMATICA® and MATLAB® performances, the tests we performed showed that both solvers are not able to integrate the original DAE system due to the incapacity of projecting the initial values into the solution space. Furthermore, the Pantelides algorithm implemented in MATLAB® can not reduce the index of the system to index-1.

5.4 ELECTRICAL CIRCUITS

Historically, the simulation of electrical networks stimulated the study of DAEs and their solutions since the early 70s [142]. DAEs encountered in this domain exhibit a distinct structure, somewhat different from those arising from mechanical systems or TPPC problems. Typically, these DAEs are large and sparse, often linear, although nonlinearities may arise from some circuit components. Our focus here is not to give a detailed account of circuit design, but rather to illustrate the types of DAEs that may arise and how various aspects of the circuit influence DAE system properties such as index, solvability, and numerical solution.

Consider an electrical network comprising b branches connected to n nodes. Assigning a current variable i_b to each branch and a voltage variable v_n to each node, the circuit equations stem from Kirchoff's laws, *i.e.*, the algebraic sum of currents into a node is zero, and the algebraic sum of the voltage drops around a loop is zero. By convention, current denotes the net flow of positive charge, with a designated current direction along each branch assigned by designating one node as negative and the other as positive (with current flowing from positive to negative). The circuit's topology is described by a $b \times n$ network incidence matrix $\mathbf{A}$. The (i, j) element of $\mathbf{A}$ is ± 1 if node j is the $\pm$ node for the i-th branch. Denoting $\mathbf{i}_b$ as the current variables vector, Kirchoff's

current law states that $\mathbf{A}^\top \mathbf{i}_b = 0$. The voltage drop across each branch is defined as the difference between the voltage at the positive node and that at the negative node. These branch voltages $\mathbf{p}_b$ are expressed in terms of the nodal voltages $\mathbf{p}_n$ as $\mathbf{p}_b = \mathbf{A}\mathbf{p}_n$. Linear circuits composed of resistors, capacitors, and inductors can result in large sparse linear DAEs. In such circuits, the voltage-current relationship across a resistor branch follows Ohm's law, $v_r = Ri_r$, with a positive resistance R. Similarly, the voltage-current characteristics of linear capacitors and inductors satisfy $i_c = C(\mathrm{d}v_c/\mathrm{d}t)$ and $v_l = L(\mathrm{d}i_l/\mathrm{d}t)$, respectively. However, including transistors or unicursal elements introduces nonlinearity into DAE systems. The solvability of DAEs arising from linear circuits lacking operational amplifiers is solely influenced by the network topology. However, circuits containing differential amplifiers, typically realized using operational amplifiers, may give rise to DAEs of arbitrarily high index. DAEs arising from linear circuits incorporating operational amplifiers depends on the specific voltage-current characteristics of the circuit components for solvability. Moreover, the number of independent ICs may vary depending on circuit parameter values. The potential for arbitrarily high-index DAEs originating from circuits is demonstrated in examples such as a cascade of differential amplifiers. Furthermore, high-index DAEs may emerge when different variables are designated as inputs and outputs. For example, whether a device functions as a differentiator or an integrator depends on the designation of inputs and outputs [9].

In the following, we present three examples of electrical circuits, the first being an eight-node transistor amplifier, the second an electric ring modulator, and the third a cascade of differential amplifiers. The first two examples are taken from [132, 134], while the third is from [9].

5.4.1 EIGHT-NODE TRANSISTOR-AMPLIFIER

This problem originates from electrical circuit analysis, and it is a model for the transistor amplifier. The circuit diagram is depicted in Figure 5.17. Here, U_e is the input signal, while the amplified output signal is found in point 8. The circuit is modeled by a system of DAEs of index-1, consisting of 8 equations, which are written in matrix form as $\mathbf{M}\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$, where $\mathbf{x} = [x_1, \ldots, x_8]^\top$, $\mathbf{M}$ and $\mathbf{f}(\mathbf{x}, t)$ given by

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

$$\mathbf{f}(\mathbf{x},t)=\begin{bmatrix} -\dfrac{U_e(t)}{R_0}+\dfrac{x_1}{R_0} \\ -\dfrac{U_b}{R_2}+x_2\left(\dfrac{1}{R_1}+\dfrac{1}{R_2}\right)-(\alpha-1)g(x_2-x_3) \\ -g(x_2-x_3)+\dfrac{x_3}{R_3} \\ -\dfrac{U_b}{R_4}+\dfrac{x_4}{R_4}+\alpha g(x_2-x_3) \\ -\dfrac{U_b}{R_6}+x_5\left(\dfrac{1}{R_5}+\dfrac{1}{R_6}\right)-(\alpha-1)g(x_5-x_6) \\ -g(x_5-x_6)+\dfrac{x_6}{R_7} \\ -\dfrac{U_b}{R_8}+\dfrac{x_7}{R_8}+\alpha g(x_5-x_6) \\ \dfrac{x_8}{R_9} \end{bmatrix},$$

where $g(x)=\beta(\exp(x/U_f)-1)\,\mathrm{A}$, and $U_e(t)=0.1\sin(200\pi t)\,\mathrm{V}$. ICs at $t=0$ and parameters are given by

$$\mathbf{x}_0=\begin{bmatrix} 0 \\ U_b/(R_2/R_1+1) \\ U_b/(R_2/R_1+1) \\ U_b \\ U_b/(R_6/R_5+1) \\ U_b/(R_6/R_5+1) \\ U_b \\ 0 \end{bmatrix}, \quad \text{and} \quad \begin{aligned} U_b &= 6\,\mathrm{V}, \\ U_f &= 0.026\,\mathrm{V}, \\ \alpha &= 0.99, \\ \beta &= 10^{-}6\,\mathrm{A}, \\ R_0 &= 1\,\mathrm{k}\Omega, \\ R_k &= 9\,\mathrm{k}\Omega \text{ with } k=1,\ldots,9, \\ C_k &= k\,\mu\mathrm{F} \text{ with } k=1,\ldots,9. \end{aligned}$$

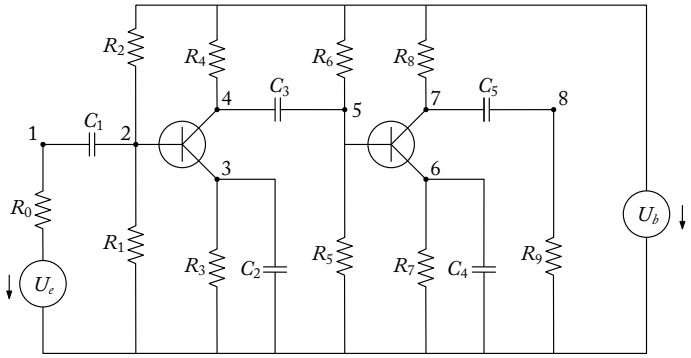


Figure 5.17: Eight-node transistor-amplifier circuit [132, 134].

The index reduction process is performed smoothly, and the complexity of the expressions encountered throughout the index reduction is detailed in Table 5.16. As we can

see, the expression complexity is not affected by the index reduction process. The numerical integration of the reduced DAE system is carried out using both the MAPLE® and INDIGO numerical solvers. In this regard, MAPLE®, MATHEMATICA®, MATLAB® (both Pantelides and Gaussian elimination), and INDIGO can integrate the original DAE system in the specified time interval $t \in [0, 0.2]$ s.

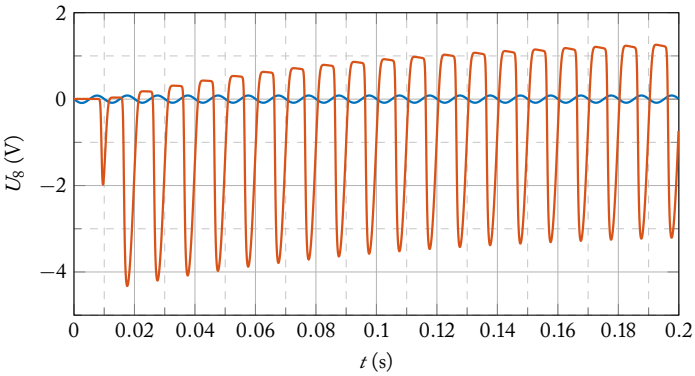


Figure 5.18: Transistor amplifier circuit integration results [132, 134] in the time interval $t \in [0, 0.2]$ s. Notice that the output signal U_8 is equal to the state variable x_8. *Legend*: ■ input signal $U_e(t)$, and ■ output signal U_8.

5.4.2 ELECTRIC RING MODULATOR

The electric ring modulator is an interesting example of a system whose structure depends on the specific values of the parameters. The system is in the form $\mathbf{M}\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$, where $\mathbf{x} = [x_1, \ldots, x_{15}]^\top$, $\mathbf{M}$ and $\mathbf{f}(\mathbf{x}, t)$ being given by

$$\mathbf{M} = \operatorname{diag}(C, C, C_s, C_s, C_s, C_s, C_p, L_b, L_b, L_{s2}, L_{s3}, L_{s2}, L_{s3}, L_{s1}, L_{s1}) ,$$

Table 5.16: Expression complexity encountered throughout the index reduction of the eight-node transistor-amplifier problem [132, 134] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Eight-Nodes Transistor-Amplifier [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 55\,\mathrm{f} + 21\,\mathrm{d} + 29\,\mathrm{m} + 41\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-1 DAEs	$5\,\mathrm{a}$	$17\,\mathrm{f} + 11\,\mathrm{d} + 22\,\mathrm{m} + 20\,\mathrm{a}$	$19\,\mathrm{f} + 12\,\mathrm{d} + 44\,\mathrm{m} + 24\,\mathrm{a}$
Index-0 DAEs	$24\,\mathrm{f} + 26\,\mathrm{d} + 24\,\mathrm{m} + 24\,\mathrm{a}$	$18\,\mathrm{f} + 12\,\mathrm{d} + 26\,\mathrm{m} + 20\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 74\,\mathrm{f} + 26\,\mathrm{d} + 87\,\mathrm{m} + 49\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 19\,\mathrm{f} + 12\,\mathrm{d} + 44\,\mathrm{m} + 26\,\mathrm{a}$

Table 5.17: Expression complexity encountered throughout the index reduction of the electric ring modulator problem [132, 134] DAE system. *Legend*: f = functions, a = additions, m = multiplications, and d = divisions.

Electric Ring Modulator [132, 134]

Original DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 116\,\mathrm{f} + 3\,\mathrm{d} + 75\,\mathrm{m} + 92\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 0$
Reduction step	$\mathbf{E}(\mathbf{x}, t)$	$\mathbf{g}(\mathbf{x}, t)$	$\mathbf{a}(\mathbf{x}, t)$
Index-2 DAEs	0	$51\,\mathrm{f} + 3\,\mathrm{d} + 41\,\mathrm{m} + 41\,\mathrm{a}$	$44\,\mathrm{f} + 32\,\mathrm{m} + 36\,\mathrm{a}$
Index-1 DAEs	$86\,\mathrm{f} + 68\,\mathrm{m} + 66\,\mathrm{a}$	$262\,\mathrm{f} + 13\,\mathrm{d} + 416\,\mathrm{m} + 220\,\mathrm{a}$	$10\,\mathrm{f} + 2\,\mathrm{d} + 18\,\mathrm{m} + 11\,\mathrm{a}$
Index-0 DAEs	$86\,\mathrm{f} + 12\,\mathrm{d} + 72\,\mathrm{m} + 70\,\mathrm{a}$	$262\,\mathrm{f} + 13\,\mathrm{d} + 416\,\mathrm{m} + 220\,\mathrm{a}$	0
Reduced DAEs	$\mathbf{F}(\mathbf{x}, \mathbf{x}', t) = 335\,\mathrm{f} + 15\,\mathrm{d} + 537\,\mathrm{m} + 256\,\mathrm{a}$		$\mathbf{h}(\mathbf{x}, t) = 54\,\mathrm{f} + 2\,\mathrm{d} + 50\,\mathrm{m} + 47\,\mathrm{a}$

$$\mathbf{f}(\mathbf{x},t)=\begin{bmatrix} x_8-(x_{10}+x_{11})/2+x_{14}-x_1/R\\ x_9-(x_{12}+x_{13})/2+x_{15}-x_2/R\\ x_{10}-q(U_{d1})+q(U_{d4})\\ -x_{11}+q(U_{d2})-q(U_{d3})\\ x_{12}+q(U_{d1})-q(U_{d3})\\ -x_{13}-q(U_{d2})+q(U_{d4})\\ -x_7/R_p+q(U_{d1})+q(U_{d2})-q(U_{d3})-q(U_{d4})\\ x_1\\ x_2\\ x_1/2-x_3-R_{g2}x_{10}\\ -x_1/2+x_4-R_{g3}x_{11}\\ x_2/2-x_5-R_{g2}x_{12}\\ -x_2/2+x_6-R_{g3}x_{13}\\ -x_1+U_{in1}(t)-(R_i+R_{g1})x_{14}\\ -x_2-(R_c+R_{g1})x_{15} \end{bmatrix},$$

where

$$\begin{aligned} U_{d1}&=x_3-x_5-x_7-U_{in2}(t)\,,\\ U_{d2}&=-x_4+x_6-x_7-U_{in2}(t)\,,\\ U_{d3}&=x_4+x_5+x_7+U_{in2}(t)\,,\\ U_{d4}&=-x_3-x_6+x_7+U_{in2}(t)\,,\\ q(U)&=\gamma(\exp(\delta U)-1)\,,\\ U_{in1}(t)&=0.5\sin(2000\pi t)\,,\\ U_{in2}(t)&=2\sin(20000\pi t)\,. \end{aligned}$$

ICs at $t=0$ are $\mathbf{x}_0=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]^\top$, while the parameters are given by

$$\begin{aligned} C&=16\,\text{nF}\,, & &\text{and} & R&=25\,\text{k}\Omega\,,\\ C_s&=0\text{ or }2\,\text{pF}\,, & & & R_p&=50\,\Omega\,,\\ C_p&=100\,\text{nF}\,, & & & R_{g1}&=36.3\,\Omega\,,\\ L_h&=4.45\,\text{H}\,, & & & R_{g2}&=17.3\,\Omega\,,\\ L_{s1}&=2\,\text{mH}\,, & & & R_{g3}&=17.3\,\Omega\,,\\ L_{s2}&=0.5\,\text{mH}\,, & & & R_i&=50\,\Omega\,,\\ L_{s3}&=0.5\,\text{mH}\,, & & & R_c&=600\,\Omega\,,\\ \gamma&=40.67286402\,, & & & \delta&=17.7493332\,. \end{aligned}$$

Interestingly, the system is an ODEs if $C_s\neq 0$, otherwise, it is an index-2 DAE system. In the original test problem in [132, 134], the authors set $C_s=2\,\text{pF}$ to obtain an ODE system. The electric ring modulator circuit is shown in Figure 5.19.

Considering $C_s=0$, the index reduction process is performed flawlessly through the presented algorithm, the complexities of the expressions encountered throughout the index reduction are detailed in Table 5.17. Numerical integration in the specified time interval $t\in[0,\,1]\,\text{ms}$ is successfully performed by INDIGO numerical solvers. Conversely, MAPLE® is not able to integrate the original DAE due to computational time

exceeding 100 s. Results of the numerical integration are shown in Figure 5.20, where the modulated output signal U_2 is represented. Notice that the modulated signal is influenced strongly by the C_s capacity value.

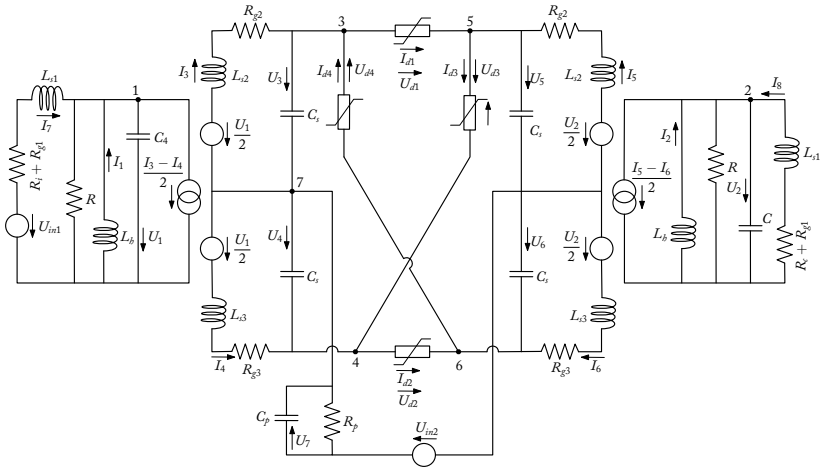


Figure 5.19: Electric ring modulator circuit [132, 134].

5.4.3 CASCADE OF DIFFERENTIAL AMPLIFIERS

The cascade of differential amplifiers is an example of a system whose index can be arbitrarily high, depending on how many operational amplifiers are cascaded. If we consider a circuit with one differential amplifier as shown in Figure 5.21a and, assuming that the operational amplifier is ideal, the circuit equations lead to the relation $x = -CRU'(t)$ where $U(t)$ is the generator voltage. Since the solution to the circuit equations involves at least one derivative of the input function, the system must be index-1. By cascading a series of $p \in \mathcal{N}$ differential amplifiers in a circuit as in Figure 5.21b, we can see that the resulting DAEs index is equal to p. Specifically, the voltage at the output of the p-th differential amplifier is given by $x_p = -C_pR_px'_{p-1}$, where $v = -C_1R_1U'(t)$. Thence the following system of equations is obtained

$$\begin{cases} x_1 = -C_1R_1U'(t) \\ x_2 = -C_2R_2x'_1 \\ \quad\vdots \\ x_p = -C_pR_px'_{p-1} \end{cases} \qquad \text{whose analytical solution is} \quad x_i = \prod_{j=1}^{i}(-C_jR_j)U^{(i)}(t)\,.$$

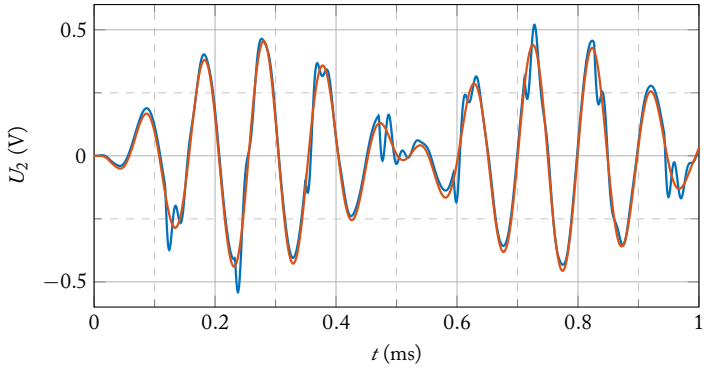
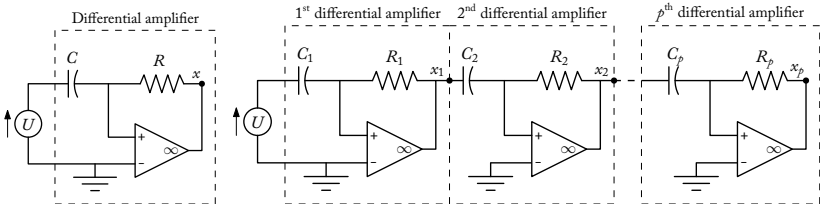


Figure 5.20: Electric ring modulator circuit integration results [132, 134] in the time interval $t \in [0, 1]$ ms. The lines represent the modulated output signal U_2, which is equal to the state variable x_2. *Legend*: ■ output signal U_2 with $C_s = 0$ pF (DAE system), ■ output signal U_2 with $C_s = 2$ pF (ODE system).

Consistent ICs are easily obtained by evaluating the analytical solution at $t = t_0$, *i.e.*,

$$\mathbf{x}_0 = \begin{bmatrix} -C_1 R_1 U'(t)\big|_{t=t_0} \\ \vdots \\ \prod_{j=1}^{p} (-C_j R_j) U^{(p)}(t)\big|_{t=t_0} \end{bmatrix}.$$



(a) Circuit with a single differential amplifier.

(b) Circuit with a cascade of p differential amplifiers.

Figure 5.21: Circuit diagrams for the cascade of differential amplifiers problem [9].

The index reduction of such a system, which falls under the category of linear DAEs, is straightforward. INDIGO can reduce to index-0 systems made of up to $p = 100$ differential amplifiers, even if the performance of the reduction process is strongly linked

to the capabilities of MAPLE® symbolic kernel to deal with sparse large matrices. An interesting aspect that this last application brings to light is how systematically the index reduction process is carried out through INDIGO. The system is transformed in the form $\mathbf{E}(\mathbf{x}, t)\,\mathbf{x}' = \mathbf{g}(\mathbf{x}, t)$ with $\mathbf{a}(\mathbf{x}, t) = \mathbf{0}$, which is explicitly given by

$$\left[\begin{array}{ccc|c} -C_2R_2 & & & 0 \\ & \ddots & & \vdots \\ & & -C_pR_p & 0 \end{array}\right] \left[\begin{array}{c} x_1' \\ \vdots \\ \hline x_{p-1}' \\ \hline x_p' \end{array}\right] = \left[\begin{array}{c} x_2 \\ \vdots \\ x_p \end{array}\right]$$

with $\mathbf{a}(\mathbf{x}, t) = [-x_1 - C_1R_1U'(t)]$. Then the index reduction is performed p times, and the system $\mathbf{E}(\mathbf{x}, t)\,\mathbf{x}' = \mathbf{g}(\mathbf{x}, t)$ at the k-th index reduction step is equal to

$$\left[\begin{array}{c|c|c} 1 & & \\ \ddots & & \\ 1 & & \\ \hline & & -C_{k+2}R_{k+2} \\ & & \ddots \\ & & -C_pR_p \\ \hline & 1 & \end{array}\right] \left[\begin{array}{c} x_1' \\ \vdots \\ \hline x_{k+1}' \\ \hline x_{k+2}' \\ \vdots \\ \hline x_{p-1}' \\ \hline x_p' \end{array}\right] = \left[\begin{array}{c} -C_1R_1U''(t) \\ \vdots \\ \hline \displaystyle\prod_{j=1}^{k-1}(-C_jR_j)U^{(k)}(t) \\ \hline x_{k+2} \\ \vdots \\ \hline x_p \\ \hline \displaystyle\prod_{j=1}^{k}(-C_jR_j)U^{(k+1)}(t) \end{array}\right],$$

with algebraic equations $\mathbf{a}(\mathbf{x}, t) = \mathbf{0}$ and hidden constraints $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$ given by

$$\mathbf{a}(\mathbf{x}, t) = \left[-x_{k+1} + \prod_{j=1}^{k+1}(-C_jR_j)U^{(k+1)}(t)\right]$$

and

$$\mathbf{h}(\mathbf{x}, t) = \left[\begin{array}{c} -x_1 - C_1R_1U'(t) \\ \vdots \\ -x_k + \displaystyle\prod_{j=1}^{k}(-C_jR_j)U^{(k)}(t) \end{array}\right].$$

Finally, at the p^{th} step, the reduced index-0 system is written as

$$\left[\begin{array}{ccc} 1 & & \\ & \ddots & \\ & & 1 \end{array}\right] \left[\begin{array}{c} x_1' \\ \vdots \\ x_p' \end{array}\right] = \left[\begin{array}{c} -C_1R_1U''(t) \\ \vdots \\ \displaystyle\prod_{j=1}^{p-1}(-C_jR_j)U^{(p)}(t) \end{array}\right]$$

with

$$\mathbf{h}(\mathbf{x}, t) = \begin{bmatrix} -x_1 - C_1 R_1 U'(t) \\ \vdots \\ -x_k + \prod_{j=1}^{p} (-C_j R_j) U^{(p)}(t) \end{bmatrix}.$$

5.5 SUMMARY OF RESULTS AND DISCUSSION

It is now time to summarize the results obtained from the experiments and discuss the performance of the presented algorithm. To do so, we will consider the following aspects: the symbolic index reduction, the numerical integration, and the overall performance of the algorithm throughout the reduction steps. The results are summarized in Table 5.18, where the solution process is for each example reported in brief. Notice that in Table 5.18, the examples are numbered according to the order in which they are presented in this chapter, which will be useful for the following discussions..

5.5.1 SYMBOLIC INDEX REDUCTION

The first aspect to consider is the symbolic index reduction process. Specifically, the presented algorithm can successfully reduce the index of all the example DAE systems here considered. The computational cost of the expressions generated during the index reduction procedure is typically comparable to the original DAE system in most of the tests. Additionally, we have seen that the specific type of matrix factorization plays a crucial role in the computational cost of the expressions generated during the index reduction process. In this regard, the "standard" LU factorization produces better results in terms of computational cost than the FFLU factorization. Nonetheless, in examples 3, 6, 7, and 11 the expressions generated during the index reduction procedure are significantly more complex than the original DAE system. In these cases, the MAPLE® symbolic computation kernel is not able to perform the simplification within 100 s of CPU time, and the raw expressions are kept in the following reduction steps. As a consequence, the computational cost increases inherently throughout the following reduction steps. Notably, the DAE systems that are hard to simplify are those having a matrix $\mathbf{A}(\mathbf{x}, t)$ that is strongly dependent on the state variables $\mathbf{x}$ or equivalently that retains complicated divisions in the vector $\mathbf{b}(\mathbf{x}, t)$. In these cases, the computational cost of the vector $\mathbf{a}(\mathbf{x}, t)$ increases significantly due to successive and repeated differentiation and symbolic factorization. In examples 3 and 11, the hierarchical pivoting strategy is used to mitigate the expression swell and ensure the successful index reduction of the DAE system. In such examples the veiling strategy proved to be of substantial help, effectively reducing the expression complexities and ensuring the successful index reduction of the DAE system. On the other hand, in examples 6 and 7, the hierarchical representation is not used on purpose to understand the impact of the expression swell on the numerical integration process.

For the sake of completeness, also the index reduction algorithms provided by MATLAB® and MATHEMATICA® are tested. The results show that both software tools can reduce the index of the DAE systems, however, the computational cost of the expressions generated during the index reduction procedure cannot be evaluated due to either the lack of a built-in function to compute the expression complexity or the inability to access the reduced DAE systems' expressions.
Another aspect to mention is the sudden increase in computational complexity of the expressions generated in the last reduction steps. Even if such a sudden increase is not critical to the correct index reduction as the presented algorithm is insensitive to expression swell, it does undermine the numerical solution efficiency of the final DAE system. This highlights the need for further research on expression swell mitigation techniques in symbolic computation (see Section 3.4), as well as the need to use integrators that can handle index-1 DAEs. Specifically, the detection of linear index-1 variables during the index reduction process would be beneficial in minimizing the final DAE system complexity. This is particularly important for the numerical integration process, as the more complex the expressions, the more computationally expensive the numerical integration process becomes. Discussions on future work (Section 6.2) will provide insights on new developments in this direction.

5.5.2 NUMERICAL INTEGRATION

The numerical integration has also been carried out, with outcomes detailed in Table 5.18. This table reports the performance comparison between the joint index reduction algorithm and numerical integration schemes offered by MAPLE® and those of INDIGO. To ensure a fair comparison, both MAPLE® and INDIGO utilized the RKF 4(5) method for numerical integration of the DAE system. Identical error tolerances are applied, with a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-7}. The results illustrate that INDIGO's index reduction algorithm implementation effectively generates numerically stable reduced-index DAEs, ensuring consistent integration across examples. However, exceptions arise in examples 3, 5, 6, 7, 9, 10, 11, 13, and 14. Specifically, in examples 3, 5, 6, 7, 9, 11, and 13, MAPLE® fails to generate code for the reduced-index system within the expected time frame thus the integration is not performed. This is due to the complexity of the expressions generated during the index reduction procedure, which overloads the MAPLE®'s `CodeGeneration` package. On the other hand, in examples 10 and 14, the numerical integration can not be performed due to the inability of MAPLE® to verify the consistency of the IVP. The INDIGO numerical integration is successful in all examples, except for example 11, where it fails to generate the code for the reduced-index system within the expected time frame. This is due to the complexity of the expressions generated during the index reduction procedure, which overloads the INDIGO's `CodeGeneration` package. Notably, in examples 5, 6, and 7 the integration of the reduced-index system is successfully performed using the non-default implicit RadauIIA5 method. Nonetheless, the high expression swell in these examples leads to a significant increase in the computational cost of the numerical integration, however,

the numerical stability is not compromised.
MATHEMATICA® and MATLAB® are also tested for numerical integration. The results show that both software tools can integrate the original DAE systems in the specified time interval. The only exception is example 11, where both fail to integrate the DAE system due to the inability to project the ICs onto the solution manifold.
It is important to highlight that during the numerical integration, the symbolic code is not regenerated, even if the DAE system may locally change its index. Indeed, for some numerical values of states and parameters, the DAEs system structure may change, leading to numerical instability. While this does pose a potential issue, we have not encountered instability in the integration process. The pivot values are continuously monitored during the numerical integration process to ensure that the DAE system structure remains consistent, thereby experimentally proving that the presented pivoting strategy is effective.

Table 5.18: Numerical integration results of the reduced-index DAE systems. The table reports the name and reference of the DAE system, the index of the system, the integration interval $t \in [t_{\text{ini}}, t_{\text{end}}]$, and the outcomes of the whole code generation and integration process for both MAPLE® and INDIGO. If not otherwise specified, the tests are integrated using an embedded RKF 4(5) method with a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-7}. The computation time limit is 1000 s, to both generate the necessary code and perform numerical computations. *Legend:* ✓ successful code generation and numerical integration, ✗ errors in the code generation, numerical integration or time expired, and ⚠ warnings encountered or non-default settings used in the code generation or numerical integration.

Integrated DAE system	Integration outcomes and errors		
1. Particle Motion	MAPLE®	✓	Success
	MATHEMATICA®	✓	Success
Index-3 $t \in [0, 400\pi]$ s	MATLAB®	✓	`reduceDAEIndex` – Success
		✓	`reduceDAEToODE` – Success
	INDIGO	✓	Success
2. Car-Axis	MAPLE®	✓	Success
	MATHEMATICA®	✓	Success
Index-3 $t \in [0, 3]$ s	MATLAB®	✓	`reduceDAEIndex` – Success
		✓	`reduceDAEToODE` – Success
	INDIGO	✓	Success
3. Flexible Slider-Crank	MAPLE®	✗	Error, time expired in `dsolve`
Index-3 $t \in [0, 0.1]$ s	INDIGO	✓	Success
4. 2-Pendula	MAPLE®	✓	Success
Index-5 $t \in [0, 60]$ s	INDIGO	✓	Success

5. 3-Pendula	MAPLE®	✗	Error, time expired in `dsolve`
Index-7 $t \in [0, 60]$ s	INDIGO	✓⚠	Success with RadauIIA5 method
6. 4-Pendula	MAPLE®	✗	Error, time expired in `dsolve`
Index-9 $t \in [0, 60]$ s	INDIGO	✓⚠	Success with RadauIIA5 method
7. 5-Pendula	MAPLE®	✗	Error, time expired in `dsolve`
Index-11 $t \in [0, 60]$ s	INDIGO	✓⚠	Success with RadauIIA5 method
8. Double-Wishbone Suspension	MAPLE®	✓	Success
Index-3 $t \in [0, 60]$ s	INDIGO	✓	Success
9. Initial St. Space Shuttle Reentry	MAPLE®	✗	Error, time expired in `dsolve`
Index-3 $t \in [332.8, 419.8]$ s	INDIGO	✓	Success
10. Final St. Space Shuttle Reentry	MAPLE®	✗	Error, in `dsolve/.../checkconstraints`
Index-2 $t \in [0, 300]$ s	INDIGO	✓	Success
11. Robotic Arm	MAPLE®	✗	Error, time expired in `dsolve`
	MATHEMATICA®	✗	Error, time expired in `NDSolve`
Index-5 $t \in [0, 2]$ s	MATLAB®	✗	`reduceDAEIndex` – Error, reduction not complete
		✗	`reduceDAEToODE` – Error, in `sym/decic`
	INDIGO	✗	Error, time expired in `CodeGeneration`
12. 8-Nodes Transistor-Amplifier	MAPLE®	✓⚠	Warning, function evaluations limit exceeded
	MATHEMATICA®	✓	Success
Index-1 $t \in [0, 0.2]$ s	MATLAB®	✓	`reduceDAEIndex` – Success
		✓	`reduceDAEToODE` – Success
	INDIGO	✓	Success
13. Electric Ring Modulator	MAPLE®	✗	Error, time expired in `dsolve`
Index-2 $t \in [0, 1]$ ms	INDIGO	✓	Success
14. Cascaded Diff. Amplifiers	MAPLE®	✗	Error, in `dsolve/.../checkconstraints`
Arbitrary index $t \in [0, 10]$ s	INDIGO	✓	Success

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

This thesis has focused on the development of a hybrid computational framework that combines symbolic computation with numerical methods for the efficient and accurate solution of DAEs. The research aimed to address the inherent challenges in solving DAEs with a broad applicability across various domains of engineering. More specifically, we have presented a methodology for the automatic index reduction of DAE systems. The index reduction algorithm is based on the splitting of the system into differential and algebraic parts with the help of symbolic matrix factorization. However, the current capabilities of MAPLE® kernel do not allow for the handling of exceedingly large expressions. This brought us to exploit the latest state-of-the-art symbolic computation techniques to manage large expressions and to reduce the computational complexity of the equations generated during the index reduction procedure. Notably, the open-source LEM and LAST libraries implement matrix factorization with large expression management techniques and form the core of the symbolic index reduction. The index reduction algorithm is implemented in the INDIGO software package, which is composed of a MAPLE® library for the symbolic analysis of DAE systems, as well as of a MATLAB® library for the numerical integration of the reduced-index DAE systems. Hence, the INDIGO software package is capable of automatically reducing the index of generic DAE systems and provides a comprehensive set of tools for their solution, ranging from the involved symbolic manipulation to the MATLAB® code generation for the numerical integration of the DAE systems.

Symbolic-numerical examples showcase the capabilities of the proposed index reduction algorithm. The results show that the presented methodology can consistently reduce high-index DAE systems to index 0 or index 1. The computational cost of the

expressions generated during the index reduction procedure is comparable to the cost of the original DAE system in the majority of cases. However, the MAPLE® symbolic computation kernel can not always perform the simplification due to either the size of the expressions or the complexity of the operations involved. Consequently, the expression size increases significantly throughout the remaining reduction steps. Yet, the presented algorithm can successfully reduce the index of the DAE system. Still, the numerical efficiency of the final DAE system degrades by the increased number of computations needed to evaluate the swollen expressions. The inclusion of large expression management techniques into the symbolic index reduction algorithm effectively limits expression swell and augments the DAE system with index-1 variables, resulting in a less complex representation of the expressions generated during the index reduction process. Indeed, tests show that the DAE systems, which present intermediate expression swell during factorization, are successfully reduced to index-1 DAEs with the aid of expression swell mitigation, while retaining a slightly more compact representation of the generated expressions compared to the reduced index-0 DAE systems without veiling variables. A comparison between the joint index reduction algorithm and numerical integration schemes offered by MAPLE®, MATHEMATICA®, and MATLAB® with those offered by INDIGO demonstrates the effectiveness of the proposed methodology and software implementation.

Further investigations aimed at increasing the simulations' efficiency in vehicle dynamics are presented in the appendices and are linked to the main topic of the thesis through the MB simulation of a vehicle model's suspension system. Nonetheless, the comprehensive analysis and simulation of tire-ground interaction and structural analysis are also crucial elements of this research, each contributing to the advancements in HRT vehicle simulation. Specifically, accurate modeling of tire-ground interaction is fundamental for realistic vehicle dynamics simulations. This research developed specialized algorithms and models to simulate tire-road interactions, focusing on the physical modeling, yet providing the RT capabilities essential for studies of advanced vehicle dynamics studies. Integrating such models into the simulation framework significantly improves the performance and accuracy of vehicle dynamics assessments, facilitating run-time modifications of the physical tire model parameters. Similarly, the symbolic-numerical analysis and solution of structures using the DSM method represents another significant contribution. Such a method enables an efficient assembly and solution of large-scale structural systems, particularly useful in design optimization, parametric studies, and model reduction.

As a concluding remark, the symbolic-numerical mixed approach we developed in this thesis addresses the challenges of complex dynamic systems' simulation. Beyond vehicle dynamics, the methodologies and developed tools provide a solid foundation for practical applications in various fields of engineering, such as robotics and aerospace, as well as in scientific computing. To this end, the software packages supporting the methodologies and algorithms developed in this research are made available to the public as open-source software [23–30].

6.2 FUTURE WORK

The symbolic index reduction algorithm presented in this thesis is a solid foundation for future research in the field of symbolic computation. Nonetheless, future efforts will focus on enhancing the algorithm to handle more complicated DAE systems. Specifically, addressing expression swell mitigation techniques while guaranteeing the numerical stability of the reduced-index DAE systems is crucial for the optimization of the algorithm's performance. Therefore, future work can take several directions. Some of the most promising ones are outlined below.

DETECTION OF LINEAR INDEX-1 VARIABLES During the index reduction procedure of MB-type DAE systems, we noticed that the expression swell occurs in the last factorization step. This is due to the presence of index-1 variables that are linearly present in the system's equations, which are explicitly substituted in the DAE system by factorization. Notably, such variables correspond to the Lagrange multipliers. The detection and exploitation of this type of linear index-1 variables would have a relevant impact on the optimization of the algorithm, as it enables us to avoid the last factorization step, which is typically the most computationally expensive step in the algorithm.

SYSTEM AUGMENTATION One may also argue that veiling variables are linear index-1 variables. However, so far, we have only considered them as an auxiliary evaluation layer for the expressions generated during the index reduction procedure, and have not exploited them as state variables in the DAE system. This system augmentation technique can be exploited whenever a "complicated" pivot is detected in the factorization process. Specifically, if a complicated expression is detected in the pivot position, we could substitute it with a veiling variable that will also be added to the system as a state variable. This has the following advantages.

- The augmentation introduces a new equation to the system that grants us a 1 in the pivot position, which allows us to continue with the factorization process by keeping the complexity of the expressions at the minimum level.
- The symbolic complexity of the reduced-index DAE system is better mitigated, as the pivot is not a complicated expression, but a 1 in the diagonal of the matrix being factorized. Nonetheless, the overhead is relegated to the right-hand side of the system, which does not affect the numerical stability of the system.

Notably, finding consistent ICs for the augmented system would not get more complicated, as the newly introduced state variables would also be stored in the veiling variable vector. This allows us to find consistent ICs for the augmented system almost as straightforwardly as before. Hence, the veils add an evaluation layer to the system and, once the ICs for the original state variables are found, the veiling variables' ICs can be computed through the evaluation of their expressions at the initial time.

COMPUTER ALGEBRA SYSTEM The MAPLE® CAS is a powerful tool for symbolic computation. However, the current capabilities of its kernel do not allow for an effective handling of large expressions. This is why we resorted to the latest state-of-the-art symbolic computation techniques to handle exceedingly large expressions and to mitigate the computational complexity of the expressions generated during the index reduction procedure. However, in some cases, the MAPLE® symbolic computation kernel still struggles to perform the index reduction procedure. This is due to the CAS's sensitivity to the size and number of the generated expressions. Despite this being a normal behavior for CASs, it would be beneficial to explore the possibility of implementing the symbolic index reduction algorithm in a modern open-source CAS. This would allow us to have more control over the symbolic computation kernel while having an in-depth understanding of the CAS' internal design. As a consequence, we would be able to exploit the full potential of the symbolic index reduction algorithm and to optimize the algorithm's performance.

6.3 FORTHCOMING USES OF THE PROPOSED TECHNIQUE

In this thesis, we have only explored the capabilities of the proposed methodology in some application examples, which are mainly focused on the validation of the symbolic index reduction algorithm implemented in the INDIGO software package. However, the proposed methodology has a wider range of applications in real-world problems. Some of the most promising are outlined below.

OPTIMAL CONTROL DAEs found extensive application also in OCP [143–145]. The proposed methodology can be used in OCPs to reduce the index of the DAE system and to obtain an index-reduced formulation of the DAE system. This would allow us to apply standard numerical integration schemes to solve the OCP. Specifically, the index-0 or index-1 DAE system can be integrated as a standard ODE, while the hidden constraints can be enforced by introducing a quadratic penalty term in the cost function. Without doubt, *ad hoc* solution schemes can be developed to handle the hidden constraints more efficiently.
Another approach would be to exploit a common interface between ODE and DAE systems. In particular, we can provide the OCP solver with the derivatives of the state variables, and with the appropriate Jacobian matrices at any given time on the solution mesh. How the Jacobian matrices are computed will thereby be hidden from the solver, which will only use them to compute the OC input regardless of the underlying system's structure. This would allow us to solve the OCP by integrating DAE systems just as ODEs, while also enforcing invariants or hidden constraints and providing the Jacobian matrices at each time. Notice that additional investigations on the numerical stability of the reduced-index system and hidden constraints' preservation may be necessary to ensure the convergence of the OCP solver.

NONLINEAR OPTIMIZATION Similarly to Shmoylova, Gerhard, Postma, and Roche [58], nonlinear Programming (NLP) problems can also be tackled by using DAE systems. Specifically, if one formulates a NLP problem through a Lagrangian function, the Karush-Kuhn-Tucker (KKT) conditions would yield an index-2 DAE system. The proposed methodology can be used to reduce the index of the DAE system and to obtain an index-0 or index-1 DAE system. This would allow us to apply numerical integration schemes to solve the NLP problem.

APPENDIX A

A SMALL 3D GEOMETRY LIBRARY

In the past few decades, the simulation of both manned and unmanned vehicles has gained increasing significance. The demand for highly efficient RT simulators underscores the necessity for algorithms that are not only efficient but also accurate in modeling vehicle movements within a virtual environment. Typically, the virtual world comprises numerous basic geometric entities capable of colliding and adjusting accordingly. The ability to quickly solve basic geometric problems is one of the most important roles in this kind of simulation. The Acme library, previously introduced in Stocco and Bertolazzi [1], is built to efficiently perform simple operations on a large number of basic geometric entities. Specifically, the library is tailored to address specific HRT tire-ground contact geometry analysis. This chapter describes the implementation details of the Acme library, exploring its data types, features, and ease of use. The software is implemented in C++ and is freely available online under the BSD 2-Clause license. Online documentation includes descriptions of the C++ and Matlab® Mex Application Programming Interfaces (APIs), along with usage examples.

A.1 COMPUTATIONAL GEOMETRY IN REAL-TIME SIMULATIONS

Over the recent decades, there has been a notable shift in the automotive manufacturing sector towards prioritizing simulation. In particular, the advent of high-performance CPUs and Graphics Processing Units (GPUs) has intensified the focus on RT simulators. These sophisticated simulators, characterized by high fidelity and full integration, demand substantial computational capabilities. Moreover, specialized codes are essential to meet the dual requirements of HRT responsiveness and high accuracy within very limited time intervals. Indeed, the time step in driving simulators

is typically set at 1 ms, which is a trade-off to capture most of the typical frequencies in vehicle subsystems.
An important aspect of simulation lies in the vehicle-environment interaction. In driving simulators the virtual environment on which the vehicle moves is made from a multitude of basic geometric entities that can intersect and evolve. Consequently, the efficient resolution of simple geometric problems assumes a crucial role in achieving high accuracy and, by extension, a realistic simulation. Specifically, when working with numerous geometric objects, it is essential to partition the 3D space with an appropriate data structure. This structured partitioning facilitates efficient access to spatial objects. The absence of spatial partitioning would necessitate scanning the entire database during any search, resulting in a significant increase in processing time.
There is a multitude of geometric libraries already implemented and capable of solving complicated geometric problems, *e.g.*, mesh-mesh intersection, re-meshing, Delaunay triangulation, and so on. The sheer size of such libraries and their high complexity make them unsuitable for application in the simulation environment introduced earlier. The need to easily maintain and correct inefficiencies has led to the development of a new geometry library. In this chapter, we introduce a C++ library named Acme, designed to efficiently address the resolution of basic 3D geometric problems at high speed. The first version of Acme was tailor-made to perform HRT tire-ground contact geometry analysis, where geometrical objects and tire-ground intersection objects were initially coexistent in the same code. The desire to bring the library to the next level made it necessary to formalize and create a more effective framework. Consequently, all the geometrical algorithms are then collected in an independent library. But why create a new library even if there are plenty of alternatives available out there? The dynamic nature of the simulation field, with the continual introduction of new features, underscores the importance of maintaining a simple yet robust minimum core. This approach enables quick response to changes. Most of the available geometry libraries are either excessively large or overly complex for this specific purpose [146, 147]. Furthermore, we aim to reduce the dependencies by relying solely on the C++ Eigen template library, which is well-recognized for its efficiency with small vectors and matrices.

A.2 A NEW GEOMETRY LIBRARY

As previously mentioned, the software is implemented in C++, a widely used and high-performance object-oriented general-purpose programming language. Since its invention by Bjarne Stroustrup in 1985, C++ has undergone significant extensions and modifications. Therefore, we chose to develop our code based on the C++11 standard [148]. The adoption of the 2011 standard introduced notable improvements in the coding style, exemplified by the introduction of the new smart pointer classes, extensively utilized in the Acme library. The source code of the software is freely available online [26] and is released under the BSD 2-Clause license. The online documentation includes descriptions of the C++ and Matlab® Mex APIs, along with usage examples. Rig-

orous testing has been conducted on MacOS®, Linux®, and Windows® Operative Systems (OSs) to ensure the software's compatibility across diverse platforms.

A.2.1 DESIGN CHOICES

This software is neither intended as a black box nor as a Graphical User Interface (GUI) based application for end-users. Instead, it is designed as an easy-to-use set of C++ classes that provides a basic and reliable foundation, which can be extended by the developers according to their specific needs. The design of the software is grounded in the following principles.

DRIVEN BY ACTUAL NEEDS The implementation focuses on a stable minimum core library, including only features that are currently in use. This deliberate choice allows for progressive testing of the software and a less concerned third-party extension process.

BUILD ON THE STATE-OF-THE-ART EIGEN LIBRARY For a flexible and extensible framework, Acme is built on the Eigen template library for linear algebra [149]. Recognized for its efficiency with small vectors and matrices, Eigen is an apt choice in the field of computational geometry where matrices and vectors are typically of limited size. Additionally, Eigen can leverage Linear Algebra PACKage (LAPACK)/Basic Linear Algebra Subprograms (BLAS) [150] for peak performance when dealing with larger matrices and vectors. Relying on this well-tested and high-performing template library allows Acme to achieve high-performance levels while maintaining an elegant and expressive API.

AVOIDING MEMORY LEAKS Managing dynamically allocated memory is one of the most critical aspects of a low-level programming language like C++. Often, the most insidious errors are due to flaws in memory allocation and release policies, resulting in excessive use of resources (*memory leak*), or irreversible error conditions that undermine program stability (*access violation*). The usage of C++11 smart pointers in Acme significantly reduces the likelihood of these errors. Smart pointers, as part of the standard library utility classes, act as wrappers for raw pointers, offering transparent memory release policies suitable for various use cases. Notably, SharedPointer objects retain shared ownership, allowing multiple objects to own the same instance. The object is only destroyed and its memory deallocated when either the last SharedPointer owning it is destroyed or reassigned to another SharedPointer. Additionally, the object can be destroyed using the `delete` expression or a custom `delete` expression.

POLYMORPHIC BEHAVIOR Acme capitalizes on C++ polymorphism as a fundamental design pattern. This polymorphic behavior greatly simplifies the management

of heterogeneous objects that share a common interface of geometric entities. Notably, the same C++ polymorphic behavior is also present in the MATLAB® MEX wrapper.

HIGH-QUALITY DOCUMENTATION Comprehensive documentation is available on the provided website, encompassing both the C++ and MATLAB® MEX APIs, along with examples. The documentation is generated using a combination of DOXYGEN and SPHINX. DOXYGEN processes annotated C++ sources to create documentation, while SPHINX enhances the graphical quality of the generated HTML code, providing a more visually appealing and graphically rich design.

A.2.2 DATA TYPES

ACME supports a limited number of geometrical entities, carefully chosen to maintain the library's essential nature for efficiency and easy maintenance. The chosen classes specifically describe and manipulate virtual ground surfaces and tires. While the library is intentionally kept minimal, it is extensible according to the needs of end-users. The geometric entities are systematically organized into classes, each being within the ACME namespace and publicly inheriting from the virtual superclass ENTITY. The derived classes, representing the homonyms geometric entities are POINT, LINE, RAY, PLANE, SEGMENT, TRIANGLE, DISK, and BALL, are integral components of the library. In Figure A.1, a representation of all ACME basic ENTITY objects is shown. A concise mathematical description of each data type in the software follows.

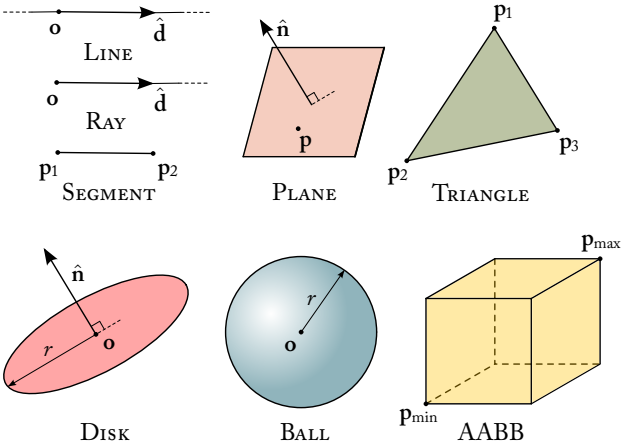


Figure A.1: Representation of all ACME basic ENTITY objects.

POINT In the 3D Euclidean space, a point represents an exact location. A point $\mathbf{p} \in \mathbb{R}^3$ is represented by an ordered triplet of coordinates

$$\mathbf{p} = [x, y, z]^\top \ .$$

The POINT class is built through public inheritance from the virtual class ENTITY and the EIGEN::MATRIXBASE template class. It is worth noting that in many C++ libraries, vectors and points are often described by the same class. However, in ACME, we have provided a clear way to distinguish them. This distinction is evident in the inheritance structure, where the POINT class inherits publicly from the ENTITY class. On the other hand, the inheritance of the MATRIXBASE template class makes it possible to easily build mathematical vectors out of point entities and vice versa. In our software, both vectors and points are represented by column sets of elements.

LINE A line ℓ is defined by an origin point $\mathbf{o}$ and a unit direction vector $\hat{\mathbf{d}}$, such that the line corresponds to the set

$$\ell(\mathbf{o}, \hat{\mathbf{d}}) = \left\{ \mathbf{o} + \hat{\mathbf{d}}t \;\middle|\; t \in \mathbb{R} \right\} .$$

RAY A ray ϱ is defined by an origin point $\mathbf{o}$ and a unit direction vector $\hat{\mathbf{d}}$, such that the ray corresponds to the set

$$\varrho(\mathbf{o}, \hat{\mathbf{d}}) = \left\{ \mathbf{o} + \hat{\mathbf{d}}t \;\middle|\; t \in \mathbb{R}_{\geq 0} \right\} .$$

PLANE A plane π is defined by a generic point on the plane $\mathbf{p}$ and a unit normal vector $\hat{\mathbf{n}}$, such that the plane corresponds to the set

$$\pi(\mathbf{p}, \hat{\mathbf{n}}) = \left\{ \hat{\mathbf{n}} \cdot (\mathbf{p} - [x, y, z]^\top) = 0 \;\middle|\; [x, y, z]^\top \in \mathbb{R}^3 \right\} .$$

SEGMENT A segment σ is defined by two points $\mathbf{p}_1$ and $\mathbf{p}_2$, such that the segment corresponds to the set

$$\sigma(\mathbf{p}_1, \mathbf{p}_2) = \left\{ \mathbf{p}_1 + (\mathbf{p}_2 - \mathbf{p}_1)\, t \;\middle|\; t \in \mathbb{R},\ 0 \leq t \leq 1 \right\} .$$

TRIANGLE A triangle τ is defined by three points $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, such that the triangle corresponds to the set

$$\tau(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = \left\{ t_1\mathbf{p}_1 + t_2\mathbf{p}_2 + t_3\mathbf{p}_3 \;\middle|\; t_1, t_2, t_3 \in \mathbb{R}_{\geq 0},\ t_1 + t_2 + t_3 \leq 1 \right\} .$$

DISK A disk ϕ is defined by a radius r, a center point $\mathbf{o}$, and a unit normal vector to the disk face $\hat{\mathbf{n}}$. Equivalently, using the same notation for the center point and the unit normal vector to the face, a disk can be defined by a radius r and a laying plane $\pi(\mathbf{o}, \hat{\mathbf{n}})$. In both cases, the disk corresponds to the set

$$\phi(r, \mathbf{o}, \hat{\mathbf{n}}) = \phi(r, \pi(\mathbf{o}, \hat{\mathbf{n}})) = \left\{ \left\| \mathbf{o} - [x, y, z]^\top \right\|_2^2 \leq r^2 \;\middle|\; [x, y, z]^\top \in \pi(\mathbf{o}, \hat{\mathbf{n}}) \right\} .$$

BALL A ball ω is defined by a radius r and a center point $\mathbf{o}$, such that it corresponds to the set

$$\omega(r, \mathbf{o}) = \left\{ \left\| \mathbf{o} - [x, y, z]^\top \right\|_2^2 \leq r^2 \mid [x, y, z]^\top \in \mathbb{R}^3 \right\} .$$

A.2.3 MESH TOOLS

In addition to the fundamental data types presented, we also provide other classes that are useful in scenarios involving mesh or manipulation of large numbers of entities. These objects include Collection, Axis-Aligned Bounding Box (AABB), and AABBtree.

COLLECTION The Collection object consists of a vector of SharedPointer to Entity type objects. This class can be used when a substantial number of Entity object instances need to be grouped into a single object. The grouping, coupled with the usage of SharedPointer objects, facilitates effective data manipulation and ensures safe memory management. Nonetheless, the Collection object is not a geometric entity and does not have any geometric meaning. It is merely a container for Entity objects that can be used to perform operations on a large number of Entity objects simultaneously. In Figure A.2, an example of two Collection objects bounded in two different AABBs is reported.

AABB An AABB β is defined by a maximum point $\mathbf{p}_{\max}$ and a minimum point $\mathbf{p}_{\min}$, which are respectively equal to

$$\mathbf{p}_{\max} = [x_{\max}, y_{\max}, z_{\max}]^\top \quad \text{and} \quad \mathbf{p}_{\min} = [x_{\min}, y_{\min}, z_{\min}]^\top .$$

The AABB corresponds to the set

$$\beta(\mathbf{p}_{\max}, \mathbf{p}_{\min}) = \left\{ [x, y, z]^\top \in \mathbb{R}^3 \;\middle|\; \begin{array}{c} x_{\min} \leq x \leq x_{\max} \\ y_{\min} \leq y \leq y_{\max} \\ z_{\min} \leq z \leq z_{\max} \end{array} \right\} .$$

Indeed, this type of geometrical entity is very simple, requiring only two Point objects to fully describe the space it occupies. Furthermore, the algorithms involved in AABB collision detection and/or intersection are highly efficient. Specifically, the basic algorithm for AABB-AABB collision detection can be executed solely through two-way comparison operators, making it lightweight and fast to perform.

AABBTREE There are plenty of possible tree structures. Some of them are suitable for a more rough spatial description with low computational complexity, while others are suitable for accurate spatial indexing but carry high computational complexity. In the Acme library, the AABB tree is chosen due to its balanced complexity-performance ratio, making it effective for RT applications. The performance of a

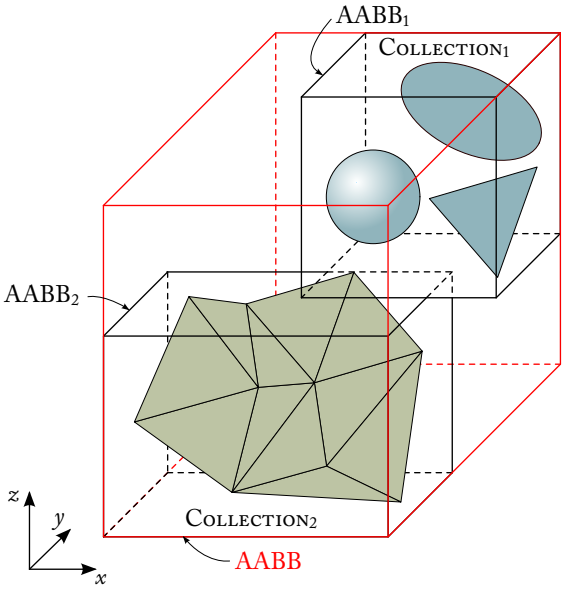


Figure A.2: Example Collection objects bounded in two different AABBs. The two AABB objects are then bounded in a master AABB depicted in red.

generic Bounding Volume Hierarchy (BVH) is generally measured by the computation time required to solve an intersection query. To enhance the BVH performance and consequently reduce the number of comparisons among pairs of Bounding Volume Primitives (BVPs), a BVH should be as compact as possible, minimizing the bounding volume contained in each BVP [151, 152]. Several techniques can be employed to build an AABB tree, with the most common being the *top-down* and *bottom-up* strategies. While the *top-down* strategy allows to easily perform the tree construction [151–153], the *bottom-up* approach usually achieves more compact trees and better performances, albeit being more intricate to construct [151, 154]. The typical average computational complexity of tree construction is $\mathcal{O}(n \log n)$, while the intersection of two AABB trees has an average cost of $\mathcal{O}(m \log n)$, where n and m are the numbers of BVPs of the two trees. If one intends to intersect two sets of BVPs without the use of the AABB tree the computational cost is always $\mathcal{O}(nm)$, as each element of the first set must be compared with all the elements of the second set [155]. Notably, The AABBtree implemented in the Acme library is directly derived from the one presented in [156, 157] and has been extended from the 2D to the 3D case (please refer to [1] for a detailed description of the AABBtree implementation).

A.2.4 BASIC INTERSECTION ALGORITHMS

Specific algorithms for basic intersections are not discussed here for the sake of brevity. It is important to note that comprehensive sources for intersection testing are limited. Exceptions include [158] and [159], which serve as extensive collections of geometric tests of various types. References [153] and [160] are equally valuable, although not as exhaustive. Individual articles on specific tests can also be found in the five-volume *Graphic Gems* series [161–165].

A.2.5 SOFTWARE FUNCTIONALITIES

The Acme geometry library consists of a C++ core and a Matlab® Mex wrapper. The library is built to efficiently *create*, *intersect* and *destroy* basic geometry entities objects. It is possible to check geometrical conditions between objects, like *parallelism*, *orthogonality*, *collinearity* and *coplanarity*. The intersections that can be performed with the Acme library are limited to those that potentially return a *single* Acme::Entity object. For example, the intersection of a coplanar disk and a triangle may potentially return a circular arc and two segments, making it unsuitable for direct execution through the Acme library. The sets of geometrical condition tests and intersections that can be performed are summarized in Tables A.1 and A.2 respectively.

Geometrical intersection tests	Point	Line	Ray	Plane	Segment	Triangle	Disk	Ball
Parallelism	–	●	●	●	●	●	●	–
Orthogonality	–	●	●	●	●	●	●	–
Collinearity	–	●	●	–	–	–	–	–
Coplanarity	–	●	●	●	●	●	●	–

Table A.1: Geometrical conditions tests that can be performed through Acme library. *Legend*: ● available test, and – not available test.

Thanks to the Matlab® Mex, objects can also be manipulated and *visualized* in the Matlab® environment. An interesting feature of the Matlab® Mex is that it preserves C++ polymorphism. In other words, when performing an intersection between two generic objects, both in C++ and in Matlab®, the software outputs the exact data type of the entity resulting from the intersection, maintaining all checks and verifications transparent to the end-user.
Table A.3 presents a comparison of timing performances between the CGAL and Acme libraries in a C++ environment. As evident from the results, there is a notable increase in speed. This could be attributed to the greater complexity of the CGAL

Geometrical intersection tests	Point	Line	Ray	Plane	Segment	Triangle	Disk	Ball
Point	•	•	•	•	•	•	•	•
Line	•	•	•	•	•	•	•	•
Ray	•	•	•	•	•	•	•	•
Plane	•	•	•	•	•	•	•	•
Segment	•	•	•	•	•	•	•	•
Triangle	•	•	•	•	•	○	○	–
Disk	•	•	•	•	•	○	○	–
Ball	•	•	•	•	•	–	–	–

Table A.2: Geometrical intersection tests that can be performed through Acme library. *Legend*: • intersection can be always performed, ○ intersection can be performed only if entities are not coplanar, and – intersection can not be performed.

library, which, in addition to having a much more intricate and comprehensive framework than Acme, likely carries out additional checks or dynamic allocations on the objects in use.

A.3 A STEP-BY-STEP EXAMPLE

We now present an example that illustrates some capabilities of the Acme library. Specifically, the same example will be presented in both the C++ language and the Matlab® environment, allowing us to understand the few differences between the two working environments. In the following C++ and Matlab® code snippets, we will create the Disk objects

$$\phi_1(r_1, \mathbf{o}_1, \hat{\mathbf{n}}_1) = \phi_1(2,\ [0,0,0]^\top, [0,1,0]^\top),$$

and

$$\phi_2(r_2, \mathbf{o}_2, \hat{\mathbf{n}}_2) = \phi_2(1,\ [0,0,0]^\top, [1,1,0]^\top),$$

that will be indicated by the variables `d1` and `d2`, respectively. Then, we will then intersect them, obtaining a geometric entity whose type is unknown to us. Subsequently, we will use the `type()` method to identify and print a string describing the type of the obtained Entity object. In both cases, the output will be the string "`segment`". Finally, we will plot the obtained Entity object in a Matlab® figure. The C++ and Matlab® code snippets are reported in the following. The visualization of the example problem is reported in Figure A.3, which is obtained through the last 6 lines of the Matlab® example code.

Intersected entities	CGAL		Acme		Speed-up
	μ (ns)	σ^2 (ns^2)	μ (ns)	σ^2 (ns^2)	($\times$)
Line-Line	18.3	0.291	2.2	0.0132	8.3
Ray-Ray	1030	0.739	8.5	0.0974	121
Segment-Segment	1050	1.05	8	0.138	131
Triangle-Triangle	2920	3.83	24	0.454	121
Line-Ray	13	0.0268	1.9	0.0164	6.8
Line-Segment	17	0.104	6.9	0.192	2.4
Line-Triangle	11.3	0.0228	5.6	0.0526	2
Ray-Triangle	11.6	2.28	25.5	0.858	-0.45
Segment-Triangle	15	0.568	13.8	0.247	1.1

Table A.3: Timing performance comparison between CGAL and Acme libraries. The test consists of 10^5 intersections between randomly created objects. Notice that intersections are only made between types of geometric entities common to the two libraries. *Legend*: μ average intersection run-time, and σ^2 intersection run-time variance.

C++

```
#include "acme.hh"
using namespace acme;
using namespace std;

int main(void){
  // Create the disks
  entity *d1 = new disk(
    2, point(0,0,0), vec3(0,1,0)
  );
  entity *d2 = new disk(
    1, point(0,0,0), vec3(1,1,0)
  );

  // Perform the intersection
  entity *e1 = intersection(d1,d2);

  // Check output entity type
  cout << e1->type() << endl;
  return 0;
}
```

Matlab®

```
% Create the disks
d1 = acme_disk( ...
  2, [0,0,0]', [0,1,0]' ...
);
d2 = acme_disk( ...
  1, [0,0,0]', [1,1,0]' ...
);

% Perform the intersection
e1 = d1.intersection(d2);

% Check output entity type
disp(e1.type());

% Plot output
f1 = figure;
grid on; grid minor;
d1.plot(f1, 'red');
d2.plot(f1, 'blue');
e1.plot(f1, 'green');
```

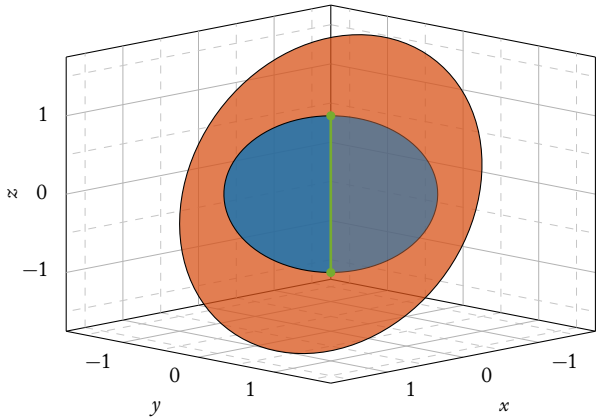


Figure A.3: Visualization of the example problem, which is obtained through the last 6 lines of the MATLAB® example code.

APPENDIX B

TIRE-GROUND ENVELOPING MODELING

As previously introduced in Appendix A, simulation has become vital in vehicle development and virtual testing, especially for autonomous vehicles. High-performance HRT simulators, which are crucial for those undertakings, require efficient algorithms to accurately model vehicle behavior within virtual environments. A prime example is tire-ground contact modeling, which is pivotal if we aim to achieve a high level of realism when simulating wheeled vehicles. Contact modeling focuses on an accurate estimation of the parameters needed to compute the forces and torques generated by vehicle-ground interaction. However, the complexity of this task is compounded by the fact that tire-ground contact is a highly nonlinear phenomenon, which is further exacerbated by the need to perform tests to fine-tune state-of-the-art tire-ground contact models. To tackle those challenges, we have developed a novel enveloping model that does not require any fitting of experimental data and is based on the 3D geometry of the intersection between undeformed volumes. In this appendix, we provide a detailed description of the algorithm's formulation, the current software implementation, which is available as a BSD 3-Clause Library under the name Enve, as well as the achieved scalability and RT performance.

B.1 INTRODUCTION TO TIRE-GROUND CONTACT MODELING

From the 1950s onwards, vehicle simulation has been crucial for evaluating ride and handling. More recently, as we shift towards autonomous vehicles, there's a growing need for high-quality simulations to develop and test them in various scenarios. Real-world testing is time-consuming and costly, making driving simulators essential for accelerating the transition to autonomous driving, enhancing safety, and reducing ex-

penses. Beyond the hardware, a simulator's essence lies in its software, responsible for shaping the virtual environment based on physics. Aside from computer graphics, this software comprises multiple subprograms, including a Multi-Body System (MBS) – used to integrate the MB car system – and several other subprograms that allow the MB system to interact with the environment. In the case of a road vehicle, interactions with the environment primarily occur through tire-ground contact, aerodynamic forces, and, less frequently, through collisions with external objects. To obtain a sufficiently realistic simulation of the vehicle it is therefore of crucial importance to correctly estimate the forces and torques generated by the tire-ground contact [166]. In order to be able to calculate the stress generated by the contact, the first step is to estimate the ground rut pose with respect to the tire reference frame [167]. An incorrect estimate of the *local contact plane* or *surface* would inevitably lead to unrealistic or unreliable simulations. Since tire-ground contact evaluation is one of the most time-consuming processes in MB simulations, algorithms designed specifically for this purpose are needed to ensure high efficiency and scalability. Specifically, in the case of Driver In the Loop (DIL), Hardware In the Loop (HIL), and/or Software In the Loop (SIL) simulations, this software must guarantee a capability that is at least as fast as RT. It is also desirable to obtain a faster than RT capability, in order to speed up costly offline simulations and to allow for the use of less powerful machines. As a matter of fact, the development of faster and at the same time more accurate contact models still remains an open research topic [168, 169].

From the scientific literature, it is known that appropriate contact methods must be applied when road unevenness is characterized by wavelengths 2–3 times smaller than the contact patch length. This occurs when riding is done either over cobblestone or Belgian block roads, or simply over road surfaces that present sharp obstacles, *e.g.*, cleats, bumps or potholes [166]. The role of the tire-ground contact model is to represent the excitation of pneumatic tires caused by uneven terrain surfaces. Although the term "uneven ground surface" comprehends any kind of unevenness, attention is mainly paid to short irregularities, as tire deformation is particularly important in those instances. The ability of the tire to deform, *i.e.*, to follow the ground surface shape, is defined as the *enveloping* property of the tire-ground contact. In specific, the enveloping is the *quasi-static* component of the tire-ground contact phenomenon, whereas the *dynamic* component comes from the carcass flexibility and is implemented in the tire force-calculation model [166, 170]. The enveloping property depends on the tire geometry and structure, and can be represented through *physical* models such as those presented in [171–175], or (*semi-*)*empirical* models such as [170, 176]. In contrast to physical models, the empirical models try to describe the contact through much simpler models designed to mimic the behavior of the tire-ground interaction.

In the early days of vehicle simulation, studies on tire behavior have mostly been limited to the case of flat and asperity-free contact surfaces, focusing on the deformation and dynamics of the tire carcass induced by the contact forces. The importance of accurately describing the local contact area has been highlighted in [167, 177], where the influence of obstacles on the overall tire behavior is studied. While the tire is riding over 3D un-

even road surfaces the local road plane shows variations in height, and in forward and banking angles. The banking slope angle gives rise to the so-called tire camber thrust, which must be taken into account in order to properly simulate the force generated by the tire. Since then, many enveloping models for arbitrarily uneven surfaces have been developed. Most of the models developed so far are based on heuristic concepts. Without a doubt, the most famous enveloping model is certainly the SWIFT® model, which allows the MagicFormulae model to be used even on rough road surfaces [170]. Simple and parameterless contact models like that used in TMEASY [176, 178] can be applied, but do not always guarantee adequate enveloping properties, which can cause sudden and unrealistic variations of the banking and slope angles in the proximity of sharp cleats or asperities. Another family of more sophisticated enveloping models is based on the physical concept of radial and radial-interradial spring tire [171–173]. Models based on the tire radial-spring behavior are a good compromise between computational effort, robustness and physical consistency. Unlike the SWIFT® model, the radial spring tire models do not show any working range limit in shape, magnitude and orientation of the incoming road obstacles [171]. For this reason, the results are physically consistent even in the case of high cleats. The state-of-the-art approach is referred to as "full-physical tire modeling". In particular, the commercial software FTIRE® [179] and CDTIRE® [168, 180] are the leading virtual tire simulation models in this category. They are proven to be multi-purpose physics-based tire models that are able to simulate nearly all tire dynamics phenomena. They combine a discrete elements description of both belt and sidewalls elements with a brush type contact, and RT capabilities [181]. In terms of enveloping behavior, they are also providing physically consistent and extremely realistic results [180, 182]. Unfortunately, little information on the actual computational time performance is available for both FTIRE® and CDTIRE®.

A very simple physical contact modeling approach we have not yet mentioned before relies on the geometry of intersection between undeformed regions, also known as displaced area models [174]. This kind of model is derived from the physics of the linear radial-spring tire model. As stated also in [171] a common approach for modeling the 2D vehicle-road contact is to assume that the tire is described by a disk moving on a ground region. Indeed, if we assume that the tire is represented by an infinite number of independent linear radial springs, the force resulting from the contact is linearly proportional to the intersection area and acts along the line passing through the centroid of the intersection region and the wheel center. Despite the simplicity of this approach and the limits that arise from neglecting the tire carcass being a unique cohesive body capable of transmitting shear forces, it provides a good approximation of the contact information. Models based on the geometry of intersection between undeformed regions are already available, but they all lack a full 3D description of the contact. This limits the comparison between other enveloping models to 2D contact scenarios. Moreover, no robust software implementation is freely available and can be used to determine the tire contact point and normal or equivalently a contact plane.

This appendix aims to present a software-based algorithm (hereafter called ENVE) to

model the 3D geometrical contact between tire and ground, represented respectively as a generic axial-symmetric surface and a triangular mesh. In particular, the adopted methodology is derived from the physics of the linear radial-spring tire model and is based on the geometry of the intersection between undeformed regions. Indeed, ENVE is an enveloping model that is fully parameterizable by the tire's external shape alone and does not require any fitting on experimental data. As will be shown in the following, the proposed approach achieves an efficient and robust modeling of the tire-ground geometrical contact with scalable precision. This model is intended to be coupled with a tire model, *i.e.*, empirical or physical tire model, with or without belt and sidewall dynamics. The tire model calculates tire-ground forces based on geometric contact parameters supplied by the ENVE software. In this study, we demonstrate the efficiency and efficacy of the software by coupling ENVE with the MAGIC FORMULA 6.2 tire model.

Remark 1. *In this appendix, the focus will only be on the enveloping properties of the tire-ground contact,* i.e., *the calculation of the so-called effective road surface. To evaluate dynamic response, one can employ a rigid ring model coupled with a tire model for force calculation, as exemplified by* MAGIC FORMULA-SWIFT® *[166, 170]. In this appendix, our aim is to determine the application point and direction of contact forces, and as such, we do not delve into the modeling of tire-ground contact forces or the tire carcass dynamics.*

This appendix has two main goals. The first is to provide a methodology for the analysis of the geometrical tire-ground contact, which takes into account the geometry of the tire cross-section and of the road surface to compute information needed by the tire model to obtain forces. The proposed enveloping model offers advantages over SWIFT® in terms of algorithm simplicity and scalability. It excels in computing contact on multiple sections, making it suitable for supporting physical-based tire models like the brush model. Additionally, it does not rely on independently identified parameters, further enhancing its effectiveness and immediacy of use. These characteristics make it suitable for HRT applications where computational efficiency is the main concern without diminishing the level of accuracy. The second is to evaluate the impact of this new approach coupled with a tire model in an advanced simulation environment with HRT scheduling. The software, which is also called ENVE, is built on the ACME geometry library (see Appendix A or Stocco and Bertolazzi [1]), and is designed to be integrated into vehicle dynamics MB kernel codes as well as into simpler simulation environments [20, 21].

B.2 TIRE-GROUND ENVELOPING MODEL

Let us consider a reference frame $\mathbf{H}xyz$ with unit vectors $(\hat{\mathbf{h}}_x, \hat{\mathbf{h}}_y, \hat{\mathbf{h}}_z)$, whose origin $\mathbf{H}$ is located in the wheel hub (see Figure B.1). The axes are oriented according to the ISO 8855:2011 standard [183]. The x-axis is directed towards the longitudinal direction of motion, the z-axis points upwards and the y-axis is oriented in such a way that

the coordinate system is right-handed. The tire is defined geometrically as an axially symmetric (around the axis $\hat{\mathbf{h}}_y$), convex and closed set $\mathcal{T}$

$$\mathcal{T} = \left\{ [x, y, z]^{\mathrm{T}} \in \mathbb{R}^3,\ y \in [y_l, y_r],\ R(y) : \mathbb{R} \mapsto \mathbb{R}_{\geq 0} \mid \sqrt{x^2 + z^2} \leq R(y) \right\},$$

whose interior and boundary are respectively denoted with $\mathring{\mathcal{T}}$ and $\partial\mathcal{T}$. The function $R(y)$ can be chosen arbitrarily to properly reconstruct the tire's external morphology. On the other hand, the ground is modeled as a closed half-space $\mathcal{G}$

$$\mathcal{G} = \left\{ [x, y, z]^{\mathrm{T}} \in \mathbb{R}^3 \mid z \leq g(x, y),\ g(x, y) : \mathbb{R}^2 \mapsto \mathbb{R},\ f(x, y) : \mathbb{R}^2 \mapsto \mathbb{R}_{\geq 0} \right\},$$

whose interior and boundary are respectively denoted with $\mathring{\mathcal{G}}$ and $\partial\mathcal{G}$. The map $g(x, y)$ is a mapping between $\mathbb{R}^2$ to $\mathbb{R}$ that defines the height of the track surface. Similarly, the map $f(x, y)$ defines the local friction coefficient scaling factor, which is used to scale the tire performance according to the local ground surface conditions. Notice that the concept behind the friction coefficient scaling factor can be extended to other relevant local properties of the ground surface, *e.g.*, temperature, water film thickness, and asphalt wear.

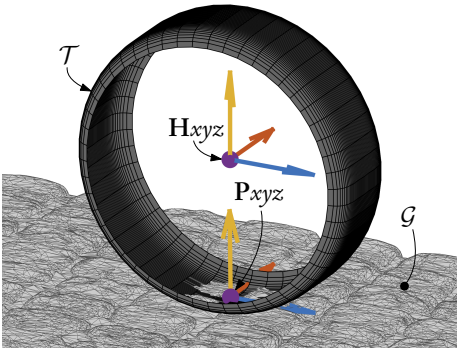


Figure B.1: Representation of the intersection between the tire $\mathcal{T}$ and a Belgian block road boundary $\mathcal{G}$ alongside the hub $\mathbf{H}xyz$ and the contact point $\mathbf{P}xyz$ reference frames.

To identify and quantify the contact between the tire and the ground we need to define what the intersection volume and the contact patch regions are. The intersection region is a closed set $\mathcal{V}$, collecting all the points of the tire $\mathcal{T}$, which are also points of the ground set $\mathcal{G}$, *i.e.*, $\mathcal{V} = \mathcal{T} \cap \mathcal{G}$ (see Figure B.2). On the other hand, the contact patch is defined as a closed set $\mathcal{P}$ that collects all the points of the tire which are also interior points of the ground boundary surface $\partial\mathcal{G}$, *i.e.*, $\mathcal{P} = \mathcal{T} \cap \partial\mathcal{G}$. The contact area A and the contact volume V are defined respectively as

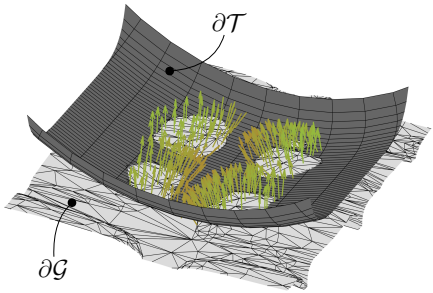


Figure B.2: Representation of the ground boundary $\partial\mathcal{G}$ normal directions (green arrows) inside the tire set boundary $\partial\mathcal{T}$. Each of these directions, together with other information, is exploited to calculate the contact point $\mathbf{P}$ and normal $\hat{\mathbf{n}}$.

$$A = \iint_{\mathcal{P}} 1\,\mathrm{d}\mathbf{x}\,, \qquad \text{(B.1)} \qquad \text{and} \qquad V = \iiint_{\mathcal{V}} 1\,\mathrm{d}\mathbf{x}\,. \qquad \text{(B.2)}$$

The tire is assumed to be modeled by an infinite number of *independent linear radial-springs*, whose stiffness k is uniform all over the tire section. The inaccuracy introduced by this assumption is recovered by the tire force-calculation model, which employs concentrated parameters to describe the non-uniformity of the carcass stiffness. Before calculating the direction $\hat{\mathbf{n}}$ and application point $\mathbf{P}$ of the resultant contact force, we have to make three important observations.

Observation 1. *The force generated by the contact is produced by the deformation of the radial springs. If the terrain is non-deformable the contact force magnitude* $\|\vec{\mathbf{F}}\|$ *is linearly proportional to the intersection volume* V, i.e.,

$$\|\vec{\mathbf{F}}\| = \iiint_{\mathcal{V}} k\,\mathrm{d}\mathbf{x} = kV \propto V.$$

Notice that the proportionality constant is the stiffness coefficient k, which is a function of the tire's internal structure and pressure. However, these dependencies are considered by the tire force-calculation model through lumped parameters. It should be pointed out that the damping effect is not considered due to the quasi-static nature of the enveloping phenomenon.

Observation 2. *Since the tire is assumed to be modeled by an infinite number of independent radial springs, the contact force* $\vec{\mathbf{F}}$ *can not produce any rolling resistance torque around the wheel hub axis* $\hat{\mathbf{h}}_y$. *In other words, the contact force acts along the direction determined by one of the families of lines passing through the contact point* $\mathbf{P}$ *and the hub axis* $\hat{\mathbf{h}}_y$.

It is also important to note that the rolling resistance is not considered in the enveloping models. It is typically considered to be part of the tire contact force model, which

must be fed with the information provided by the enveloping model. For instance, the SWIFT® model is coupled with the MagicFormulae model to compute the contact forces and torques generated by the tire-ground contact.
Thanks to these observations, the duality between the physical and geometrical models of the tire-ground contact can be exploited. Hence, the contact point $\mathbf{P}$ is calculated as the weighted average of contact surface points. The weight is proportional to the infinitesimal radial volume interested in the tire-ground contact. This infinitesimal radial volume is calculated as the intersection of the contact volume with the line $\ell(\mathbf{x}, \hat{\mathbf{h}}_y)$ passing through the point $\mathbf{x} = [x,\ y,\ x]^{\mathrm{T}}$ and matching the axes of the wheel $\hat{\mathbf{h}}_y$ orthogonally. If $\mathrm{proj}(\mathbf{x}, \partial\mathcal{G})$ defines the intersection point of the line $\ell(\mathbf{x}, \hat{\mathbf{h}}_y)$ with $\partial\mathcal{G}$, then $\mathbf{P}$ is calculated as

$$\mathbf{P} = \frac{1}{V} \iiint_{\mathcal{V}} \mathrm{proj}(\mathbf{x}, \partial\mathcal{G})\, \mathrm{d}\mathbf{x}\,. \tag{B.3}$$

The application direction of the resultant tire-ground contact force is denoted by $\hat{\mathbf{n}}$. It is calculated as the average of the directions $\hat{\mathbf{d}}(\mathbf{x}, \hat{\mathbf{h}}_y)$ of the lines $\ell(\mathbf{x}, \hat{\mathbf{h}}_y)$, *i.e.*,

$$\vec{\mathbf{n}} = \iiint_{\mathcal{V}} \hat{\mathbf{d}}(\mathbf{x}, \hat{\mathbf{h}}_y)\, \mathrm{d}\mathbf{x}, \quad \text{where} \quad \hat{\mathbf{n}} = \frac{\vec{\mathbf{n}}}{\|\vec{\mathbf{n}}\|}\,. \tag{B.4}$$

Similarly, the overall tire friction coefficient scaling factor λ is calculated as

$$\lambda = \frac{1}{V} \iiint_{\mathcal{V}} f(\mathbf{x})\, \mathrm{d}\mathbf{x}\,. \tag{B.5}$$

The solution of the here defined integrals (B.3), (B.4), and (B.5) is not straightforward. This is why, in the next section, we propose a numerical approximation of these integrals.
It should be pointed out that the local ground plane is hereafter identified by the reference frame $\mathbf{P}xyz$ whose origin point is $\mathbf{P}$ and axes directions are

$$\hat{\mathbf{e}}_z = \hat{\mathbf{n}}\,, \quad \hat{\mathbf{e}}_x = \frac{\hat{\mathbf{h}}_y \times \hat{\mathbf{n}}}{\|\hat{\mathbf{h}}_y \times \hat{\mathbf{n}}\|}\,, \quad \text{and} \quad \hat{\mathbf{e}}_y = \hat{\mathbf{n}} \times \hat{\mathbf{e}}_x\,.$$

The β_x and β_y angles, which are used to describe the local ground surface orientation with respect to the wheel hub, are calculated as the Euler angles (zxy sequence) between the hub reference frame $\mathbf{H}xyz$ and the contact point reference frame $\mathbf{P}xyz$ (see Figure B.3).

B.3 ALGORITHM IMPLEMENTATION

So far, we have not yet described the actual shapes of the sets $\mathcal{T}$ and $\mathcal{G}$, which represent the tire and the ground sets respectively. Since we will deal with complicated obstacle shapes it is important to restate the presented enveloping model in a formulation that does not limit its numerical efficiency, robustness, and scalability. For this reason, a

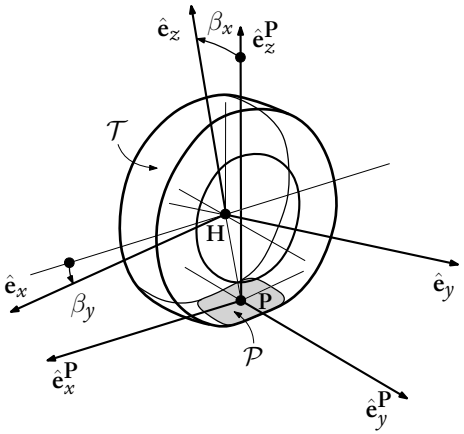


Figure B.3: Representation of the contact pose **P***xyz* along with the hub reference frame **H***xyz* according to the ISO standard [183]. The constant pose is defined by the contact point **P** and the normal $\hat{\mathbf{e}}_z = \hat{\mathbf{n}}$, which also allows us to identify the forward and banking angles β_x and β_y.

discretized representation of tire and ground elements is required both to eventually solve the contact problem and to offer a highly scalable framework to work with. More specifically, the tire's external shape roughly corresponds to the external tread pattern profile revolved around the hub axis $\hat{\mathbf{h}}_y$. The tire set is discretized into a series of laterally lumped *disks*, also called *ribs*. On the other hand, the ground boundary is represented by a triangular mesh (see Figure B.4). Both tire and ground descriptions are fully scalable as the density of the ribs and the triangles can be adjusted according to the accuracy and execution speed needed for the specific simulation. This representation approach turns out to be useful in the case of HRT simulation (where the execution speed represents an important requirement that needs to be satisfied) but also in the case of offline simulations (where the demanded precision is usually higher).
The representation of tire and ground allows us to restate the enveloping model presented in Section B.2 in a more "software-friendly" formulation. We will consider the discretized tire $\mathcal{S}$ as a set of n tire ribs ϕ, *i.e.*, $\mathcal{S} = \{\phi_1, \phi_2, \ldots \phi_{n-1}, \phi_n\}$. The generic tire rib ϕ is defined geometrically as a disk of radius r, whose center is $\mathbf{O}$, and whose face normal corresponds to $\hat{\mathbf{h}}_y$. In addition, the rib also retains its "virtual" width $w \in \mathbb{R}_{\geq 0}$, which will be used in the numerical integration processes

$$\begin{aligned} \phi = \Big\{ & w \in \mathbb{R}_{\geq 0},\ \mathbf{x} = [x, y, z]^{\mathrm{T}} \mid \mathbf{O} = [0,\ y,\ 0]^{\mathrm{T}}, \ldots \\ & r = R(y),\ \|\mathbf{O} - \mathbf{x}\| \leq r,\ \hat{\mathbf{h}}_y \cdot (\mathbf{O} - \mathbf{x}) = 0 \Big\}. \end{aligned} \tag{B.6}$$

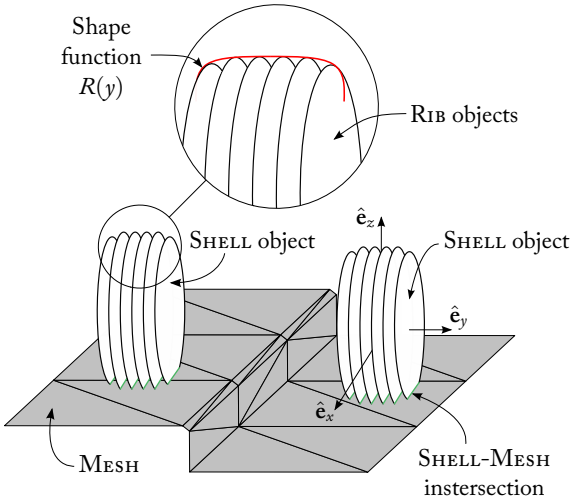


Figure B.4: Tire-road intersection with multiple instances of Shell objects. Notice that the Mesh is modeled as a multitude of TriangleGround objects, while the Shell object is made of a set of Ribs. The Ribs diameters and positions are extracted directly from the custom function $R(y)$ defined in the Shape object.

Similarly, the discretized ground boundary $\partial\mathcal{G}$ is represented through a triangular mesh set $\mathcal{M}$ of m triangles τ, *i.e.*, $\mathcal{M} = \{\tau_1, \tau_2, \ldots, \tau_{m-1}, \tau_m\}$, where

$$\tau = \Big\{ \lambda \in \mathbb{R}_{\geq 0},\ \mathbf{x} \in \mathbb{R}^3 \ \Big|\ \mathbf{x} = t_1\,\mathbf{p}_1 + t_2\,\mathbf{p}_2 + t_3\,\mathbf{p}_3,\ \ldots \\ t_1 + t_2 + t_3 \leq 1,\ t_1 \geq 0,\ t_2 \geq 0,\ t_3 \geq 0 \Big\}, \tag{B.7}$$

and where $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are the three vertices of the triangle. Notice that the mesh triangle τ also carries a friction coefficient scaling factor $\lambda \in \mathbb{R}_{\geq 0}$, which is considered constant over the triangle surface.

The intersection between the set of ribs $\mathcal{S}$ and the ground mesh triangles $\mathcal{M}$ allows us to identify the contact parameters, namely the contact patch area A, the intersection volume V, the contact point location $\mathbf{P}$, and the contact force direction $\hat{\mathbf{n}}$. Trivially, if a generic rib ϕ is intersected with a generic mesh triangle τ, the result will be a set $\sigma = \phi \cap \tau$. This intersection can lead to four different cases, in which the set σ can be: (1) an empty set, if ϕ does not touch τ at all; (2) a point, if ϕ intersects τ one of its vertices; (3) a segment, if ϕ intersects τ or a portion of it; and (4) a convex hull, if ϕ and τ are coplanar and intersect. Cases 1 and 2 are not considered since they are not relevant to the modeled contact. Case 4 corresponds to a tire lying on the ground, which is not relevant for a vehicle simulation in which the tire is always rolling. Hence,

case 3 is the only relevant one for the modeled contact. In this case, the intersection region is a segment σ such that

$$\sigma = \left\{ \mathbf{x} \in \mathbb{R}^3 \;\middle|\; \mathbf{x} = (1-t)\,\mathbf{p}_a + t\,\mathbf{p}_b,\ t \in [0,\,1] \right\},$$

where $\mathbf{p}_a$ and $\mathbf{p}_b$ are the two extrema points of the segment. We can now calculate the contact patch area A by restating (B.1) as the summation

$$A = \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \left\| \mathbf{p}_{b,ij} - \mathbf{p}_{a,ij} \right\|_2 , \tag{B.8}$$

where w_i is the "virtual" width of the i-th rib.
To solve integrals (B.2), (B.3) and (B.4), they are first transformed to cylindrical coordinates. Moreover, $\bar{\sigma}$ denotes the transformation to into cylindrical coordinates of σ. The radius $\bar{r}(\theta)$ represents the distance from the rib center point $\mathbf{O}$ to a generic point of the intersection segment $\bar{\sigma}$. Denote with r_i the radius of the i-th rib and with $\bar{r}_{ij}(\theta)$ the distance between the i-th rib center point and the segment $\bar{\sigma}_{ij}$; where the segment $\bar{\sigma}_{ij}$ is identified as the intersection of the i-th rib with the j-th triangle. The range of the angle θ for the generic segment $\bar{\sigma}$ is denoted with $[\theta^a, \theta^b] \subset (\pi, 2\pi)$, which are implicitly defined as

$$\begin{bmatrix} \cos\theta^a \\ \sin\theta^a \end{bmatrix} = \frac{\mathbf{p}_a - \mathbf{O}}{\|\mathbf{p}_a - \mathbf{O}\|}, \quad \text{and} \quad \begin{bmatrix} \cos\theta^b \\ \sin\theta^b \end{bmatrix} = \frac{\mathbf{p}_b - \mathbf{O}}{\|\mathbf{p}_b - \mathbf{O}\|}.$$

With this notation the integrals (B.2), (B.3), (B.4) and (B.5) are rewritten as

$$v_{ij}(\theta) = \frac{r_i^2 - \bar{r}_{ij}(\theta)^2}{2}, \tag{B.9}$$

$$V = \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \int_{\theta_{ij}^a}^{\theta_{ij}^b} v_{ij}(\theta)\,\mathrm{d}\theta, \tag{B.10}$$

$$\mathbf{P} = \frac{1}{V} \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \int_{\theta_{ij}^a}^{\theta_{ij}^b} \begin{bmatrix} \bar{r}_{ij}(\theta)\cos(\theta) v_{ij}(\theta) \\ y_i v_{ij}(\theta) \\ \bar{r}_{ij}(\theta)\sin(\theta) v_{ij}(\theta) \end{bmatrix} \mathrm{d}\theta, \tag{B.11}$$

$$\vec{\mathbf{n}} = \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \int_{\theta_{ij}^a}^{\theta_{ij}^b} \begin{bmatrix} \cos(\theta) v_{ij}(\theta) \\ 0 \\ \sin(\theta) v_{ij}(\theta) \end{bmatrix} \mathrm{d}\theta, \quad \hat{\mathbf{n}} = \frac{\vec{\mathbf{n}}}{\|\vec{\mathbf{n}}\|}, \tag{B.12}$$

$$\lambda = \frac{1}{V} \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \int_{\theta_{ij}^a}^{\theta_{ij}^b} \lambda v_{ij}(\theta)\,\mathrm{d}\theta. \tag{B.13}$$

Notice that the intermediate variable $v_{ij}(\theta)$ in (B.9) represents the length of the generic segment radially connecting the $\bar{\sigma}_{ij}$ to the external tire boundary at the angle θ (see Figure B.5). From a physical perspective, $v_{ij}(\theta)$ is the deflection of the infinitesimal radial springs at the angle θ. It is possible to find a closed-form solution for $\bar{r}_{ij}(\theta)$. However, the analytical expression of the resulting $\mathbf{P}$ and $\hat{\mathbf{n}}$ is excessively long. For practical purposes, it is better to use a quadrature formula to numerically approximate the integrals. In this work, Simpson's 1/3 rule is used.

Remark 2. *Simpson's* 1/3 *rule quadrature formula of* $f(x)$ *for the interval* $[a, b]$ *reads as*

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{6}\left[f(a) + 4f(c) + f(b)\right] - \frac{h^5}{2880}f^{(4)}(\xi)\,,$$

where $h = b - a$, $c = (a + b)/2$, *and* $\xi \in [a, b]$ *[184]. Notice that if* $r \leq 1$, $\theta^b - \theta^a \leq \pi/6$, *and* $1 \geq \|\mathbf{O} - \mathbf{p}_{a,b}\| \geq 4/5$, *the relative error of the numerically computed integrals in* (B.10), (B.11), *and* (B.12) *is below* 1%.

Remark 3. *The midpoint* $\bar{r}((\theta^a + \theta^b)/2)$ *is computed as*

$$\bar{r}\left(\frac{\theta^b - \theta^a}{2}\right) = \frac{2\bar{r}(\theta^a)\bar{r}(\theta^b)}{\bar{r}(\theta^a) + \bar{r}(\theta^b)} \cos\left(\frac{\theta^b - \theta^a}{2}\right).$$

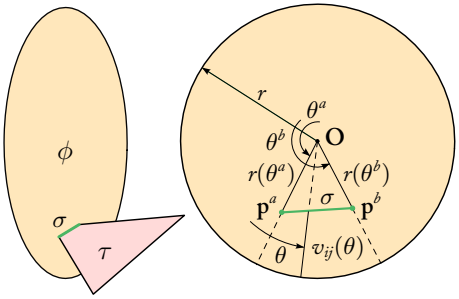


Figure B.5: Representation of the intersection between a tire rib ϕ ( ) and a ground mesh triangle τ ( ). The intersection set σ ( ) is a segment with vertices $\mathbf{p}_a$ and $\mathbf{p}_b$, at θ^a and θ^b angles respectively.

B.4 SOFTWARE ARCHITECTURE

The ENVE algorithm-based software is written in C++ (2011 standard [148]), which is one of the most widely supported, common and fast among object-oriented general-purpose programming languages. We have only reduced the dependencies on the EIGEN [149] and ACME [1] libraries: this provides us with a flexible and extensible

framework. EIGEN is a template library for linear algebra. It is very efficient for small vectors and matrices and can exploit LAPACK/BLAS [150] for peak performance when matrices and vectors have a large size. ACME, on the other hand, is an easy-to-use geometry library that aims to be a minimal tool for HRT scheduling applications. Furthermore, MATLAB® MEX and SIMULINK® S-FUNCTION extensions are provided to allow the integration of ENVE in more complex simulation environments. The software is distributed under the BSD 3-Clause License and is freely available online [27].

B.4.1 DATA TYPES

The software is built on a limited number of classes. We have chosen to keep the library as essential as possible for ease of maintenance and efficiency. For this reason, we use the necessary classes to describe and manipulate virtual ground surfaces and tires. Each geometrical entity representing the tire and ground is organized in a specific class. A brief description of the data types used in the software is now given in this section.

TRIANGLEGROUND The generic ground triangle τ is defined by the three points $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$. In this way, it geometrically corresponds to the TRIANGLE object described in the ACME library [1]. For this reason, the TRIANGLEGROUND object publicly inherits from the ACME::TRIANGLE object. Since the TRIANGLEGROUND object describes the generic triangle representing a local road region, this class also carries a scaling factor for the friction coefficient λ. The mathematical representation corresponds to the definition in (B.7).

MESH The mesh represents the discretized ground set $\mathcal{M}$ on which the tire rolls. It is composed of a multitude of TRIANGLEGROUND objects. The size of these triangles is determined according to end-use requirements and desired accuracy. As a consequence, the number of triangles can range from a few dozen to tens of thousands. For this reason, the ACME library's AABBTREE data structure is used to efficiently manage the intersection between the tire and the ground, and substantial acceleration of the overall algorithm is achieved.

FLAT Using a mesh to represent a flat ground surface is not always the most efficient choice, as this can prolong the time needed for computation unnecessarily. With a MESH the software is forced to go through the creation of an AABBTREE and to make an evaluation of the intersection of the same tree with the AABB containing the tire. To eliminate this source of inefficiency, the FLAT class has been implemented, to represent a perfectly flat terrain. The FLAT object publicly inherit from the ACME::PLANE object and a friction coefficient scaling factor λ is added to the class description

$$\pi = \left\{ \lambda \in \mathbb{R}_{\geq 0},\ \mathbf{x}, \mathbf{p}, \hat{\mathbf{n}} \in \mathbb{R}^3 \;\middle|\; \hat{\mathbf{n}} \cdot (\mathbf{p} - \mathbf{x}) = 0 \right\},$$

where $\mathbf{p}$ is a generic point on the plane and $\hat{\mathbf{n}}$ is a unit normal vector.

SHAPE The SHAPE object describes the overall external shape of the tire $\mathcal{T}$ and of all the ribs in the set $\mathcal{S}$. The external boundary of the tire $\partial\mathcal{T}$ is defined as a surface generated by the revolution of an arbitrary function $R(y)$. By using the proper surface of revolution, it is possible to represent a wide variety of tire morphologies, ranging from car tires to motorcycle tires (see Figure B.1).

RIB As mentioned before, the rib is considered a small section of the overall tire. Geometrically the rib is defined as a disk ϕ of radius r, whose center $\mathbf{O}$ is located in $[0,\ y,\ 0]^{\mathrm{T}}$ in the $\mathbf{H}xyz$ wheel hub reference frame and whose lateral coordinate y is constant throughout the whole disk area. The RIB object publicly inherit from the ACME::DISK object. In addition, the RIB class also carries its "virtual" width $w \in \mathbb{R}_{\geq 0}$, which is stored as a class member. The mathematical representation of this set corresponds to that in (B.6).

SHELL The shell object encapsulates the set of RIB objects $\mathcal{S}$, along with a shared pointer to a SHAPE object and an object of the type ACME::AABB. This last object represents the bounding box that contains the tire at every instant of time. The SHELL object also contains all the attributes and secondary objects that are used to describe the tire and its intersection with the terrain. Whenever an intersection between the tire and the terrain occurs, the affine transformation describing the tire pose as well as the tire AABB is updated. Once these basic operations are completed, the bounding box is intersected with the ACME::AABBTREE of the mesh. A list of shared pointers to TRIANGLEGROUND objects that are internal or that simply touch the tire bounding box is generated. This list is supplied one by one to the RIB objects to find the intersection parameters described in the previous section. To avoid unnecessary recalculations, the intersection parameters are stored in members that are internal to the SHELL class; they are then eventually extracted with methods specifically designed for this purpose.

B.4.2 ALGORITHM WORKFLOW

The workflow of the algorithm is highly streamlined. It is divided into three main algorithms that are briefly described below briefly described and that are also reported in the pseudocodes. Other minor algorithms are used to support the main ones and are not reported here for the sake of brevity.

SHELL INITIALIZATION The initialization of the SHELL object is done by passing a SHAPE object and the number of ribs n as input parameters. Firstly, the SHAPE object is used to reconstruct the tire's external shape; then the RIB objects are instantiated by discretizing the tire into n ribs.

Algorithm 11 Initialization of a SHELL object.

1: **Require:** The tire SHAPE object, and the number of ribs n
2: **function** SHELL(SHAPE, n) ▷ SHELL class constructor

```
3:      S ← ∅                                          ▷ Initialize the set of ribs
4:      w ← Shape.width()/n                            ▷ The ribs width
5:      for i ← 1 to n do
6:          y_i ← w/2 + (i − 1) w                      ▷ The i-th rib lateral coordinate
7:          r_i ← Shape.radius(y_i)                    ▷ The i-th rib radius
8:          S_i ← Rib(r_i, y_i, w)                     ▷ The i-th rib
9:      end for
10:     F ← I_{4×4}                                    ▷ The default affine transformation
11:     B ← AABB(S, F)                                 ▷ The bounding box of the tire
12:     store S, B, F                                  ▷ Store the data in class members
13: end function
```

MESH INITIALIZATION The Mesh object is initialized by passing a Road Data Format or Wavefront OBJ extension file containing the triangles' vertices and faces as input. The file is first parsed to instantiate the mesh TriangleGround objects and then used to build the AABBtree, which will later be used to accelerate the intersection process as well as the calculation of contact parameters.

Algorithm 12 Initialization of a Mesh object.

```
1:  Require: The mesh file path
2:  function Mesh(path)                                ▷ Mesh class constructor
3:      V ← ∅                                          ▷ Initialize ground mesh vertices
4:      F ← ∅                                          ▷ Initialize mesh faces
5:      λ ← ∅                                          ▷ Initialize mesh λs
6:      V, F, λ ← Mesh.parse(file)                     ▷ Parse the mesh file
7:      M ← ∅                                          ▷ Initialize the set of triangles
8:      B ← ∅                                          ▷ Initialize the set of triangles' AABBs
9:      n ← F.size()                                   ▷ The number of triangles
10:     for i ← 1 to n do
11:         p_1 ← V(F(i, 1))                           ▷ The 1st triangle vertex
12:         p_2 ← V(F(i, 2))                           ▷ The 2nd triangle vertex
13:         p_3 ← V(F(i, 3))                           ▷ The 3rd triangle vertex
14:         M_i ← TriangleGround(p_1, p_2, p_3, λ(i))
15:         B_i ← AABB(M_i)                            ▷ The i-th triangle AABB
16:     end for
17:     T ← AABBtree(M, B)                             ▷ The mesh's AABBtree
18:     store M, T                                     ▷ Store the data in class members
19: end function
```

CONTACT PARAMETERS CALCULATION The contact parameters are calculated by intersecting the Shell object and the Mesh object. This is done by calling the "setup" method of the Shell object. Firstly, this method intersects the mesh's AABBtree with the tire's AABB and returns a list of triangles that are potentially intersecting the tire. Then, the tire ribs are intersected with the triangles and the contact parameters are calculated as described in Section B.2.

Algorithm 13 Contact parameters calculation.

Require: The new tire affine transformation F, and the MESH object $\mathcal{M}$
function SHELL.setup(F, $\mathcal{M}$) ▷ The intersection method
  $B \leftarrow$ SHELL.update_aabb(F) ▷ Update the tire AABB
  $L \leftarrow$ MESH.intersect(B) ▷ The intersecting triangles
  $r \leftarrow \mathcal{S}$.size() ▷ The number of tire ribs
  $n \leftarrow L$.size() ▷ The number of intersecting triangles
  $A \leftarrow 0$ ▷ Initialize the contact patch area
  $V \leftarrow 0$ ▷ Initialize the intersection volume
  $\mathbf{P} \leftarrow \mathbf{0}$ ▷ Initialize the contact point
  $\hat{\mathbf{n}} \leftarrow \mathbf{0}$ ▷ Initialize the contact normal
  $\lambda \leftarrow 0$ ▷ Initialize the friction coefficient scaling factor
  for $i \leftarrow 1$ to r **do**
    for $j \leftarrow 1$ to n **do**
      $\sigma \leftarrow \mathcal{S}_i \cap \mathcal{M}_{L(j)}$ ▷ The intersection set (segment)
      $A \leftarrow A + \sigma$.area() ▷ The contact area (B.8)
      $V \leftarrow V + \sigma$.volume() ▷ The contact volume (B.10)
      $\mathbf{P} \leftarrow \mathbf{P} + \sigma$.point() ▷ The contact point (B.11)
      $\hat{\mathbf{n}} \leftarrow \hat{\mathbf{n}} + \sigma$.normal() ▷ The contact normal (B.12)
      $\lambda \leftarrow \lambda + \sigma$.lambda() ▷ The contact λ (B.13)
    end for
  end for
  $\lambda \leftarrow \lambda / V$ ▷ The average λ
  return A, V, $\mathbf{P}$, $\hat{\mathbf{n}}$, λ ▷ The contact parameters
end function

B.5 SIMULATIONS AND DISCUSSION

In this section, we present some simulations to prove the capabilities of the proposed model. The simulations are divided into two categories: quasi-static simulations, and RT performance assessments. Quasi-static simulations are carried out according to the SAE J2731 standard [185] to assess the enveloping capabilities at low speed of the presented model, while dynamic simulations highlight the outcomes of the tire-ground enveloping model in a dynamic scenario. In both types of simulations, a comparison with the SWIFT® model [170] and the TMEASY model [178] is provided. Simulations are performed by modeling the tire as a 205/60R15 (2.2 bar inflation pressure) passenger car tire, which is one of the tires used in [170] to validate the SWIFT® model. The tire is discretized into 10×10 cams for the SWIFT® model and 10 ribs for the ENVE model.

B.5.1 QUASI-STATIC SIMULATIONS ON UNEVEN ROAD SURFACES

As mentioned before the aim of the first set of simulations is to assess the enveloping capabilities of the presented model at a low speed according to SAE J2731 standard [185]. As motivated in the SAE standard, we want to assess under quasi-static considerations the ability of the model to correctly estimate the contact point height z[1] and the forward and lateral slope angles β_x and β_y respectively. In this regard, we have considered

[1]The contact point height is defined as the z-axis component of the contact point position $\mathbf{P}$ in the absolute reference frame.

three different road surfaces, characterized by different degrees of unevenness. On each of these surfaces, we have performed a set of simulations with constant vertical loads F_z of 2, 4 and 6 kN. It must be pointed out that the TMEasy model output does not change with the vertical load F_z and is thus reported with a single line in the plots. The first simulation is performed on a series of different obstacles positioned orthogonally to the forward direction of the tire. Specifically, the obstacles' cross-sections are depicted at the top of Figure B.6. The results of the simulation are reported in the last two plots of Figure B.6. As it can be noticed, the behavior of the presented model is similar to that of the SWIFT® model, which is considered to be very close to the ground truth (see [170] for an in-depth discussion on the SWIFT® model and its accuracy). The performance of ENVE depends on the geometry of the obstacles. Indeed, in the presented model the contact parameters are defined only by the geometry of the obstacles, which is not entirely true in the real world. Tire nonlinear radial stiffness and inflation pressure also play a non-negligible role in defining the overall tire-ground contact behavior. However, given the simplicity of the model, the results are quite satisfactory. Conversely, the TMEasy model does not produce realistic results, causing abrupt changes in the contact parameters: z, β_x, and β_y.
The last simulation is performed on a rough Belgian block road surface depicted in Figure B.1. The results reported in Figure B.7, show that the ENVE model behavior is very close to that of the SWIFT® model in terms of contact point height z and forward slope angle β_y. However, the ENVE model is not able to correctly estimate the lateral slope angle β_x. Even on the rough Belgian block road surface, the TMEasy model is unable to correctly estimate the contact point height z and the lateral slope angle β_x. Moreover, it strongly overestimates the forward slope angle β_y.

B.5.2 FULL-VEHICLE MODEL SIMULATION

To evaluate the behavior of the presented tire-ground enveloping model in a typical use case, it is integrated into a complete RT vehicle simulation framework, and compared with the SWIFT® [170] and TMEasy [176, 178] enveloping models. The used framework consists of a DIL/SIL/HIL simulator based on a custom high-fidelity vehicle model. Vehicle dynamics is described by a 14 DOFs full-vehicle model able to accurately simulate the impact of suspension kinematics and compliance on vehicle behavior within the typical range of frequencies for the ride and handling analysis. Contrary to widespread practice, the vehicle dynamic equations are not linearized. This improves model accuracy in nonlinear regions and vehicle attitude even in abnormal conditions. The described formulation enables the vehicle model for 3D road simulations. It is worth noting that additional DOFs are used to describe internal components' dynamics, such as the powertrain, the steering system or the tire subsystems. The structure of the model is organized in a modular fashion mirroring the mechanical interfaces of vehicle components. This modular structure makes the integration of third-party software or, more generally, of any external system a fairly straightforward process. As a result, it is possible to easily configure the vehicle model for HIL, SIL and/or DIL sim-

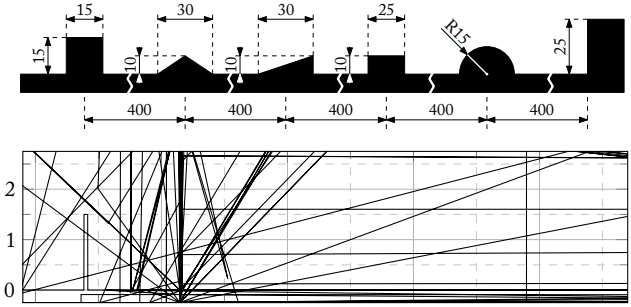


Figure B.6: Quasi-static simulations of a 205/60R15 tire rolling over a stepped road surface. The TMEasy model output does not change with the vertical load F_z and is reported with a single line. *Colors legend*: ■ $F_z = 2\,\text{kN}$, ■ $F_z = 4\,\text{kN}$, and ■ $F_z = 6\,\text{kN}$. *Lines legend*: — Enve, −− Swift®, and ■ TMEasy, and ■ road surface cross-section.

ulations. The developed enveloping model is thus interfaced, as a subsystem with the vehicleMB system through a pre-defined model interface. Inputs of the tire-ground contact subsystem are the wheels' reference frames as well as the wheels' geometrical characteristics and the 3D road surface mesh. The outputs of the subsystem are, for each wheel, the equivalent tire contact point, the local contact plane, the contact penetration and the local friction coefficient. Road contact parameters, extracted from the tire-ground enveloping model, are then used by the Magic Formula 6.2 tire model to compute contact forces.

The modeled vehicle is a passenger car with sedan-like characteristics. The vehicle features a fully electric powertrain with a front-wheel drive transmission layout. The modeled suspensions are a MacPherson configuration for the front suspensions and a Multi-Link scheme for the rear suspensions. For the sake of brevity, only the most relevant vehicle parameters are listed in Table B.1.

Model accuracy and robustness are widely tested with DIL simulations by driving the vehicle in urban scenarios, characterized by speed bumps, sidewalks or, generally, uneven road surfaces. In this work, we present the results obtained by driving the vehicle at a longitudinal speed of 10 km/h over a 45 deg oblique and 1 cm high step. The

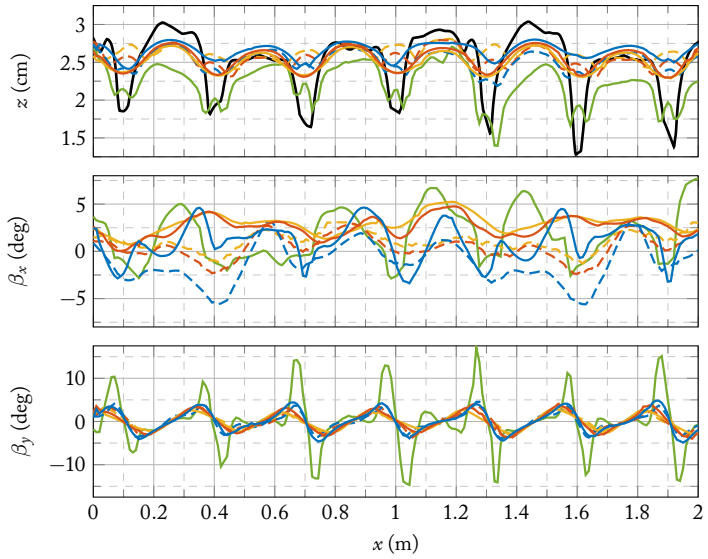


Figure B.7: Quasi-static simulations of a 205/60R15 tire rolling over a Belgian block road surface. The TMEasy model output does not change with the vertical load F_z and is reported with a single line. *Colors legend*: ■ $F_z = 2\,\text{kN}$, ■ $F_z = 4\,\text{kN}$, and ■ $F_z = 6\,\text{kN}$. *Lines legend*: — Enve, −− Swift®, and ■ TMEasy, and ■ road surface cross-section.

reported results are the chassis accelerations (Figure B.8), front-left tire forces (Figure B.9), and the tires deflections (Figure B.10). Note that Enve and Swift® enveloping models present a similar behavior even in the case of a simulation of a full-vehicle model. In fact, the generated chassis accelerations, the tires' forces, and deflections are in agreement with each other. The discontinuities in both the contact point height and the banking and forward slope angles of the TMEasy model reflect on the full-vehicle model simulations with abrupt changes contact forces as well as chassis acceleration not seen in experimental measurements. Notice that according to the results reported in Figure B.7, the Enve model is able to correctly estimate the contact point height, but it slightly underestimates the banking angle. This causes the correct prediction of the vertical contact force magnitude, but a slight underestimation of the lateral contact force magnitude. This error may be emphasized or reduced depending on the tire's properties.

Parameter description	Value
Total mass of the vehicle	1300 kg
Center-of-mass height	0.30 m
Front axle distance from the center-of-mass	1.25 m
Rear axle distance from the center-of-mass	1.45 m
Wheelbase	2.70 m
Yaw inertia	1400 kg m^2
Track width	1.50 m
Steering ratio	20
Maximum torque at wheel	1200 N m
Maximum motor power	150 kW
Wheel inertia around rotation axis	1.40 kg m^2
Size specification of the tires	205/60R15
Spring stiffness at wheel	3530 kN/m
Damping coefficients at wheel for jounce	789 N s/m
Damping coefficients at wheel for rebound	1578 N s/m

Table B.1: Main parameters of the modeled vehicle.

B.5.3 REAL-TIME PERFORMANCE

To access the computations' performance of the Enve C++ software implementation, we will make use of theReal-Time Factor (RTF) metric. Specifically, the RTF is defined as the ratio between the time needed to process the input and the input duration. In order to consider a system a RT system, RTF should be ≤ 1. We performed a test in which the triangles inside the AABB of the tire are increased from 1 up to 10^4, while the number of ribs is increased from 1 to 10. The simulation platform is an iHawk^{TM} Concurrent Real-Time provided with 2.5 GHz Intel Xeon Silver 4215 8 Core, 11 MB cache, 32 GB DDR4 Random Access Memory (RAM), and 64 bit RedHawk Linux RTOS with CentoOS distribution. All tests are performed on a shielded CPU. The results reported in Figure B.11, show that the RTF of Enve is mostly impacted by the number of triangles inside the AABB of the tire rather than the number of ribs. Overall, the RTF of the software is ≤ 1 for a number of triangles inside the AABB of the tire up to ≈ 1000, which corresponds to a regular grid discretization of ≈ 1.5 cm spacing. However, in a typical vehicle simulation, only a part of the time step can be used to perform the tire-ground contact calculations. The available time depends on several factors, such as the complexity of both the vehicle and the tire models. To the author's knowledge, a good practice is to use roughly 25% of the integration time step to establish the contact parameters. This means that the RTF of the software should be ≤ 0.25 to be considered RT. In this case, the maximum number of triangles inside the tire AABB drops to ≈ 200, which corresponds to a regular grid discretization of ≈ 3.5 cm spacing. Overall, these performances are more than sufficient for most RT

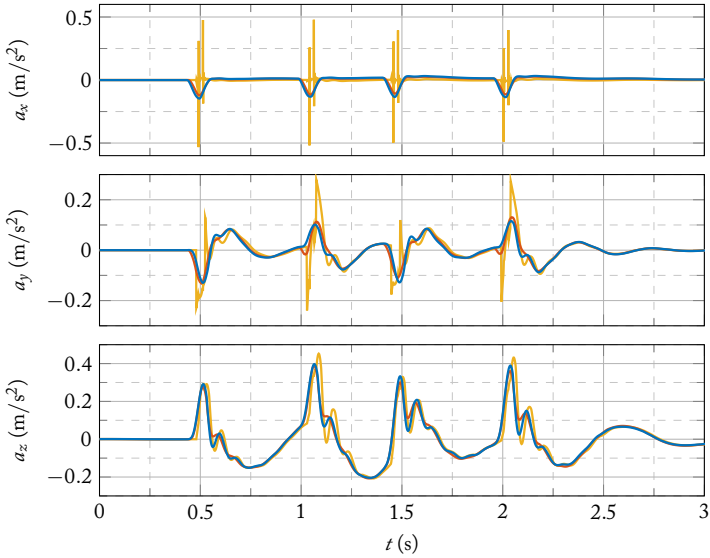


Figure B.8: Chassis accelerations of the full-vehicle model driving over a 45 deg oblique step of 1 cm at a longitudinal speed of 10 km/h. *Legend*: ■ ENVE, ■ SWIFT®, and ■ TMEASY.

applications. Further improvements can be achieved by parallelizing the AABBTREE-AABB and the RIB-TRIANGLEGROUND intersection algorithms, which are currently implemented in a serial fashion. Moreover, different instances of ENVE objects can run concurrently in separated CPUs to further improve the overall performance of the full-vehicle model.

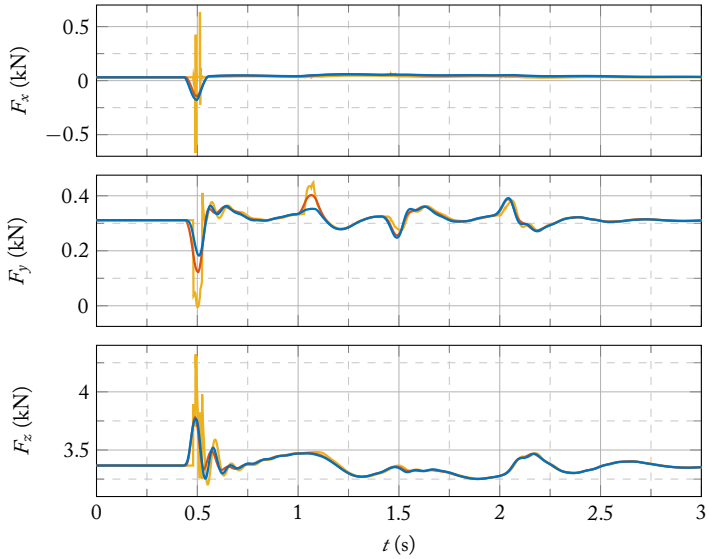


Figure B.9: Front-left tire forces of the full-vehicle model driving over a 45 deg oblique step of 1 cm at a longitudinal speed of 10 km/h. *Legend*: ■ Enve, ■ Swift®, and ■ TMEasy.

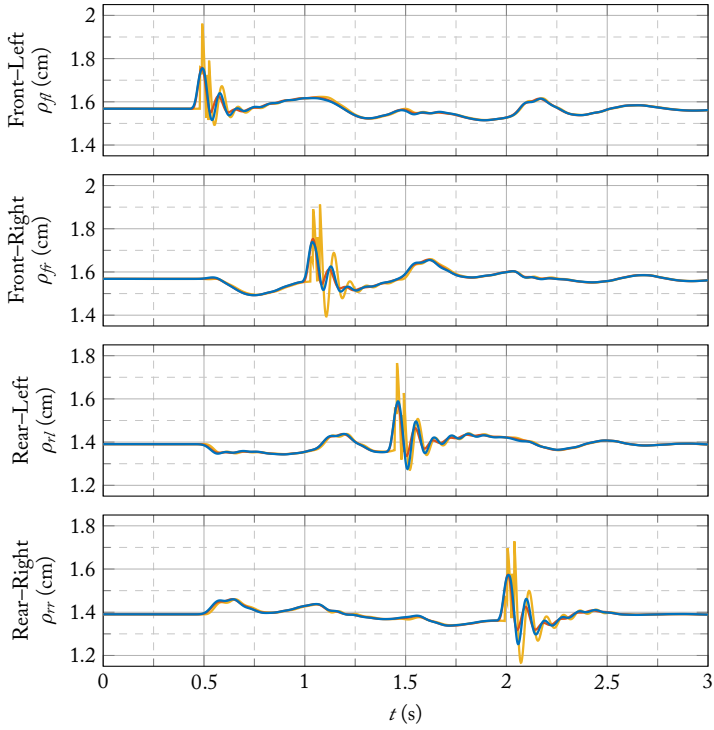


Figure B.10: Tires deflections of the full-vehicle model driving over a 45 deg oblique step of 1 cm at a longitudinal speed of 10 km/h. *Legend*: ■ Enve, ■ Swift®, and ■ TMEasy.

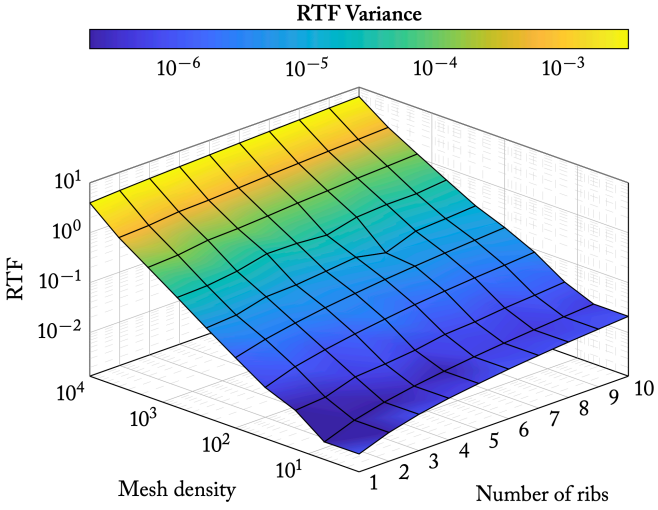


Figure B.11: Real-time factor of the ENVE algorithm C++ implementation [27] as a function of the number of triangles in the tire AABB and the number of ribs which discretize the tire.

APPENDIX C

TIRE PHYSICAL MODELING

We present a novel tire brush model with carcass flexibility, primarily designed for RT applications while preserving the physical significance of its parameters. The model has been already presented in [5], and it is here reported to provide a comprehensive overview of the tire modeling approach. Specifically, it is built upon the methodologies in [1, 4], also outlined in Appendix A and Appendix B. The tire's geometry is discretized and represented using a series of ribs, each intersecting the local road surface independently. The carcass in-plane fore-aft displacement, in-plane deflection, and out-of-plane sidewall torsion are approximated through a second-order polynomial. Forces and torques originating from the tire-road contact area are described using brush mechanics. Notably, the kinematics of the bristles are directly linked to carcass deformation. Extensive effort is dedicated to understanding the possible sources of instability, as well as describing and implementing a robust numerical scheme to effectively solve the nonlinear system of equations arising from the modeling. Numerical performance results are provided to demonstrate the suitability of the presented model for demanding HRT simulations. Validation of the model here presented is carried out by fitting experimental data and comparing it with the state-of-the-art Magic Formula model, proving its reliability and accuracy in reproducing tire behavior.

C.1 INTRODUCTION TO TIRE MODELING

Numerous investigations and experiments have been conducted over the past century to understand tire characteristics and behavior, revealing a close relationship between tire performance and the technologies, techniques, and materials used in manufacturing [22, 186]. However, accurately modeling tire characteristics from construction specifications is challenging due to the complex tire internal structure. To tackle this

challenge, two main modeling approaches have been proposed so far: *empirical* and *physical* [166, 187–189].

The empirical approach is popular because of its simplicity, low computational cost, and ability to capture accurate tire behavior. In this approach, tire-road interaction forces are approximated using a combination of functions derived from physical insights, as well as years of experimental campaigns and experience. The identification process for computing the set of parameters that best fit experimental data for a specific tire is one of the biggest challenges for these models. The Magic Formula model, initially developed by Bakker, Nyborg, and Pacejka [190] and later improved multiple times [166], is the most-known and state-of-the-art empirical model. It fits experimental data through a sequence of numerical optimizations on predetermined curve shapes [191]. Due to its diffusion, low computational complexity and overall performance, the Magic Formula model is considered the standard in both data fitting and vehicle simulation [166, 187].

The physical approach involves direct geometrical and physical tire modeling, providing a deeper understanding of the complex phenomena related to tire behavior [22]. While the computational burden heavily depends on the desired accuracy, this approach can better generalize behavior without fitting a large experimental dataset for every specific tire. Additionally, physical models usually depend on a smaller number of parameters that have a direct physical meaning. The FE method represents a physical approach to tire modeling primarily used, though not exclusively, for assessing static characteristics, stiffnesses, resonant frequencies, and vibration modes [192]. Another physical modeling technique is based on the Discrete Element (DE) method [193], which employs a limited number of interconnected elements and nodes to mimic the degrees of freedom and constraints of real tire carcass structures [179, 180, 194]. Prominent DE models like FTire® and CDTire® are capable of running under RT environments [168, 195], and encompass advanced features such as detailed external geometry description, flexible and visco-plastic rim, air cavity vibration, deformable and visco-elastic ground soil compatibility, as well as temperature and wear modeling. However, the availability of detailed information regarding the computational time performance of both FTire® and CDTire® remains limited.

A big branch of physical modeling relies on the brush mechanics theory [166], which originates from a simplified version of Kalker's conformal contact theory [196–201]. This theory assumes the road surface to be non-deformable, and describes the tire behavior within the contact patch area using one or more rows of radial bristles along or parallels to its equator, having a frictional constraint with the road, and deflecting according to kinematic equations. Over the years, several extensions of the original brush model have emerged, addressing dynamic friction behavior [202–205], realistic contact pressure distribution [206–208], high camber angles and high steering speeds [209, 210], carcass deformation influence [186, 206, 208, 211–217], effects of temperature and wear [207, 218], and a more accurate description of external tire morphology [219, 220]. Brush models are effective at efficiently capturing the behavior of the tire tread layer, and they can also be coupled to a simplified carcass deformation model under

proper assumptions. Sakai's [221–224], Neo-Fiala [206, 212–215], TreadSim [225], and TaMeTire® [207] are three notable advanced tire models that combine a brush model with a carcass deflection and, although different in design, they can accurately describe significant tire characteristics that are beyond the capabilities of empirical models. These include: (1) advanced frictional properties at the tire-road interface, such as local contact pressure and velocity dependency of the friction coefficient; (2) description of the tangential stress within the contact patch, crucial for understanding the tire's lateral and longitudinal force generation; and (3) carcass in-plane fore-aft displacement and deflection, and out-of-plane sidewall torsion, often described through a beam on elastic foundation or a second-order polynomial approximation.
Despite the extensive literature on tire modeling, the author aims to develop a novel tire model suitable for RT vehicle simulations, highlighting the scope for further enhancement in the field of tire modeling. This new model is built upon a minimal set of physical parameters rather than empirical formulas, to characterize tire properties. For this reason, the contributions of this thesis to the existing state-of-the-art tire modeling include the development of a fully scalable physical tire model capable of running within a time step of 1 ms. Such a time step is a typical requirement for RT simulations that aim at reproducing stiff subsystems with high-frequency dynamics [166]. The tire model is structured modularly, facilitating straightforward extension to incorporate additional features such as the effect of temperature and wear, and not less importantly, to achieve the desired computational performance through the newly developed numerical solution scheme. Possible sources of instability are also analyzed, and practical solutions to ensure robustness are discussed. In summary, this study aims to advance the current understanding of flexible carcass tire model behavior and provide a numerical framework enabling the newly developed physical tire model's integration into HRT simulations. To demonstrate the suitability of the proposed model in such demanding conditions, timing and numerical scheme performance outcomes are provided. Validation of the model is performed by fitting experimental data provided by the Tire Research Facility (TIRF) for the Formula SAE (FSAE) Tire Test Consortium (TTC) testing program (refer to Kasprzak and Gentz [226] for a comprehensive description FSAE TTC's testing set-up, procedures, and data acquisition), as well as by comparing it with the state-of-the-art Magic Formula model, ensuring its reliability and accuracy in reproducing tire behavior.

C.2 TIRE MODELING

Pneumatic radial tires are inflated composite structures whose primary structural subcomponents are the *rim*, the *carcass*, and the *tread*. While the rim maintains tire shape, the carcass constitutes the main tire structure connecting the rim to the tread. The tread, on the other hand, serves as the outer rubber layer providing a link between the road and the carcass. In the following sections, we will describe the tire modeling approach, focusing on the carcass and tread, as the rim is considered a rigid body.

C.2.1 TIRE AND ROAD GEOMETRIC REPRESENTATION

Let us consider a reference frame $\mathbf{H}xyz$ with unit vectors $(\hat{\mathbf{h}}_x, \hat{\mathbf{h}}_y, \hat{\mathbf{h}}_z)$, having its origin $\mathbf{H}$ located at the wheel hub. The axes are oriented according to the ISO 8855:2011 standard [183]. The x-axis is directed towards the longitudinal direction of motion, the z-axis points upward, and the y-axis is oriented so that the coordinate system is right-handed (refer to Figure C.1). Geometrically, the tire is characterized as axially symmetric (about the axis $\hat{\mathbf{h}}_y$), convex, and closed set $\mathcal{T}$ defined as

$$\mathcal{T} = \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3,\ y \in [y_l, y_r]\,,\ R(y) : \mathbb{R} \mapsto \mathbb{R} \;\middle|\; \sqrt{x^2 + z^2} \leq R(y) \right\}.$$

Here, the function $R(y)$ can be chosen arbitrarily to reconstruct the desired outermost shape of the tire. On the other hand, the road is modeled as a closed half-space $\mathcal{R}$ whose boundary is defined by the plane passing through the point $\mathbf{p}$ with unit normal $\hat{\mathbf{n}}$, *i.e.*,

$$\mathcal{R} = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid (\mathbf{x} - \mathbf{p}) \cdot \hat{\mathbf{n}} \leq 0 \right\},$$

where $\mathbf{x} = [x, y, z]^\top$. The contact patch region is defined as a closed set $\mathcal{P}$ comprising all the points of the tire $\mathcal{T}$ which are also interior points of the road boundary surface $\partial\mathcal{R}$, *i.e.*, $\mathcal{P} = \mathcal{T} \cap \partial\mathcal{R}$. The contact patch reference frame $\mathbf{O}xyz$ has the origin in $\mathbf{O}$, which corresponds to the centroid of the region $\mathcal{P}$. Its unit vectors $(\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z)$ are defined as $\hat{\mathbf{e}}_x = \hat{\mathbf{h}}_x$, $\hat{\mathbf{e}}_y = \hat{\mathbf{n}} \times \hat{\mathbf{e}}_x$, and $\hat{\mathbf{e}}_z = \hat{\mathbf{n}}$.

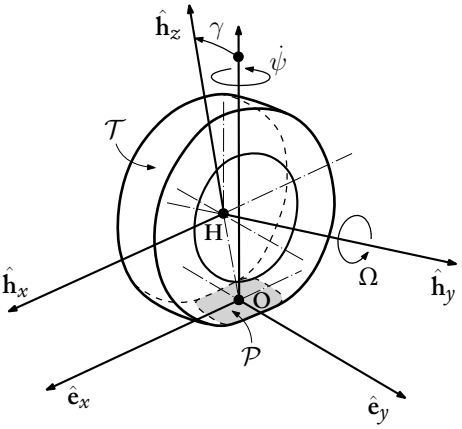


Figure C.1: Tire-road schematics according to ISO coordinate system [183].

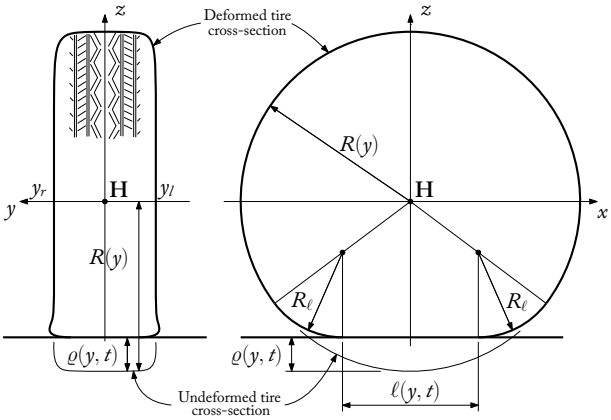


Figure C.2: Illustration of the tire-road contact geometry parameters.

C.2.2 TIRE-ROAD VERTICAL CONTACT MECHANICS

When the contact patch set is not empty (*i.e.*, $\mathcal{P} \neq \varnothing$) there is a multitude of contact points $\mathbf{x} = [x, y, z]^\top \in \mathcal{P}$. At each point, the stress component along the z-axis (also known as normal stress) $q_z(\mathbf{x}, t)$ is applied. The integral of the normal stress distribution over the entire contact patch region is equal to the tire's vertical load $F_z(t)$. The vertical normal stress distribution is extremely difficult to calculate analytically, and it depends on the tread pattern properties, carcass stiffness, friction and inflation pressure [22]. To easily model the tire's vertical characteristics, we employ a single-contact point approach. The total vertical tire stiffness is calculated as

$$k_z(\omega, \gamma, p) = k_z^s + k_z^p p + k_z^\omega \omega + k_z^\gamma \gamma^2 .$$

Here, k_z^s represents the structural vertical stiffness, k_z^p denotes the vertical stiffness dependence on inflation pressure, k_z^ω indicates the vertical stiffness dependence on rolling speed, and k_z^γ is the vertical stiffness dependence on camber angle. The total vertical force is thereby calculated as

$$F_z(t) = k_z(\omega(t), \gamma(t), p)\rho(t) + c_z\dot{\rho}(t) + k_z^b \max(0, \rho(t) - \rho_b) , \tag{C.1}$$

where $F_z \in \mathbb{R}_{\geq 0}$, $\rho(t) \in \mathbb{R}_{\geq 0}$ represents the deflection at the tire contact point, ρ_b represents the bottoming threshold, c_z is the vertical damping coefficient, and k_z^b indicates the bottoming vertical stiffness. Specifically, the tire contact point deflection $\rho(t)$ is calculated with the method outlined in [1, 4]. This technique not only facilitates efficient and robust computation of the contact point deflection $\rho(t)$ but also allows us to compute the tire deflection $\varrho(y, t)$ at a specific lateral coordinate y and time t.

Alternatively, for cases where the technique outlined in [4] is not employed, a viable approximation, particularly for small camber angles ($|\gamma(t)| < \pi/10$), is

$$\rho(t) = \max_{y \in [y_l, y_r]} \varrho(y, t) \quad \text{(refer to Figure C.2)}.$$

C.2.2.1 CONTACT PATCH LENGTH

The geometrical intersection between the tire set $\mathcal{T}$ and road boundary surface $\partial\mathcal{R}$ provides us only a rough approximation of the tire-road contact patch area. Based on the findings in [227, 228], we introduce a contact transition radius $R_\ell \in \mathbb{R}_{\geq 0}$ to take the tire belt stiffness into account and get a more realistic result (see Figure C.2). The contact length at any given lateral coordinate y on the contact patch reference system can be computed as

$$\ell(y, t) = 2\sqrt{\big(2(R(y) - R_\ell) - \varrho(y, t)\big)\,\varrho(y, t)}\,, \tag{C.2}$$

where $\varrho(y, t)$ is the tire radial deflection. Notice that the equation (C.2) can be easily found from the study of Rhyne [228] by imposing null radial counter-deflection.

C.2.2.2 CONTACT PRESSURE DISTRIBUTION

The pressure distribution is one of the most complex, challenging-to-model factors significantly impacting tire behavior [229]. Both experimental results and FE analyses reveal that the pressure trend over the contact patch is strongly three-dimensional and is influenced by numerous factors, including carcass structural stiffness, contact pressure, relative camber angle, tread pattern topology, and local road macro-roughness. Experimental evidence suggests that adequately inflated modern radial tires typically exhibit a plateau in the pressure distribution at the central part of the contact patch [22, Chapter 5]. Conversely, overinflated and underinflated radial tires may have more complicated distribution shapes [22, 230]. Recent efforts have been aimed at more detailed modeling of the contact patch to better match experimental results [206–208]. An interesting approach involves modeling the contact pressure distribution using a quartic polynomial function $\Upsilon(\chi) = c_0 + c_1\chi^1 + c_2\chi^2 + c_3\chi^3 + c_4\chi^4$. This function represents the normalized contact pressure distribution shape along the normalized longitudinal coordinate $\chi \in [0, 1]$. To determinate the polynomial coefficients $c_0, \ldots, c_4$ the following constraints are imposed

$$\begin{cases} \dfrac{\mathrm{d}^2}{\mathrm{d}\chi^2}\Upsilon\left(\dfrac{1}{2}\right) = -\lambda \\ \displaystyle\int_0^1 \Upsilon(\chi)\,\mathrm{d}\chi = 1 \\ \displaystyle\int_0^1 \Upsilon(\chi)\chi\,\mathrm{d}\chi = \frac{1}{2} - \delta \\ \Upsilon(0) = \Upsilon(-1) = 0 \end{cases} \tag{C.3}$$

Then, we solve for $c_0, \ldots, c_4$, and the solution of the system is

$$\begin{aligned} c_0 &= 0\,, \\ c_1 &= -\frac{1}{3}\lambda + 10 + 60\delta\,, \\ c_2 &= 2\lambda - 30 - 180\delta\,, \\ c_3 &= -\frac{10}{3}\lambda + 40 + 120\delta\,, \\ c_4 &= \frac{5}{3}\lambda - 20\,. \end{aligned}$$

Here, $\lambda \in \mathbb{R}$ and $\delta \in \mathbb{R}$ respectively denote the convexity and longitudinal barycentre shift of the base curve $\Upsilon(\chi)$. The only drawback of this approach is that for certain values of δ and λ, the condition of positive pressure $\Upsilon(\chi) \geq 0$ within the range $\chi \in [0, 1]$ is not always guaranteed and must be numerically checked in the software implementation. Thus, a careful selection of these parameters is essential. On the other hand, the use of a polynomial to describe the pressure distribution over the contact patch allows us to adopt some numerically robust algorithms and efficient evaluations for some of the calculations that will follow. Given the objective of achieving RT applicability for the numerical tire model, this aspect should not be overlooked. The vertical contact stress distribution $q_z(\mathbf{x}, t)$ is then written as

$$q_z(\mathbf{x}, t) = \frac{F_z(t)}{A_{\mathcal{P}}(t)} \Upsilon \left(\frac{\ell(y, t)/2 - x}{\ell(y, t)} \right), \tag{C.4}$$

where $\mathbf{x} = [x, y, z]^\top \in \mathcal{P}$, F_z denotes the tire's vertical load, and $A_{\mathcal{P}}$ represents the contact patch area, defined as

$$A_{\mathcal{P}}(t) = \int_{y_r}^{y_l} \ell(y, t)\, \mathrm{d}y\,.$$

It must be stressed that the contact patch region $\mathcal{P}$ is a function of time t and of the lateral coordinate y. This variability arises because the contact patch is not fixed but rather a function of the tire-road contact geometry.

C.2.3 TIRE-ROAD TANGENTIAL CONTACT MECHANICS

In the present work, we assume that the vertical problem can be decoupled from the tangential, implying that the tangential stresses do not affect the vertical one, but not vice versa. Additionally, the vertical and lateral carcass flexibility characteristics are assumed to be fully independent. In simpler terms, we assume that neither friction-related phenomena nor carcass deformation affect the vertical pressure distribution q_z. Consequently, we can analyze tire-road tangential contact mechanics sequentially from the vertical component, while still maintaining the tangential contact forces proportionality to the vertical load F_z. It is worth noting that this assumption greatly

simplifies the numerical solution of the tire model, as it avoids the need to find the vertical and tangential forces through a system of nonlinear equations, which would be computationally expensive.

C.2.3.1 CARCASS DEFORMATION MODEL

In numerous tire models, the carcass is modeled as a beam on elastic foundation [22, 186, 225, 231]. Experimental findings from the studies [186, 231] demonstrate that carcass deflection comprises both bending and shears components. The significance of the shear contribution varies depending on the carcass ply cord angles. Consequently, formulations like the Timoshenko-Ehrenfest or Euler-Bernoulli beam are particularly suited to describe the lateral bending of carcass structure in the cornering conditions [186]. Models like those in [186, 206–208, 213, 214, 221–224, 232] adopt a second-order polynomial (parabola) as a local approximation of the beam on elastic foundation in the proximity of contact patch area. This approach provides a straightforward yet accurate description of the deformed carcass centreline within the contact region. However, the models currently present in the literature are only able to fully describe the lateral deformation and do not allow for modeling the longitudinal displacement. Indeed, during traction, braking and cornering, the contact patch is subject to non-negligible fore-aft, lateral, bending and torsional displacements. These displacements strongly impact the generation of longitudinal, lateral, and vertical forces, as well as rolling resistance, overturning, and self-alignment torques. To take into account the longitudinal displacement of the carcass, we define the two-dimensional carcass centreline as

$$\mathcal{L}(\mathbf{x}, \vec{\mathbf{c}}) = \begin{bmatrix} \mathcal{L}_x \\ \mathcal{L}_y \end{bmatrix} = \begin{bmatrix} x_c \\ y_c + \theta_c(x - x_c) - y_c \dfrac{\Psi}{2}(x - x_c)^2 \end{bmatrix}, \tag{C.5}$$

where time-dependent parameters x_c and y_c represent the carcass longitudinal and the lateral translating deformations, respectively, θ_c denotes the carcass twisting angle, and Ψ is the lateral bending shape factor. As depicted in Figure C.3, we model the carcass deformation through a point $\vec{\mathbf{c}} = [x_c, y_c, \theta_c]^\top$ described within the contact patch reference frame $\mathbf{O}xyz$. The carcass point $\mathbf{c}$ is connected through a set of independent spring elements to the point $[0, 0, -R_z]^\top$, defined in the hub reference frame $\mathbf{H}xyz$, where $R_z = \hat{\mathbf{n}} \cdot (\mathbf{H} - \mathbf{O})$.

As mentioned at the beginning of this, the carcass is the subcomponent of a tire that connects the rim to the tread and, it has a significant impact on the tire's overall behavior. From a pure physics perspective, the carcass is a system that is rigidly connected to the rim and has to deform in order to accommodate the tread's deformation. This involves balancing forces generated by the carcass state against the stresses of the tread state deformation. Thence, the carcass equilibrium equations in matrix form are written as

$$\vec{\mathbf{G}}(\vec{\mathbf{c}}) = \mathbf{K}(\vec{\mathbf{c}})\vec{\mathbf{c}} - \vec{\mathbf{F}}(\vec{\mathbf{c}}, t) = 0\,, \tag{C.6}$$

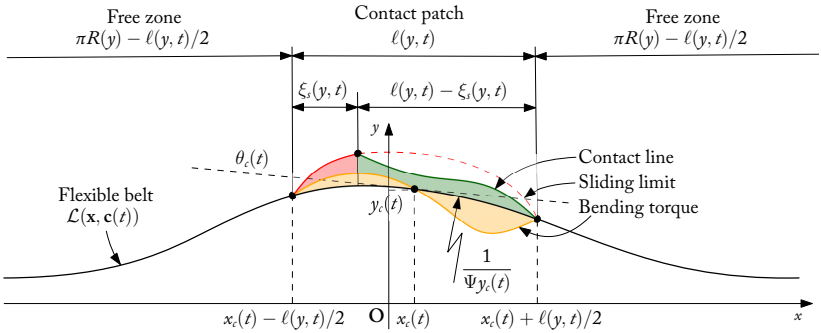


Figure C.3: Depictions of the deformed carcass baseline $\mathcal{L}(\mathbf{x}, \vec{\mathbf{c}})$, along with the contact line of bristles, adherence limit, distribution of bending torque, and the adherence and sliding regions.

where

$$\vec{\mathbf{F}}(\vec{\mathbf{c}}, t) = \begin{bmatrix} F_{ax}(\vec{\mathbf{c}}, t) \\ F_{ay}(\vec{\mathbf{c}}, t) \\ M_{az}(\vec{\mathbf{c}}, t) \end{bmatrix} + \begin{bmatrix} F_{sx}(\vec{\mathbf{c}}, t) \\ F_{sy}(\vec{\mathbf{c}}, t) \\ M_{sz}(\vec{\mathbf{c}}, t) \end{bmatrix},$$

and the matrix $\mathbf{K}$ represents the carcass stiffness matrix, which is assumed to be a diagonal matrix of the form

$$\mathbf{K}(\vec{\mathbf{c}}) = \begin{bmatrix} K_x(p, x_c) & 0 & 0 \\ 0 & K_y(p, y_c) & 0 \\ 0 & 0 & K_\theta(p, \theta_c) \end{bmatrix}.$$

The entries $K_x(p, x_c)$, $K_y(p, t_c)$, and $K_\theta(p, \theta_c)$ are the longitudinal, lateral and twisting carcass stiffnesses, respectively. These values are influenced by the tire construction technology and inflation pressure p, as well as the deformation of the tire carcass. They can be expressed as follows

$$\begin{aligned} K_x(p, x_c) &= k_x^s + p\, k_x^p + k_x^b\, h(x_c, x_b, h_x), \\ K_y(p, y_c) &= k_y^s + p\, k_y^p + k_y^b\, h(y_c, y_b, h_y), \\ K_\theta(p, \theta_c) &= k_\theta^s + p\, k_\theta^p + k_\theta^b\, h(\theta_c, \theta_b, h_\theta). \end{aligned}$$

Here, the function $h(x, b, h)$ is defined as

$$h(x, b, h) = (\mathrm{pos}(-x - b, h) + \mathrm{pos}(x - b, h))^2,$$

where

$$\mathrm{pos}(x,h) = \begin{cases} \dfrac{\sqrt{x^2+h^2}}{2} + x & x > 0 \\ \dfrac{h^2}{2(\sqrt{x^2+h^2}-x)} & \text{otherwise} \end{cases}$$

is a regularized function of the positive part. Notice that the deformation of the tire carcass has a physical limit given by the maximum deformation of the sidewalls. For this reason, the parameters k_x^b, k_y^b, and k_θ^b are introduced to model the longitudinal, lateral and twisting bottoming stiffnesses, respectively.

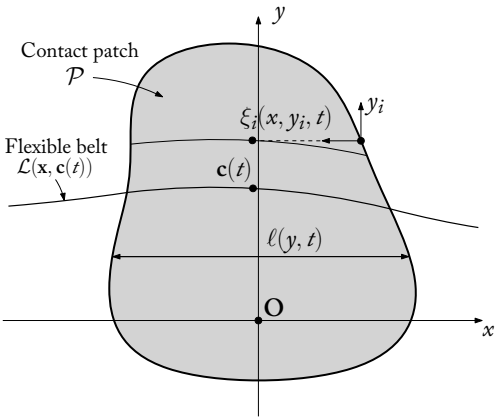


Figure C.4: Representation of the contact patch region $\mathcal{P}$ and contact patch reference frame $\boldsymbol{\xi}_i(y_i, t)$. Notice that the contact patch is defined as a D-convex and arbitrary region, whose shape is described along the y-axis by the quantity $\ell(y, t)$.

C.2.3.2 TIRE-ROAD TANGENTIAL CONTACT MECHANICS EQUATIONS

For each point $\mathbf{x} \in \mathcal{P}$, given a carcass pose $\vec{\mathbf{c}}$, we associate a speed field $\mathbf{v}(\mathbf{x}, t)$ and a displacement vector $\vec{\mathbf{u}}(\mathbf{x}, \vec{\mathbf{c}}, t)$. The speed field $\mathbf{v}$ represents the relative speed between the outer tread layer and the road, while the displacement vector $\vec{\mathbf{u}}$ represents the deformation of the material point (also referred to as a *bristle*) located at the coordinate $\mathbf{x}$. Additionally, each material point is also subjected to an external force $\vec{\mathbf{q}}(\mathbf{x}, \vec{\mathbf{c}}, t)$ generated by the tire-road contact.

We define the generic tangential position as $\mathbf{t} = x\hat{\mathbf{e}}_x + y\hat{\mathbf{e}}_y$, the tangential velocity field as $\vec{\mathbf{v}}_\mathbf{t}(\mathbf{x}, \vec{\mathbf{c}}, t) = v_x(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_x + v_y(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_y$, the tangential displacement vectors as $\vec{\mathbf{u}}_\mathbf{t}(\mathbf{x}, \vec{\mathbf{c}}, t) = u_x(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_x + u_y(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_y$, and the tangential stress vector as $vtq_\mathbf{t}(\mathbf{x}, \vec{\mathbf{c}}, t) =$

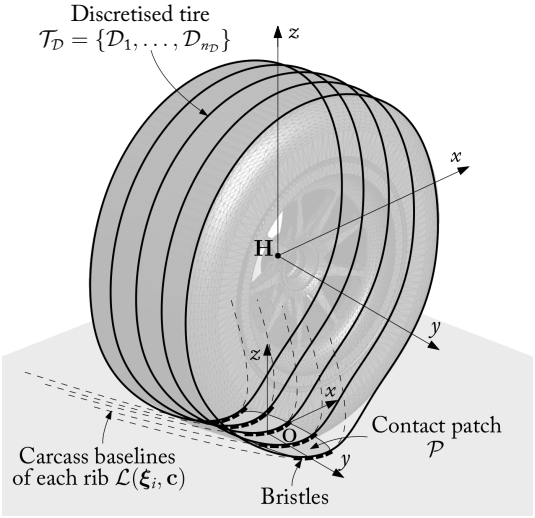


Figure C.5: Illustration of the brush model. The tire $\mathcal{T}$ is discretized into a series of $n_\mathcal{D}$ ribs. Carcass and bristles are represented as the main components of the model. Conversely, the rim is assumed to be a rigid body and thus has no impact on the tire-road contact.

$q_x(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_x + q_y(\mathbf{x}, \vec{\mathbf{c}}, t)\hat{\mathbf{e}}_y$, in a such a way that

$$\vec{\mathbf{v}} = \begin{bmatrix} \vec{\mathbf{v}}_\mathbf{t} \\ v_z \end{bmatrix}, \quad \vec{\mathbf{u}} = \begin{bmatrix} \vec{\mathbf{u}}_\mathbf{t} \\ u_z \end{bmatrix}, \quad \text{and} \quad \vec{\mathbf{q}} = \begin{bmatrix} \vec{\mathbf{q}}_\mathbf{t} \\ q_z \end{bmatrix}. \tag{C.7}$$

The bristles tangential *micro-sliding speed* is defined as

$$\vec{\mathbf{v}}_s(\mathbf{x}, \vec{\mathbf{c}}, t) = \begin{bmatrix} v_{sx}(\mathbf{x}, \vec{\mathbf{c}}, t) \\ v_{sy}(\mathbf{x}, \vec{\mathbf{c}}, t) \end{bmatrix},$$

which represents the relative speed between a material point within the contact patch $\mathcal{P}$ and the road surface $\partial\mathcal{R}$. In the following, we will employ an isotropic steady-state Coulomb friction model, which enables us to write the equations governing the contact mechanics between the tire tread layer and the road surface as

$$\vec{\mathbf{v}}_s = 0 \iff \|\mathbf{q}_\mathbf{t}\| \leq \mu_s q_z \quad \text{and} \quad \vec{\mathbf{q}}_\mathbf{t} = -\mu_d q_z \frac{\vec{\mathbf{v}}_s}{\|\vec{\mathbf{v}}_s\|} \iff \vec{\mathbf{v}}_s \neq 0\,. \tag{C.8}$$

In these equations, μ_s represents the static friction coefficient and $\mu_d(\mathbf{x}, \vec{\mathbf{c}}, t)$ denotes the "dynamic" friction coefficient. It is important to note that (C.8) is only valid under the assumption of memory-less friction, meaning there is a one-to-one map between

the tangential stress and the micro-sliding speed [233]. To solve (C.8), two additional sets of equations are necessary. The first set consists of the so-called *tire-road kinematic equations*, which link the wheel hub's relative motion to the deformation field over the contact patch area. The second set includes the *constitutive relations*, which correlate the bristles deformation field to the generated tangential stress $\vec{\mathbf{q}}_\mathbf{t}$.

C.2.3.3 RUBBER FRICTION

Friction significantly affects tire behavior. As noted in prior works [234–237], it is influenced by various factors such as tread rubber compound, road surface roughness at both micro and macro levels, relative speed of sliding surfaces, and temperature conditions. To keep the complexity under an acceptable level, we adopt a steady-state Coulomb friction model. In doing so, the dynamic characteristic of friction, known as *frictional memory*, is disregarded. Such a behavior arises from the fact that the friction coefficient depends not only on the relative sliding speed but also on the contact history, *i.e.*, the sliding trajectory [238]. Including this phenomenon would significantly increase the complexity of the whole tire model, necessitating a dense discretization of the tire tread layer.

Numerous physical formulations aim to capture the relationship between the friction coefficient and sliding speed. One of the most important is the *Savkoor friction law*, which includes the so-called *Stribeck effect* [229, 235, 236]. When accounting for contact pressure dependency, the "dynamic" friction coefficient $\mu_d(\mathbf{x})$ can be expressed as

$$\mu_d = \mu_w(\mu_{p_0} + \mu_{p_1}\exp(-\mu_{p_2}q_z))(\mu_k + (\mu_s - \mu_k))\ldots \\ \exp\left(-\mu_{v_0}^2\log^2\left(\frac{\|\vec{\mathbf{v}}_s\|}{v_\mu} + \mu_{v_1}\exp\left(-\frac{\|\vec{\mathbf{v}}_s\|}{v_\mu}\right)\right)\right),$$

where

- μ_w represents the friction coefficient scaling factor, used to tune the friction coefficient under different conditions, such as wet, dry, icy, etc.;
- μ_s is the static friction coefficient;
- μ_k is the kinetic friction coefficient;
- μ_{p_0}, μ_{p_1}, μ_{p_2} are parameters aimed at fitting the rubber-asphalt contact friction pressure dependency;
- μ_{v_0} and μ_{v_1} are parameters used to reproduce the rubber-asphalt contact friction velocity dependency;
- v_μ is the Stribeck velocity.

C.2.3.4 TIRE-ROAD KINEMATIC EQUATIONS

The relative velocity between the bristles and the road, first introduced with the name of micro-sliding velocity $\vec{\mathbf{v}}_s$, are expressed in Eulerian form similar to that in [166, 187, 210, 216, 217, 239, 240]. Hence, using (C.5) and (C.7), $\vec{\mathbf{v}}_s$ can be written as

$$\vec{\mathbf{v}}_s = V_r \left(\boldsymbol{\sigma} + \varphi \hat{\mathbf{e}}_z \times (\mathcal{L} + \mathbf{t}) + \frac{\mathrm{d}}{\mathrm{d}x}\mathcal{L} \right) + \frac{\mathrm{d}}{\mathrm{d}t}\vec{\mathbf{u}}_\mathbf{t}\,, \tag{C.9}$$

where the material point deformation $\vec{\mathbf{u}}_\mathbf{t}$ is assumed to be small. For a comprehensive description and derivation of such sliding refer to [166, 187, 188, 210, 216, 239, 240]. The *theoretical translational slips* $\boldsymbol{\sigma}$ represent the normalized difference between the longitudinal and lateral components of the speed of the wheel hub and that of the tire periphery $V_r(y,t) = \omega(t)R_r(y,t)$. Therefore, they are defined as

$$\boldsymbol{\sigma}(y,t) = \begin{bmatrix} \sigma_x(y,t) \\ \sigma_y(y,t) \end{bmatrix} = -\frac{1}{|V_r(y,t)|} \begin{bmatrix} V_x(t) - V_r(y,t) + \dot{x}_c(t) \\ V_y(t) - R_z(t)\dot{\gamma}(t) + \dot{y}_c(t) \end{bmatrix},$$

where σ_x and σ_y are referred to as the *longitudinal* and the *lateral theoretical slips*. The *theoretical spin slip* φ is obtained from

$$\varphi(y,t) = -\frac{\dot{\psi}(t) + \omega(t)\sin(\gamma(t)) + \dot{\theta}_c(t)}{|V_r(y,t)|}\,.$$

The quantity R_r represents the so-called *real rolling radius*, which is calculated as

$$R_r(y,t) = R(y) - \frac{\varrho(y,t)}{3}\,.$$

Other definitions of slips are available in the literature, such as the *practical longitudinal slip* κ and *slip angle* α [166], which are defined as

$$\begin{bmatrix} \kappa(y,t) \\ \alpha(y,t) \end{bmatrix} = -\frac{1}{|V_x(t)|} \begin{bmatrix} V_x(t) - V_r(y,t) + \dot{x}_c(t) \\ \arctan V_y(t) - R_z(t)\dot{\gamma}(t) + \dot{y}_c(t) \end{bmatrix},$$

and thus

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \frac{1}{1+|\kappa|} \begin{bmatrix} \kappa \\ -\tan(\alpha) \end{bmatrix}.$$

Since we are using the same Eulerian form of [210], the total time derivative is given by

$$\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \vec{\mathbf{v}}_\mathbf{t} \cdot \nabla\mathbf{t}\,,$$

where $\nabla\mathbf{t}$ collects the tangential components of the gradient, and $\partial/\partial t = 0$ since we are neglecting the viscous component of the Kelvin-Voigt element representing the rubber material point [241]. Neglecting the viscous effect of the tread rubber material eliminates any explicit time dependence in the formulation of brush model forces and

torques. Consequently, the tread rubber always operates under stationary conditions. Considering these points, the total derivative of the material point can be explicitly expressed as

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{\mathbf{u}}_{\mathbf{t}} = \underbrace{\frac{\partial}{\partial t}\vec{\mathbf{u}}_{\mathbf{t}}}_{=0} + \vec{\mathbf{v}}_{\mathbf{t}} \cdot \nabla \mathbf{t}\vec{\mathbf{u}}_{\mathbf{t}}\,,$$

where the tangential gradient is given by $\nabla \mathbf{t} = [\partial/\partial x, \partial/\partial y]^{\top}$. The velocity field $\vec{\mathbf{v}}_{\mathbf{t}}$ of the tread rubber is formulated as

$$\vec{\mathbf{v}}_{\mathbf{t}} = -V_r \frac{\dfrac{\mathrm{d}}{\mathrm{d}x}\mathcal{L}}{\left\| \dfrac{\mathrm{d}}{\mathrm{d}x}\mathcal{L} \right\|}\,.$$

Given the assumption of small carcass deformations (approximatively $|x_c| < \rho_b$, $|y_c| < \rho_b$, and $|\theta_c| < \pi/10$) and small tire camber angles ($|\gamma(t)| < \pi/10$) the following relationships holds [242]

$$\frac{\mathrm{d}}{\mathrm{d}x}\mathcal{L}_x \approx 1 \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}x}\mathcal{L}_y \approx 0\,.$$

The trajectories of the bristles moving within the contact patch are straightened in the longitudinal direction. Therefore, the formulation is constrained to a velocity field of the form $\vec{\mathbf{v}}_{\mathbf{t}} = [-V_r, 0]^{\top}$
By separating the longitudinal and tangential displacements of the bristles, we derive the bristle micro-sliding speed formulated as

$$\vec{\mathbf{v}}_s = V_r \left(\vec{\boldsymbol{\sigma}} + \varphi \hat{\mathbf{n}}_z \times (\mathcal{L} + \mathbf{t}) + \frac{\mathrm{d}}{\mathrm{d}x}\mathcal{L} - \frac{\mathrm{d}}{\mathrm{d}x}\vec{\mathbf{u}}_{\mathbf{t}} \right)\,. \tag{C.10}$$

C.2.3.5 CONSTITUTIVE RELATIONS

In the present work, we assume that – from a frictional perspective – two regions are identified along the contact patch region $\mathcal{P}$. The first is the adhesion region $\mathcal{A}$, where bristle tips do not slide with respect to the ground, generating the so-called adhesive forces. The second is the sliding region $\mathcal{S}$, where bristles tips slide on the ground surface, producing instead the sliding forces. The boundary separating these two regions is called transition line $\xi_s(\boldsymbol{\xi}, \vec{\mathbf{c}}, t)$. Although this assumption may be not realistic in the case of high spin slip value, it greatly simplifies the problem [242]. By adopting this approach, the domain corresponds to the contact patch interior $\mathring{\mathcal{P}}$, where the partial differential equations ruling the tire-road kinematics are defined, and the corresponding BCs can be uniquely formulated on $\partial\mathcal{P}$ [210]. This research aims to develop a formulation that strikes a balance between realism and computational efficiency, rather than striving for a rigorous closed-form expression for the tangential bristle contact mechanics.

ADHESION REGION Each material point, acting independently of others, enters the contact patch through the leading edge in an undeformed state. As a result of van der Waals molecular bonding and rubber indentation at the rubber/ground interface, the tip of the bristles sticks to the ground. However, due to the micro-sliding velocity $\vec{\mathbf{v}}_s$ between the root of the bristles and the road, a tangential deflection $\vec{\mathbf{u}}_t$ starts to build up, along with tangential stress $\vec{\mathbf{q}}_t$. Initially null, this bristle deformation increases gradually along the contact length until it reaches the transition line $\xi_s(\boldsymbol{\xi}, \vec{\mathbf{c}})$. This transition line represents the contact length coordinate at which the local force exerted by the bristle deformation in adhesion equals the maximum possible static friction force.

It is important to note that within the adhesion region, the tips of the bristles do not slide relative to the ground, resulting in a zero micro-sliding velocity. Consequently, the deformation of the bristles increases along the contact length direction with a spatial derivative equal to

$$\frac{\mathrm{d}}{\mathrm{d}x}\vec{\mathbf{u}}_t = \frac{\mathrm{d}}{\mathrm{d}x}\begin{bmatrix} u_x \\ u_y \end{bmatrix}, \tag{C.11}$$

which is easily found from (C.10) by imposing $\vec{\mathbf{v}}_s = 0$.

We now introduce a new contact patch coordinate system (also depicted in Figure C.4)

$$\boldsymbol{\xi}(y,t) = \begin{bmatrix} \xi(x,y,t) \\ \eta \\ \zeta \end{bmatrix} = \begin{bmatrix} \ell(y,t)/2 - x \\ y \\ z \end{bmatrix}.$$

The variable ξ represents the distance from the entrance to the longitudinal coordinate, measured longitudinally from the contact patch leading edge. From now on, we will use the superscript "$\boldsymbol{\xi}$" to denote variables described in reference system $\boldsymbol{\xi}$, *e.g.*, $\vec{\mathbf{u}}_t(\boldsymbol{\xi}, \vec{\mathbf{c}})$. Substituting $\boldsymbol{\xi}$ in (C.11) we obtain

$$\frac{\mathrm{d}}{\mathrm{d}\xi}\vec{\mathbf{u}}_t^{\boldsymbol{\xi}} = \frac{\mathrm{d}}{\mathrm{d}\xi}\begin{bmatrix} u_x^{\boldsymbol{\xi}} \\ u_y^{\boldsymbol{\xi}} \end{bmatrix}. \tag{C.12}$$

The specific deformation $\vec{\mathbf{u}}_t^{\boldsymbol{\xi}}$ at a contact length coordinate ξ and lateral coordinate y is obtained from integrating (C.12) over space, while imposing the boundary condition $\vec{\mathbf{u}}_t(\mathbf{x} = [0, y, 0]^\top, \vec{\mathbf{c}}) = 0$, namely

$$\vec{\mathbf{u}}_t^{\boldsymbol{\xi}} = \int_0^{\xi} \frac{\mathrm{d}}{\mathrm{d}\zeta}\vec{\mathbf{u}}_t(\zeta, \vec{\mathbf{c}})\,\mathrm{d}\zeta. \tag{C.13}$$

The infinitesimal adherence force at a specific contact length coordinate ξ is determined by the product between (C.13) and the tread tangential stiffness matrix $\mathbf{k}$

$$\vec{\mathbf{q}}_t^{\boldsymbol{\xi}} = -\nu \mathbf{K}_t \vec{\mathbf{u}}_t^{\boldsymbol{\xi}}.$$

Here, ν represents the tread pattern void ratio, and $\mathbf{K}_\mathrm{t}$ denotes the tangential stiffness matrix. This latter is typically expressed as

$$\mathbf{K}_\mathrm{t} = \begin{bmatrix} k_{xx} & k_{yx} \\ k_{yx} & k_{yy} \end{bmatrix},$$

where $k_{xy} \approx k_{yx} \ll k_{xx} \approx k_{yy}$, as proofed by Okonieski, Moseley, and Cai [243]. The total bristles force induced by the deformation in adherence is obtained by integrating the following over the entire adhesion region $\mathcal{A}$

$$\vec{\mathbf{F}}_a(\vec{\mathbf{c}}, t) = \int_{\mathcal{A}} \vec{\mathbf{q}}_\mathrm{t}^{\boldsymbol{\xi}} \, \mathrm{d}\boldsymbol{\xi} \,. \tag{C.14}$$

In the original brush model presented in [166], the self-aligning moment is attributed solely to the asymmetrical distribution of lateral shear stress along the contact length. However, in this study, we will also consider that the difference in longitudinal force on each side of the contact patch contributes to the generation of the self-aligning moment. Thereby, the adhesive self-aligning moment is defined as the cross-product between forces induced by deformation and the position vector (the sum of bristle position and carcass baseline position vectors), which reads

$$\vec{\mathbf{M}}_a(\vec{\mathbf{c}}, t) = \int_{\mathcal{A}} \vec{\mathbf{q}}_\mathrm{t}^{\boldsymbol{\xi}} \times \left(\mathcal{L}^{\boldsymbol{\xi}} + \boldsymbol{\xi} \right) \mathrm{d}\boldsymbol{\xi} \,. \tag{C.15}$$

SLIDE REGION When the local force induced by bristle deformation in adhesion is greater than the local maximum explicable static friction force, the bristle tips begin to slide over the ground with a micro-sliding velocity $\vec{\mathbf{v}}_s^{\boldsymbol{\xi}} \neq 0$. The local force exerted by bristle deformation in sliding opposes the micro-sliding velocity. Thus, its direction is denoted as

$$\vec{\mathbf{d}}_\mathrm{t}(\boldsymbol{\xi}, \vec{\mathbf{c}}) = \begin{bmatrix} d_{tx}(\boldsymbol{\xi}, \vec{\mathbf{c}}) \\ d_{ty}(\boldsymbol{\xi}, \vec{\mathbf{c}}) \end{bmatrix} = -\frac{\vec{\mathbf{v}}_s^{\boldsymbol{\xi}}}{\| \vec{\mathbf{v}}_s^{\boldsymbol{\xi}} \|} \,.$$

The total bristles force induced by the deformation in sliding is determined by integrating the following expression over the entire sliding region $\mathcal{S}$

$$\vec{\mathbf{F}}_s(\vec{\mathbf{c}}, t) = \int_{\mathcal{S}} \mu_d^{\boldsymbol{\xi}} \, q_z^{\boldsymbol{\xi}} \, \vec{\mathbf{d}}_\mathrm{t}^{\boldsymbol{\xi}} \, \mathrm{d}\boldsymbol{\xi} \,. \tag{C.16}$$

Similar to the adhesive scenario, the self-aligning moment is defined as the cross-product between the local force induced by the sliding and the position vector

$$\vec{\mathbf{M}}_s(\vec{\mathbf{c}}, t) = \int_{\mathcal{S}} \mu_d^{\boldsymbol{\xi}} \, q_z^{\boldsymbol{\xi}} \, \vec{\mathbf{d}}_\mathrm{t}^{\boldsymbol{\xi}} \times \left(\mathcal{L}^{\boldsymbol{\xi}} + \boldsymbol{\xi} \right) \mathrm{d}\boldsymbol{\xi} \,. \tag{C.17}$$

C.3 A NUMERICAL SOLUTION APPROACH

Due to the complexity of the problem outlined in the preceding sections, finding an analytical solution is not feasible. Therefore, we need to further analyze the contact zone between the tire and the road and set some more assumptions that will allow us to obtain an effective numerical scheme.

C.3.1 TIRE-ROAD CONTACT DISCRETIZATION

An essential characteristic of the contact patch set $\mathcal{P}$ is its constant compactness and D-convexity along the $\hat{\mathbf{e}}x$ direction [210, 244]. Leveraging the D-convexity properties enables us to discretize the contact patch into longitudinal sections. To optimize the intersection of the tire and road sets, we discretized the tire set $\mathcal{T}$ into a series of $n_{\mathcal{D}}$ ribs, similarly to the approach in [4, 219]. Each tire rib, denoted as $\mathcal{D}$, is characterized as a conical frustum which is located in $\mathbf{o} = [0, y_{\mathcal{D}}, 0]^\top$ in the $\mathbf{H}xyz$ coordinate system, with a radius of $R(y_{\mathcal{D}})$ and a width of $w_{\mathcal{D}}$

$$\mathcal{D} = \left\{ w_{\mathcal{D}}, y_{\mathcal{D}} \in \mathbb{R}, \mathbf{x} = [x, y, z]^\top \in \mathbb{R}^3 \mid \sqrt{x^2 + z^2} \leq R(y_{\mathcal{D}}), |y - y_{\mathcal{D}}| \leq \frac{w_{\mathcal{D}}}{2} \right\} .$$

This new representation allows us to build the discretized tire set $\mathcal{T}_{\mathcal{D}}$ as

$$\mathcal{T}_{\mathcal{D}} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{n_{\mathcal{D}}-1}, \mathcal{D}_{n_{\mathcal{D}}}\} .$$

Notice that we will employ a set of contact patch coordinate systems $\boldsymbol{\xi}_i$ equal to the ribs $n_{\mathcal{D}}$. These coordinate systems are redefined as

$$\boldsymbol{\xi}_i(x, y_i, t) = \begin{bmatrix} \xi_i(x, y_i, t) \\ \eta_i \\ \zeta \end{bmatrix} = \begin{bmatrix} \ell(y_i, t)/2 - x \\ y_i \\ z_i(x) \end{bmatrix} ,$$

where $z_i(x)$ represents the height of the rib i at the point x, considering the possibility of the contact patch being sloped in the longitudinal direction. Similar to our previous approach, the variables described in the reference system $\boldsymbol{\xi}_i$ will be denoted with the superscript "$\boldsymbol{\xi}_i$", *e.g.*, $\vec{\mathbf{u}}_{\mathbf{t}}(\boldsymbol{\xi}_i, \vec{\mathbf{c}})$. It is worth noting that this tire representation is fully adjustable in scale, allowing for the density of ribs representing the tire to be tailored based on the required accuracy and computational speed for a specific simulation.
By applying the considerations and assumptions outlined in Section C.2.3.5 to the constitutive relations, we can express the integrals of forces and moments relative to the adhesion and sliding zones as summations of the components from each rib. Specifically, if the transition coordinate ξ_s^i is assumed to be known, the integrals (C.14) and (C.15) can be analytically solved. Conversely, integrals (C.16) and (C.17) must be numerically calculated.

C.3.2 THE TRANSITION COORDINATE

Starting from the beginning of each rib contact region, the bristle tips stick to the ground up to the transition coordinate ξ_s^i. Determining ξ_s^i entails locating the precise point where the local force induced by bristle deformation in adhesion equals the local maximum static friction force that can be sustained. This requires applying the adhesion criteria outlined in (C.8)

$$\left(q_x^{\boldsymbol{\xi}^i}\right)^2 + \left(q_y^{\boldsymbol{\xi}^i}\right)^2 = \left(\mu_s q_z^{\boldsymbol{\xi}^i}\right)^2 . \tag{C.18}$$

Due to the complexity of the pressure distribution model and the shear field description, finding an analytical solution to equation (C.18) is not feasible. However, through calculus, it can be demonstrated that the longitudinal and lateral tangential stresses $q_x^{\boldsymbol{\xi}^i}$, $q_y^{\boldsymbol{\xi}^i}$, and normal stress $q_z^{\boldsymbol{\xi}^i}$ can be described by polynomial functions. Consequently, both the right- and left-hand sides of (C.18) are also polynomials. We can therefore exploit many powerful and robust root-finding algorithms, like Sturm's sequence and the Bisection methods [245] or the Algorithm 748 by Alefeld, Potra, and Shi [246].

C.3.3 TIRE NONLINEAR SYSTEM AND SOLUTION METHOD

As mentioned earlier, tire behavior is greatly influenced by carcass compliance. However, most tire models similar to the one presented in this paper (*i.e.*, Sakai's [221–224], Neo-Fiala [206, 212–215], TreadSim [225], and TaMeTire® [207]) lack or miss the sufficient discussion on the numerical method's performance in computing carcass deformation. This is a critical aspect, as the carcass deformation is the most challenging part of the tire model to tackle. Indeed, to solve the nonlinear system arising from carcass modeling (C.6), an efficient iterative approach satisfying RT constraints is necessary. Many tire models, such as those in [206, 212–215, 247, 248], employ Picard's Iteration Method (also known as the Fixed-Point Method), which is simple to implement but suffers from a slow convergence rate, potentially compromising model robustness. Alternatively, TaMeTire® employs a mixed Newton/quasi-Newton iterative method [207]. If the Jacobian $\mathbf{J_G}$[1] is unavailable, *quasi-Newton* methods can be utilized, albeit with a slight decrease in convergence rate and robustness. Notice that the Frobenius norm[2] of the Jacobian $\mathbf{J_F}$[3] is typically small ($\|\mathbf{J_F}\|_\mathrm{F} \ll 1$), while the Frobenius norm of the carcass stiffness matrix Jacbobian $\mathbf{J_K}$ is usually large ($\|\mathbf{J_K}\|_\mathrm{F} \gg 1$). Therefore, a good first approximation of $\mathbf{J_G}$ is given by

$$\mathbf{J_G} = \mathbf{J_K} - \mathbf{J_F} \approx \mathbf{J_K} .$$

If we set the initial iteration point at $\vec{\mathbf{c}} = [0, 0, 0]^\top$, the bottoming effect of the carcass stiffness is absent. As a consequence, the initial estimation of the Jacobian $\mathbf{J_G}$ can

[1] $\mathbf{J_G}(\vec{\mathbf{c}}) = \partial\vec{\mathbf{G}}(\vec{\mathbf{c}})/\partial\vec{\mathbf{c}}$.

[2] For a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$, the Frobenius norm is defined as $\|\mathbf{A}\|_\mathrm{F} = \sqrt{\sum_{ij} A_{ij}^2}$.

[3] $\mathbf{J_F}(\vec{\mathbf{c}}) = \partial\vec{\mathbf{F}}(\vec{\mathbf{c}})/\partial\vec{\mathbf{c}}$.

be straightforwardly computed by accounting solely for the structural and inflation pressure components of the carcass stiffness.

C.4 NUMERICAL PERFORMANCE AND VALIDATION

In this section, we present numerical simulations to validate the tire model and the numerical solution approach. Specifically, we will compare the performance of the quasi-Newton methods used to solve the nonlinear system of equations (C.6), as well as the fitting quality of the tire model to experimental data. A comparison with the Magic Formula is also provided.

C.4.1 PERFORMANCE OF THE QUASI-NEWTON METHODS

We evaluate the performance of the quasi-Newton numerical methods mentioned earlier. Specifically, we aim to identify the most suitable method for solving the nonlinear system of equations (C.6) among Picard Iteration Method (PIM), Greenstadt's 1st Method (G1M) [249], Greenstadt's 2nd Method (G2M) [249], Eirola-Nevanlinna Method (ENM) [250], Broyden's Bad Method (BBM) [251], Broyden's Good Method (BGM) [251], Broyden's Combined Method (BCM) [252], and their respective dumped versions, denoted with the prefix "D-". We will compare these algorithms based on function evaluations, iterations, convergence speed, and success ratio. Table C.1 presents the performance data of the quasi-Newton solvers. The tests are conducted on a set of 10^5 simulations with inputs distributed evenly within the following ranges $\sigma_x \in [-1, 1]$, $\sigma_y \in [-1, 1]$, $\varphi \in [-0.1, 0.1]$, $F_z \in [0, 5 \cdot 10^5]$, and $\gamma \in [-\pi/10, \pi/10]$, which represent typical operating conditions for a passenger car tire. All tests are performed on a shielded CPU. The simulation platform is a Concurrent® iHawk™ with 2.5 GHz Intel® Xeon® Silver 4215 8-core, 11 MB cache, 32 GB DDR4 RAM, and 64 bit RedHawk™ Linux RT operative system with CentOS distribution.

As depicted in Table C.1, both the PIM and D-PIM fail to converge to the solution with the required tolerance. The G1M and D-G1M can not complete the tests due to overflow issues. Conversely, the G2M and ENM, and their dumped version, D-G2M and D-ENM, show a good success ratio but exhibited high average and variance iterations and function evaluations. Although the pure BBM suffered from overflow issues, its dumped version, D-BBM, successfully converged to the solution with the required tolerance and maintained an acceptable success ratio. The BGM and BCM, as well as their dumped versions, D-BGM and D-BCM, demonstrated the best performance overall. They exhibited the highest success ratio and the lowest average and variance in both iterations and function evaluations. Notably, the BCM showed superior computational efficiency. It is worth mentioning that no relaxation steps were accepted in D-BCM. Consequently, the performance of the BCM method is deemed excellent for meeting the demands of HRT simulations, *i.e.*, achieving a computational time of less than 1 ms. Only a small percentage of the tests required and a high number of

iterations, however, in these cases the vertical load and resulting carcass deformations exceeded the structural limits, leading to instability in the iterative process.
The bottom part of Table C.1 presents the performance of the BCM in relation to the number of ribs. Notably, the number of ribs does not impact the efficacy of the iterative method but merely affects the computational time. This arises from the fact that a change in the number of ribs does not alter the number of unknowns in (C.6). Consequently, the computational time scales linearly with the number of ribs. However, it is worth noting that the variance of the computational time increases more than linearly due to small yet significant variations in the performance of Algorithm 748.

C.4.2 VALIDATION OF THE TIRE MODEL

To demonstrate the effectiveness of the proposed fitting procedure, we validate the proposed tire model under pure slip conditions by comparing its predictions with those of a Magic Formula 5.2 [166] on the fitting of experimental data for a Hoosier® 18x6-10 LCO tire. As previously mentioned in the introduction, the dataset is obtained from the Calspan TIRF for the FSAE TTC testing program [226]. In Figures C.6, C.7 and C.8, we compare the fitting of the longitudinal and lateral forces and aligning moment for the Hoosier® tire at several discrete values of vertical load with zero camber angle. The results indicate that the proposed model provides data fitting comparable to the Magic Formula 5.2 for the lateral force, while a higher Root Mean Square (RMS) error is observed for the longitudinal force and aligning moment. Given the limited number of parameters and the simplicity of the proposed model, these results are considered satisfactory.

Table C.1: Comparison between quasi-Newton methods' performance based on a dataset comprising 10^5 tests. The inputs are evenly distributed within the ranges $\sigma_x \in [-1,1]$, $\sigma_y \in [-1,1]$, $\varphi \in [-0.1,0.1]$, $F_z \in [0, 5 \cdot 10^5]$, and $\gamma \in [-\pi/10, \pi/10]$, using a 325/660R13 race car tire discretized with 5 ribs. Convergence is considered achieved when $\|\mathbf{G}(\vec{c})\|_2 \leq 10^{-8}$ and $\|\vec{c}_{k+1} - \vec{c}_k\|_2 \leq 10^{-16}$. Maximum algorithm iteration and relaxation steps are set to 100 and 10, respectively. The bottom part of the table demonstrates the impact of changing the number of ribs on the BCM method's performance.
Symbols legend: min minimum number of evaluations/iterations, max maximum number of evaluations/iterations, μ average evaluations/iterations, σ^2 evaluations/iterations variance.

Quasi-Newton Methods Performance with 5 Ribs

Method	Time (μs)				Success
	min	max	μ	σ^2	
PIM	518.0	7949.0	617.2	13953.2	0.0%
G1M	–	–	–	–	Overflow
G2M	23.5	1089.4	69.0	5252.0	99.5%
ENM	25.0	2025.0	81.5	11002.8	99.4%
BBM	–	–	–	–	0.0%
BGM	22.6	948.1	62.5	1973.0	99.9%
BCM	22.9	264.4	55.7	885.9	100.0%
D-PIM	507.0	5383.0	1151.9	568491	0.0%
D-G1M	–	–	–	–	Overflow
D-G2M	21.5	3195.5	80.4	40321.9	99.5%
D-ENM	27.0	5207.7	97.5	52752.6	99.5%
D-BBM	22.3	1284.0	76.9	5126.3	100.0%
D-BGM	21.3	3238.9	67.0	8142.9	99.9%
D-BCM	22.5	316.8	62.1	1615.1	100.0%

Broyden Combined Method Performance

Ribs	Time (μs)				Success
	min	max	μ	σ^2	
1	12.1	89.0	19.4	20.6	100.0%
5	22.9	264.4	55.7	885.9	100.0%
10	106.9	361.8	167.3	1746.22	100.0%
15	158.0	523.5	249.4	3878.2	100.0%
20	210.1	693.3	332.9	7028.7	100.0%
25	262.7	858.4	414.0	10708.3	100.0%

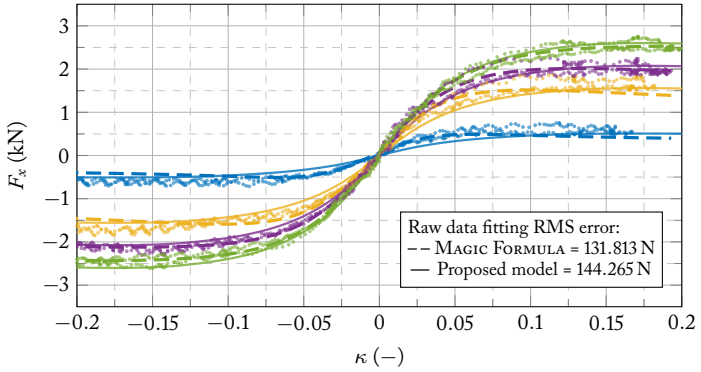


Figure C.6: Comparison between the Magic Formula and the proposed model regarding the fitting of longitudinal force experimental data for a Hoosier® 18x6-10 LCO tire. Experiments are performed under pure longitudinal slips at several discrete values of vertical load with zero camber angle. The dataset is provided by the Calspan TIRF for the FSAE TTC testing program [226]. *Lines and marks legend*: • experimental data, — proposed model, and -- Magic Formula 5.2. *Colors legend*: ■ $F_z = 222\,\text{N}$, ■ $F_z = 444\,\text{N}$, ■ $F_z = 667\,\text{N}$, ■ $F_z = 890\,\text{N}$, and ■ $F_z = 1120\,\text{N}$.

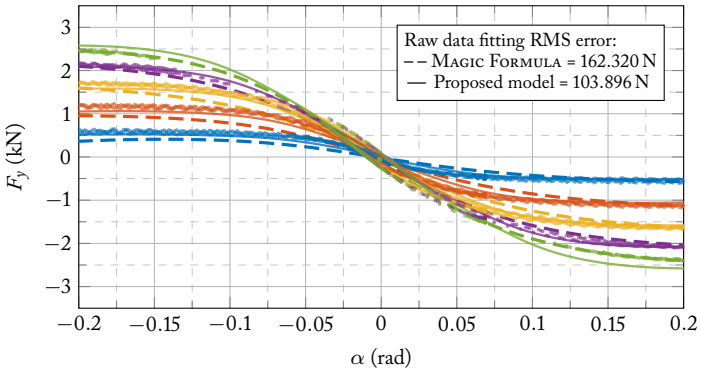


Figure C.7: Comparison between the Magic Formula and the proposed model regarding the fitting of lateral force experimental data for a Hoosier® 18x6-10 LCO tire. Experiments are performed under pure lateral slips at several discrete values of vertical load with zero camber angle. The dataset is provided by the Calspan TIRF for the FSAE TTC testing program [226]. *Lines and marks legend*: • experimental data, — proposed model, and -- Magic Formula 5.2. *Colors legend*: ■ $F_z = 222\,\text{N}$, ■ $F_z = 444\,\text{N}$, ■ $F_z = 667\,\text{N}$, ■ $F_z = 890\,\text{N}$, and ■ $F_z = 1120\,\text{N}$.

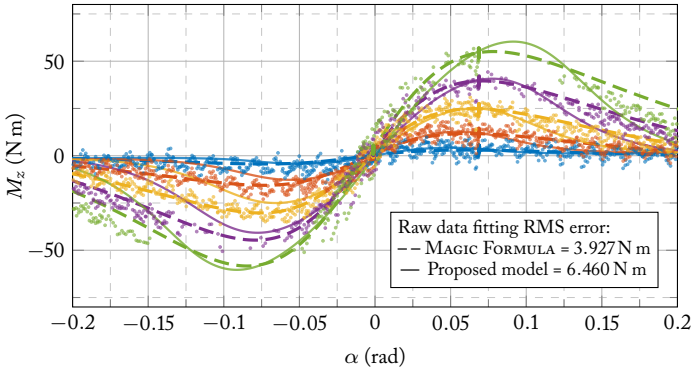


Figure C.8: Comparison between the Magic Formula and the proposed model regarding the fitting of self-aligning moment experimental data for a Hoosier® 18x6-10 LCO tire. Experiments are performed under pure lateral slips at several discrete values of vertical load with zero camber angle. The dataset is provided by the Calspan TIRF for the FSAE TTC testing program [226]. *Lines and marks legend*: • experimental data, — proposed model, and – – Magic Formula 5.2. *Colors legend*: ■ $F_z = 222$ N, ■ $F_z = 444$ N, ■ $F_z = 667$ N, ■ $F_z = 890$ N, and ■ $F_z = 1120$ N.

APPENDIX D

SYMBOLIC-NUMERICAL ANALYSIS AND SOLUTION OF STRUCTURES

Understanding how structures react to external forces and imposed displacements is fundamental in engineering. This knowledge is essential for predicting system behavior, optimizing performance, and ensuring safety. Structural analysis involves evaluating deformations and internal reactions under specified BCs, a process foundational in structural mechanics. Typically, structural analysis is carried out numerically using the DSM, a Finite Element Method (FEM) implementation known for solving large systems of equations. However, this method's principles can also be applied in a symbolic or hybrid symbolic-numerical fashion. This dual approach is valuable for reducing computational load, as symbolic solutions can be simplified and translated into efficient code for simulations. Moreover, symbolic DSM facilitates model reduction, enabling the creation of smaller-scale models for faster simulations. Despite its advantages, symbolic computation has limitations, particularly in solving large linear systems of equations, which can be computationally intensive and may exceed software capabilities. This chapter introduces TrussMe-Fem, a toolbox built on the DSM, harnessing symbolic computation from Maple® and numerical capabilities from Matlab® for structural analysis. The toolbox facilitates both symbolic and hybrid symbolic-numerical analyses, streamlining code generation for efficient simulations. To address challenges in the symbolic solution of large linear systems, TrussMe-Fem incorporates the routines for symbolic matrix factorization previously presented in Chapter 3, employing hierarchical representation for managing large expressions in the symbolic solution.

D.1 INTRODUCTION TO SYMBOLIC STRUCTURAL ANALYSIS

Structural mechanics is foundational in engineering for predicting system behavior under external forces. Achieving optimal performance in mechanical systems requires refining analytical processes to ensure precision and reliability, especially with the increasing adoption of complex structures. Structural analysis typically involves evaluating deformations and internal reactions under specified BCs, such as applied loads and displacements. Accurately predicting these responses is crucial for identifying critical areas and optimizing performance during the design process. Moreover, it is essential for simulating dynamic behavior during the simulation phase.

Traditionally, deformation and internal reaction analysis are performed numerically using FEM, which was introduced in the mid-20^{th} century and is now widely utilized for analyzing complex structures with relative ease [253]. It is important to note that FEM is a general concept based on discretizing structures into finite elements, leading to multiple implementation approaches with varying performance and capabilities across software platforms. Specifically, the FEM implementation used in this chapter is the DSM, introduced by Turner in 1959 [254] and further refined in 1964 [255]. DSM involves assembling stiffness matrices of individual elements into a global stiffness matrix, which is then used to solve equations relating node displacements and rotations to internal and external forces and moments. BCs are enforced by appropriately modifying the global stiffness matrix, force, and displacement vectors. Consequently, DSM is a versatile method applicable to various structures, including trusses, beams, frames, and plates.

The drive to optimize product performance while minimizing design and production costs necessitates parametric studies and optimization techniques. However, traditional numeric design optimization brings complexities, requiring numerous simulations and substantial computational resources. In response, symbolic computation emerges as a valuable alternative. While the DSM is typically associated with the numerical solution of large equation systems, its theoretical foundations enable symbolic or hybrid symbolic-numerical structural analysis. Symbolic solutions provide crucial derivatives of the solution concerning structural parameters, essential for fast optimizations. Additionally, symbolic approaches facilitate lean code generation, exploiting expression simplification for efficient simulations. Symbolic DSM also aids model reduction by deriving smaller models to reduce simulation time. Despite its advantages, symbolic computation poses challenges, particularly in solving large linear equation systems due to software limitations [103, 105] (see Chapter 3).

Symbolic computation in structural analysis has been explored in literature [101, 256]. Successful attempts include symbolic manipulation in stiffness matrix generation [257] and solving equation systems [258, 259]. However, as highlighted in [256], the symbolic structural solution remains limited by symbolic kernel capabilities, restricting the analysis to relatively small structures. Nonetheless, recent decades have seen significant advancements in solving large linear equation systems symbolically [103, 105]. These advances, utilizing symbolic matrix factorization with hierarchical representation, en-

able solutions previously deemed impractical.
The absence of a symbolic analysis and solution package for structures within the MAPLE® environment spurred the development of this toolbox[1]. Specifically, the authors' necessity for a tool to symbolically represent, solve, and generate code for modeling, simulating, and optimizing compliant mechanisms prompted the creation of TRUSSME-FEM. Leveraging the symbolic computation capabilities of MAPLE® and the numerical efficiency of MATLAB®, this toolbox constitutes a framework for symbolic-numerical analysis and solution of structures. Grounded in the aforementioned DSM, the solution method enables modeling, analysis, and resolution of structures. Depending on the symbolic kernel's capabilities, available computation time, and problem complexity, the symbolic solution can be found either symbolically in closed form or numerically. In both scenarios, a MATLAB® class can be generated to efficiently evaluate the symbolic solution or solve the problem numerically. Throughout code generation, model inputs and class internal data are also mapped into the generated code. Particular emphasis is placed on the symbolic solution of linear equation systems arising from the DSM, where the symbolic matrix factorization routines from Chapter 3 are employed. Notably the toolbox, named TRUSSME-FEM, along with its optional dependencies(LEM [23] and LAST [24]), is released as open-source software under the BSD 3-Clause License [28].

D.2 THE DIRECT STIFFNESS METHOD

The forthcoming discussion of the solution method is indispensable for grasping the toolbox implementation and is not intended to offer an exhaustive review of the DSM. For a more thorough understanding of DSM theory, please consult [254, 255, 261, 262]. As delineated in [263], DSM furnishes a systematic approach to analyzing statically determinate and indeterminate structures. The structural analysis problem can be formulated as follows: given a structure subjected to external loads and/or prescribed displacements, determine the displacements and rotations of the nodes, along with the internal forces and moments within the elements. These displacements and rotations of nodes are termed DOFs. The relationship between DOFs and the forces and moments acting on the structure (both internal and external) is expressed in the system of equations

$$\mathbf{K}^n \, \mathbf{d}^n = \mathbf{f}^n \, . \tag{D.1}$$

Here, $\mathbf{K}^n \in \mathbb{R}^{N \times N}$ represents the stiffness matrix, $\mathbf{d}^n \in \mathbb{R}^N$ denotes the displacement vector, $\mathbf{f}^n \in \mathbb{R}^N$ signifies the force vector, and N signifies the number of DOFs of the structure. The superscript n indicates that these quantities are expressed in the nodes' reference frames. Subsequent sections briefly outline the derivation of the components of the system of equations (D.1). Furthermore, the toolbox's solution method for effectively handling the system of equations is later discussed.

[1] It is worth noting that, unlike MAPLE®, MATHEMATICA® features a pre-existing suite of packages tailored for symbolic analysis of elastic structural elements, known as the *Structural Mechanics* package [260].

D.2.1 ELEMENT STIFFNESS MATRIX

To derive the global stiffness matrix $\mathbf{K}^n$ in (D.1), we must initially define the stiffness matrix $\mathbf{K}_e \in \mathbb{R}^{N_e \times N_e}$ for individual elements $e \in \mathcal{E}$, which connect a subset of the structure's nodes. Here, $\mathcal{E}$ denotes the set of elements in the structure, and N_e signifies the number of DOFs of the element. Unless specified otherwise, the matrix $\mathbf{K}_e$ is always assumed to be expressed in the element reference frame. Stiffness matrices are derived from various theories, such as the Euler-Bernoulli and the Timoshenko-Ehrenfest beam theories, the plane stress theory, etc. [262]. However, describing the derivation of such matrices $\mathbf{K}_e$ is beyond our scope, and they are assumed to be known.

D.2.2 ELEMENT STIFFNESS MATRIX CONDENSATION

The stiffness matrix $\mathbf{K}_e$ delineates how an element reacts when subjected to unit displacements imposed at each of its nodes' DOFs. Typically, when deriving such a matrix, it is assumed that the element is rigidly connected at each of its nodes' DOFs, thus possessing no internal DOFs. However, in certain structures, internal DOFs may arise due to links that allow for specific movements at nodes while constraining others. To address this scenario, the matrix $\mathbf{K}_e$ must be adjusted accordingly [261]. This adjustment, known as *condensation*, involves reordering rows and columns of the matrix $\mathbf{K}_e$ such that rows and columns corresponding to free nodal DOFs are relocated to the bottom-right corner of the matrix. This reordering, executed via a permutation matrix $\mathbf{P}_e$, facilitates the elimination of stiffness entries associated with free nodal DOFs from the original matrix $\mathbf{K}_e$. Consequently, a new condensed stiffness matrix $\mathbf{K}_{e,c} \in \mathbb{R}^{N_{e,c} \times N_{e,c}}$ is derived, where $N_{e,c}$ denotes the number of constrained DOFs of the element. The condensed stiffness matrix $\mathbf{K}_{e,c}$ is then computed through the following operations

$$\mathbf{P}_e \, \mathbf{K}_e \, \mathbf{P}_e^\top = \begin{bmatrix} \mathbf{K}_{e,11} & \mathbf{K}_{e,12} \\ \mathbf{K}_{e,21} & \mathbf{K}_{e,22} \end{bmatrix} \quad \xrightarrow[\text{condensation}]{\text{stiffness}} \quad \mathbf{P}_e \, \mathbf{K}_{e,c} \, \mathbf{P}_e^\top = \begin{bmatrix} \mathbf{K}_c & 0 \\ 0 & 0 \end{bmatrix} .$$

where $\mathbf{K}_c = \mathbf{K}_{e,11} - \mathbf{K}_{e,12} \, {\mathbf{K}_{e,22}}^{-1} \, \mathbf{K}_{e,21}$. It is important to apply condensation to each element of the structure as needed. Subsequently, the subscript c in $\mathbf{K}_{e,c}$ is dropped, and the condensed stiffness matrix is denoted simply as $\mathbf{K}_e$.

D.2.3 GLOBAL STIFFNESS MATRIX ASSEMBLAGE

The global stiffness matrix $\mathbf{K}^g$ is constructed by aggregating the stiffness contributions from individual elements of the structure. The superscript g signifies that these quantities are expressed in the global reference frame. To achieve this, the element's stiffness matrix $\mathbf{K}_e^g$ is initially derived by transforming $\mathbf{K}_e$ from the element reference frame to the global reference frame as follows

$$\mathbf{K}_e^g = \mathbf{T}_e^\top \, \mathbf{K}_e \, \mathbf{T}_e \, , \quad \text{where} \quad \mathbf{T}_e = \operatorname{diag}\left(\mathbf{R}_e, \ldots, \mathbf{R}_e\right) .$$

Note that $\mathbf{R}_e$ represents the rotation matrix from the e-th element reference frame to the global reference frame. Each transformed stiffness matrix $\mathbf{K}_e^g$ is then integrated into the global stiffness matrix $\mathbf{K}^g$ by incorporating the contributions of individual elements into their respective positions corresponding to the structure's DOFs. Consequently, the matrix $\mathbf{K}_e^g$ is multiplied by the injection matrix $\mathbf{J}_e \in \mathbb{R}^{N_e \times N}$, which is a matrix of zeros except for the positions corresponding to the structure's DOFs, which are set to one. Thus, the global stiffness matrix $\mathbf{K}^g$ is formulated as follows

$$\mathbf{K}^g = \sum_{e \in \mathcal{E}} \mathbf{J}_e^\top \, \mathbf{K}_e^g \, \mathbf{J}_e = \sum_{e \in \mathcal{E}} \mathbf{J}_e^\top \, \mathbf{T}_e^\top \, \mathbf{K}_e \, \mathbf{T}_e \, \mathbf{J}_e \,,$$

where $\mathcal{E}$ is the set of elements of the structure.
Following this transformation, we arrive at the linear system of equations (D.1) expressed in the global reference frame, denoted as $\mathbf{K}^g \, \mathbf{d}^g = \mathbf{f}^g$, where $\mathbf{d}^g$ and $\mathbf{f}^g$ represent the displacement and force vectors in the global reference frame, respectively. However, BCs for displacements and forces are typically applied in the nodes' reference frames. Consequently, to account for this, the system of equations (D.1) must be adjusted by incorporating the nodes' rotations $\mathbf{Q}_i$ with respect to the global reference frame, which are expressed in the block diagonal matrix $\mathbf{Q}$. This adjustment allows us to express the system (D.1) as

$$\underbrace{\mathbf{Q} \, \mathbf{K}^g \, \mathbf{Q}^\top}_{\mathbf{K}^n} \, \underbrace{\mathbf{Q} \, \mathbf{d}^g}_{\mathbf{d}^n} = \underbrace{\mathbf{Q} \, \mathbf{f}^g}_{\mathbf{f}^n} \,, \quad \text{where} \quad \mathbf{Q} = \operatorname{diag}\left(\mathbf{Q}_1, \ldots, \mathbf{Q}_{\mathcal{N}}\right) \,,$$

and $\mathcal{N}$ is the number of nodes of the structure.
To construct the nodal force vector $\mathbf{f}^n$ and displacement vector $\mathbf{d}^n$, it is necessary to incorporate the individual nodes' BCs into their corresponding DOFs. This process involves multiplying the i-th nodal force vector $\mathbf{f}_i^n$ and displacement vector $\mathbf{d}_i^n$ by the injection matrix $\mathbf{J}_i \in \mathbb{R}^{N_i \times N}$, where N_i represents the number of DOFs of the node. Consequently, the vectors $\mathbf{f}^n$ and $\mathbf{d}^n$ are respectively derived as

$$\mathbf{f}^n = \sum_{i \in \mathcal{N}} \mathbf{J}_i^\top \, \mathbf{f}_i^n \quad \text{and} \quad \mathbf{d}^n = \sum_{i \in \mathcal{N}} \mathbf{J}_i^\top \, \mathbf{d}_i^n \,.$$

D.2.4 LINEAR SYSTEM PARTITIONING AND SOLUTION

To solve the system of equations (D.1), it is beneficial to split it into two distinct sets of equations: the first set is solved for the free displacements $\mathbf{d}_f^n$, while the second set is solved for the reaction (internal) forces $\mathbf{f}_s^n$. This division is accomplished by rearranging and partitioning the stiffness matrix $\mathbf{K}^n$ into four submatrices $\mathbf{K}_{ff}^n$, $\mathbf{K}_{fs}^n$, $\mathbf{K}_{sf}^n$, and $\mathbf{K}_{ss}^n$, where the subscripts f and s denote the free and specified (or subject to BCs) DOFs, respectively. Similarly, the displacement and force vectors are partitioned accordingly, *e.g.*, $\mathbf{d}_f^n$, $\mathbf{d}_s^n$, and $\mathbf{f}_f^n$, $\mathbf{f}_s^n$. Subsequently, the system of equations (D.1) can be expressed

as

$$\underbrace{\begin{bmatrix} \mathbf{K}^n_{ff} & \mathbf{K}^n_{fs} \\ \mathbf{K}^n_{sf} & \mathbf{K}^n_{ss} \end{bmatrix}}_{\mathbf{P}^n\,\mathbf{K}^n\,(\mathbf{P}^n)^\top} \underbrace{\begin{bmatrix} \mathbf{d}^n_f \\ \mathbf{d}^n_s \end{bmatrix}}_{\mathbf{P}^n\,\mathbf{d}^n} = \underbrace{\begin{bmatrix} \mathbf{f}^n_f \\ \mathbf{f}^n_s + \mathbf{f}^n_r \end{bmatrix}}_{\mathbf{P}^n\,\mathbf{f}^n}, \tag{D.2}$$

where $\mathbf{P}^n$ denotes the permutation matrix responsible for rearranging the rows and columns of the stiffness matrix $\mathbf{K}^n$. It is noteworthy that the vector $\mathbf{f}^n_r$ is introduced to conveniently represent the forces and moments applied to the nodes' DOFs that are also subject to BCs. This vector is henceforth referred to as the "remainder" force vector. The linear system (D.2) is subsequently solved for the unknowns $\mathbf{d}^n_f$ and $\mathbf{f}^n_s$ through the following operations

$$[\text{solve for } \mathbf{d}^n_f] \quad \mathbf{K}^n_{ff}\mathbf{d}^n_f = \mathbf{f}^n_f - \mathbf{K}^n_{fs}\,\mathbf{d}^n_s\,, \tag{D.3a}$$

$$[\text{compute } \mathbf{f}^n_s] \quad \mathbf{f}^n_s = \mathbf{K}^n_{sf}\mathbf{d}^n_f + \mathbf{K}^n_{ss}\,\mathbf{d}^n_s - \mathbf{f}^n_r\,. \tag{D.3b}$$

As will be elucidated later, obtaining a symbolic solution for (D.3a) is not always feasible due to limitations in software capabilities. In such instances, one can resort to the numerical solution by performing a numerical factorization of the $\mathbf{K}^n_{ff}$ matrix and evaluating both (D.3a) and (D.3b). If one's sole interest lies in the free displacements $\mathbf{d}^n_f$ and not in the specified forces (or support reactions) $\mathbf{f}^n_f$, it suffices to compute the solution for (D.3a) only.

D.3 SOFTWARE DESCRIPTION

This section outlines the implementation of the TrussMe-Fem toolbox, which comprises two main components: the symbolic and numerical parts. The symbolic component, developed in Maple®, facilitates the symbolic analysis of the structure. Conversely, the numerical component, developed in Matlab®, is utilized for numerically evaluating the symbolic expressions generated earlier. Each component is discussed in detail, accompanied by usage examples.

D.3.1 THE SYMBOLIC COMPUTATION IN MAPLE

As previously stated, the TrussMe-Fem package serves as a symbolic implementation of the DSM method for assembling and solving structures. The package's interface is designed with the philosophy of offering users a collection of user-friendly functions that can be seamlessly combined to conduct structural analysis. TrussMe-Fem is structured to be employed in a step-by-step manner. Users are prompted to define the nodes, elements, and loads of the structure, followed by the generation of the structure's model. To accomplish this, the following procedure is recommended.

1. **Node Definition**: Specify the nodes of the structure, including their names, coordinates, reference frames, DOFs, and BCs on nodal displacements and rotations.

2. **Material Definition**: Define the materials used in the structure, providing information such as name, Young's modulus, Poisson's ratio, and density.
3. **Element Definition**: Define the elements connecting the nodes, specifying their names, node connections, reference frames, material properties, and cross-section details. Available element types include spring, rod, beam, or a generic element which serves as a base class for custom element definitions with user-defined stiffness matrices.
4. **Load Definition**: Specify the loads acting on the structure, including their names, application nodes, and magnitudes.

5a. **Model Generation**: Combine the defined nodes, elements, and loads to create a unified model of the structure. This model encapsulates components such as the global stiffness matrix, load vector, displacement vector, and permutation matrix used to reorder the system of equations in the form (D.2).

5b. **Solvability Check (Optional)**: Analyze the generated FE model to ascertain if the structure retains under-constrained DOFs. This step involves automatically checking the solvability of the system of equations (D.3a). If the system is singular but solvable, indicating zero entries in the stiffness matrix and load vector corresponding to under-constrained DOFs, the TrussMe-Fem package can identify and address this issue, issuing a warning message to notify the user.

6. **System Solution**: Solve the system of equations to obtain the solution in symbolic form. If a closed-form solution is unavailable or if numerical evaluation is desired, the code-generation process can be employed to obtain the solution in numerical form.

D.3.1.1 SYMBOLIC COMPUTATION USAGE EXAMPLE

Consider the depicted simple structure illustrated in Figure D.1. This structure comprises three nodes (N_1, N_2, N_3) interconnected by two elements (E_1, E_1). Nodes N_1 and N_3 are fully constrained in all DOFs, while node N_2 serves as a hinge connecting the two elements. A vertical load F is applied at node N_2. Both elements are beams characterized by a cross-section A and moment of inertia I_z, and are constructed from a material with Young's modulus E.
To describe the structure in TrussMe-Fem, we begin by defining the nodes. The `MakeNode` function is utilized for this purpose.

```
> N1 := MakeNode("N1", <0,0,0>, dofs=<0,0,0,0,0,0>):
> N2 := MakeNode("N2", <L_1,0,0>, dofs=<1,1,1,1,1,1>):
> N3 := MakeNode("N3", <L_1+L_2,0,0>, dofs=<0,0,0,0,0,0>):
```

This function requires the node name, coordinates, and DOFs as inputs. In this case, we have not specified reference frames, so the default ground frame (the identity matrix) is chosen for both nodes. Similarly, no constraints on displacements are specified,

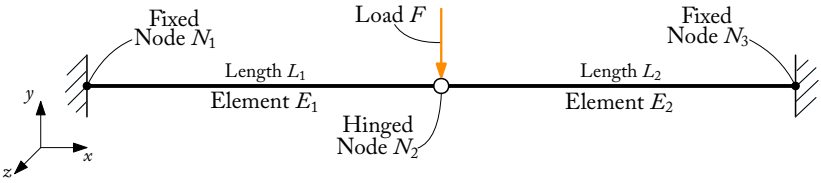


Figure D.1: Simple structure used to demonstrate the usage of the TrussMe-Fem package.

so they are assumed to be zero. The DOFs are set using a six-element vector. The first three elements represent displacements along the x-, y-, and z-axes, while the remaining three represent rotations about these axes. A value of 0 indicates a constrained DOF, while 1 indicates a free DOF. Here, nodes N_1 and N_3 are constrained in all directions, while N_2 is free. The hinge modeling at node N_2 will be explained later in this section.

Before defining the elements, the material properties must be specified. Custom materials can be defined using the `MakeMaterial` function, which requires the material name, Young's modulus, Poisson's ratio, and density as inputs. If the shear modulus is not provided, it is calculated from Young's modulus and Poisson's ratio. In this case, the material is defined as follows.

```
> M := MakeMaterial(name="GenericMaterial", elastic_modulus=E,
shear_modulus=G, poisson_ratio=nu, density=rho):
```

Next, the `MakeBeam` function is employed, necessitating the element's name, nodes at its ends, material, and cross-section properties.

```
> E1 := MakeBeam("E1", N1, [N2, <0,0,0,0,0,0>], material=M, area=A,
    inertia=[I_x,I_y,I_z], frame=GenerateFrameXY(N1["coordinates"],
    N2["coordinates"], [0,0,1])):
> E2 := MakeBeam("E2", [N2, <0,0,0,0,0,1>], N3, material=M, area=A,
    inertia=[I_x,I_y,I_z], frame=GenerateFrameXY(N2["coordinates"],
    N3["coordinates"], [0,0,1])):
```

During the element definition, the reference frame is also specified. Here, the `GenerateFrameXY` function aligns the coordinate system axes based on the coordinates of the two nodes. Additional details about how the element ends are connected to the nodes can be provided using an augmented six-element vector, such as `[N, <1,1,1,1,1,1>]`. If this vector is not provided, the element ends are assumed to be rigidly connected to the nodes in all directions. In the provided code snippet, element E_1 is connected rigidly to nodes N_1 and N_2, while element E_2 is hinged to node N_2 with constrained translational DOFs and free rotational DOFs. Element E_2 is then rigidly connected to node N_3. This hinge connection is achieved by specifying the

free rotational DOFs of the element ends. In this case, element E_2 can rotate freely around the z-axis.
Once the nodes and elements are defined, the loads acting on the structure can be specified using the `MakeLoad` function.

```
> F := MakeLoad("F", N2, <0,-P,0,0,0,0>):
```

Similarly, the `MakeLoad` function requires the load name, the node where the load is applied, and its magnitude as inputs. Optionally, the load reference frame can be specified using the `frame` parameter. If not specified, the node's reference frame is used. In this scenario, the load is applied to node N_2 in the y-direction.
Subsequently, the FE model of the structure can be generated using the `GenerateFEM` function, followed by solving the linear system of equations with `SolveFEM`.

```
> fem := GenerateFEM([N1,N2,N3], [E1,E2], [F], tryhard):
> SolveFEM(fem, use_LEM=false, use_LAST=false):
```

When using the `tryhard` flag, the solvability check described in step (5b) of the list above is activated. To enhance performance, the TrussMe-Fem package is designed to complement the LEM and LAST packages, which help manage expression expansion during the symbolic solution procedure. Detailed instructions on utilizing these packages can be found in Section 3.4 and Section 3.5, as well as in their respective documentation [23, 24]. If both libraries are unavailable, the toolbox defaults to utilizing the linear algebra routines built into Maple®. If the LEM and LAST packages are preferred for solving the linear system of equations, the `use_LEM` and `use_LAST` flags must be set to `true`. In such cases, the solution may be obtained in a veiled form. To unveil the solution, the `use_LEM` flag must be set to `false`. Once the linear system of equations is solved, the solution is stored in the `fem` table.

```
> f = fem["f"]^%T; d = fem["d"]^%T;
```

$$f = \left[0,\ \frac{PL_2^3}{L_1^3+L_2^3},\ 0,\ 0,\ 0,\ \frac{PL_1L_2^3}{L_1^3+L_2^3},\ 0,\ -P,\ 0,\ 0,\ 0,\ 0,\ 0,\ \frac{PL_1^3}{L_1^3+L_2^3},\ 0,\ 0,\ 0,\ -\frac{PL_1^3L_2}{L_1^3+L_2^3} \right]$$

$$d = \left[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ \frac{PL_1^3L_2^3}{3EI_z(L_1^3+L_2^3)},\ 0,\ 0,\ 0,\ \frac{PL_1^2L_2^2}{2EI_z(L_1^3+L_2^3)}\ 0,\ 0,\ 0,\ 0,\ 0,\ 0 \right]$$

In cases where no symbolic solution could be obtained, or if the user just prefers to assess the solution numerically, the `GenerateMatlabCode` function can be employed.

```
> GenerateMatlabCode("FemClass", fem, path="./dir/", info="Usage example
    class", vars=[P], data=[I_x=4.0e-4, I_y=2.0e-4, I_z=2.0e-4, E=210.0e6,
    nu=0.33, A=5.0e-3, L_1=1.0, L_2=1.0]);
```

This function creates the `FemClass.m` file in the `./dir/` directory, which contains a class definition of the aforementioned FE model. During the code generation process, users can establish default class properties in the `data` field and variable parameters in the `vars` field. The latter is intended to represent parameters that may vary during the structure analysis, such as the load magnitude P. Further insights into the generated class and its application are elaborated in the subsequent section.

D.3.2 THE NUMERICAL COMPUTATION IN MATLAB

Since the symbolic solution of (D.3a) may not always be feasible, users have the option to employ a numerical solution. This involves numerically factorizing the $\mathbf{K}_{ff}^{n}$ matrix and evaluating (D.3). The numerical solution can be achieved within the MAPLE® environment through variable substitution or in MATLAB® post the code generation process. In the latter scenario, the TRUSSME-FEM MATLAB® toolbox comes into play. This toolbox is built upon the `TrussMe.System` base class, which is utilized to define the components of (D.3) and to establish a unified framework that can be leveraged by inherited classes. Within this abstract class, various methods are present, including several virtual methods that need to be implemented in the inherited classes:

- stiffness matrix $\mathbf{K}^{n}$;
- free-free stiffness matrix $\mathbf{K}_{ff}^{n}$;
- free-specified stiffness matrix $\mathbf{K}_{fs}^{n}$;
- specified-free stiffness matrix $\mathbf{K}_{sf}^{n}$;
- specified-specified stiffness matrix $\mathbf{K}_{ss}^{n}$;
- displacement vector $\mathbf{d}^{*}$;
- free displacement vector $\mathbf{d}_{f}^{n*}$;
- specified displacement vector $\mathbf{d}_{s}^{n}$;
- force vector $\mathbf{f}^{*}$;
- free force vector $\mathbf{f}_{f}^{n}$;
- specified force vector $\mathbf{f}_{s}^{n*}$;
- remainder force vector $\mathbf{f}_{r}^{n}$;
- DOFs permutation $\mathbf{P}^{n}$;
- getters and setters for the system data;

where (*) denotes that the functionality is accessible only when the symbolic solution of the system is available; otherwise, an empty vector is returned.

D.3.2.1 NUMERICAL COMPUTATION USAGE EXAMPLE

The TRUSSME-FEM MATLAB® toolbox serves to numerically assess the problem's solution. Utilizing the toolbox is straightforward, particularly when the code is generated using the TRUSSME-FEM MAPLE® package. Suppose we have generated the `FemClass.m` file through the TRUSSME-FEM MAPLE® package; this file encompasses the class definition of the structure. To employ the generated class, we first instantiate it.

```
>>> fem = FemClass();
```

Optionally, during instantiation, we have the liberty to assign values to the class's internal data.

```
>>> data.I_x = 4.0e-4; data.I_y = 2.0e-4; data.I_z = 2.0e-4; ...
    data.E = 210.0e9; data.E = 210.0e9; data.A = 5.0e-3; ...
    data.L_1 = 1.0; data.L_2 = 1.0;
>>> fem = FemClass(data);
```

Following instantiation, we can manipulate the internal data values either by setting or retrieving them using dedicated methods.

```
>>> fem.set_data_field('I_x', 4.0e-4);
>>> I_x = fem.get_data_field('I_x');
>>> fem.set_data(data);
>>> data = fem.get_data();
```

Once instantiated, we can acquire the components of the system of equations (D.3) through the following methods.

```
>>> x = [1000];
>>> v = fem.v(x);
>>> d = fem.d(x,v);
>>> f = fem.f(x,v);
```

It is worth noting that the vector `x` includes the system's parameters, such as the load value P of 1000 N, while the vector `v` holds the veiling variables that might have been retained during the symbolic solution computation. In cases where the symbolic solution is unavailable, numerical computation becomes the only viable alternative, which is achieved by invoking the `compute_d` and `compute_f` methods.

```
>>> x = [1000];
>>> v = fem.v(x);
>>> d = fem.compute_d(x,v);
>>> f = fem.compute_f(x,v);
```

The least squares solution can also be acquired by incorporating the tolerance value and maximum number of iterations into the `compute_d` and `compute_f` methods, such as `fem.compute_f(x, v, tol, iter)`. Alternative numerical solution methods, relying on constrained minimization of energy functional, can be utilized to solve the system of equations (D.3) [262]. However, these methods are not currently integrated into the TrussMe-Fem Matlab® toolbox.

D.4 EXAMPLE APPLICATIONS

In this section, we briefly showcase two example applications of the TrussMe-Fem toolbox. All examples revolve around the rear left double wishbone suspension of the Formula SAE *E-Agle Trento Racing Team* (*University of Trento*) vehicle [137]. The suspension system is depicted in Figure D.2. On the left side, a rendering of the system is shown along with the names of its main components, while on the right side, a schematic representation of the suspension system is presented. In the subsequent paragraphs, we delve into how the TrussMe-Fem toolbox can facilitate the design optimization of the suspension system and effectively incorporate the kinematics and compliance of the suspension system through model reduction. The outcomes obtained using the TrussMe-Fem toolbox are validated against results obtained from the commercial software Ansys®. For a comprehensive analysis of the results from the example applications, please refer to [8].

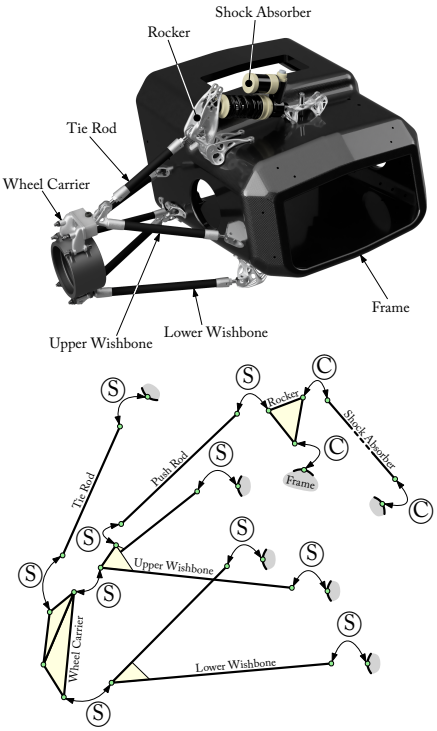


Figure D.2: Rendering and schematic representation of the rear left double wishbone suspension. The spherical and cylindrical constraints are respectively represented by the symbols Ⓢ and Ⓒ.

Before delving into the example applications, it is essential to clarify the workflow involving the TrussMe-Fem package. In this scenario, the TrussMe-Fem package is utilized to symbolically assemble the structure and simplify the expressions of its linear system components. Subsequently, the symbolic code is exported into a MATLAB® class, where internal data and input parameters of the system are specified. This MATLAB® class is then employed to numerically evaluate the solution of the problem within the SIMULINK® environment. This approach becomes necessary due to the significant slowdown of the symbolic kernel caused by the size and complexity of the resulting linear system.

D.4.1 DESIGN OPTIMIZATION

Mechanisms involving motion are prevalent in engineering applications. Typically, the design of such mechanisms assumes rigid bodies. However, in certain instances, the flexibility of the mechanism can significantly impact its performance. Optimization emerges as a potent tool for designing structures with optimal performance. The TrussMe-Fem toolbox facilitates shape optimization aimed at reducing the compliance of the mechanism and minimizing internal forces. Specifically, the presented optimizations aim to minimize both the wheel compliance hub angle θ_z and the tie rod axial force F_a by varying the x- and z- coordinates of the hard point connecting the tie rod to the chassis, referred to as point P_5 according to [8]. The results of these optimizations are depicted in Figures D.3 and D.4. These optimization demonstrations underscore that the current design does not represent the optimal solution in terms of minimum tie rod axial force and minimum wheel compliance hub angle. Optimum conditions are attained through a combination of values indicated by the green point. It is noteworthy that these optimization examples serve solely as a proof of concept for the model's parametric characteristics. In a real-world scenario, a multi-objective optimization considering compliance, structural analysis, and kinematic characteristics of the suspension would be imperative.

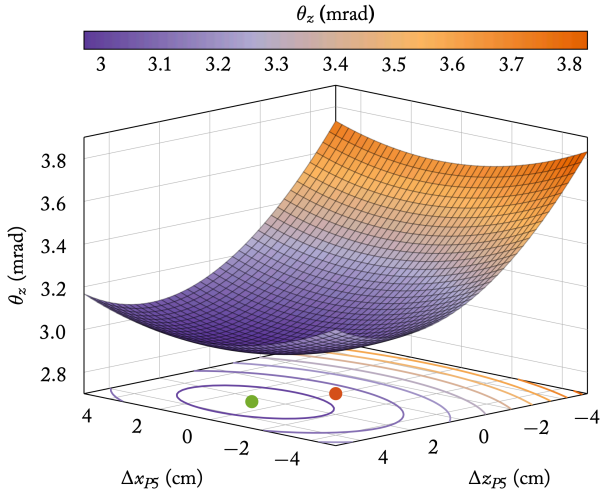


Figure D.3: Optimization of the point P_5 x- and z-coordinates to minimize the wheel compliance hub angle θ_z. The experiments are conducted under the application of a constant torque M_z of 0.4 kN m. *Marks legend:* ● current design, ● optimality condition.

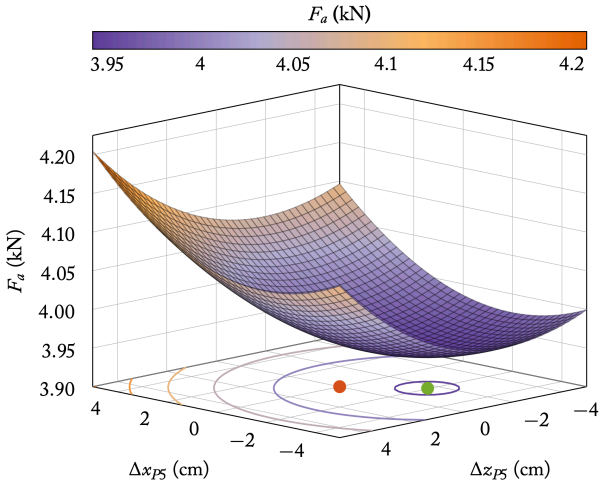


Figure D.4: Optimization of the point P_5 x- and z-coordinates to minimize the tie rod axial force F_a. The experiments are conducted under the application of a constant torque M_z of 0.4 kN m. *Marks legend:* ● current design, ● optimality condition.

D.4.2 MODEL REDUCTION

This example application explores the inclusion of suspension compliance in vehicle simulation using a hybrid symbolic-numerical approach. This methodology facilitates easy generalization and RT modification of model parameters without the need for code regeneration. The suspension's dynamic characteristics are modeled through a system of differential-algebraic equations, integrated using methods described in [8]. Depending on the modeling approach, suspension pick-up points' positions and tire force at the hub are extracted either through semi-analytical solutions or numerical integration. These are then utilized to calculate the suspension's compliance characteristics. The compliance contribution can be incorporated into the overall suspension system displacement as either a *steady-state* or *full dynamic* contribution [8]. The former approach reduces computational costs, while the latter yields accurate simulations at a higher computational expense. Results from this approach are compared with simulation data from commercial software Ansys®, showing good agreement under both static (Figure D.6 and Figure D.5) and dynamic conditions (Figure 5.9 in Chapter 5).

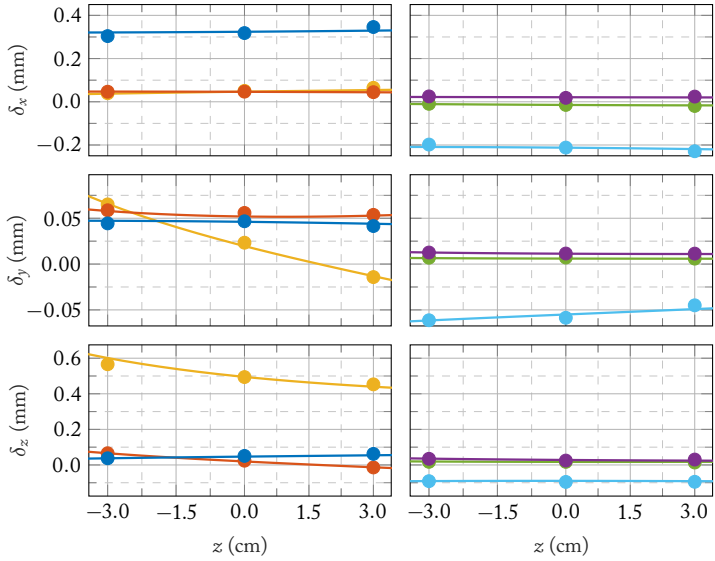


Figure D.5: Displacements of the wheel carrier reference frame in the x-, y-, and z-axes directions for different loads applied at the wheel hub. In the left-hand side of the figure, the forces are applied and torques are null. Conversely, the right-hand half side of the figure reports the results where torques are applied and forces are null. *Legend:* $F_x = 4.0\,\text{kN}$, $F_y = 4.0\,\text{kN}$, $F_z = 4.0\,\text{kN}$, with $M_x = M_y = M_z = 0.0\,\text{kN m}$. $M_x = 0.4\,\text{kN m}$, $M_y = 0.4\,\text{kN m}$, $M_z = 0.4\,\text{kN m}$, with $F_x = F_y = F_z = 0.0\,\text{kN}$.

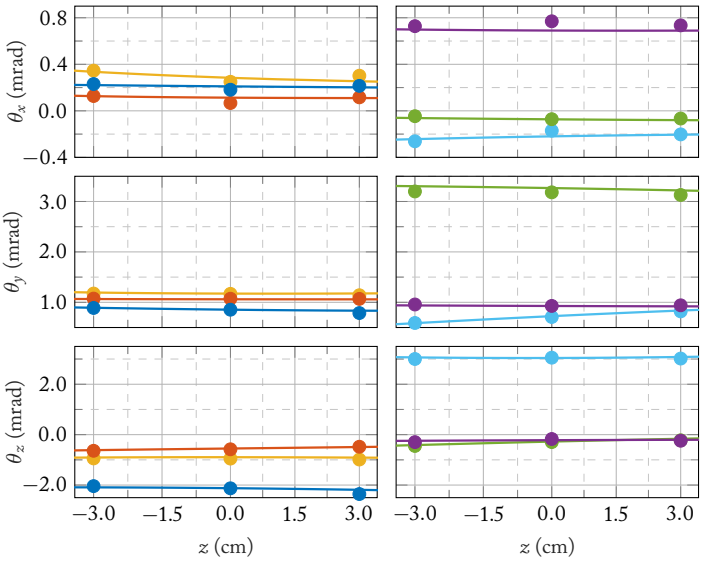


Figure D.6: Rotations of the wheel carrier reference frame around the x-, y-, and z-axes directions for different loads applied at the wheel hub. In the left-hand side of the figure, the forces are applied and torques are null. Conversely, the right-hand half side of the figure reports the results where torques are applied and forces are null. *Legend:* $F_x = 4.0\,\text{kN}$, $F_y = 4.0\,\text{kN}$, $F_z = 4.0\,\text{kN}$, with $M_x = M_y = M_z = 0.0\,\text{kN m}$. $M_x = 0.4\,\text{kN m}$, $M_y = 0.4\,\text{kN m}$, $M_z = 0.4\,\text{kN m}$, with $F_x = F_y = F_z = 0.0\,\text{kN}$.

BIBLIOGRAPHY

[1] D. Stocco and E. Bertolazzi. "Acme: A small 3D geometry library". In: *SoftwareX* 16 (Dec. 2021), p. 100845. ISSN: 2352-7110. DOI: 10.1016/j.softx.2021.100845.

[2] D. Stocco and M. Larcher. "A Symbolic-Numerical Approach to Index Reduction and Solution of Differential-Algebraic Equations". In: *Numerical Computations: Theory and Algorithms: Fourth International Conference, NUMTA 2023, Pizzo Calabro, Italy, June 14–20, 2023, Revised Selected Papers*. Lecture Notes in Computer Science. In press. Springer. 2024.

[3] D. Stocco and E. Bertolazzi. "Symbolic Matrix Factorization for Differential-Algebraic Equations Index Reduction". In: *Journal of Computational and Applied Mathematics* 448 (Oct. 2024), p. 115898. ISSN: 0377-0427. DOI: 10.1016/j.cam.2024.115898.

[4] D. Stocco, M. Larcher, F. Biral, and E. Bertolazzi. "A Novel Approach for Real-Time Tire-Ground Enveloping Modeling". In: *Journal of Computational and Nonlinear Dynamics* 19.6 (May 2024), p. 061005. ISSN: 1555-1423. DOI: 10.1115/1.4065439.

[5] D. Stocco, F. Biral, and E. Bertolazzi. "A physical tire model for real-time simulations". In: *Mathematics and Computers in Simulation* 223 (Sept. 2024), pp. 654–676. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2024.04.020.

[6] D. Stocco†, M. Larcher†, M. Tomasi, and E. Bertolazzi. "TrussMe-FEM: A Toolbox for Symbolic-Numerical Analysis and Solution of Structures". In: *Computer Physics Communications*. Submitted for publication.

[7] D. Stocco, M. Larcher, F. Biral, and E. Bertolazzi. "Solution of Differential-Algebraic Equations Through Symbolic Index Reduction". In: *Proceedings of the ASME International Mechanical Engineering Congress and Exposition, Portland, Oregon, USA, November 15-21, 2024*. Accepted for publication.

[8] M. Larcher†, D. Stocco†, M. Tomasi, and F. Biral. "A Symbolic-Numerical Approach to Suspension Kinematics and Compliance Analysis". In: *Proceed-*

ings of the ASME International Mechanical Engineering Congress and Exposition, Portland, Oregon, USA, November 15–21, 2024. Accepted for publication.

[9] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Jan. 1995. ISBN: 978-1611971224. DOI: 10.1137/1.9781611971224.

[10] C. de Dieuleveult, J. Erhel, and M. Kern. "A global strategy for solving reactive transport equations". In: *Journal of Computational Physics* 228.17 (Sept. 2009), pp. 6395–6410. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2009.05.044.

[11] M. Burger and M. Gerdts. "DAE Aspects in Vehicle Dynamics and Mobile Robotics". In: *Differential-Algebraic Equations Forum*. Springer, 2018, pp. 37–80. ISBN: 978-3030037185. DOI: 10.1007/11221_2018_6.

[12] Michael Blundell and Damian Harty. *Multibody systems approach to vehicle dynamics*. Elsevier, 2004. Chap. 5. ISBN: 978-0750651127. DOI: 10.1016/b978-0-7506-5112-7.x5000-3.

[13] J. G. De Jalon and E. Bayo. *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*. Springer, 1994. ISBN: 978-1461226000. DOI: 10.1007/978-1-4612-2600-0.

[14] L. Petzold. "Differential/Algebraic Equations are not ODE's". In: *SIAM Journal on Scientific and Statistical Computing* 3.3 (Sept. 1982), pp. 367–384. ISSN: 2168-3417. DOI: 10.1137/0903023.

[15] E. Griepentrog and R. März. *Differential-algebraic equations and their numerical treatment*. Leipzig: Teubner, 1986.

[16] J. S. Cohen. *Computer Algebra and Symbolic Computation: Elementary Algorithms*. AK Peters/CRC Press, 2002. ISBN: 978-0429064753. DOI: 10.1201/9781439863695.

[17] J. S. Cohen. *Computer Algebra and Symbolic Computation: Mathematical Methods*. AK Peters/CRC Press, 2003. ISBN: 978-0429064753. DOI: 10.1201/9781439863695.

[18] J. Andreasson, N. Machida, M. Tsushima, J. Griffin, and P. Sundström. "Deployment of high-fidelity vehicle models for accurate real-time simulation". In: *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*. Linköping University Electronic Press, May 2016. DOI: 10.3384/ecp1612478.

[19] E. Pankiewicz and W. Rulka. "From Off-Line to Real Time Simulations by Model Reduction and Modular Vehicle Modelling". In: *Proceedings of DETC03: Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Chicago, Illinois, USA, September 2-6, 2003*. ASME, Jan. 2003. DOI: 10.1115/detc2003/vib-48348.

[20] M. Piccinini, M. Larcher, E. Pagot, D. Piscini, L. Pasquato, and F. Biral. "A predictive neural hierarchical framework for on-line time-optimal motion planning and control of black-box vehicle models". In: *Vehicle System Dynamics* 61.1 (Feb. 2022), pp. 83–110. ISSN: 1744-5159. DOI: 10.1080/00423114.2022.2035776.

[21] M. Piccinini, S. Taddei, M. Larcher, M. Piazza, and F. Biral. "A Physics-Driven Artificial Agent for Online Time-Optimal Vehicle Motion Planning and Control". In: *IEEE Access* 11 (2023), pp. 46344–46372. ISSN: 2169-3536. DOI: 10.1109/access.2023.3274836.

[22] Y. Nakajima. *Advanced Tire Mechanics*. Springer, 2019. ISBN: 978-9811357992. DOI: 10.1007/978-981-13-5799-2.

[23] D. Stocco, E. Bertolazzi, and M. Larcher. *LEM GitHub Repository*. 2024. URL: http://github.com/stoccodavide/lem/.

[24] D. Stocco, E. Bertolazzi, and M. Larcher. *LAST GitHub Repository*. 2024. URL: http://github.com/stoccodavide/last/.

[25] D. Stocco and E. Bertolazzi. *Indigo GitHub Repository*. 2024. URL: http://github.com/stoccodavide/indigo/.

[26] D. Stocco and E. Bertolazzi. *ACME GitHub Repository*. 2024. URL: http://stoccodavide.github.io/acme.

[27] D. Stocco, E. Bertolazzi, and M. Larcher. *ENVE GitHub Repository*. 2024. URL: http://stoccodavide.github.io/enve/.

[28] D. Stocco and M. Larcher. *TrussMe-FEM Repository*. 2024. URL: http://github.com/stoccodavide/trussme-fem/.

[29] E. Bertolazzi and D. Stocco. *Lime GitHub Repository*. 2024. URL: http://github.com/ebertolazzi/lime/.

[30] E. Bertolazzi and D. Stocco. *LimeRickey GitHub Repository*. 2024. URL: http://github.com/ebertolazzi/limerickey/.

[31] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 1993. DOI: 10.1007/978-3-540-78862-1.

[32] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, 1996. ISBN: 978-3642052217. DOI: 10.1007/978-3-642-05221-7.

[33] V. Mehrmann. "Index Concepts for Differential-Algebraic Equations". In: *Encyclopedia of Applied and Computational Mathematics*. Springer, 2015, pp. 676–681. ISBN: 978-3540705291. DOI: 10.1007/978-3-540-70529-1_120.

[34] R. Lamour and R. März. "Detecting structures in differential algebraic equations: Computational aspects". In: *Journal of Computational and Applied Mathematics* 236.16 (Oct. 2012), pp. 4055–4066. ISSN: 0377-0427. DOI: 10.1016/j.cam.2012.03.009.

[35] W. C. Rheinboldt. "Differential-algebraic systems as differential equations on manifolds". In: *Mathematics of Computation* 43.168 (1984), pp. 473–482. ISSN: 1088-6842. DOI: 10.1090/s0025-5718-1984-0758195-5.

[36] C. C. Pantelides. "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2 (Mar. 1988), pp. 213–231. ISSN: 2168-3417. DOI: 10.1137/0909014.

[37] J. D. Pryce. "A Simple Structural Analysis Method for DAEs". In: *BIT Numerical Mathematics* 41.2 (2001), pp. 364–394. ISSN: 0006-3835. DOI: 10.1023/a:1021998624799.

[38] A. Benveniste, B. Caillaud, and M. Malandain. *Structural Analysis of Multimode DAE Systems: summary of results*. Tech. rep. RR-9387. France: Inria Rennes – Bretagne Atlantique, 2021, p. 27.

[39] G. Reißig, W. S. Martinson, and P. I. Barton. "Differential–Algebraic Equations of Index 1 May Have an Arbitrarily High Structural Index". In: *SIAM Journal on Scientific Computing* 21.6 (Jan. 2000), pp. 1987–1990. ISSN: 1095-7197. DOI: 10.1137/s1064827599353853.

[40] J. Unger, A. Kröner, and W. Marquardt. "Structural analysis of differential-algebraic equation systems—theory and applications". In: *Computers & Chemical Engineering* 19.8 (Aug. 1995), pp. 867–882. ISSN: 0098-1354. DOI: 10.1016/0098-1354(94)00094-5.

[41] S. E. Mattsson and G. Söderlind. "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives". In: *SIAM Journal on Scientific Computing* 14.3 (May 1993), pp. 677–692. ISSN: 1095-7197. DOI: 10.1137/0914043.

[42] R. Lamour, R. März, and C. Tischendorf. *Differential-Algebraic Equations: A Projector Based Analysis*. Springer, 2013. ISBN: 978-3642275555. DOI: 10.1007/978-3-642-27555-5.

[43] Roswitha März. "The index of linear differential algebraic equations with properly stated leading terms". In: *Results in Mathematics* 42.3-4 (Nov. 2002), pp. 308–338. ISSN: 1420-9012. DOI: 10.1007/bf03322858.

[44] R. März. "Characterizing differential algebraic equations without the use of derivative arrays". In: *Computers & Mathematics with Applications* 50.7 (Oct. 2005), pp. 1141–1156. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2005.08.015.

[45] R. Lamour, R. März, and C. Tischendorf. *Computational linear algebra aspects of projector based treatment of DAEs*. Tech. rep. Technical Report 11-17, Humboldt-University, Department of Mathematics, 2011.

[46] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Jan. 1998. ISBN: 978-1611971392. DOI: 10.1137/1.9781611971392.

[47] U. M. Ascher and L. R. Petzold. "Projected Implicit Runge-Kutta Methods for Differential-Algebraic Equations". In: *SIAM Journal on Numerical Analysis* 28.4 (Aug. 1991), pp. 1097–1120. ISSN: 1095-7170. DOI: 10.1137/0728059.

[48] J. Martín-Vaquero. "A 17th-order Radau IIA method for package RADAU. Applications in mechanical systems". In: *Computers & Mathematics with Applications* 59.8 (Apr. 2010), pp. 2464–2472. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2009.12.025.

[49] P. Amodio and F. Mazzia. "Boundary value methods for the solution of differential-algebraic equations". In: *Numerische Mathematik* 66.1 (Dec. 1993), pp. 411–421. ISSN: 0945-3245. DOI: 10.1007/bf01385705.

[50] P. Amodio and L. Brugnano. "Parallel implementation of block boundary value methods for ODEs". In: *Journal of Computational and Applied Mathematics* 78.2 (Feb. 1997), pp. 197–211. ISSN: 0377-0427. DOI: 10.1016/s0377-0427(96)00112-4.

[51] P. Amodio and F. Mazzia. "An algorithm for the computation of consistent initial values for differential-algebraic equations". In: *Numerical Algorithms* 19.1/4 (1998), pp. 13–23. ISSN: 1017-1398. DOI: 10.1023/a:1019175027639.

[52] E. Hairer and G. Wanner. "Stiff differential equations solved by Radau methods". In: *Journal of Computational and Applied Mathematics* 111.1-2 (Nov. 1999), pp. 93–111. ISSN: 0377-0427. DOI: 10.1016/s0377-0427(99)00134-x.

[53] E. Eich-Soellner and C. Führer. *Numerical Methods in Multibody Dynamics*. Springer, 1998. ISBN: 978-3663098287. DOI: 10.1007/978-3-663-09828-7.

[54] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software* 31.3 (Sept. 2005), pp. 363–396. ISSN: 1557-7295. DOI: 10.1145/1089014.1089020.

[55] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos. "Enabling New Flexibility in the SUNDIALS Suite of Nonlinear and Differential/Algebraic Equation Solvers". In: *ACM Transactions on Mathematical Software* 48.3 (Sept. 2022), pp. 1–24. ISSN: 1557-7295. DOI: 10.1145/3539801.

[56] J. Tolsma and P. I. Barton. "DAEPACK: An Open Modeling Environment for Legacy Models". In: *Industrial & Engineering Chemistry Research* 39.6 (May 2000), pp. 1826–1839. ISSN: 1520-5045. DOI: 10.1021/ie990734o.

[57] G. Bader and U. M. Ascher. "A New Basis Implementation for a Mixed Order Boundary Value ODE Solver". In: *SIAM Journal on Scientific and Statistical Computing* 8.4 (July 1987), pp. 483–500. ISSN: 2168-3417. DOI: 10.1137/0908047.

[58] E. Shmoylova, J. Gerhard, E. J. Postma, and A. D. Roche. "Simplification of Differential Algebraic Equations by the Projection Method". In: *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, April 19, University of Nottingham (UK)*. 2013, pp. 87–96. ISBN: 978-9175196213.

[59] S. L. Campbell and C. W. Gear. "The index of general nonlinear DAEs". In: *Numerische Mathematik* 72.2 (Dec. 1995), pp. 173–196. ISSN: 0945-3245. DOI: 10.1007/s002110050165.

[60] J. D. Pryce. "Solving high-index DAEs by Taylor series". In: *Numerical Algorithms* 19.1/4 (1998), pp. 195–211. ISSN: 1017-1398. DOI: 10.1023/a:1019150322187.

[61] R. März. "Differential-Algebraic Equations from a Functional-Analytic Viewpoint: A Survey". In: *Differential-Algebraic Equations Forum*. Springer, Nov. 2014, pp. 163–285. ISBN: 978-3319110509. DOI: 10.1007/978-3-319-11050-9_4.

[62] I. Higueras and R. März. "Differential algebraic equations with properly stated leading terms". In: *Computers & Mathematics with Applications* 48.1-2 (July 2004), pp. 215–235. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2003.05.010.

[63] Mathworks Matlab. *Solve Differential Algebraic Equations*. url: https://it.mathworks.com/help/matlab/math/solve-differential-algebraic-equations-daes.html. Online February 5, 2024. 2023.

[64] S. E. Mattsson and H. Elmqvist. "Modelica - An International Effort to Design the Next Generation Modeling Language". In: *IFAC Proceedings Volumes* 30.4 (Apr. 1997), pp. 151–55. ISSN: 1474-6670. DOI: 10.1016/s1474-6670(17)43628-7.

[65] S. E. Mattsson, H. Elmqvist, and M. Otter. "Physical system modeling with Modelica". In: *Control Engineering Practice* 6.4 (Apr. 1998), pp. 501–510. ISSN: 0967-0661. DOI: 10.1016/s0967-0661(98)00047-1.

[66] Yingbo Ma, Shashi Gowda, Ranjan Anantharaman, Chris Laughman, Viral Shah, and Chris Rackauckas. *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. 2021. arXiv: 2103.05244 [cs.MS].

[67] Wolfram Mathematica. *Numerical Solution of Differential-Algebraic Equations*. url: https://reference.wolfram.com/language/tutorial/NDSolveDAE.html. Online February 5, 2024. 2023.

[68] S. Chowdhry, H. Krendl, and A. A. Linninger. "Symbolic Numeric Index Analysis Algorithm for Differential Algebraic Equations". In: *Industrial & Engineering Chemistry Research* 43.14 (May 2004), pp. 3886–3894. ISSN: 1520-5045. DOI: 10.1021/ie0341754.

[69] Maplesoft Maple. *Introduction to Numerical Differential-algebraic Equation Solvers*. url: https://fr.maplesoft.com/support/helpJP/Maple/view.aspx?path=examples/numeric_DAE. Online February 5, 2024. 2023.

[70] E. J. Postma, J. Gerhard, E. Shmoylova, and A. D. Roche. *Exact parameter space reduction for numerically integrating parameterized differential equations*. US20120123746A1. May 2012.

[71] E. Shmoylova, J. Gerhard, E. J. Postma, and A. D. Roche. *Method and system for simplifying models*. US8483999B2. July 2013.

[72] E. J. Postma, J. Gerhard, E. Shmoylova, and A. D. Roche. *Exact parameter space reduction*. US20150142399A1. May 2015.

[73] N. S. Nedialkov, J. D. Pryce, and G. Tan. "Algorithm 948: DAESA—A Matlab Tool for Structural Analysis of Differential-Algebraic Equations: Software". In: *ACM Transactions on Mathematica Software* 41.2 (Feb. 2015). ISSN: 0098-3500. DOI: 10.1145/2700586.

[74] G. Tan, N. S. Nedialkov, and J. D. Pryce. "Symbolic-Numeric Methods for Improving Structural Analysis of Differential-Algebraic Equation Systems". In: *Mathematical and Computational Approaches in Advancing Modern Science and Engineering*. Springer, 2016, pp. 763–773. ISBN: 978-3319303796. DOI: 10.1007/978-3-319-30379-6_68.

[75] N. S. Nedialkov and J. D. Pryce. "Solving Differential-Algebraic Equations by Taylor Series (I): Computing Taylor Coefficients". In: *BIT Numerical Mathematics* 45.3 (Sept. 2005), pp. 561–591. ISSN: 1572-9125. DOI: 10.1007/s10543-005-0019-y.

[76] N. S. Nedialkov and J. D. Pryce. "Solving differential-algebraic equations by Taylor series (II): Computing the System Jacobian". In: *BIT Numerical Mathematics* 47.1 (Nov. 2006), pp. 121–135. ISSN: 1572-9125. DOI: 10.1007/s10543-006-0106-8.

[77] N. S. Nedialkov and J. D. Pryce. "Solving differential-algebraic equations by Taylor series (III): The DAETS code". In: *JNAIAM* 3.1-2 (2008), pp. 61–80. ISSN: 1790-8140.

[78] D. Estévez Schwarz and R. Lamour. "InitDAE: Computation of consistent values, index determination and diagnosis of singularities of DAEs using automatic differentiation in Python". In: *Journal of Computational and Applied Mathematics* 387 (May 2021), p. 112486. ISSN: 0377-0427. DOI: 10.1016/j.cam.2019.112486.

[79] P. Kunkel, V. Mehrmann, and I. Seufer. *GENDA: A software package for the solution of General Nonlinear Differential-Algebraic equations*. Tech. rep. 730-02. Institut für Mathematik, Technische Universität Berlin, 2002.

[80] P. Kunkel and V. Mehrmann. "A New Class of Discretization Methods for the Solution of Linear Differential-Algebraic Equations with Variable Coefficients". In: *SIAM Journal on Numerical Analysis* 33.5 (Oct. 1996), pp. 1941–1961. ISSN: 1095-7170. DOI: 10.1137/s0036142994240364.

[81] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations: Analysis and Numerical Solution*. EMS Press, Feb. 2006. ISBN: 978-3037195178. DOI: 10.4171/017.

[82] D. Estévez Schwarz and R. Lamour. "Diagnosis of singular points of properly stated DAEs using automatic differentiation". In: *Numerical Algorithms* 70.4 (Feb. 2015), pp. 777–805. ISSN: 1572-9265. DOI: 10.1007/s11075-015-9973-x.

[83] J. Bojarincev. *Regular, Singular Systems of Linear Ordinal Differential Equations*. 1980.

[84] C. W. Gear and L. R. Petzold. "ODE Methods for the Solution of Differential/Algebraic Systems". In: *SIAM Journal on Numerical Analysis* 21.4 (Aug. 1984), pp. 716–728. ISSN: 1095-7170. DOI: 10.1137/0721048.

[85] E. Griepentrog and R. März. "Basic Properties of Some Differential-Algebraic Equations". In: *Zeitschrift für Analysis und ihre Anwendungen* 8.1 (Feb. 1989), pp. 25–40. ISSN: 1661-4534. DOI: 10.4171/zaa/334.

[86] R. D. Jenks and R. S. Sutor. *Axiom, The Scientific Computation System*. Springer, 1992. ISBN: 978-1461229407. DOI: 10.1007/978-1-4612-2940-7.

[87] M. J. Wester. *Computer algebra systems: a practical guide*. Wiley, 1999.

[88] A. Heck. *Introduction to Maple*. Springer, 2003. ISBN: 978-1461300236. DOI: 10.1007/978-1-4613-0023-6.

[89] S. Wolfram. *The Mathematica Book*. 5th. Wolfram Media, Inc., 2003, p. 1488. ISBN: 978-1579550226.

[90] C. Creutzig and W. Oevel. *MuPAD Tutorial*. Springer, 2004. ISBN: 978-3642593048. DOI: 10.1007/978-3-642-59304-8.

[91] G. Rayna. *Reduce, Software for Algebraic Computation*. Springer, 1987. ISBN: 978-1461248064. DOI: 10.1007/978-1-4612-4806-4.

[92] W. H. Jeffreys. "TRIGMAN". In: *SIGSAM Bulletin* 24 (Oct. 1972), pp. 20–21. ISSN: 0163-5824. DOI: 10.1145/1086793.1086800.

[93] M. Gastineau and J. Laskar. "TRIP: a computer algebra system dedicated to celestial mechanics and perturbation series". In: *ACM Communications in Computer Algebra* 44.3/4 (Jan. 2011), pp. 194–197. ISSN: 1932-2240. DOI: 10.1145/1940475.1940518.

[94] H. Strubbe. "Presentation of the SCHOONSCHIP system". In: *SIGSAM Bulletin* 8.3 (Dec. 1974), pp. 55–60. ISSN: 0163-5824. DOI: 10.1145/1086837.1086845.

[95] S. R. Bourne. "CAMAL". In: *SIGSAM Bulletin* 24 (Oct. 1972), pp. 8–11. ISSN: 0163-5824. DOI: 10.1145/1086793.1086795.

[96] I. Frick. *The computer algebra system SHEEP, what it can and cannot do in general relativity*. Tech. rep. 1977.

[97] Xiaoye S. L. and J. W. Demmel. "Making Sparse Gaussian Elimination Scalable by Static Pivoting". In: *Proceedings of the IEEE/ACM SC98 Conference*. IEEE, 1998, pp. 34–34. DOI: 10.1109/sc.1998.10030.

[98] D. R. Stoutemyer. "A radical proposal for computer algebra in education". In: *SIGSAM Bulletin* 18-19.4-1 (Nov. 1984), pp. 40–53. ISSN: 0163-5824. DOI: 10.1145/1089355.1089366.

[99] R. Pavelle. "MACSYMA: Capabilities and applications to problems in engineering and the sciences". In: *Lecture Notes in Computer Science*. Springer, 1985, pp. 19–32. ISBN: 978-3540396840. DOI: 10.1007/3-540-15983-5_2.

[100] P. Mitic and P. G. Thomas. "Pitfalls and limitations of computer algebra". In: *Computers & Education* 22.4 (May 1994), pp. 355–361. ISSN: 0360-1315. DOI: 10.1016/0360-1315(94)90057-4.

[101] A. K. Noor and C. M. Andersen. "Computerized symbolic manipulation in structural mechanics—Progress and potential". In: *Computers & Structures* 10.1-2 (Apr. 1979), pp. 95–118. ISSN: 0045-7949. DOI: 10.1016/0045-7949(79)90077-4.

[102] A. R. Korncoff and S. J. Fenves. "Symbolic generation of finite element stiffness matrices". In: *Computers & Structures* 10.1-2 (Apr. 1979), pp. 119–124. ISSN: 0045-7949. DOI: 10.1016/0045-7949(79)90078-6.

[103] J. Carette, W. Zhou, J. Jeffrey D, and M. B. Monagan. "Linear algebra using Maple's LargeExpressions package". In: *Proceedings of Maple Conference*. 2006, pp. 14–25.

[104] W. Zhou, J. Carette, D. J. Jeffrey, and M. B. Monagan. "Hierarchical Representations with Signatures for Large Expression Management". In: *Lecture Notes in Computer Science*. Springer, 2006, pp. 254–268. ISBN: 978-3540397304. DOI: 10.1007/11856290_22.

[105] W. Zhou. "Symbolic Computation Techniques for Solving Large Expression Problems from Mathematics and Engineering". PhD thesis. University of Western Ontario, 2007.

[106] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Springer, 1992. ISBN: 978-0585332475. DOI: 10.1007/b102438.

[107] R. A. Demillo and R. J. Lipton. "A probabilistic remark on algebraic program testing". In: *Information Processing Letters* 7.4 (June 1978), pp. 193–195. ISSN: 0020-0190. DOI: 10.1016/0020-0190(78)90067-4.

[108] J. T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *Journal of the ACM* 27.4 (Oct. 1980), pp. 701–717. ISSN: 1557-735X. DOI: 10.1145/322217.322225.

[109] R. Zippel. "Probabilistic algorithms for sparse polynomials". In: *Lecture Notes in Computer Science*. EUROSAM '79. Springer, 1979, pp. 216–226. ISBN: 978-3540351283. DOI: 10.1007/3-540-09519-5_73.

[110] B. W. Char, G. J. Fee, K. O Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *On the design and performance of the Maple system*. Tech. rep. University of Waterloo, Department of Computer Science, 1984.

[111] G. H. Gonnet. "Determining equivalence of expressions in random polynomial time". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing - STOC '84*. ACM Press, 1984, pp. 334–341. DOI: 10.1145/800057.808698.

[112] G. H. Gonnet. "New results for random determination of equivalence of expressions". In: *Proceedings of the fifth ACM symposium on Symbolic and algebraic computation - SYMSAC '86*. ACM Press, 1986, pp. 127–131. DOI: 10.1145/32439.32465.

[113] M. B. Monagan and G. H. Gonnet. "Signature functions for algebraic numbers". In: *Proceedings of the international symposium on Symbolic and algebraic computation - ISSAC '94*. ACM Press, 1994, pp. 291–296. DOI: 10.1145/190347.190430.

[114] W. Zhou, D. J. Jeffrey, G. J. Reid, C. Schmitke, and J. McPhee. "Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems". In: *Lecture Notes in Computer Science*. Springer, 2005, pp. 31–43. ISBN: 978-3540321194. DOI: 10.1007/11499251_4.

[115] D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2015. ISBN: 978-3319167213. DOI: 10.1007/978-3-319-16721-3.

[116] A. George. "Nested Dissection of a Regular Finite Element Mesh". In: *SIAM Journal on Numerical Analysis* 10.2 (Apr. 1973), pp. 345–363. ISSN: 1095-7170. DOI: 10.1137/0710032.

[117] R. J. Lipton, D. J. Rose, and R. E. Tarjan. "Generalized Nested Dissection". In: *SIAM Journal on Numerical Analysis* 16.2 (Apr. 1979), pp. 346–358. ISSN: 1095-7170. DOI: 10.1137/0716027.

[118] H. M. Markowitz. "The Elimination form of the Inverse and its Application to Linear Programming". In: *Management Science* 3.3 (Apr. 1957), pp. 255–269. ISSN: 1526-5501. DOI: 10.1287/mnsc.3.3.255.

[119] D. J. Rose. "Symmetric elimination on sparse positive definite systems and the potential flow network problem". PhD thesis. Harvard University, 1970.

[120] M. Giesbrecht and N. Pham. "A Symbolic Approach to Compute a Null-Space Basis in the Projection Method". In: *Computer Mathematics*. Springer, 2014, pp. 243–259. ISBN: 978-3662437995. DOI: 10.1007/978-3-662-43799-5_19.

[121] W. Zhou and D. J. Jeffrey. "Fraction-free matrix factors: new forms for LU and QR factors". In: *Frontiers of Computer Science in China* 2.1 (Mar. 2008), pp. 67–80. ISSN: 1673-7466. DOI: 10.1007/s11704-008-0005-z.

[122] S. L. Campbell. "High-Index Differential Algebraic Equations". In: *Mechanics of Structures and Machines* 23.2 (Jan. 1995), pp. 199–222. ISSN: 0890-5452. DOI: 10.1080/08905459508905235.

[123] P. G. Thomsen and C. Bendtsen. *Numerical solution of differential algebraic equations (IMM-REP-1999-8)*. Tech. rep. DK-2800, Lyngby, Denmark: Technical University of Denmark, May 1999.

[124] J. Baumgarte. "Stabilization of constraints and integrals of motion in dynamical systems". In: *Computer Methods in Applied Mechanics and Engineering* 1.1 (June 1972), pp. 1–16. ISSN: 0045-7825. DOI: 10.1016/0045-7825(72)90018-7.

[125] W. Zhou, D. J. Jeffrey, and G. J. Reid. "Symbolic Computation Sequences and Numerical Analytic Geometry Applied to Multibody Dynamical Systems". In: *Symbolic-Numeric Computation*. Ed. by D. Wang and L. Zhi. Basel: Birkhäuser Basel, 2007, pp. 335–347. ISBN: 978-3764379841. DOI: 10.1007/978-3-7643-7984-1_20.

[126] E. Bayo, J. Garcia De Jalon, and M. A. Serna. "A modified Lagrangian formulation for the dynamic analysis of constrained mechanical systems". In: *Computer Methods in Applied Mechanics and Engineering* 71.2 (Sept. 1988), pp. 183–195. ISSN: 0045-7825. DOI: 10.1016/0045-7825(88)90085-0.

[127] R. A. Wehage and E. J. Haug. "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems". In: *Journal of Mechanical Design* 104.1 (Jan. 1982), pp. 247–255. ISSN: 0161-8458. DOI: 10.1115/1.3256318.

[128] R. McKenzie and J. Pryce. "Structural analysis based dummy derivative selection for differential algebraic equations". In: *BIT Numerical Mathematics* 57.2 (Feb. 2017), pp. 433–462. ISSN: 1572-9125. DOI: 10.1007/s10543-016-0642-9.

[129] S. Iwata, T. Oki, and M. Takamatsu. "Index Reduction for Differential-algebraic Equations with Mixed Matrices". In: *Journal of the ACM* 66.5 (July 2019), pp. 1–34. ISSN: 1557-735X. DOI: 10.1145/3341499.

[130] E. Hairer. "Symmetric Projection Methods for Differential-Algebraic Equations on Manifolds". In: *BIT Numerical Mathematics* 40.4 (2000), pp. 726–734. ISSN: 0006-3835. DOI: 10.1023/a:1022344502818.

[131] S. L. Campbell and E. Moore. "Constraint preserving integrators for general nonlinear higher index DAEs". In: *Numerische Mathematik* 69.4 (Feb. 1995), pp. 383–399. ISSN: 0945-3245. DOI: 10.1007/s002110050099.

[132] W. M. Lioen and J. J. B. de Swart. *Test set for initial value problem solvers*. Tech. rep. Centrum voor Wiskunde en Informatica, 1998.

[133] F. Mazzia, F. Iavernaro, and C Magherini. *Test set for initial value problem solvers*. Tech. rep. Department of Mathematics, University of Bari, 2003.

[134] F. Mazzia and C. Magherini. *Test set for initial value problem solvers*. Tech. rep. Department of Mathematics, University of Bari, 2008.

[135] F. Mazzia, J. R. Cash, and K. Soetaert. "A Test Set for stiff Initial Value Problem Solvers in the open source software R: Package deTestSet". In: *Journal of Computational and Applied Mathematics* 236.16 (Oct. 2012), pp. 4119–4131. ISSN: 0377-0427. DOI: 10.1016/j.cam.2012.03.014.

[136] MathWorks. *Solve Differential Algebraic Equations (DAEs)*. 2024. URL: http://it.mathworks.com/help/symbolic/solve-differential-algebraic-equations.html.

[137] University of Trento. *E-Agle Trento Racing Team*. www.eagletrt.it. 2023.

[138] K. E. Brenan. "Numerical simulation of trajectory prescribed path control problems by the backward differentiation formulas". In: *IEEE Transactions on Automatic Control* 31.3 (Mar. 1986), pp. 266–269. ISSN: 0018-9286. DOI: 10.1109/tac.1986.1104236.

[139] K. E. Brenan. "Stability and Convergence of Difference Approximations for Higher Index Differential-Algebraic Systems with Applications in Trajectory Control". PhD thesis. University of California, 1983.

[140] S. L. Campbell. "A general method for nonlinear descriptor systems: an example from robotic path control". In: *Proceedings of the 27th IEEE Conference on Decision and Control*. IEEE. DOI: 10.1109/CDC.1988.194386.

[141] D. Schwarz Estévez, R. Lamour, and R. März. "Singularities of the Robotic Arm DAE". In: *Differential-Algebraic Equations Forum*. Springer, 2020, pp. 433–480. ISBN: 978-3030539054. DOI: 10.1007/978-3-030-53905-4_14.

[142] C. Gear. "Simultaneous Numerical Solution of Differential-Algebraic Equations". In: *IEEE Transactions on Circuit Theory* 18.1 (1971), pp. 89–95. ISSN: 0018-9324. DOI: 10.1109/tct.1971.1083221.

[143] M. Gerdts. *Optimal Control of ODEs and DAEs*. De Guyter, Dec. 2012. ISBN: 978-3110249958. DOI: 10.1515/9783110249996.

[144] M. Gerdts. "Optimal control and real-time optimization of mechanical multibody systems". In: *ZAMM - Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 83.10 (Sept. 2003), pp. 705–719. ISSN: 1521-4001. DOI: 10.1002/zamm.200310067.

[145] M. Gerdts. "Gradient evaluation in DAE optimal control problems by sensitivity equations and adjoint equations". In: *PAMM* 5.1 (Nov. 2005), pp. 43–46. ISSN: 1617-7061. DOI: 10.1002/pamm.200510012.

[146] The CGAL Project. *CGAL User and Reference Manual*. 5.6. CGAL Editorial Board, 2023. URL: http://doc.cgal.org/5.6/Manual/packages.html.

[147] A. Jacobson, D. Panozzo, et al. *libigl: A simple C++ geometry processing library*. 2018. URL: http://libigl.github.io/.

[148] B. Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley Professional, 2013. ISBN: 978-0275967307.

[149] G. Guennebaud, B. Jacob, et al. *Eigen v3*. 2010. URL: http://eigen.tuxfamily.org.

[150] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al. *LAPACK Users' Guide*. 3rd. Philadelphia, Pennsylvania, USA: SIAM, 1999.

[151] H. Asyrani and A. Bade. "Bounding Volume Hierarchies for Collision Detection". In: *Computer Graphics*. InTech, Mar. 2012. DOI: 10.5772/35555.

[152] N. W. Eloe, J. A. Steurer, J. L. Leopold, and C. L. Sabharwal. "Dual graph partitioning for Bottom-Up BVH construction". In: *Journal of Visual Languages and Computing* 25.6 (Dec. 2014), pp. 764–771. ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2014.09.014.

[153] C. Ericson. *Real-Time Collision Detection*. USA: CRC Press, 2004. ISBN: 978-1558607323.

[154] S. M. Omohundro. *Five balltree construction algorithms*. Tech. rep. TR-89-063. Dec. 1989.

[155] Y.-S. Xing, X. P. Liu, and S.-P. Xu. "Efficient collision detection based on AABB trees and sort algorithm". In: *International Conference on Control and Automation (ICCA) 2010*. IEEE, June 2010. DOI: 10.1109/icca.2010.5524093.

[156] M. Frego and E. Bertolazzi. "Point-Clothoid Distance and Projection Computation". In: *SIAM Journal on Scientific Computing* 41.5 (Jan. 2019), A3326–A3353. ISSN: 1095-7197. DOI: 10.1137/18m1200439.

[157] E. Bertolazzi, P. Bevilacqua, and M. Frego. "Efficient intersection between splines of clothoids". In: *Mathematics and Computers in Simulation* 176 (Oct. 2020), pp. 57–72. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2019.10.001.

[158] P. J. Schneider and D. Eberly. *Geometric Tools for Computer Graphics*. USA: Elsevier, 2002. ISBN: 978-1558605947.

[159] D. Eberly. *Robust and Error-Free Geometric Computing*. 1st. CRC Press, 2020. ISBN: 978-0367352943. DOI: 10.1201/9780429330506.

[160] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. CRC Press, Oct. 2003. ISBN: 978-0429176364. DOI: 10.1201/9781482297997.

[161] A. S. Glassner, ed. *Graphics gems*. New York, USA: Academic Press Professional, Inc., 1990. ISBN: 978-0122861659.

[162] J. Arvo. *Graphic Gems II*. New York, USA: Academic Press Professional, Inc., 1991. ISBN: 978-0120644810.

[163] D. Kirk. *Graphic Gems III*. New York, USA: Academic Press Professional, Inc., 1992. ISBN: 978-0124096707.

[164] P. Heckbert. *Graphic Gems IV*. New York, USA: Academic Press Professional, Inc., 1994. ISBN: 978-0123361554.

[165] A. Paeth. *Graphic Gems V*. New York, USA: Academic Press Professional, Inc., 1995. ISBN: 978-0123991843.

[166] H. B. Pacejka. *Tire and Vehicle Dynamics*. 3rd. Butterworth-Heinemann, 2012. ISBN: 978-0080970165.

[167] H. B. Pacejka. "Spin: camber and turning". In: *Vehicle System Dynamics* 43.sup1 (Jan. 2005), pp. 3–17. ISSN: 1744-5159. DOI: 10.1080/00423110500140013.

[168] A. Gallrein, M. Bäcker, M. Burger, and A. Gizatullin. "An Advanced Flexible Realtime Tire Model and its Integration Into Fraunhofer's Driving Simulator". In: *SAE Technical Paper Series*. SAE International, Apr. 2014. DOI: 10.4271/2014-01-0861.

[169] R. Serban, J. Taves, and Z. Zhou. "Real-Time Simulation of Ground Vehicles on Deformable Terrain". In: *Journal of Computational and Nonlinear Dynamics* 18.8 (May 2023). ISSN: 1555-1423. DOI: 10.1115/1.4056851.

[170] A. J. C. Schmeitz. "A semi-empirical three-dimensional model of the pneumatic tyre rolling over arbitrarily uneven road surfaces". PhD thesis. TU Delft, 2004.

[171] D. C. Davis. "A Radial-Spring Terrain-Enveloping Tire Model". In: *Vehicle System Dynamics* 4.1 (Mar. 1975), pp. 55–69. ISSN: 1744-5159. DOI: 10.1080/00423117508968461.

[172] J. M. Badalamenti and G. R. Doyle. "Radial-Interradial Spring Tire Models". In: *Journal of Vibration and Acoustics* 110.1 (Jan. 1988), pp. 70–75. ISSN: 1528-8927. DOI: 10.1115/1.3269483.

[173] D. Negrut and J. S. Freeman. "Dynamic Tire Modelling for Application with Vehicle Simulations Incorporating Terrain". In: *SAE Transactions* 103 (1994), pp. 96–103. ISSN: 0148-7191. DOI: 10.4271/940223.

[174] R. Mousseau and G. Markale. "Obstacle Impact Simulation of an ATV Using an Efficient Tire Model". In: *Tire Science and Technology* 31.4 (Oct. 2003), pp. 248–269. ISSN: 0090-8657. DOI: 10.2346/1.2135271.

[175] S. Kim, P. E. Nikravesh, and G. Gim. "A two-dimensional tire model on uneven roads for vehicle dynamic simulation1". In: *Vehicle System Dynamics* 46.10 (Oct. 2008), pp. 913–930. ISSN: 1744-5159. DOI: 10.1080/00423110701729994.

[176] G. Rill. "TMeasy – A Handling Tire Model based on a three-dimensional slip approach". In: *Proceedings of the XXIII International Symposium on Dynamic of Vehicles on Roads and on Tracks. Quingdao, China*. 2013, pp. 19–23.

[177] I. Kageyama and S. Kuwahara. "A study on tire modeling for camber thrust and camber torque". In: *JSAE Review* 23.3 (July 2002), pp. 325–331. ISSN: 0389-4304. DOI: 10.1016/s0389-4304(02)00204-7.

[178] G. Rill. "Sophisticated but quite simple contact calculation for handling tire models". In: *Multibody System Dynamics* 45.2 (May 2018), pp. 131–153. ISSN: 1573-272X. DOI: 10.1007/s11044-018-9629-4.

[179] M. Gipser. "FTire: a physically based application-oriented tyre model for use with detailed MBS and finite-element suspension models". In: *Vehicle System Dynamics* 43.1 (Jan. 2005), pp. 76–91. ISSN: 1744-5159. DOI: 10.1080/00423110500139940.

[180] A. Gallrein and M. Bäcker. "CDTire: a tire model for comfort and durability applications". In: *Vehicle System Dynamics* 45.sup1 (Jan. 2007), pp. 69–77. ISSN: 1744-5159. DOI: 10.1080/00423110801931771.

[181] B. Rieff and M. Gipser. "FTire – Full tire dynamics in Real-Time on driving simulators". In: *Proceedings of the 20th Driving Simulation and Virtual Reality Conference and Exhibition, Munich, Germany*. 2021.

[182] M. Gipser. "FTire: 10 years of Development and Application". In: *Vehicle System Dynamics* 45 (2008).

[183] ISO 8855:2011. *Road vehicles – Vehicle dynamics and road-holding ability – Vocabulary*. 2011.

[184] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 2002. ISBN: 978-0387217383. DOI: 10.1007/978-0-387-21738-3.

[185] SAE International. *J2731 – "Low Speed Enveloping Test with Perpendicular and Inclined Cleats"*. 2018. DOI: 10.4271/j2731_201811.

[186] G. Gil and S. Lee. "In-Plane Bending and Shear Deformation of Belt Contributions on Tire Cornering Stiffness Characteristics". In: *Tire Science and Technology* 49.4 (July 2020), pp. 276–297. ISSN: 0090-8657. DOI: 10.2346/tire.20.190213.

[187] M. Guiggiani. *The Science of Vehicle Dynamics: Handling, Braking, and Ride of Road and Race Cars*. Springer, 2014. ISBN: 978-9401785334. DOI: 10.1007/978-94-017-8533-4.

[188] G. Rill and A. A. Castro. *Road Vehicle Dynamics: Fundamentals and Modeling with MATLAB®*. CRC Press, May 2020. ISBN: 978-0429244476. DOI: 10.1201/9780429244476.

[189] C. Oertel. "75 Years of Tyre Modelling for Vehicle System Dynamics Analysis: Review and Future Challenges". In: *Proceedings of the 4th International Tyre Colloquium: Tyre Models for Vehicle Dynamics Analysis: 20-21 April 2015, University of Surrey, Guildford, United Kingdom*. University of Surrey, 2015, pp. 11–20. ISBN: 978-1844690329.

[190] E. Bakker, L. Nyborg, and H. B. Pacejka. "Tyre Modelling for Use in Vehicle Dynamics Studies". In: *SAE Technical Paper Series* (Feb. 1987). ISSN: 0148-7191. DOI: 10.4271/870421.

[191] P. Bayle, J. F. Forissier, and S. Lafon. "A new tyre model for vehicle dynamics simulations". In: *Automotive Technology International* 93 (1993), pp. 193–198.

[192] S. Taheri, C. Sandu, S. Taheri, E. Pinto, and D. Gorsich. "A technical survey on Terramechanics models for tire-terrain interaction used in modeling and simulation of wheeled vehicles". In: *Journal of Terramechanics* 57 (Feb. 2015), pp. 1–22. ISSN: 0022-4898. DOI: 10.1016/j.jterra.2014.08.003.

[193] E. Karpman, J. Kövecses, D. Holz, and K. Skonieczny. "Discrete element modelling for wheel-soil interaction and the analysis of the effect of gravity". In: *Journal of Terramechanics* 91 (Sept. 2020), pp. 139–153. ISSN: 0022-4898. DOI: 10.1016/j.jterra.2020.06.002.

[194] H. Yamashita, P. Jayakumar, and H. Sugiyama. "Physics-Based Flexible Tire Model Integrated With LuGre Tire Friction for Transient Braking and Cornering Analysis". In: *Journal of Computational and Nonlinear Dynamics* 11.3 (Mar. 2016). ISSN: 1555-1423. DOI: 10.1115/1.4032855.

[195] Cosin Scientific Software. *FTire Description*. url: https://www.cosin.eu. Online February 10, 2024. 2024.

[196] J. J. Kalker. "On the rolling contact of two elastic bodies in the presence of dry friction". PhD thesis. Technische Universiteit Delft, 1967.

[197] J. J. Kalker. "Transient Rolling Contact Phenomena". In: *ASLE Transactions* 14.3 (Jan. 1971), pp. 177–184. ISSN: 0569-8197. DOI: 10.1080/05698197108983240.

[198] J. J. Kalker. "Survey of Wheel—Rail Rolling Contact Theory". In: *Vehicle System Dynamics* 8.4 (Nov. 1979), pp. 317–358. ISSN: 1744-5159. DOI: 10.1080/00423117908968610.

[199] J. J. Kalker. "Railway Wheel and Automotive Tyre". In: *Vehicle System Dynamics* 15.5 (Jan. 1986), pp. 255–269. ISSN: 1744-5159. DOI: 10.1080/00423118608968855.

[200] J. J. Kalker, F. M. Dekking, and E. A. H. Vollebregt. "Simulation of Rough, Elastic Contacts". In: *Journal of Applied Mechanics* 64.2 (June 1997), pp. 361–368. ISSN: 1528-9036. DOI: 10.1115/1.2787315.

[201] S. Z. Meymand, A. Keylin, and M. Ahmadian. "A survey of wheel-rail contact models for rail vehicles". In: *Vehicle System Dynamics* 54.3 (Jan. 2016), pp. 386–428. ISSN: 1744-5159. DOI: 10.1080/00423114.2015.1137956.

[202] J. Deur, J. Asgari, and D. Hrovat. "A 3D Brush-type Dynamic Tire Friction Model". In: *Vehicle System Dynamics* 42.3 (Dec. 2004), pp. 133–173. ISSN: 1744-5159. DOI: 10.1080/00423110412331282887.

[203] J. Deur, V. Ivanović, M. Troulis, C. Miano, D. Hrovat, and J. Asgari. "Extensions of the LuGre tyre friction model related to variable slip speed along the contact patch length". In: *Vehicle System Dynamics* 43.sup1 (Jan. 2005), pp. 508–524. ISSN: 1744-5159. DOI: 10.1080/00423110500229808.

[204] E. Velenis, P. Tsiotras, C. Canudas-de-Wit, and M. Sorine. "Dynamic tyre friction models for combined longitudinal and lateral vehicle motion". In: *Vehicle System Dynamics* 43.1 (Jan. 2005), pp. 3–29. ISSN: 1744-5159. DOI: 10.1080/00423110412331290464.

[205] R. Kikuuwe. "A Brush-Type Tire Model with Nonsmooth Representation". In: *Mathematical Problems in Engineering* 2019 (Nov. 2019), pp. 1–13. ISSN: 1563-5147. DOI: 10.1155/2019/9747605.

[206] N. Miyashita, K. Kabe, and M. Mizuno. *Tire characteristic calculation method, tire dynamic element parameter value derivation method, vehicle traveling simu-*

lation method, and tire designing method and vehicle designing method in which consideration is given to tire friction ellipse. US7778809B2. Aug. 2008.

[207] P. Fevrier and M. Hervé. *Method for the simulation of the physical behavior of a tire rolling on the ground*. US8560289B2. Oct. 2013.

[208] N. Xu, K. Guo, X. Zhang, and H. R. Karimi. "An Analytical Tire Model with Flexible Carcass for Combined Slips". In: *Mathematical Problems in Engineering* 2014 (2014), pp. 1–9. ISSN: 1563-5147. DOI: 10.1155/2014/397538.

[209] A. Higuchi. "Transient response of tyres at large wheel slip and camber." PhD thesis. Technische Universiteit Delft, 1999.

[210] L. Romano, F. Timpone, F. Bruzelius, and B. Jacobson. "Analytical results in transient brush tyre models: theory for large camber angles and classic solutions with limited friction". In: *Meccanica* 57.1 (Dec. 2021), pp. 165–191. ISSN: 1572-9648. DOI: 10.1007/s11012-021-01422-3.

[211] J. Svendenius and M. Gäfvert. "A semi-empirical dynamic tire model for combined-slip forces". In: *Vehicle System Dynamics* 44.2 (Feb. 2006), pp. 189–208. ISSN: 1744-5159. DOI: 10.1080/00423110500385659.

[212] N. Miyashita, T. Kawazura, and K. Kabe. "Analytical model of μ–S curve using generalized skewed-parabola". In: *JSAE Review* 24.1 (Jan. 2003), pp. 87–92. ISSN: 0389-4304. DOI: 10.1016/s0389-4304(02)00240-0.

[213] K. Kabe and N. Miyashita. "A New Analytical Tire Model for Cornering Simulation. Part II: Cornering Force and Self-aligning Torque". In: *Tire Science and Technology* 34.2 (June 2006), pp. 100–118. ISSN: 0090-8657. DOI: 10.2346/1.2204753.

[214] K. Kabe and N. Miyashita. "A New Analytical Tire Model for Cornering Simulation. Part I: Cornering Power and Self-Aligning Torque Power". In: *Tire Science and Technology* 34.2 (June 2006), pp. 84–99. ISSN: 0090-8657. DOI: 10.2346/1.2204752.

[215] N. Miyashita. "A study of transient cornering property by use of an analytical tyre model". In: *Proceedings of the 4th International Tyre Colloquium: tyre models for vehicle dynamics analysis: 20-21 April, University of Surrey, Guildford, United Kingdom*. University of Surrey, 2015, pp. 372–381. ISBN: 978-1844690329.

[216] L. Romano, A. Sakhnevych, S. Strano, and F. Timpone. "A novel brush-model with flexible carcass for transient interactions". In: *Meccanica* 54.10 (Aug. 2019), pp. 1663–1679. ISSN: 1572-9648. DOI: 10.1007/s11012-019-01040-0.

[217] L. Romano, F. Bruzelius, and B. Jacobson. "Unsteady-state brush theory". In: *Vehicle System Dynamics* 59.11 (June 2020), pp. 1643–1671. ISSN: 1744-5159. DOI: 10.1080/00423114.2020.1774625.

[218] D. Harsh and B. Shyrokau. "Tire Model with Temperature Effects for Formula SAE Vehicle". In: *Applied Sciences* 9.24 (Dec. 2019), p. 5328. ISSN: 2076-3417. DOI: 10.3390/app9245328.

[219] H. Chollet. "A 3D model for rubber tyres contact, based on Kalker's methods through the STRIPES model". In: *Vehicle System Dynamics* 50.1 (Jan. 2012), pp. 133–148. ISSN: 1744-5159. DOI: 10.1080/00423114.2011.575945.

[220] P. Riehm, H.-J. Unrau, F. Gauterin, S. Torbrügge, and B. Wies. "3D brush model to predict longitudinal tyre characteristics". In: *Vehicle System Dynamics* 57.1 (2019), pp. 17–43. DOI: 10.1080/00423114.2018.1447135.

[221] H. Sakai. "Theoretical and experimental studies on the dynamic properties of tyres Part 1: Review of theories of rubber friction". In: *International Journal of Vehicle Design* 2.1 (1981), pp. 78–110. ISSN: 0143-3369. DOI: 10.1504/IJVD.1981.061246.

[222] H. Sakai. "Theoretical and experimental studies on the dynamic properties of tyres. Part 2: Experimental investigation of rubber friction and deformation of a tyre". In: *International Journal of vehicle design* 2.2 (1981), pp. 182–226. ISSN: 0143-3369. DOI: 10.1504/IJVD.1981.061416.

[223] H. Sakai. "Theoretical and experimental studies on the dynamic properties of tyres: Part 3: Calculation of the six components of force and moment of a tyre". In: *International Journal of Vehicle Design* 2.3 (1981), pp. 335–372. ISSN: 0143-3369. DOI: 10.1504/IJVD.1981.061417.

[224] H. Sakai. "Theoretical and experimental studies on the dynamic properties of tyres. Part 4: investigations of the influences of running conditions by calculation and experiment". In: *International Journal of vehicle design* 3.3 (1982), pp. 333–375. ISSN: 0143-3369. DOI: 10.1504/IJVD.1982.061275.

[225] J. de Hoogh. "Implementing inflation pressure and velocity effects into the Magic Formula tyre model". MA thesis. Technische Universiteit Eindhoven, 2005.

[226] E. M. Kasprzak and D. Gentz. "The Formula SAE Tire Test Consortium-Tire Testing and Data Handling". In: *SAE Technical Paper Series*. SAE International, Nov. 2006, p. 12. DOI: 10.4271/2006-01-3606.

[227] F. Koutny. *Geometry and mechanics of pneumatic tires*. Tech. rep. Zlín, 2007.

[228] T. B. Rhyne. "Development of a Vertical Stiffness Relationship for Belted Radial Tires". In: *Tire Science and Technology* 33.3 (July 2005), pp. 136–155. ISSN: 0090-8657. DOI: 10.2346/1.2174340.

[229] A. R. Savkoor. "Some aspects of friction and wear of tyres arising from deformations, slip and stresses at the ground contact". In: *Wear* 9.1 (Jan. 1966), pp. 66–78. ISSN: 0043-1648. DOI: 10.1016/0043-1648(66)90015-9.

[230] H. Sakai. "Measurement and Visualization of the Contact Pressure Distribution of Rubber Disks and Tires". In: *Tire Science and Technology* 23.4 (Oct. 1995), pp. 238–255. ISSN: 0090-8657. DOI: 10.2346/1.2137506.

[231] P. Sarkisov. "Physical Understanding of Tire Transient Handling Behavior". PhD thesis. University of Göttingen, 2019.

[232] E. Fiala. "Seitenkraften am rollenden Luftreifen". In: *VDI Zeitschrift* 96 (1954), pp. 973–979.

[233] C. Canudas-de-Wit, P. Tsiotras, E. Velenis, M. Basset, and G. Gissinger. "Dynamic Friction Models for Road/Tire Longitudinal Interaction". In: *Vehicle System Dynamics* 39.3 (Mar. 2003), pp. 189–226. ISSN: 0042-3114. DOI: 10.1076/vesd.39.3.189.14152.

[234] M. Selig, B. Lorenz, D. Henrichmöller, K. Schmidt, A. Ball, and B. N. J. Persson. "Rubber Friction and Tire Dynamics: A Comparison of Theory with Experimental Data". In: *Tire Science and Technology* 42.4 (Oct. 2014), pp. 216–262. ISSN: 0090-8657. DOI: 10.2346/tire.14.420403.

[235] A. R. Savkoor. "On the friction of rubber". In: *Wear* 8.3 (May 1965), pp. 222–237. ISSN: 0043-1648. DOI: 10.1016/0043-1648(65)90161-4.

[236] A. R. Savkoor. "Dry adhesive friction of elastomers". PhD thesis. Technische Universiteit Delft, 1987.

[237] A. Tiwari, L. Dorogin, B. Steenwyk, A. Warhadpande, M. Motamedi, G. Fortunato, V. Ciaravola, and B. N. J. Persson. "Rubber friction directional asymmetry". In: *EPL (Europhysics Letters)* 116.6 (Dec. 2016), p. 66002. ISSN: 1286-4854. DOI: 10.1209/0295-5075/116/66002.

[238] B. N. J. Persson. "Rubber friction and tire dynamics". In: *Journal of Physics: Condensed Matter* 23.1 (Dec. 2010), p. 015003. ISSN: 1361-648X. DOI: 10.1088/0953-8984/23/1/015003.

[239] L. Romano, F. Bruzelius, and B. Jacobson. "Brush tyre models for large camber angles and steering speeds". In: *Vehicle System Dynamics* 60.4 (Dec. 2020), pp. 1341–1392. ISSN: 1744-5159. DOI: 10.1080/00423114.2020.1854320.

[240] D. J. N. Limebeer and M. Massaro. *Dynamics and Optimal Control of Road Vehicles*. Oxford University PressOxford, Sept. 2018. ISBN: 978-0191864636. DOI: 10.1093/oso/9780198825715.001.0001.

[241] M. A. Meyers and K. K. Chawla. *Mechanical Behavior of Materials*. 2nd. Cambridge University Press, 2008.

[242] L. Romano. *Advanced Brush Tyre Modelling*. Briefs in Applied Sciences and Technology. Springer, 2022. ISBN: 978-3030984359. DOI: 10.1007/978-3-030-98435-9.

[243] R. E. Okonieski, D. J. Moseley, and K. Y. Cai. "Simplified Approach to Calculating Geometric Stiffness Properties of Tread Pattern Elements". In: *Tire Science and Technology* 31.3 (July 2003), pp. 132–158. ISSN: 0090-8657. DOI: 10.2346/1.2135265.

[244] J. Matoušek. "On directional convexity". In: *Discrete Computational Geometry* 25.3 (Apr. 2001), pp. 389–403. ISSN: 1432-0444. DOI: 10.1007/s004540010069.

[245] R. Bulirsch and J. Stoer. *Introduction to Numerical Analysis*. Springer, 2002. ISBN: 978-0387217383. DOI: 10.1007/978-0-387-21738-3.

[246] G. E. Alefeld, F. A. Potra, and Y. Shi. "Algorithm 748: enclosing zeros of continuous functions". In: *ACM Transactions on Mathematical Software* 21.3 (Sept. 1995), pp. 327–344. ISSN: 1557-7295. DOI: 10.1145/210089.210111.

[247] P. Gruber, R. S. Sharp, and A. D. Crocombe. "Normal and shear forces in the contact patch of a braked racing tyre. Part 1: results from a finite-element model". In: *Vehicle System Dynamics* 50.2 (Feb. 2012), pp. 323–337. ISSN: 1744-5159. DOI: 10.1080/00423114.2011.586428.

[248] P. Gruber, R. S. Sharp, and A. D. Crocombe. "Normal and shear forces in the contact patch of a braked racing tyre. Part 2: development of a physical

tyre model". In: *Vehicle System Dynamics* 50.3 (Mar. 2012), pp. 339–356. ISSN: 1744-5159. DOI: 10.1080/00423114.2011.586429.

[249] E. Spedicato and J. Greenstadt. "On some classes of variationally derived Quasi-Newton methods for systems of nonlinear algebraic equations". In: *Numerische Mathematik* 29.4 (Apr. 1978), pp. 363–380. ISSN: 0945-3245. DOI: 10.1007/bf01432875.

[250] T. Eirola and O. Nevanlinna. "Accelerating with rank-one updates". In: *Linear Algebra and its Applications* 121 (Aug. 1989), pp. 511–520. ISSN: 0024-3795. DOI: 10.1016/0024-3795(89)90719-2.

[251] C. G. Broyden. "A class of methods for solving nonlinear simultaneous equations". In: *Mathematics of Computation* 19.92 (1965), pp. 577–593. ISSN: 1088-6842. DOI: 10.1090/s0025-5718-1965-0198670-6.

[252] J. M. Martinez and L. S. Ochi. "Sobre dois métodos de Broyden". In: *Matemática Aplicáda e Computacional* 1.2 (1982), pp. 135–141.

[253] C. A. Felippa. "A historical outline of matrix structural analysis: a play in three acts". In: *Computers & Structures* 79.14 (June 2001), pp. 1313–1324. ISSN: 0045-7949. DOI: 10.1016/s0045-7949(01)00025-6.

[254] M. J. Turner and Boeing Company. *The Direct Stiffness Method of Structural Analysis*. Boeing Airplane Company, 1959.

[255] M. J. Turner, H. C. Martin, and R. C. Weikel. "Further development and applications of the stiffness method". In: *Matrix methods of structural analysis* 1 (1964), pp. 203–266.

[256] M. N. Pavlović. "Symbolic computation in structural engineering". In: *Computers & structures* 81.22-23 (2003), pp. 2121–2136. DOI: 10.1016/S0045-7949(03)00286-4.

[257] M. M. Cecchi and C. Lami. "Automatic generation of stiffness matrices for finite element analysis". In: *International Journal for Numerical Methods in Engineering* 11.2 (1977), pp. 396–400. DOI: 10.1002/NME.1620110216.

[258] N. I. Ioakimidis. "Application of mathematica to the direct semi-numerical solution of finite element problems". In: *Computers & Structures* 45.5-6 (Dec. 1992), pp. 833–839. ISSN: 0045-7949. DOI: 10.1016/0045-7949(92)90043-y.

[259] A. I. Beltzer. *Variational and Finite Element Methods*. Springer, 1990. ISBN: 978-3642839146. DOI: 10.1007/978-3-642-83914-6.

[260] Wolfram Mathematica. *Structural Mechanics*. 2024. URL: https://reference.wolfram.com/applications/structural/index.html.

[261] D. L. Logan. *A first course in the finite element method*. 6th. Thomson, 2015. ISBN: 978-1305635111.

[262] D. V. Hutton. *Fundamentals of finite element analysis*. McGraw Hill, 2004. ISBN: 978-0072395365.

[263] A. Samuelsson and O. C. Zienkiewicz. "History of the stiffness method". In: *International Journal for Numerical Methods in Engineering* 67.2 (June 2006), pp. 149–157. ISSN: 1097-0207. DOI: 10.1002/nme.1510.