Numerical and Symbolic Methods for Dynamic Optimization

Magnusson, Fredrik

Numerical and Symbolic Methods for Dynamic Optimization

Fredrik Magnusson

LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis TFRT-1115
ISBN 978-91-7753-004-6 (print)
ISBN 978-91-7753-005-3 (web)
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2016

Abstract

Mathematical optimization is becoming increasingly important for engineering in general and control in particular. This thesis deals with numerical methods, primarily direct collocation, and symbolic methods, primarily block-triangular ordering and tearing, for numerical solution of general dynamic optimization problems involving dynamical systems modeled by large-scale differential-algebraic equations (DAE). These methods have been implemented in a software framework in the open-source JModelica.org platform, which is a software tool for simulation- and optimization-based analysis of DAEs described in the modeling language Modelica. The framework relies heavily upon the open-source, third-party software packages CasADi for symbolic operations and algorithmic differentiation and IPOPT for solving the resulting nonconvex optimization problems.

Modelica is a standardized modeling language, which permeates the thesis. One of the many benefits of Modelica is that it is supported by several different tools, allowing implemented models to be used for different purposes. However, Modelica models are often developed for dynamic simulation and sometimes with little regard for numerics, which is enabled by the power of the available simulation software. Consequently, the models may be difficult to reuse for dynamic optimization, which is one of the challenges addressed by this thesis.

The application of direct collocation to DAE-constrained optimization problems is conventionally done by discretizing the full DAE. This often turns out to be inefficient, especially for DAEs originating from Modelica code. The thesis proposes various schemes to symbolically eliminate many of the algebraic variables in a preprocessing step before discretization to improve the efficiency of numerical methods for dynamic optimization, in particular direct collocation. These techniques are inspired by the causalization and tearing techniques often used when solving DAE initial-value problems in the Modelica community. Since sparsity is crucial for some dynamic optimization methods, we also propose a novel approach to preserving sparsity during this procedure.

A collection of five computationally challenging and industrially relevant optimal control problems is presented. The collection is used to evaluate the performance of the methods. We consider both computational time and probability of solving problems in a timely manner. We demonstrate that the proposed methods often are an order of magnitude faster than the standard way of discretizing the full DAE, and that they also increase probability of successful convergence significantly. It is also demonstrated that the methods are beneficial not only for DAEs originating from Modelica code, but also for more conventional textbook DAEs that have been developed specifically for optimization purposes.

Acknowledgments

I am most grateful to my supervisor Johan Åkesson, who convinced me to pursue a PhD in the first place. He also guided me all the way from start to finish, and somehow managed to maintain his interest in my work despite leaving the world of academia. I am also appreciative of my two other supervisors, Anders Rantzer and Bo Bernhardsson, who have always shared their different—but valuable—perspectives on my work with enthusiasm.

I have also had occasional opportunities to bask in the wisdom of Claus Führer and Carl Laird, who shared their expertise on differential-algebraic equations and optimization, respectively (although both of them like to think that they know a bit about the other topic as well).

My years at the Department of Automatic Control have been delightful, in no small part due to the many great colleagues. Special thanks go to Jacob Bergstedt, who endured my constant company for years. He always took my many trifling questions seriously (and gave me ample opportunity to return the favor). Jacob has also—together with Erik Henningsson, Christian Grussler, Mahdi Ardakani, and whomever else I managed to engage—satisfied my basic need to regularly discuss mathematics for no particular reason.

My many research collaborators have given me very valuable experiences and also inspired my own work. I am especially grateful to Roel De Coninck, who proved the industrial applicability of my research, as well as Anders Holmqvist (and his protégé Anton Sellberg), who time and again managed to concoct crazy application ideas and somehow make them work despite my recurring initial skepticism.

The people of Modelon have been a great aid to my work, with the model developers contributing valuable use cases and the tool developers getting JModelica.org to behave as I desire (and sometimes aligning my desires with how JModelica.org behaves). Special thanks go to Toivo Henningsson, whom I very much enjoyed working with while it lasted.

I also thank Joel Andersson and Joris Gillis for developing the great

tool that is CasADi and helping me get the most out of it.

To the benefit of you, the reader, I have received many valuable suggestions for improvements of the thesis manuscript from Leif Andersson, Karl Berntorp, Erik Henningsson, and Björn Olofsson.

I thank my friends of HALT HAMMERZEIT and Nöbbelövs spelförening (with an honorary mention to Ylva Johansson) for making everything not related to this thesis more fun.

Finally, I thank my parents for never helping me any less than I want them to.

Contents

1. Introduction **9**
- 1.1 Motivation . . . 10
- 1.2 Contributions . . . 12
- 1.3 Publications . . . 12
- 1.4 Notation . . . 15

2. Background **16**
- 2.1 Hierarchical Acausal Modeling with Modelica . . . 16
- 2.2 Differential-Algebraic Equations . . . 20
- 2.3 Dynamic Optimization . . . 26
- 2.4 Nonlinear Programming . . . 32

3. Dynamic Optimization in JModelica.org **36**
- 3.1 Problem Formulation . . . 36
- 3.2 Related Software and Languages . . . 39
- 3.3 Direct Local Collocation . . . 45
- 3.4 Implementation . . . 53
- 3.5 Additional Features . . . 55
- 3.6 Example . . . 58
- 3.7 Conclusion . . . 60

4. Symbolic Elimination Based on Block-Triangular Ordering **62**
- 4.1 Illustrative Example . . . 63
- 4.2 Related Work . . . 65
- 4.3 Causalization, Tearing, and Pivot Selection . . . 66
- 4.4 Symbolic Elimination for Dynamic Optimization . . . 71
- 4.5 Conclusion . . . 79

5. Problem Suite **81**
- 5.1 The Many Colors of Block-Triangular Decompositions . . . 82
- 5.2 Car . . . 82
- 5.3 Combined-Cycle Power Plant . . . 86
- 5.4 Double Pendulum . . . 90
- 5.5 Fourbar1 . . . 93

5.6 Distillation Column 96
5.7 Conclusion 99
5.8 Heat Recovery Steam Generator 100

6. Scheme Benchmarks **105**
6.1 Benchmark Setup 105
6.2 Problem Results 108
6.3 Performance Profiles 112
6.4 Computation Times 114
6.5 Global Collocation 116
6.6 Conclusion 117

7. Conclusion **118**
7.1 Summary 118
7.2 Directions for Future Research 119

Bibliography **120**

1

Introduction

The application of optimization to large-scale dynamical systems has become more common in both industry and academia during the last decades. Dynamic optimization problems occur in many different fields and contexts, including optimal control, design optimization, parameter estimation, and state estimation. Examples of applications are minimization of material and energy consumption during setpoint transitions in power plants [Krüger et al., 2004] and chemical processes [Prata et al., 2008], estimating occupancy and ambient air flow in buildings [Zavala, 2014], and optimal experimental design for estimation of kinetic parameters in fed-batch processes [Baltes et al., 1994].

Optimization enables the exploration of the limits of performance of a system, which often elucidates the need for performing tradeoffs. An example of this is the use of chromatography in chemical engineering for separation of components from multicomponent mixtures, where a tradeoff between target component purity and the two competing objectives of (normalized) production rate and recovery yield arises. Performing this tradeoff optimally is considered in [Holmqvist and Magnusson, 2016] based on a novel control scheme, with the resulting Pareto frontiers being shown in Figure 1.1. We see how an increase in recovery yield comes at the expense of production rate, and also how a lower bound on purity affects the efficiency of the system.

The applications of dynamic optimization are diverse and occur in both online and offline settings. Online refers to the case in which an optimization problem needs to be solved in real time (often repeatedly) to find the desired operating conditions of a system, adapting to the changing state of the system. In these cases the execution time for algorithms is often a performance bottleneck. Conversely, offline refers to the case in which all computations can be performed before the system is up and running, and consequently without any hard time constraints. Algorithm execution times can however still be important for the sake of reducing engineering research and development costs.

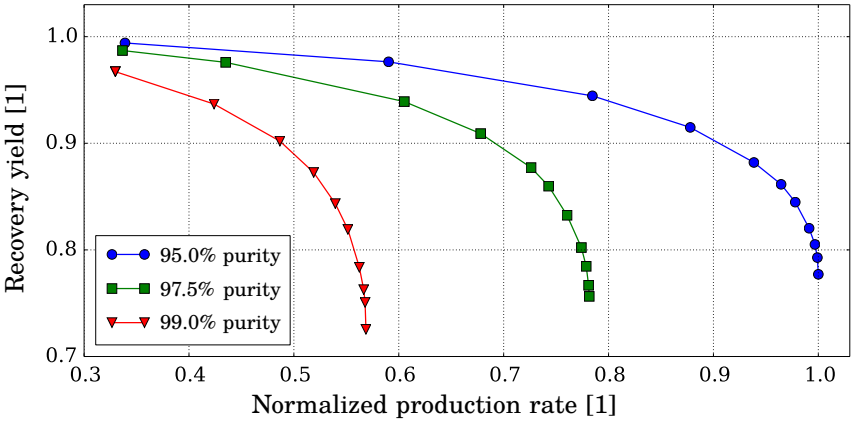

Figure 1.1 Pareto frontiers for chromatographic separation of chemical components: A tradeoff between production rate, recovery yield, and purity.

Online optimal control is usually done in the form of Model Predictive Control (MPC) [Maciejowski, 2002] and online state estimation based on dynamic optimization is usually done in the form of Moving Horizon Estimation (MHE) [Allgöwer et al., 1999]. Offline applications include finding optimal trajectories, which can be used either as a reference during manual control [Larsson, 2011] or as reference trajectories combined with online feedback to handle deviations caused by model uncertainty and disturbances [Giselsson et al., 2009]. The study of Pareto frontiers is also usually done offline.

This thesis concerns the development of algorithms for general-purpose dynamic optimization of systems described by Differential-Algebraic Equations (DAE).

1.1 Motivation

The purpose of this thesis is twofold: To build and improve upon the state of the art of algorithms for solving dynamic optimization problems and to provide a software implementation of these that is viable both for academic research and for industrial application.

As computers and algorithms become more powerful, so too do the complexity and size of problems that can be successfully solved increase. However, dealing with large, heterogeneous problems often requires significant expertise and effort from the user. There are abundant opportunities for failure when solving dynamic optimization problems, and the symptoms of failure are often difficult to diagnose. In particular, a sufficiently good initial

guess is often needed for iterative methods to converge to a local solution to nonconvex optimization problems, let alone a desirable one. Implementing accurate and efficient large-scale models and using them to solve dynamic optimization problems is therefore demanding work.

While there already exist many efficient software packages for dynamic optimization, almost all of them are inherently tied to a specific—and often limited—model representation. Some tools rely on general-purpose programming languages such as C++, MATLAB, or Python, to represent models, which are ill-suited for modeling large-scale, heterogeneous dynamical systems. Other tools support dedicated modeling languages; however, these languages are typically tied to that specific tool. Considering the expense of implementing large-scale physical models, it is undesirable to have the model implementation tied to a specific tool or even algorithm. The algorithms of this thesis have thus been implemented for use with Modelica [Mattsson et al., 1998; Fritzson, 2015], which is a tool-agnostic and standardized language for modeling of heterogeneous dynamical systems that greatly facilitates model reuse through acausal, hierarchical modeling. Using Modelica allows the model implementations to be used not only in all of the several different tools that support Modelica, but also for purposes other than dynamic optimization. The most important of such purposes is dynamic simulation. Examples of other ones are linear and nonlinear controller design [Thümmel et al., 2005], parametric sensitivity analysis [Elsheikh and Wiechert, 2008], and analysis using formal methods [Klenk et al., 2014].

While the acausal modeling approach of Modelica is a great aid to developers of models, it shifts some of the burden of analysis onto the algorithms that are used to simulate or otherwise "solve" the system. Applying state-of-the-art dynamic optimization algorithms out-of-the-box on models from typical Modelica libraries is often met with failure, typically in the form of iterative methods failing to converge. One reason for this is that Modelica components are often developed for general purposes, modeling phenomena that may have no relevance for a particular application, thus blowing up the size of the model. Another reason for the blown-up size is the multitude of algebraic equations that result from acausal, hierarchical modeling. The models may therefore end up being overly complicated for the purposes of dynamic optimization. This thesis attempts to overcome these issues, primarily through the use of symbolic elimination and automated initialization and scaling based on dynamic simulation.

However, there are limits to what can be achieved by automated algorithms. Hence, it is important to provide advanced users with the means to manually investigate convergence issues and other numerical problems. This is achieved by providing a symbolic interface to the equations to be solved, both before and after their discretization.

1.2 Contributions

The main contributions of this thesis are:

- An efficient, open-source implementation of a direct collocation algorithm for dynamic optimization and its integration in a Modelica-based toolchain. This is the focus of Chapter 3.
- Methods for symbolic elimination of algebraic variables in DAEs and their tailoring for dynamic optimization. In particular, a novel scheme for preservation of sparsity is presented. This is the focus of Chapter 4.
- A suite of five computationally challenging and industrially relevant optimal control problems, which is used to demonstrate the efficiency of the methods and their implementation. The suite is presented in Chapter 5 and the method benchmark is performed in Chapter 6.

All implementations of this thesis are integrated and distributed together with the open-source Modelica platform JModelica.org [Åkesson et al., 2010a] under the GNU General Public License and code for reproducing the key results of this thesis is distributed with JModelica.org. The most significant contributions to the JModelica.org codebase are the Python modules `pyjmi.casadi_interface`, `pyjmi.optimization.casadi_collocation`, and `pyjmi.symbolic_elimination`.

The framework of Chapter 3 has seen widespread use in both academia and industry. Section 1.3 lists publications using the framework in which the author has been involved. Dozens of other publications have also made use of the framework independently of the author, such as [Norén, 2013; Yang et al., 2014; Belkhir et al., 2015; Maree and Imsland, 2016].

1.3 Publications

This section lists all of the (to be) peer-reviewed publications by the author. The thesis is primarily based on the following two publications.

Magnusson, F. and J. Åkesson (2015). “Dynamic optimization in JModelica.org”. *Processes* **3**:2, pp. 471–496.

Magnusson, F. and J. Åkesson (2016). “Symbolic elimination in dynamic optimization based on block-triangular ordering”. *Optimization Methods and Software*. Submitted for publication.

The first publication serves as a basis for Chapter 3 and the second publication for Chapters 4 and 6. Chapter 5 is largely unpublished material, although a brief description of the same suite can be found in the

second publication. Most of the overarching ideas are primarily due to J. Åkesson, while working out the details has been a joint effort between the two authors. An important idea that is due to F. Magnusson is the sparsity preserving aspect of the symbolic elimination. F. Magnusson has made the implementations and written the manuscripts.

The following two publications are the basis of the two former publications.

Magnusson, F. and J. Åkesson (2012). "Collocation methods for optimization in a Modelica environment". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 649–658.

Magnusson, F., K. Berntorp, B. Olofsson, and J. Åkesson (2014). "Symbolic transformations of dynamic optimization problems". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 1027–1036.

The following 14 publications have, for the most part, not been included in the thesis.

Axelsson, M., F. Magnusson, and T. Henningsson (2015). "A framework for nonlinear model predictive control in JModelica.org". In: *Proceedings of the 11th International Modelica Conference*. Paris, France, pp. 301–310.

Berntorp, K. and F. Magnusson (2015). "Hierarchical predictive control for ground-vehicle maneuvering". In: *2015 American Control Conference*. Chicago, IL, pp. 2771–2776.

De Coninck, R., F. Magnusson, J. Åkesson, and L. Helsen (2014). "Grey-box building models for model order reduction and control". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 657–666.

De Coninck, R., F. Magnusson, J. Åkesson, and L. Helsen (2016). "Toolbox for development and validation of grey-box building models for forecasting and control". *Journal of Building Performance Simulation* **9**:3, pp. 288–303.

Fouquet, M., F. Magnusson, H. Guéguen, S. Velut, D. Faille, D. Dumur, and T. Henningsson (2016). "Hybrid dynamic optimization of power plants based on physical models and the collocation method". *Journal of Process Control*. Accepted subject to major revision.

Ghazaei Ardakani, M. M. and F. Magnusson (2016). "Ball and finger system: modeling and optimal trajectories". *Robotics and Autonomous Systems*. To be submitted.

Holmqvist, A., C. Andersson, F. Magnusson, and J. Åkesson (2015a). "Methods and tools for robust optimal control of batch chromatographic separation processes". *Processes* **3**:3, pp. 568–606.

Holmqvist, A. and F. Magnusson (2016). “Open-loop optimal control of batch chromatographic separation processes using direct collocation”. *Journal of Process Control* **46**, pp. 55–74.

Holmqvist, A., F. Magnusson, and B. Nilsson (2015b). “Dynamic multi-objective optimization of batch chromatographic separation processes”. In: *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering*. Vol. 37, pp. 815–820.

Holmqvist, A., F. Magnusson, and S. Stenström (2014a). “Scale-up analysis of continuous cross-flow atomic layer deposition reactor designs”. *Chemical Engineering Science* **117**, pp. 301–317.

Holmqvist, A., T. Törndahl, F. Magnusson, U. Zimmermann, and S. Stenström (2014b). “Dynamic parameter estimation of atomic layer deposition kinetics applied to in situ quartz crystal microbalance diagnostics”. *Chemical Engineering Science* **111**, pp. 15–33.

Larsson, P.-O., F. Casella, F. Magnusson, J. Andersson, M. Diehl, and J. Åkesson (2013). “A framework for nonlinear model-predictive control using object-oriented modeling with a case study in power plant start-up”. In: *Proceedings of the 2013 IEEE Multi-Conference on Systems and Control*. Hyderabad, India, pp. 346–351.

Magnusson, F., K. Palmer, L. Han, and G. Bollas (2015). “Dynamic parametric sensitivity optimization using simultaneous discretization in JModelica.org”. In: *2015 International Conference on Complex Systems Engineering*, pp. 37–42.

Sellberg, A., A. Holmqvist, F. Magnusson, C. Andersson, and B. Nilsson (2016). “Discretized multi-level elution trajectory: model calibration, optimization, and validation”. *Journal of Chromatography A*. Submitted for publication.

Most of these publications concern application-oriented work in which the results of this thesis have been used, primarily that of Chapter 3. The only one of these publications to utilize Chapter 4 is [Berntorp and Magnusson, 2015]. While virtually all of these publications are mainly application-oriented, some of them have elements of method development for dynamic optimization, in particular [Axelsson et al., 2015; Magnusson et al., 2015; Holmqvist and Magnusson, 2016; Fouquet et al., 2016]. Two of these publications have made contributions to Chapter 5, namely [Larsson et al., 2013] and [Berntorp and Magnusson, 2015]. The author’s contribution to most of these works has been to aid the formulation of the desired optimization problems and to overcome numerical issues encountered in their solution.

1.4 Notation

Scalars and scalar-valued functions are denoted by regular italic letters x. Vectors and vector-valued functions are denoted by bold italic letters $\boldsymbol{x}$. The dimension of $\boldsymbol{x}$ (or in the case of functions, its codomain) is denoted by n_x and component k of $\boldsymbol{x}$ is denoted by x_k. Matrices are denoted by bold Roman letters $\mathbf{A}$ and element (i, j) of $\mathbf{A}$ is denoted by $A_{i,j}$. Row i and column j of $\mathbf{A}$ are denoted by $\boldsymbol{A}_{i,:}$ and $\boldsymbol{A}_{:,j}$, respectively. The diagonal matrix with $\boldsymbol{x}$ along the diagonal is denoted by $\operatorname{diag} \boldsymbol{x}$.

Systems of equations $\boldsymbol{f} = \mathbf{0}$ that are to be considered as being parametrized by $\boldsymbol{p}$ and having $\boldsymbol{x}$ as unknowns are denoted by $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}) = \mathbf{0}$. The Jacobian of $\boldsymbol{f}$ with respect to $\boldsymbol{x}$ is denoted by $\nabla_{\boldsymbol{x}} \boldsymbol{f}$. The integer interval from $a \in \mathbb{Z}$ to $b \in \mathbb{Z}$ (inclusive) is denoted by $[a..b]$.

2

Background

> If I have seen further it is by standing on the sholders [sic] of Giants.
>
> Isaac Newton

In this chapter we review relevant background material, with the purpose of introducing well-known core concepts that are fundamental to the rest of the thesis. Details are sometimes omitted, but can be found in cited references.

2.1 Hierarchical Acausal Modeling with Modelica

The classical equation-based approach to modeling dynamical systems using software is that of causal, block-oriented modeling, which is used in general-purpose tools such as Simulink [MathWorks, 2016]. The corresponding mathematical formalism is that of explicit ordinary differential equations, which facilitates efficient numerical solution, but may be inconvenient for modeling systems that are more naturally described by DAEs. Furthermore, causal modeling inhibits model reuse because of the need of defining causal input-output connections between components, see [Fritzson, 2015, Section 2.7.1] for an illustrative example of this.

A more modern—but by now well-established—approach to modeling is the use of acausal, declarative equations, which is used in tools such as SPICE [Nagel and Pederson, 1973] and ASCEND [Piela et al., 1991].

2.1.1 Modelica

Most acausal modeling tools and languages are domain-specific, focusing on, for example, electrical circuits, or chemical processes. A more general-purpose language for acausal modeling is Modelica [Mattsson et al., 1998; Fritzson, 2015]. Modular modeling is supported in Modelica by defining

acausal physical ports for elementary components, building systems by hierarchical aggregation of subsystems, and managing model variants by replacing some parts of the model by others that share the same physical interface.

Modelica is a textual and graphical language standardized and developed by the Modelica Association [Modelica Association, 2016b], a nonprofit organization. Several different software tools exist based on Modelica. Its development started in 1996, although it has roots dating back to the 70s [Elmqvist, 1978]. An important part of Modelica is the description of hybrid systems; that is, systems with both continuous and discrete dynamics. Since such systems are outside the scope of this thesis, these aspects of Modelica will not be considered.

As an example of Modelica's syntax, consider the Van der Pol oscillator, which is a single-input, explicit ordinary differential equation described by

$$\begin{aligned} \dot{x}_1(t) &= \left(1 - x_2^2(t)\right) x_1(t) - x_2(t) + u(t), & x_1(0) &= 0, \\ \dot{x}_2(t) &= x_1(t), & x_2(0) &= 1, \end{aligned} \tag{2.1}$$

where we have imposed initial conditions at $t = 0$ to get a unique solution given u. This system can be encoded in Modelica as shown in Listing 2.1.

Listing 2.1 Modelica implementation of the Van der Pol oscillator.

```
model VDP
    Real x1(start=0, fixed=true);
    Real x2(start=1, fixed=true);
    input Real u;
equation
    der(x1) = (1-x2^2)*x1 - x2 + u;
    der(x2) = x1;
end VDP;
```

This is the simplest form of Modelica models, consisting first of a declaration of the system variables—with specifications of initial state and which variables are top-level inputs—and then the equations.

2.1.2 Object-Oriented Modeling

Typical Modelica models look nothing like Listing 2.1. They are hierarchical, rather than flat, meaning that rather than only having declarations of variables and equations, they also have declarations of components—which themselves are made up of components, variables, and equations—and the connections between them. They also consist of implicit DAEs, rather than explicit ODEs, because of the acausal connections of components and common occurrence of physical phenomena that are more naturally described by algebraic rather than differential equations.

For a more typical Modelica example, consider a kinematic loop formed by connecting four rigid bars with revolute joints. Such a system can be encoded in Modelica as shown in Listing 2.2. The web version of the thesis contains an appendix with a three-dimensional animation of the mechanical system. The appendix is not available in the printed version of the thesis. The model consists of predefined components for the bars and

Listing 2.2 Textual representation of the Modelica example `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar1`. Slight modifications of the original have been made for aesthetic purposes.

```
model Fourbar1
  "One kinematic loop with four bars (with only revolute joints)"
  inner Modelica.Mechanics.MultiBody.World world;
  Modelica.Mechanics.MultiBody.Joints.Revolute j1(
    n={1,0,0},
    stateSelect=StateSelect.always,
    phi(fixed=true),
    w(displayUnit="deg/s",
      start=5.235987755982989,
      fixed=true));
  Modelica.Mechanics.MultiBody.Joints.Prismatic j2(
    n={1,0,0}, s(start=-0.2), boxWidth=0.05);
  Modelica.Mechanics.MultiBody.Parts.BodyCylinder b1(
    r={0,0.5,0.1}, diameter=0.05);
  Modelica.Mechanics.MultiBody.Parts.BodyCylinder b2(
    r={0,0.2,0}, diameter=0.05);
  Modelica.Mechanics.MultiBody.Parts.BodyCylinder b3(
    r={-1,0.3,0.1}, diameter=0.05);
  Modelica.Mechanics.MultiBody.Joints.Revolute rev(n={0,1,0});
  Modelica.Mechanics.MultiBody.Joints.Revolute rev1;
  Modelica.Mechanics.MultiBody.Joints.Revolute j3(n={1,0,0});
  Modelica.Mechanics.MultiBody.Joints.Revolute j4(n={0,1,0});
  Modelica.Mechanics.MultiBody.Joints.Revolute j5(n={0,0,1});
  Modelica.Mechanics.MultiBody.Parts.FixedTranslation b0(
    animation=false, r={1.2,0,0});
equation
  connect(j2.frame_b, b2.frame_a);
  connect(j1.frame_b, b1.frame_a);
  connect(rev.frame_a, b2.frame_b);
  connect(rev.frame_b, rev1.frame_a);
  connect(rev1.frame_b, b3.frame_a);
  connect(world.frame_b, j1.frame_a);
  connect(b1.frame_b, j3.frame_a);
  connect(j3.frame_b, j4.frame_a);
  connect(j4.frame_b, j5.frame_a);
  connect(j5.frame_b, b3.frame_b);
  connect(b0.frame_a, world.frame_b);
  connect(b0.frame_b, j2.frame_a);
end Fourbar1;
```

joints. This is a more typical usage of Modelica, where libraries defining physical components from various physical domains are relied upon for creating a system model. This example is created using the Modelica Standard Library (MSL), and is one of the examples in MSL called `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar1`. MSL is a freely available library developed by the Modelica Association. There are many other libraries [Modelica Association, 2016a]—some of which are open source and freely available and others commercial—from various engineering domains, including thermodynamics, electronics, and control.

While Modelica is primarily a textual language, models like Listing 2.2 are usually not made textually, but rather graphically. The graphical representation of Listing 2.2 in Dymola [Dassault Systèmes, 2016], the most widely used Modelica tool, is shown in Figure 2.1.

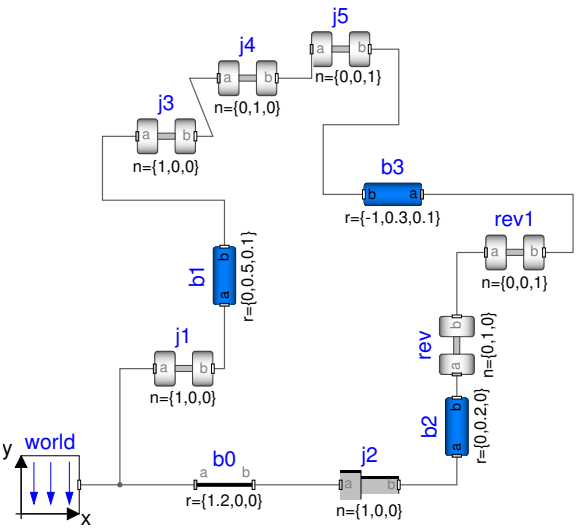

Figure 2.1 Dymola object diagram of a four-bar system in Modelica. The closed kinematic loop is easily constructed by graphically connecting components from the Modelica Standard Library for rigid bodies as well as revolute and prismatic joints. No particular knowledge is required of the user; no cut-joints or a spanning tree has to be determined.

While acausal modeling is convenient for the modeler, the resulting mathematical model is often unwieldy in its raw form and requires symbolic processing before it can be simulated efficiently using numerical methods.

This holds especially true when the model is the result of object-oriented aggregation of many subcomponents, which gives rise to a multitude of variables and equations through connection equations. This challenge is further exacerbated when the used components have been developed for general purposes, requiring the computation of quantities which may hold no relevance for a particular application. The seemingly simple example of Listing 2.2 is a case in point: The resulting DAE system has 2670 equations and variables prior to symbolic processing. Chapter 4 reviews techniques that often are used to alleviate these issues and how they can be adapted for the purposes of dynamic optimization. While these techniques often are paramount to efficiently treat hierarchical, acausal models, we will see that they hold merit also when applied to examples of specialized, flat models in the context of dynamic optimization.

2.2 Differential-Algebraic Equations

The mathematical formalism of causal, block-oriented modeling is explicit ordinary differential equations (ODE) of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{2.2}$$

where t is time—the sole independent variable—$\boldsymbol{x}$ is the state variable, $\boldsymbol{u}$ is the system input, and $\boldsymbol{p}$ is system parameters that are to be optimized. Such systems are usually readily solved numerically with run-of-the-mill ODE solvers. On the other hand, acausal, hierarchical modeling gives rise to implicit DAEs of the form

$$\boldsymbol{\Phi}(t, \dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, \tag{2.3}$$

where $\boldsymbol{\xi}$ is the differential variable and $\boldsymbol{v}$ is the algebraic variable. DAEs are a generalization of explicit ODEs in that the differential variables occur implicitly. Consequently, the equations can depend nonlinearly on the differential variables, or not at all, giving rise to purely algebraic equations and an accompanying world of trouble when combined with differential equations. For simplicity we assume that the system is balanced; that is, that the dimension of the codomain of $\boldsymbol{\Phi}$ equals the sum of the dimensions of $\boldsymbol{\xi}$ and $\boldsymbol{v}$.

2.2.1 DAE Index

An important notion in the analysis and numerical solution of DAEs is the index [Brenan et al., 1996; Hairer and Wanner, 1996]. There are many different, nonequivalent definitions of the DAE index [Campbell and Gear, 1995]. The perhaps most illuminating and easily understood—but typically

not the most useful for analysis and classification purposes—is the differentiation index. Assuming sufficient differentiability, the differentiation index is the smallest integer ν such that $\dot{\boldsymbol{\xi}}$ and $\dot{\boldsymbol{v}}$ are uniquely determined as continuous functions of

$$\boldsymbol{\xi},\ \boldsymbol{v},\ \boldsymbol{u},\ \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u},\ldots,\ \frac{\mathrm{d}^{\nu}}{\mathrm{d}t^{\nu}}\boldsymbol{u},\ \boldsymbol{p},\ t \tag{2.4}$$

by the derivative array equations

$$\boldsymbol{\Phi}(t,\dot{\boldsymbol{\xi}}(t),\boldsymbol{\xi}(t),\boldsymbol{v}(t),\boldsymbol{u}(t),\boldsymbol{p})=\boldsymbol{0}, \tag{2.5a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\Phi}(t,\dot{\boldsymbol{\xi}}(t),\boldsymbol{\xi}(t),\boldsymbol{v}(t),\boldsymbol{u}(t),\boldsymbol{p})=\boldsymbol{0}, \tag{2.5b}$$

$$\vdots$$

$$\frac{\mathrm{d}^{\nu}}{\mathrm{d}t^{\nu}}\boldsymbol{\Phi}(t,\dot{\boldsymbol{\xi}}(t),\boldsymbol{\xi}(t),\boldsymbol{v}(t),\boldsymbol{u}(t),\boldsymbol{p})=\boldsymbol{0}, \tag{2.5ν}$$

which should be considered as a static system of equations with derivatives of $\boldsymbol{\xi}$ and $\boldsymbol{v}$ of order up to $\nu+1$ and ν, respectively, as unknowns. Note that $\frac{\mathrm{d}}{\mathrm{d}t}$ denotes the total—rather than partial—derivative with respect to t. Hence, the differentiation index ν can be appropriately viewed as the number of times that the equations need to be differentiated to transform the system into an implicit ODE[1]. The resulting ODE, which is called the underlying ODE, is equivalent to the original DAE in the sense that there exists a bijection between the solution sets.

2.2.2 System State

The notion of state is another important concept of dynamical systems. The state is a minimal set of system variables such that if their current values are known and all future system input values are known, then the future behavior of the system is uniquely determined. For explicit ODEs, the differential variable $\boldsymbol{x}$ is a natural and valid candidate for the system state. For typical low-index DAEs—index 0 or 1—the differential variable $\boldsymbol{\xi}$ is still a suitable state (for counterexamples, see [Pantelides, 1988, Example 2; Campbell and Gear, 1995, Example 4] for counterexamples), and will henceforth usually be referred to as such. However, for high-index DAEs—index 2 or higher—the system state is overdetermined by the differential variables; only a proper subset of them will yield a suitable state representation. We use $\boldsymbol{x}$ to denote such a subset, and its construction is discussed further in Sections 2.2.3 and 2.2.5.

[1] By implicit ODE we mean an implicit DAE such that $\nabla_{\dot{\boldsymbol{\xi}}}\boldsymbol{\Phi}(t,\dot{\boldsymbol{\xi}},\boldsymbol{\xi},\boldsymbol{v},\boldsymbol{u},\boldsymbol{p})$ is nonsingular in

The problem of selecting state variables for high-index DAEs is related to the method of index reduction described below and does in general not have a unique solution. Also, certain selections can have better numerical properties. Automated selection of state variables is a difficult problem, so algorithms often need input from the modeler to select appropriate state variables. In general, it is not even possible to preserve solvability with a static choice of state variables [Cellier and Kofman, 2006; Mattsson et al., 2000]. Such cases are however outside the scope of this thesis.

In order for the explicit ODE (2.2) to have a unique solution, it needs to be combined with boundary conditions; for example, by prescribing the value of the state $\boldsymbol{x}$ at some initial time t_0 according to

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{2.6}$$

the state trajectory $\boldsymbol{x}$ is uniquely determined (assuming that $\boldsymbol{f}$ is sufficiently regular). The combination of (2.2) and (2.6) is called an initial-value problem, which we sometimes also will refer to as the problem of (dynamic) simulation.

Initial conditions prescribing the initial state is not always convenient for DAEs, since the modeler is not always aware of which variables will be selected as states. Furthermore, it may be desirable to initialize the system in steady state—that is, $\dot{\boldsymbol{x}}(t_0) = \boldsymbol{0}$—rather than in a specific point. Ergo, rather than (2.6), we consider general initial conditions

$$\boldsymbol{\Phi}_0(\dot{\boldsymbol{\xi}}(t_0), \boldsymbol{\xi}(t_0), \boldsymbol{v}(t_0), \boldsymbol{u}(t_0), \boldsymbol{p}) = \boldsymbol{0}, \tag{2.7a}$$

which together with

$$\boldsymbol{\Phi}(t_0, \dot{\boldsymbol{\xi}}(t_0), \boldsymbol{\xi}(t_0), \boldsymbol{v}(t_0), \boldsymbol{u}(t_0), \boldsymbol{p}) = \boldsymbol{0} \tag{2.7b}$$

form the initial equations that should uniquely determine consistent initial values $\dot{\boldsymbol{\xi}}(t_0)$, $\boldsymbol{\xi}(t_0)$, and $\boldsymbol{v}(t_0)$. We assume that the dimension of the codomain of $\boldsymbol{\Phi}_0$ equals the dimension of the state.

2.2.3 Index Reduction

The concept of differentiation index and underlying ODE suggests an approach to numerically solving DAEs, which is called index reduction: Differentiate the equations until the underlying ODE is obtained, and then solve the underlying ODE. Pantelides's algorithm [Pantelides, 1988] can be used to determine the minimal number of differentiations needed to find the underlying ODE.

a neighborhood of the solution. Consequently by the implicit function theorem, an implicit DAE is an implicit ODE if and only if it is index 0.

An alternative approach is to use specialized DAE solvers [Brenan et al., 1996; Hairer and Wanner, 1996]. However, such solvers are restricted to certain classes of DAEs. There are no satisfactory methods for solving initial-value problems for the general class of high-index DAEs on the form (2.3). In particular, the discretization methods of Chapter 3 have capabilities for solving certain high-index systems, which are further enhanced in the context of dynamic optimization [Campbell and Betts, 2016]. However, applying the methods of Chapter 3 on high-index systems can cause reduced order of convergence or even numerical instability. Furthermore, the methods of Chapter 4 are only applicable on low-index systems. Therefore, we utilize methods based on index reduction.

While index reduction is more widely applicable than typical high-index DAE solvers, it has two potentially major drawbacks: inefficiency and inaccuracy. The inefficiency is caused by the abundant number of state variables that result from differentiating index-1 equations, as this transforms all algebraic variables $\boldsymbol{v}$ to state variables. This issue can be alleviated by stopping the index reduction once an index-1 DAE is obtained and then instead rely on causalization techniques, which is the topic of Chapter 4. The second issue of inaccuracy is caused by the loss of information that occurs when replacing an equation by its differentiated equivalent. The algebraic relations of the original DAE are only implicitly preserved in the underlying ODE as solutions invariants, which in general will not be preserved under discretization. The result is that the numerical solution will drift away from the solution manifold.

There are various remedies to the issue of numerical drift. Most of them are specialized to certain classes of DAEs and based on techniques such as constraint stabilization [Baumgarte, 1972] or projections [Campbell, 1985]. Another way of preventing numerical drift is the method of dummy derivatives [Mattsson and Söderlind, 1993], which is applicable to a large class of DAEs. This method is related to the approach of index reduction outlined above, but retains both the original and differentiated equations and introduces new variables called dummy derivatives to obtain a balanced system. This method however requires the selection of state variables.

2.2.4 Example

To illustrate the concept of DAE index and the method of dummy derivatives, we review the introductory example of [Mattsson and Söderlind, 1993]. Consider the DAE

$$\dot{x}(t) = y(t), \tag{2.8a}$$
$$\dot{y}(t) = z(t), \tag{2.8b}$$
$$x(t) = u(t), \tag{2.8c}$$

where x and y are differential variables, z is an algebraic variable, and u is a prescribed system input. This example may be thought of as a prescribed-trajectory problem in mechanics, where x, y, and z are position, velocity and force per unit mass, respectively, and u is the desired trajectory. This is an index-3 DAE, since by differentiating (2.8a) once and (2.8c) twice we obtain

$$\ddot{x}(t) = \dot{y}(t), \tag{2.9a}$$
$$\dot{y}(t) = z(t), \tag{2.9b}$$
$$\ddot{x}(t) = \ddot{u}(t), \tag{2.9c}$$

which can be seen to be a second-order index-1 DAE.

Numerical drift may actually not be an issue in the index-reduced formulation (2.9), because of the linearity of (2.8). Had it however been nonlinear, the numerical solution of (2.9) would not necessarily have satisfied $x(t) = u(t)$ even if $x(t_0) = u(t_0)$. The method of dummy derivatives deals with this by first combining the original DAE with the index-reduced equations, in this case yielding

$$\dot{x}(t) = y(t), \tag{2.10a}$$
$$\dot{y}(t) = z(t), \tag{2.10b}$$
$$x(t) = u(t), \tag{2.10c}$$
$$\ddot{x}(t) = \dot{y}(t), \tag{2.10d}$$
$$\dot{x}(t) = \dot{u}(t), \tag{2.10e}$$
$$\ddot{x}(t) = \ddot{u}(t). \tag{2.10f}$$

This system is however overdetermined, having six equations but only the three unknowns $\ddot{x}$, $\dot{y}$, z (the highest-order derivative of each noninput variable). This is resolved by for each differentiated equation, designate one of the noninput variables occurring in the differentiated equation as a dummy derivative, replacing it by a new nondifferentiated variable. In general there are multiple possible selection of dummy derivatives, but in our example there are only three alternatives—$\ddot{x}$, $\dot{x}$, and $\dot{y}$—which is exactly how many dummy derivatives we need, since we have exactly three differentiated equations: (2.10d)–(2.10f). Consequently, we replace these three variables by their corresponding dummy derivatives ddx, dx, and dy—which become algebraic

variables and hence are not subject to discretization—yielding

$$dx(t) = y(t), \tag{2.11a}$$
$$dy(t) = z(t), \tag{2.11b}$$
$$x(t) = u(t), \tag{2.11c}$$
$$ddx(t) = dy(t), \tag{2.11d}$$
$$dx(t) = \dot{u}(t), \tag{2.11e}$$
$$ddx(t) = \ddot{u}(t). \tag{2.11f}$$

The differential variables whose derivative have not been selected as dummy derivatives form a natural choice of state variables. As it turns out in this very special example, there are no differential variables left, and the system consequently has no internal state. Indeed, the position, velocity, and acceleration (force) is entirely determined by the prescribed trajectory u for the position; no initial conditions are needed to define a unique solution.

2.2.5 Summary

Applying the method of dummy derivatives on the high-index DAE (2.3) results in a balanced, (usually) low-index DAE

$$\boldsymbol{F}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}) = 0, \tag{2.12}$$

where $\boldsymbol{x}$ is the differential variable, which is the subset of $\boldsymbol{\xi}$ whose derivatives have not been designated as dummy derivatives, and $\boldsymbol{y}$ is the remainder of $\boldsymbol{\xi}$ as well as all of $\boldsymbol{v}$. We will in general use $\boldsymbol{\xi}$ and $\boldsymbol{v}$ to denote the differential and algebraic, respectively, variables of a general—possibly high-index—DAE $\boldsymbol{\Phi}$, as well as $\boldsymbol{x}$ and $\boldsymbol{y}$ to denote the differential (state) and algebraic, respectively, variables of a low-index DAE $\boldsymbol{F}$.

The DAE (2.12) resulting from the method of dummy derivatives will satisfy that the Jacobian

$$\left[\nabla_{\dot{\boldsymbol{x}}} \boldsymbol{F}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}) \quad \nabla_{\boldsymbol{y}} \boldsymbol{F}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p})\right] \tag{2.13}$$

is structurally nonsingular—that is, it can be permuted to have a diagonal with no identically zero elements—which is a prerequisite for numerical nonsingularity. What the method does not guarantee, but that we nevertheless further assume, is that the Jacobian is nonsingular in all relevant points, which is a sufficient condition for $\boldsymbol{x}$ to be a possible static choice of state. It is also sufficient for $\boldsymbol{F}$ to be index 1.

With the outlined method of dummy derivatives we have a way of treating a wide class of high-index implicit DAEs by transforming it to a class of low-index DAEs that lends itself better to numerical solution by general-purpose solvers.

2.2.6 Semi-Explicit Form

Another form of DAEs is the semi-explicit form

$$\dot{\boldsymbol{\xi}}(t) = \boldsymbol{f}(t, \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{2.14a}$$

$$\boldsymbol{g}(t, \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, \tag{2.14b}$$

which often lends itself better to analysis; for example, it is index 1 if and only if

$$\nabla_{\boldsymbol{v}} \boldsymbol{g}(t, \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t), \boldsymbol{p}) \tag{2.15}$$

is nonsingular. While most literature on dynamic optimization focuses on the semi-explicit form, this thesis focuses on the fully implicit form (2.3), because of the goal of treating general Modelica models. Although the fully implicit form (2.3) is more general, it can always be transformed to the semi-explicit form (2.14), which however comes at the price of increased index and number of variables and equations.

2.3 Dynamic Optimization

Before we consider general formulations of dynamic optimization problems, we will have a look at a simple example of optimal control.

2.3.1 Example

Consider again the Van der Pol (VDP) oscillator (2.1). The problem we will solve is to drive the state (x_1, x_2) from $(0, 1)$ toward the origin using a quadratic Lagrange cost on the state and input on a fixed time horizon going from $t_0 = 0$ to $t_f = 10$ using different weights $r \in (0, \infty)$ for the input cost. We will also have an upper limit on the input u of 0.8. The resulting problem is

$$\begin{aligned}
&\text{minimize} && \int_0^{10} \left(x_1^2(t) + x_2^2(t) + r u^2(t)\right) \mathrm{d}t, && (2.16a)\\
&\text{with respect to} && \boldsymbol{x} : [0, 10] \to \mathbb{R}^2, \quad u : [0, 10] \to \mathbb{R}, \\
&\text{subject to} && \dot{x}_1(t) = \left(1 - x_2^2(t)\right) x_1(t) - x_2(t) + u(t), && (2.16b)\\
& && \dot{x}_2(t) = x_1(t), && (2.16c)\\
& && x_1(0) = 0, \qquad x_2(0) = 1, && (2.16d)\\
& && u(t) \leq 0.8, && (2.16e)\\
& && \forall t \in [0, 10].
\end{aligned}$$

This problem is readily solved by JModelica.org, and code for doing so will be given in Chapter 3. For now, we just look at the solution in Figure 2.2, where we see that smaller values of r yield better state control at the expense of larger control action.

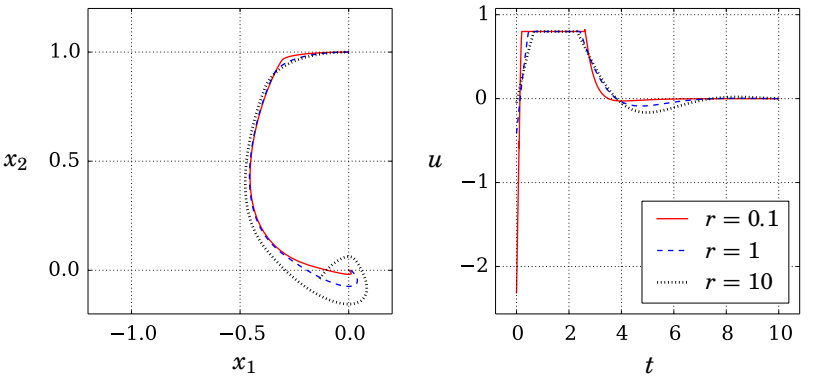

Figure 2.2 Optimal control of the Van der Pol oscillator on a fixed time horizon of 10 seconds. The initial state is (0, 1) and the objective is a quadratic penalty on the state and input, with the input penalty being weighted by r.

2.3.2 A General Case

A fairly general formulation of dynamic optimization problems (DOP) similar to what is found in most literature [Liberzon, 2012; Biegler, 2010; Betts, 2010] is

$$
\begin{aligned}
&\text{minimize} && \phi(t_f, \boldsymbol{x}(t_f)) + \int_{t_0}^{t_f} L\big(\boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t)\big)\,\mathrm{d}t, && \text{(2.17a)}\\
&\text{with respect to} && \boldsymbol{x} : [t_0, t_f] \to \mathbb{R}^{n_x}, \quad \boldsymbol{y} : [t_0, t_f] \to \mathbb{R}^{n_y},\\
& && \boldsymbol{u} : [t_0, t_f] \to \mathbb{R}^{n_u}, \quad t_f \in \mathbb{R}, \quad \boldsymbol{p} \in \mathbb{R}^{n_p},\\
&\text{subject to} && \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}), && \text{(2.17b)}\\
& && \boldsymbol{g}(t, \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}) = \boldsymbol{0}, && \text{(2.17c)}\\
& && \boldsymbol{x}(t_0) = \boldsymbol{x}_0, && \text{(2.17d)}\\
& && \boldsymbol{h}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}) \leq \boldsymbol{0}, && \text{(2.17e)}\\
& && \boldsymbol{H}(t_f, \boldsymbol{x}(t_f), \boldsymbol{y}(t_f), \boldsymbol{p}) \leq \boldsymbol{0}, && \text{(2.17f)}\\
& && \forall t \in [t_0, t_f],
\end{aligned}
$$

where the objective (2.17a) comprises the Mayer term ϕ and Lagrange integrand L, (2.17b) and (2.17c) are the low-index, semi-explicit system dynamics, (2.17d) is prescribed state initial conditions, (2.17e) is path inequality constraints, and (2.17f) is the terminal constraints. The optimization variables are $\boldsymbol{x}$ (which inherently determines $\dot{\boldsymbol{x}}$), $\boldsymbol{y}$, $\boldsymbol{u}$, t_f, and $\boldsymbol{p}$. The degrees of freedom are induced by $\boldsymbol{u}$, t_f, and $\boldsymbol{p}$, with $\boldsymbol{x}$ and $\boldsymbol{y}$ being determined by the system dynamics (2.17b) and (2.17c). It is possible to also consider

equality versions of the path and terminal constraints (2.17e) and (2.17f), which further reduces the available degrees of freedom.

Problem (2.16) is an example of a control problem. Dual to such problems are dynamic estimation problems, where unknown variables and parameters are estimated by fitting the model response to measurement data. There are many similarities between these two classes of problems, allowing them both to be captured within the general problem (2.17).

We will in Chapter 3 introduce a more general version of (2.17), which the thesis focuses on. An important and common generalization of (2.17) that we however do not consider is the introduction of multiple phases, in which the time horizon $[t_0, t_f]$ is divided into subphases and the constraints, objective, and variables are allowed to be redefined within each phase. We choose to neglect this generalization mainly because of laborious implementation rather than theoretical difficulties. Another important generalization that the thesis does not consider (although the author has done some work related to this [Fouquet et al., 2016; Ghazaei Ardakani and Magnusson, 2016]) is the inclusion of hybrid dynamics, which often is computationally intractable when combined with large-scale, nonlinear dynamics.

2.3.3 Optimica

Modelica is designed mainly with simulation-based analysis in mind, and lacks native support for optimization formulations. In the encoding of (2.17), we can therefore only rely on Modelica for (2.17b) to (2.17d). To accommodate the need for formulation of DOPs based on Modelica code, the language extension Optimica [Åkesson, 2008; Åkesson, 2007] was developed and integrated with JModelica.org. Optimica defines new syntax and semantics for specifying constraints and an objective.

Example We revisit the VDP example (2.16). Listing 2.1 was used to encode (2.16b) to (2.16d) in Modelica. The remainder of (2.16) is encoded in Optimica in Listing 2.3, where we inherit the model and add the time horizon, Lagrange cost, and input bound on top of it. Note the clear separation between model and optimization formulation offered by Optimica, facilitating the reuse of the model for purposes entirely different than dynamic optimization.

Listing 2.3 Optimica implementation of VDP oscillator optimal control.

```
optimization VDP_DOP(finalTime=10,
                     objectiveIntegrand=x1^2 + x2^2 + r*u^2)
    extends VDP(u(max=0.8));
    parameter Real r = 1; // Can be changed after compilation
end VDP_DOP;
```

Syntax Optimica introduces the specialized class `optimization`, which behaves similarly to `model` from Modelica but with some extra constructs. The class has four[2] new attributes: `objective`, `objectiveIntegrand`, `startTime`, and `finalTime`. The terms ϕ and L in the objective (2.17) are encoded using `objective` and `objectiveIntegrand`, respectively. The time horizon $[t_0, t_f]$ is encoded using `startTime` and `finalTime`.

Two new attributes are defined for the built-in type `Real`: `free` and `initialGuess`. The Boolean attribute `free` is used to turn a regular parameter into an optimization variable; that is, make it a part of $\boldsymbol{p}$. It can also be used for `startTime` and `finalTime` to signify free time horizon endpoints. Initial guesses for each variable that are constant with respect to time can be provided using the `initialGuess` attribute. It is however often both more effective and convenient to rely on dynamic simulation to generate initial guesses, as will be discussed in Section 3.5.1.

A new section called `constraint` is introduced to encode (2.17e) and (2.17f). This is similar to the `equation` section of Modelica, but also includes inequalities and also constraints that only hold at discrete points in time.

Finally, the JModelica.org implementation of Optimica also gives new semantic meaning to the existing Modelica variable attributes `min` and `max` to encode lower and upper variable bounds, respectively. While such constraints are subsumed by (2.17e), there are reasons—which will become clear in Section 2.4.1—for separating the simple variable bounds. While this new attribute meaning often is convenient and useful, it can lead to problematic semantic clashes, since in Modelica these attributes are mainly used to specify model validity regions rather than constraints to be enforced. This problem will be further discussed in Section 4.4.1

Proliferation Optimica was previously used only in JModelica.org but has since also been adopted by OpenModelica [Bachmann et al., 2012] to solve optimal control problems described in Modelica using direct multiple shooting or collocation. Optimica also served as a basis for IDOS [Pytlak et al., 2014], an online environment for solving a wide variety of optimal control problems using different techniques.

Although the idea of applying Modelica for dynamic optimization has existed for many years, with [Krüger et al., 2004; Pettersson et al., 2005] being early examples, the interest within the Modelica community ever increases. Consequently, there were efforts within the MODRIO project [ITEA, 2015] to standardize DOP formulations for Modelica. Incorporating Optimica into the Modelica standard was however deemed too complex due to the many additional keywords, which would encumber the already saturated seman-

[2] Optimica also defines a fifth attribute called `static`, which is used to formulate static, rather than dynamic, optimization problems. Such problems—despite their relative simplicity—are however outside the scope of both this thesis and today's JModelica.org.

tics of Modelica. The MODRIO efforts thus focused on the use of custom annotations [Zimmer et al., 2014].

2.3.4 Numerical Methods

The domain of (2.17)—that is, functions on $[t_0, t_f]$—is infinite-dimensional. To (approximately) find a solution, we rely on numerical methods. The feasible set of (2.17) will in general also be nonconvex because of the nonlinearity of $\boldsymbol{f}$ and $\boldsymbol{g}$. We will not endeavor to find a global optimum. We will instead rely on first-order, necessary optimality conditions to numerically find a local optimum.

There are many approaches to numerically solving DOPs, which stem from the theory of optimal control. The earliest methods date back to the 1950s and are based on Bellman's dynamic programming, of which a modern description can be found in for example [Bertsekas, 2005]. The main result on dynamic programming for continuous-time systems is the Hamilton-Jacobi-Bellman equation, which is a nonlinear partial differential equation. The dynamic programming framework is theoretically appealing, due to providing sufficient conditions for global optimality and also yielding solutions in the form of state feedback laws. However, in practice it suffers from the curse of dimensionality: The dimension of the domain of the partial differential equation increases with the dimension of the system state. Therefore, numerical methods based on dynamic programming are only computationally tractable for small-scale problems.

The most widely used numerical techniques for optimal control today are instead based on first-order necessary conditions for local optimality. Surveys of these are available in for example [Betts, 1998; Rao, 2009; Biegler, 2010] and an overview is illustrated in Figure 2.3. They can be categorized according to their respective answers to two questions: When to discretize the system dynamics, and how to discretize them? There are essentially two answers to the first question, which have led to the two categories of indirect and direct methods. Indirect methods start by establishing the optimality conditions for (2.17), and then discretize the resulting differential equations to find a numerical solution. Direct methods instead first discretize the dynamics and then establish the optimality conditions. Indirect methods are based on calculus of variations and Pontryagin's maximum principle [Liberzon, 2012], which provide optimality conditions in the form of boundary-value problems. Standard numerical methods for boundary-value problems can then be employed to solve the problem numerically. Direct methods instead reduce the DOP to a nonlinear program (see Section 2.4) by discretization. The optimality conditions are then given by the Karush-Kuhn-Tucker (KKT) conditions, as will be discussed in Section 2.4.

Both direct and indirect approaches discretize differential equations, and

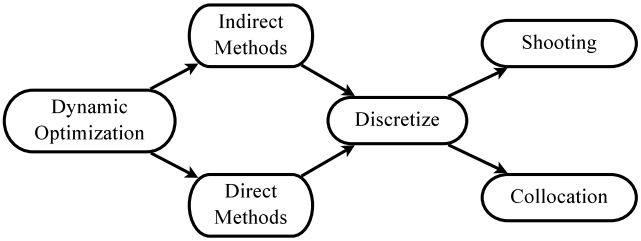

Figure 2.3 Numerical methods for dynamic optimization. Indirect methods establish optimality conditions first and then discretize the differential equations, whereas direct methods first discretize and then optimize. Both categories of methods use essentially the same discretization techniques, of which the most common ones are single shooting, multiple shooting, and collocation. This thesis focuses on direct collocation.

the same discretization methods are commonly used for both approaches. The most common methods belong to one of two families: shooting and collocation. These two families of methods are sometimes also referred to as sequential and simultaneous, respectively, with similar meaning. The simplest form of shooting is single shooting, which parametrizes the input—explicitly for the direct method and implicitly by the maximum principle and costate initial values for the indirect method—and then numerically integrates to t_f and iteratively updates the control based on sensitivities. The numerical robustness of single shooting can be improved by dividing the time horizon into subintervals. Single shooting is then applied within each subinterval, by introducing the subinterval boundary values as variables and imposing linking constraints between the subintervals. This is called multiple shooting, which essentially decouples the dynamics between the subintervals. Multiple shooting is a hybrid between sequential and simultaneous methods, which brings us to the second family of discretization methods.

Simultaneous methods simultaneously discretize the differential equations over the entire time horizon, using implicit Runge-Kutta methods in the case of collocation. Consequently, they do not rely on external numerical integrators, because after the full discretization of the differential equations the optimality conditions are reduced to an algebraic root finding problem. Collocation methods can be either local, where the time horizon is divided into elements and low-order polynomials are used to approximate the trajectories within each element, or global, where a single high-order polynomial is used over the entire time horizon. Global methods are also synonymously referred to as pseudospectral methods.

Shooting methods (especially single shooting) lead to optimization problems with few variables but highly nonlinear functions—because of their

internalization of the differential equations and numerical integrators—whereas simultaneous methods lead to problems with less severe nonlinearities but many variables. Indirect methods need good initial guesses of the costates and also identification of the switching structure of inequality constraints, both of which require proficiency in the maximum principle for all but the most simple problems. Single shooting suffers from the numerical sensitivity mentioned above. Thus, direct multiple shooting and direct collocation appear to be the most suitable methods to be used in a high-level, general-purpose framework for large-scale dynamic optimization. This thesis focuses on direct collocation, the details of which are discussed in Chapter 3.

2.4 Nonlinear Programming

The notation used in this subsection is independent of the rest of the thesis. For example, $\boldsymbol{x}$ refers to variables of a nonlinear program, rather than the state of a dynamical system.

Direct methods—in particular direct collocation—for dynamic optimization transcribe the infinite-dimensional DOP (2.17) into a finite-dimensional nonlinear program (NLP), which has the general form

$$\begin{aligned}
&\text{minimize} && f(\boldsymbol{x}), && \text{(2.18a)}\\
&\text{with respect to} && \boldsymbol{x} \in \mathbb{R}^{n_x}, &&\\
&\text{subject to} && \boldsymbol{x}_L \leq \boldsymbol{x} \leq \boldsymbol{x}_U, && \text{(2.18b)}\\
& && \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}, && \text{(2.18c)}\\
& && \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0}, && \text{(2.18d)}
\end{aligned}$$

where the bounds (2.18b) are separated from the general nonlinear inequalities (2.18d) to allow for more efficient treatment of them. The lower and upper bounds, $\boldsymbol{x}_L$ and $\boldsymbol{x}_U$, can attain infinite values, in which case the corresponding components of $\boldsymbol{x}$ are unbounded.

There are many methods available for solving (2.18) [Nocedal and Wright, 2006]. The most common ones are based on either active-set sequential quadratic programming, in which the NLP is iteratively approximated by a quadratic program, or interior-point methods, which can be viewed as approximating the NLP by an equality-constrained NLP. Since this thesis mostly makes use of interior-point methods, we will review the basics of these.

2.4.1 Interior-Point Methods

A common transformation of (2.18) is to get rid of the general inequality constraints by introducing a slack variable $\boldsymbol{s}$ to obtain

$$\begin{aligned}
&\text{minimize} && f(\boldsymbol{x}), && (2.19a)\\
&\text{with respect to} && \boldsymbol{x} \in \mathbb{R}^{n_x}, \quad \boldsymbol{s} \in \mathbb{R}^{n_s},\\
&\text{subject to} && \boldsymbol{x}_L \leq \boldsymbol{x} \leq \boldsymbol{x}_U, && (2.19b)\\
& && \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}, && (2.19c)\\
& && \boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{s} = \boldsymbol{0}, && (2.19d)\\
& && \boldsymbol{s} \leq \boldsymbol{0}. && (2.19e)
\end{aligned}$$

For the sake of clarity, we will consider the simpler NLP

$$\begin{aligned}
&\text{minimize} && f(\boldsymbol{x}), && (2.20a)\\
&\text{with respect to} && \boldsymbol{x} \in \mathbb{R}^{n_x},\\
&\text{subject to} && \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}, && (2.20b)\\
& && \boldsymbol{x} \geq \boldsymbol{0}. && (2.20c)
\end{aligned}$$

Although (2.19) can be transformed to (2.20), the treatment of (2.18) typically instead relies on straightforward generalizations of the steps below.

A common variant of interior-point methods that uses logarithmic functions can be viewed as transforming (2.20) to the equality-constrained NLP

$$\begin{aligned}
&\text{minimize} && f(\boldsymbol{x}) - \mu \sum_{i=1}^{n} \ln(x_i), && (2.21a)\\
&\text{with respect to} && \boldsymbol{x} \in \mathbb{R}^{n_x},\\
&\text{subject to} && \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}, && (2.21b)
\end{aligned}$$

where μ is the barrier parameter. By letting μ tend to zero, the solution of (2.21) approaches the solution of (2.20) under mild assumptions, see for example [Biegler, 2010, Theorem 6.7].

2.4.2 Karush-Kuhn-Tucker Conditions

To solve (2.21), we first introduce the Lagrangian function

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) := f(\boldsymbol{x}) - \mu \sum_{i=1}^{n_x} \ln(x_i) + \boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{\lambda}, \tag{2.22}$$

where $\boldsymbol{\lambda}$ is the dual variable, whose dimension equals the dimension of the codomain of $\boldsymbol{g}$. Assuming that some constraint qualifications are satisfied,

first-order local optimality conditions for (2.21) are given by the Karush-Kuhn-Tucker (KKT) conditions

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \nabla f(\boldsymbol{x}) - \mu \boldsymbol{x}^{-1} + \nabla \boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{\lambda} = \mathbf{0}, \tag{2.23a}$$

$$\boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}, \tag{2.23b}$$

where $\boldsymbol{x}^{-1}$ denotes elementwise inversion. A common, but relatively restrictive, constraint qualification is the Linear Independence Constraint Qualification (LICQ), which holds at a point of (2.18) if the Jacobian of the equality and active inequality constraints at that point has full row rank.

The KKT conditions (2.23) are called the primal equations, and unfortunately suffer from numerical ill-conditioning as x_i approaches its lower bound of 0 because of the $\mu \boldsymbol{x}^{-1}$ term. The equivalent primal-dual equations

$$\nabla_{\boldsymbol{x}} \bar{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{z}) = \nabla f(\boldsymbol{x}) - \boldsymbol{z} + \nabla \boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{\lambda} = \mathbf{0}, \tag{2.24a}$$

$$\boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}, \tag{2.24b}$$

$$\operatorname{diag}(\boldsymbol{z})\boldsymbol{x} = \mu \mathbf{1} \tag{2.24c}$$

alleviate this issue, where $\bar{\mathcal{L}}$ is implicitly defined by (2.24a) and $\boldsymbol{z}$ can be seen as the dual variable of the original bound (2.20c) when $\mu = 0$. The primal-dual equations lend themselves better to algorithmic development, and also present a different interpretation of interior-point methods: The primal-dual equations are a perturbation of the KKT conditions for (2.20), which are recovered by letting μ equal zero.

2.4.3 Newton's Method

We can thus find a local solution of (2.20) by solving the square system of equations (2.24). This can be done iteratively using Newton's method, in which we in the current iterate $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{z}_k)$ need to solve the linear system of equations

$$\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} \bar{\mathcal{L}}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{z}_k) & \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}_k)^T & -\mathbf{I} \\ \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}_k) & \mathbf{0} & \mathbf{0} \\ \operatorname{diag} \boldsymbol{z}_k & \mathbf{0} & \operatorname{diag} \boldsymbol{x}_k \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{z} \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{x}} \bar{\mathcal{L}}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{z}_k) \\ \boldsymbol{g}(\boldsymbol{x}_k) \\ \operatorname{diag}(\boldsymbol{z}_k)\boldsymbol{x}_k - \mu \mathbf{1} \end{bmatrix}, \tag{2.25}$$

where $\Delta \boldsymbol{x}$, $\Delta \boldsymbol{\lambda}$, and $\Delta \boldsymbol{z}$ are the primal and dual search directions.

In order to successfully do this, $\mathcal{L}$ needs to be twice continuously differentiable, and by extension also $\boldsymbol{f}$ and $\boldsymbol{g}$. We will in Chapter 3 see the implications this assumption has regarding differentiability of the functions in the DOP (2.17), but in short, they also need to be twice continuously differentiable.

2.4.4 Barrier Parameter Selection

An important part of interior-point methods is the selection of the barrier parameter μ in each iteration. A simple but effective method is to do it monotonously: Fix μ, solve (2.21) to some tolerance, decrease μ and repeat until μ is smaller than some tolerance.

An alternative approach is to instead adaptively choose μ in each iteration depending on the progress of the algorithm, which often yields better performance than the monotone strategy. These strategies usually select μ proportional to the complementarity; that is,

$$\mu_k = \sigma_k \frac{\boldsymbol{x}_k^T \boldsymbol{z}_k}{n_x}, \tag{2.26}$$

where σ_k is the centering parameter. Several different strategies have been proposed for selection of σ_k [Nocedal and Wright, 2006; Nocedal et al., 2009].

3

Dynamic Optimization in JModelica.org

> [...] developers of models should consider in the initial stages the ultimate need to solve an optimization problem, since it is unlikely that optimization software will ever reach the state wherein a general routine can be used with impunity.
>
> [Gill et al., 1981]

This chapter presents the current open-source toolchain in JModelica.org for numerically solving large-scale dynamic optimization problems. Its development has been an ongoing effort during the thesis work. The chapter focuses on the direct collocation algorithm used to transcribe dynamic optimization problems to nonlinear programs.

3.1 Problem Formulation

In Section 2.3.2 we discussed the general DOP formulation (2.17). This section presents a further generalized DOP formulation, which is the target of the remainder of the thesis. This formulation is strongly correlated with what can be formulated using Optimica.

3.1.1 High-Index Formulation

For ease of notation, we introduce compositions of the time-dependent variables:

$$\boldsymbol{\zeta}(t) := \left(\dot{\boldsymbol{\xi}}(t), \boldsymbol{\xi}(t), \boldsymbol{v}(t), \boldsymbol{u}(t)\right), \tag{3.1a}$$

$$\boldsymbol{z}(t) := \left(\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t)\right), \tag{3.1b}$$

where $\boldsymbol{z}$ is used for low-index DAEs and $\boldsymbol{\zeta}$ is used for general DAEs. The considered problem—whose concepts and notation are discussed below—is

$$\begin{aligned}
&\text{min.} && \phi(t_0, t_f, \boldsymbol{\zeta}_T, \boldsymbol{p}) + \int_{t_0}^{t_f} L(t, \boldsymbol{\zeta}(t), \boldsymbol{\zeta}_T, \boldsymbol{p})\,\mathrm{d}t, && \text{(3.2a)}\\
&\text{w.r.t.} && \boldsymbol{\zeta} : [t_0, t_f] \to \mathbb{R}^{n_\zeta}, \quad t_0 \in \mathbb{R}, \quad t_f \in \mathbb{R}, \quad \boldsymbol{p} \in \mathbb{R}^{n_p}, &\\
&\text{s.t.} && \boldsymbol{\Phi}(t, \boldsymbol{\zeta}(t), \boldsymbol{p}) = \boldsymbol{0}, \qquad \boldsymbol{\Phi}_0(t_0, \boldsymbol{\zeta}(t_0), \boldsymbol{p}) = \boldsymbol{0}, && \text{(3.2b)}\\
& && \boldsymbol{g}_e(t_0, t_f, t, \boldsymbol{\zeta}(t), \boldsymbol{\zeta}_T, \boldsymbol{p}) = \boldsymbol{0}, \qquad \boldsymbol{g}_i(t_0, t_f, t, \boldsymbol{\zeta}(t), \boldsymbol{\zeta}_T, \boldsymbol{p}) \le \boldsymbol{0}, && \text{(3.2c)}\\
& && \boldsymbol{G}_e(t_0, t_f, \boldsymbol{\zeta}_T, \boldsymbol{p}) = \boldsymbol{0}, \qquad \boldsymbol{G}_i(t_0, t_f, \boldsymbol{\zeta}_T, \boldsymbol{p}) \le \boldsymbol{0}, && \text{(3.2d)}\\
& && \boldsymbol{\zeta}_L \le \boldsymbol{\zeta}(t) \le \boldsymbol{\zeta}_U, \qquad \boldsymbol{p}_L \le \boldsymbol{p} \le \boldsymbol{p}_U, && \text{(3.2e)}\\
& && t_{0,L} \le t_0 \le t_{0,U}, \qquad t_{f,L} \le t_f \le t_{f,U}, && \text{(3.2f)}\\
& && \forall t \in [t_0, t_f]. &
\end{aligned}$$

The time horizon endpoints t_0 and t_f may be either free or fixed. The (possibly) high-index system dynamics (2.3) are incorporated in (3.2b) together with the fully implicit initial conditions (2.7). Note that solving the initial equations for the initial state will be done as a part of solving the optimization problem, which for example enables the treatment of problems where the initial state is not known a priori.

3.1.2 Objective and Timed Variables

The objective (3.2a) is a typical optimal control objective, but general enough to cover other problems of interest, such as parameter estimation. The Mayer term has been generalized compared to the standard one in (2.17a). The essence of the generalization is that instead of depending on only the terminal state, it depends on the state at an arbitrary (but finite) number of time points within the time horizon. This gives rise to the notion of timed variables, which we denote by $\boldsymbol{\zeta}_T$, which we define by first denoting the needed time points by $T_1, T_2, \ldots, T_{n_T}$, where $n_T \in \mathbb{Z}$ is the number of such time points. We require each such time point to be equal to a convex combination of t_0 and t_f; that is,

$$\forall j \in [1..n_T]\ \exists \theta_j \in [0, 1] : T_j = (1 - \theta_j)t_0 + \theta_j t_f. \tag{3.3}$$

For problems with a fixed time horizon, this simply means that $T_i \in [t_0, t_f]$. For problems with a free time horizon, this implies that the locations of the time points depend on the optimal t_0 and t_f. We then let

$$\boldsymbol{\zeta}_T := \big(\boldsymbol{\zeta}(T_1), \boldsymbol{\zeta}(T_2), \ldots, \boldsymbol{\zeta}(T_{n_T})\big) \in \mathbb{R}^{n_T n_\zeta}. \tag{3.4}$$

For later use, $\boldsymbol{z}_T$ is defined analogously.

The standard Mayer term only involves the single time point $T_1 = t_f$. One application of the generalized Mayer term is the formulation of parameter estimation problems, where there typically is measurement data for the system outputs at discrete time points, which is used to penalize the deviation of the model output from the data values at these points. An alternative approach in this case is to interpolate the measurement data to form a continuous-time measurement trajectory. This trajectory can then instead be used to form a Lagrange integrand which penalizes the deviation of the model output from the measurements. The occurrence of the timed variables $\boldsymbol{\zeta}_T$ is not restricted to the Mayer term. The timed variables can also be used in the Lagrange integrand and the constraints, as will be discussed in Section 3.1.3. The introduction of timed variables can be considered to be subsumed by the more common generalization of multiple phases, as discussed in Section 2.3.2.

3.1.3 Constraints

The constraints (3.2c) and (3.2d) are path constraints and point constraints, respectively, in both inequality and equality form. These are essentially the same as (2.17e) and (2.17f), although they also include the timed variables $\boldsymbol{\zeta}_T$. Hence, the number of time points n_T is not only the number of time points involved in the Mayer term, but also includes the number of time points needed to formulate the path and point constraints as well as the Lagrange term.

The constraints (3.2e) and (3.2f) are variable bounds, where $\boldsymbol{\zeta}_L, \boldsymbol{p}_L, t_{0,L}$, and $t_{f,L}$ are lower bounds and $\boldsymbol{\zeta}_U, \boldsymbol{p}_U, t_{0,U}$, and $t_{f,U}$ are upper bounds. While serving a similar purpose as the path inequalities, they are separated for the same reasons as the bounds in (2.18) were separated from the inequalities. Likewise, the lower and upper bounds can attain infinite values.

3.1.4 Differentiability

As discussed in Section 2.4, we assume that the objective functions ϕ and L, the DAE system and initial condition residuals $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}_0$, the path constraint functions $\boldsymbol{g}_e$ and $\boldsymbol{g}_i$ as well as the point constraint functions $\boldsymbol{G}_e$ and $\boldsymbol{G}_i$ are all sufficiently differentiable. The details of this assumption will be specified in Section 3.3.

3.1.5 Low-Index Formulation

While the toolchain targets the high-index formulation (3.2), the methods of this thesis target low-index problems. Hence, we rely on the JModelica.org compiler, which will be described in Section 3.2.3, to perform index reduction using the method of dummy derivatives outlined in Section 2.2. The result is a low-index version of (3.2), in which essentially $\boldsymbol{\Phi}$, $\boldsymbol{\Phi}_0$, $\boldsymbol{\zeta}$, $\boldsymbol{\zeta}_T$ are replaced

by $\boldsymbol{F}$, $\boldsymbol{F}_0$, $\boldsymbol{z}$, and $\boldsymbol{z}_T$, respectively, yielding

$$\begin{aligned}
\text{min.} \quad & \phi(t_0, t_f, \boldsymbol{z}_T, \boldsymbol{p}) + \int_{t_0}^{t_f} L(t, \boldsymbol{z}(t), \boldsymbol{z}_T, \boldsymbol{p})\,\mathrm{d}t, && && \text{(3.5a)}\\
\text{w.r.t.} \quad & \boldsymbol{z} : [t_0, t_f] \to \mathbb{R}^{n_z}, \quad t_0 \in \mathbb{R}, \quad t_f \in \mathbb{R}, \quad \boldsymbol{p} \in \mathbb{R}^{n_p}, && &&\\
\text{s.t.} \quad & \boldsymbol{F}(t, \boldsymbol{z}(t), \boldsymbol{p}) = \boldsymbol{0}, && \boldsymbol{F}_0(t_0, \boldsymbol{z}(t_0), \boldsymbol{p}) = \boldsymbol{0}, && \text{(3.5b)}\\
& \boldsymbol{g}_e(t_0, t_f, t, \boldsymbol{z}(t), \boldsymbol{z}_T, \boldsymbol{p}) = \boldsymbol{0}, && \boldsymbol{g}_i(t_0, t_f, t, \boldsymbol{z}(t), \boldsymbol{z}_T, \boldsymbol{p}) \le \boldsymbol{0}, && \text{(3.5c)}\\
& \boldsymbol{G}_e(t_0, t_f, \boldsymbol{z}_T, \boldsymbol{p}) = \boldsymbol{0}, && \boldsymbol{G}_i(t_0, t_f, \boldsymbol{z}_T, \boldsymbol{p}) \le \boldsymbol{0}, && \text{(3.5d)}\\
& \boldsymbol{z}_L \le \boldsymbol{z}(t) \le \boldsymbol{z}_U, && \boldsymbol{p}_L \le \boldsymbol{p} \le \boldsymbol{p}_U, && \text{(3.5e)}\\
& t_{0,L} \le t_0 \le t_{0,U}, && t_{f,L} \le t_f \le t_{f,U}, && \text{(3.5f)}\\
& \forall t \in [t_0, t_f], && &&
\end{aligned}$$

where there is a slight abuse of notation in the use of $\boldsymbol{g}_e$, $\boldsymbol{g}_i$, $\boldsymbol{G}_e$, $\boldsymbol{G}_i$, ϕ, and L, as the dimension of $\boldsymbol{z}$ is generally larger than the dimension of $\boldsymbol{\zeta}$. It could be worthwhile to also consider index reduction of (3.5c) and (3.5e)—especially in combination with multiple phases, which would enable identification of switching points—which however is outside the scope of this thesis.

3.2 Related Software and Languages

In this section we first give an overview of software tools that are available for numerical solution of dynamic optimization problems. We then discuss the software and languages used in the framework for dynamic optimization in JModelica.org.

3.2.1 Tools for Dynamic Optimization

One approach to solving the DOP (3.2) numerically is to manually discretize the dynamics and then encode the discretized problem in an algebraic modeling language for optimization. Mature examples of such languages are AMPL [Fourer et al., 2003] and GAMS [Brooke et al., 1988], whereas Pyomo [Hart et al., 2012] is a recent example. Another approach is to use a tool tailored for dynamic optimization in which DOPs can be formulated in their natural, undiscretized form, which is more convenient. The tool then handles the details of the discretization. An important dichotomy of such tools is whether they use existing general-purpose programming languages (such as Python, C++, or MATLAB) or dedicated modeling languages to describe the system dynamics. Some examples of modern dynamic optimization tools of the former category are ACADO Toolkit [Houska et al., 2011], PSOPT [Becerra, 2010], and PROPT [Tomlab Optimization, 2016]. ACADO Toolkit is an open-source, self-contained C++ package for dynamic

optimization. It primarily uses direct multiple shooting. It is designed for implementation of online MPC or MHE on embedded hardware. PSOPT is another open-source C++ package for dynamic optimization supporting many different flavors of simultaneous discretization with elaborate support for multiple phases. PROPT is a commercial package for MATLAB based on TOMLAB [Holmström, 1999] using direct global collocation. It supports a wide range of dynamic optimization problems, including problems with multiple phases and integer variables.

Component-based modeling of large-scale, complex dynamical systems benefits greatly from the expressiveness offered by dedicated dynamical modeling languages. It also decouples the modeling process from the computational aspects, allowing the same model implementation to be used for multiple purposes, such as simulation and optimal control. Examples of modern dynamic optimization tools that utilize a dedicated language for modeling are APMonitor [Hedengren et al., 2014], gPROMS [Process Systems Enterprise, 2016], and JModelica.org [Åkesson et al., 2010a]. APMonitor is freely available and uses its own modeling language and direct local collocation. It has a tight integration between simulation, estimation, and control of dynamical systems, both dynamically and in steady state. It offers interfaces to Python, MATLAB, Julia, and web browsers. gPROMS is a large family of commercial products for model-based chemical engineering, which are based on gPROMS's powerful object-oriented modeling language. Although not its primary focus, it has capabilities for dynamic optimization using shooting algorithms.

3.2.2 JModelica.org

JModelica.org [Åkesson et al., 2010a] is an open-source platform for dynamic optimization and simulation of Modelica models. It originally started as an academic effort [Åkesson, 2007] but is now mainly developed in industry by Modelon AB, although still in collaboration with academia (as exemplified by this thesis).

The use of Modelica in this thesis and JModelica.org serves two purposes: First, it gives the user access to a powerful modeling language, which is important for large-scale, component-based system modeling. Second, it allows existing Modelica models to be reused for dynamic optimization. The second feature is, however, a double-edged sword. Typical Modelica models are intended for high-fidelity simulation. Consequently, they are often too complex for optimization purposes out-of-the-box, to the dismay of many JModelica.org users. As pointed out by this chapter's introductory quote, it can be necessary to rethink the modeling before it can be applied for optimization. For example, unnecessary discontinuities should be avoided, with the remaining ones being dealt with appropriately, and variables—

especially differential ones—should only be introduced if the change in dynamics have significant impact on the objective.

JModelica.org originally had a unified C-based interface for dynamic simulation and optimization based on DAEs called JModelica.org Modeling Interface (JMI) [Åkesson et al., 2010a]. It later became apparent that efficient simulation and optimization had quite separate needs, resulting in two separate toolchains (although both utilize the JModelica.org compiler, which will be described in Section 3.2.3): one for optimization and one for simulation. An overview of these are given below. Further documentation is available in [Modelon AB, 2016].

The previous optimization framework that was a part of JMI had several drawbacks compared with the implementation of this thesis. It relied heavily on code generation, not supporting the modification of parameter values after compilation, much less the modification of the equations, constraints, and objective of a DOP. A different tool was used for algorithmic differentiation [Griewank and Walther, 2008], which led to considerably slower NLP function evaluation and consequently longer solution times. Finally, it only supported computation of first-order derivatives. Thus, it relied on the use of limited-memory BFGS [Nocedal and Wright, 2006], often with unsatisfactory performance.

Optimization The new framework for dynamic optimization is based on CasADi [Andersson, 2013]. The framework's first prototype is described in [Andersson et al., 2011]. A core element of this framework is a direct collocation algorithm, which was later refined and extended in [Magnusson, 2012; Magnusson and Åkesson, 2012].

The original coupling between JModelica.org and CasADi was based on an XML representation of (3.5) [Parrotto et al., 2010]. While large parts of Modelica can be mapped to this XML format, some crucial parts are missing, most notably Modelica functions. Modelica functions are an important and difficult part of the language, because of their use of imperative constructs, as opposed to the declarative nature of both CasADi and Modelica. These XML limitations limited the class of Modelica models supported by the framework. While the XML format could be extended as needed to represent additional Modelica constructs, such an extension would require significant development efforts not only from the JModelica.org developers, but also the CasADi developers.

After some time, the need for supporting Modelica functions in the framework became too pressing. By then, the CasADi developers had other priorities than extended Modelica support. A new effort was then made to internalize the model representation within JModelica.org, so that CasADi no longer had to be relied upon for XML import. This new framework is called CasADi Interface and is described later in this section and also

in [Lennernäs, 2013]. The resulting toolchain is described in this chapter (and also in [Magnusson and Åkesson, 2015]), with emphasis on the further extended direct collocation algorithm. Although CasADi is no longer relied upon for XML import, it still plays a central role in the framework. The model representation is built up by CasADi constructs and CasADi is also subsequently used in the implementation of the direct collocation algorithm.

Simulation JModelica.org also has a strong framework for dynamic simulation. While dynamic simulation is an end goal in itself, it also serves important needs in a framework for dynamic optimization; for example, it can be used to generate initial guesses (that do not have to be feasible) and verify fixed-step discretization as will be discussed in Section 3.6. While methods exist for incorporating adaptive step lengths into direct collocation, either by repeatedly solving the problem and updating the discretization [Betts and Huffman, 1998], which is called mesh refinement, or introducing the element lengths as NLP variables and bounds or penalties on the discretization error estimate [Vasantharajan and Biegler, 1990], these are not considered in this thesis. However, the author has done some work related to this topic [Fouquet et al., 2016].

The simulation framework is based on the Functional Mockup Interface (FMI) [Blochwitz et al., 2011; Blochwitz et al., 2012] for tool-independent exchange of models. FMI started as an effort within the Modelica community, but has quickly gained traction outside of it in industry and is now supported to some extent in many different simulation tools. JModelica.org is a strong supporter both in terms of import and export of Functional Mockup Units (FMU), which are archive files comprising an XML description of the system variables and code (in binary or source form) for the system equations.

Unlike Modelica, the mathematical paradigm of FMI is that of explicit ODEs. The burden of DAE analysis—involving for example the index reduction and causalization techniques of Sections 2.2 and 4.3.1, respectively—is consequently borne by the tool that exports the FMU rather than the one that imports it. JModelica.org's FMU export procedure is similar to that of other Modelica tools: A compiler transforms the implicit DAE to an explicit ODE and then generates C code for efficient evaluation of the ODE right-hand side. JModelica.org's import of FMUs is based on PyFMI [Andersson et al., 2016], which connects an FMU to an ODE solver available from a large suite of solvers through Assimulo [Andersson et al., 2015], most notably CVODE from SUNDIALS [Hindmarsh et al., 2005].

3.2.3 Software Used to Implement Framework

JModelica.org integrates many different software packages. This section presents the most important ones, especially those that are prominent in

the framework for dynamic optimization.

CasADi Newton-based iterative solution of the primal-dual equations (2.24) requires first- and second-order derivatives of the NLP cost and constraint functions with respect to the NLP variables. The framework uses CasADi to obtain these. CasADi [Andersson, 2013] (**C**omputer **a**lgebra **s**ystem with **A**lgorithmic **Di**fferentiation) is a low-level tool for efficiently computing derivatives using algorithmic differentiation (AD) based on source-code transformation and is tailored for dynamic optimization. Once a symbolic representation of an NLP has been created using CasADi primitives, the needed derivatives are efficiently and conveniently obtained and sparsity patterns are computed and preserved. CasADi also offers interfaces to numerical optimization solvers, allowing for seamless integration with, for example, IPOPT [Wächter and Biegler, 2006] and WORHP [Büskens and Wassel, 2013].

CasADi utilizes two different graph representations for symbolic expressions. The first is SX, a scalar representation, where all atomic operations are scalar-valued, as is typical for AD tools. The second is MX, a sparse matrix representation where all atomic operations instead are multiple-input, multiple-output, and matrix-valued. The MX representation is more general and allows for efficient—especially in terms of memory and possibly construction of expression graphs for derivatives—representation of high-level operations, such as matrix multiplication and function calls. For example, an SX graph for multiplying a dense matrix of size $n \times n$ with a vector of size n uses $\mathcal{O}(n^2)$ nodes, whereas an MX graph would only use $\mathcal{O}(1)$ nodes. On the other hand, the SX representation offers faster evaluation times by reducing overhead and performing additional symbolic simplifications.

The JModelica.org Compiler The JModelica.org compiler [Åkesson et al., 2010b] is implemented in the compiler construction framework JastAdd [Ekman and Hedin, 2007]. JastAdd is an extension to Java and focuses on modular extensible compiler construction by aspect orientation. The compiler process is illustrated in Figure 3.1. The compiler first creates an internal representation of the Modelica and Optimica code in the form of an abstract syntax tree (AST). The AST is then used to perform standard compiler operations such as name and type analysis, but also Modelica-specific operations as described below.

To get a representation of a hierarchical Modelica model that is closer to a mathematical DAE system, one of the first steps, called flattening, is to resolve the class inheritance and instantiation in the model to arrive at a flat representation of the model. The flat representation essentially consists of only variable declarations, equations, and functions. Before the DAE system is interfaced with a numerical solver, various symbolic transformations are performed on it, of which the following are of importance to us:

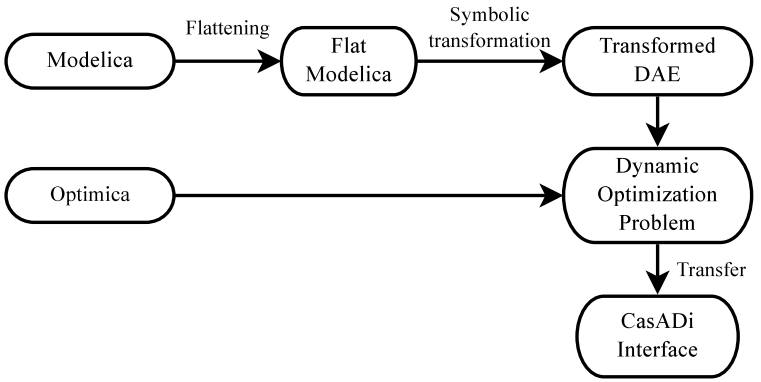

Figure 3.1 The compilation process in JModelica.org for DOPs. The process starts with the user-provided Modelica and Optimica code and ends with a symbolic representation of the DOP in CasADi Interface, which serves as an interface between dynamic optimization algorithms and the Modelica and Optimica code.

1. Alias elimination
2. Variability propagation
3. Index reduction
4. Causalization
5. Tearing

Alias elimination and variability propagation are conceptually simple and serve to eliminate algebraic variables described by trivial equations. Alias elimination identifies variables occurring in equations of the form $x \pm y = 0$, which are ubiquitous in object-oriented modeling, and eliminates one of them. Variability propagation identifies algebraic equations which are independent of time, such as $y = p + 1$, where p is a parameter, allowing the corresponding algebraic variables to be eliminated by solving the static equations. Index reduction is done using the method of dummy derivatives as outlined in Section 2.2. Causalization and tearing are by default not performed in the dynamic optimization framework. We return to this topic in Chapter 4.

Once all the symbolic transformations have been performed on the DAE system, it is coupled with the DOP formulation in the Optimica code. The AST is then used to transfer the resulting DOP (3.5) to CasADi Interface by defining symbolic CasADi variables and using them to build up the expressions represented by the AST.

CasADi Interface [Lennernäs, 2013] is a C++ package that enables the symbolic creation of DOPs using CasADi constructs. It serves as an interface between DOPs formulated using Modelica and Optimica and the optimization algorithms that can be used to solve them, in particular the one described in this chapter and the one in [Lazutkin et al., 2015]. When using CasADi Interface, the JModelica.org compiler creates CasADi expressions for the DOP, effectively mapping the Modelica and Optimica languages onto CasADi constructs, which then can be used to obtain the derivative information that is typically needed by numerical optimizers. While CasADi Interface is designed with Modelica and Optimica in mind, there is nothing in it that is inherently dependent on these languages. Hence, it could potentially serve as an interface to other modeling languages as well.

Nonlinear Programming and Linear Solvers To numerically solve the NLP arising from applying direct local collocation to (3.5), JModelica.org uses third-party solvers. The use of IPOPT [Wächter and Biegler, 2006] and WORHP [Büskens and Wassel, 2013] is supported through CasADi's NLP solver interface. IPOPT is a primal-dual interior-point method and WORHP is an active-set sequential quadratic programming method. Both solvers are designed for large-scale and sparse nonlinear programs. IPOPT is open source, whereas WORHP is commercial but offers free academic licenses. Both solvers use Newton's method to solve KKT conditions similar to (2.24), and hence need to solve a linear equation system in each iteration. They utilize external sparse linear solvers for this purpose. Both IPOPT and WORHP have interfaces to the open-source linear solver MUMPS, and also the commercial HSL library [HSL, 2016], which has free academic licenses, among others. Another important package related to linear algebra that they make use of is MeTiS [Karypis and Kumar, 1998] to generate fill-reducing orderings.

3.3 Direct Local Collocation

This section presents the details of the direct local collocation algorithm that is implemented in JModelica.org and used to solve (3.5). Although this section is largely a review of the literature on such methods [Biegler, 2010; Betts, 2010], we emphasize the particulars of the JModelica.org implementation and also the treatment of the uncommon constructs of (3.5), in particular timed variables and implicit initial conditions.

The fundamental idea is to discretize the differential equations using finite differences, thus transforming the infinite-dimensional DOP into a finite-dimensional NLP. The discretization scheme is based on collocation methods, which are special cases of implicit Runge-Kutta methods and

are also commonly used for numerical solution of DAE and stiff ODE systems [Hairer and Wanner, 1996].

3.3.1 Collocation Polynomials

The optimization time horizon is divided into n^e elements. Let h_i denote the length of element i, which has been normalized so that the sum of all element lengths is one. This normalization facilitates the solution of problems with free endpoints by keeping the normalized element lengths constant and instead varying t_0 and t_f. The time is normalized in element i according to

$$\tilde{t}_i(\tau) := t_{i-1} + h_i \cdot (t_f - t_0)\tau, \quad \forall \tau \in [0, 1], \quad \forall i \in [1..n^e], \tag{3.6}$$

where τ is the normalized time, $\tilde{t}_i(\tau)$ is the corresponding unnormalized time, and t_i is the mesh point (right boundary) of element i. This normalization enables a treatment of the below interpolation conditions that is homogeneous across elements.

Within element i, the time-dependent system variable $\boldsymbol{z}$ is approximated using a polynomial in the local time τ denoted by

$$\boldsymbol{z}_i = (\dot{\boldsymbol{x}}_i, \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{u}_i) : [0, 1] \to \mathbb{R}^{n_z}, \tag{3.7}$$

which is called the collocation polynomial for that element. The collocation polynomials are formed by choosing n^c collocation points, which in this work are restricted to be the same for all elements. We use Lagrange interpolation polynomials to represent the collocation polynomials, using the collocation points as interpolation points. Let $\tau_k \in [0, 1]$ denote collocation point $k \in [1..n^c]$, and let

$$\boldsymbol{z}_{i,k} = (\dot{\boldsymbol{x}}_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{y}_{i,k}, \boldsymbol{u}_{i,k}) \in \mathbb{R}^{n_z} \tag{3.8}$$

denote the value of $\boldsymbol{z}_i(\tau_k)$.

Since the state variable $\boldsymbol{x}$ needs to be continuous on $[t_0, t_f]$, we introduce an additional interpolation point at the start of each element for the corresponding collocation polynomials, denoted by $\tau_0 := 0$. Hence, we get the collocation polynomials

$$\boldsymbol{x}_i(\tau) = \sum_{k=0}^{n^c} \boldsymbol{x}_{i,k} \cdot \tilde{\ell}_k(\tau), \tag{3.9a}$$

$$\boldsymbol{y}_i(\tau) = \sum_{k=1}^{n^c} \boldsymbol{y}_{i,k} \cdot \ell_k(\tau), \tag{3.9b}$$

$$\boldsymbol{u}_i(\tau) = \sum_{k=1}^{n^c} \boldsymbol{u}_{i,k} \cdot \ell_k(\tau), \tag{3.9c}$$

for all $i \in [1..n^e]$, where $\tilde{\ell}_k$ and ℓ_k are the Lagrange basis polynomials, respectively with and without the additional interpolation point τ_0. The basis polynomials are defined as

$$\tilde{\ell}_k(\tau) := \prod_{l \in [0..n^c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [0..n^c], \tag{3.10a}$$

$$\ell_k(\tau) := \prod_{l \in [1..n^c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [1..n^c]. \tag{3.10b}$$

Note that the basis polynomials are the same for all elements, due to the normalized time. These basis polynomials satisfy

$$\ell_k(\tau_j) = \begin{cases} 1, & \text{if } j = k, \\ 0, & \text{if } j \neq k. \end{cases} \tag{3.11}$$

The collocation polynomials are thus parametrized by the values $\boldsymbol{z}_{i,k} = \boldsymbol{z}_i(\tau_k)$.

To obtain the collocation polynomial for the state derivative $\dot{\boldsymbol{x}}$ in element i, the collocation polynomial $\boldsymbol{x}_i$ is differentiated with respect to time. Using (3.6), (3.9a), and the chain rule, we obtain

$$\dot{\boldsymbol{x}}_i(\tau) = \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\tilde{t}_i}(\tau) = \frac{\mathrm{d}\tau}{\mathrm{d}\tilde{t}_i}\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\tau}(\tau) = \frac{1}{h_i \cdot (t_f - t_0)} \sum_{k=0}^{n^c} \boldsymbol{x}_{i,k} \cdot \frac{\mathrm{d}\tilde{\ell}_k}{\mathrm{d}\tau}(\tau). \tag{3.12}$$

There are different schemes for choosing the collocation points τ_k, with different numerical properties—in particular regarding stability and order of convergence. The most common ones are called Gauss, Radau, and Lobatto collocation [Hairer and Wanner, 1996; Biegler, 2010]. The framework in JModelica.org has support for Radau and Gauss points. For brevity, we will in the next subsection present a transcription based on Radau collocation and then briefly comment on how it relates to Gauss and Lobatto collocation.

3.3.2 Transcription of the Dynamic Optimization Problem

In this section, (3.5) is transcribed into an NLP, using the collocation polynomials constructed above. The optimization domain of functions on $[t_0, t_f]$, which is infinite-dimensional, is thus reduced to a domain of finite dimension by approximating the trajectory $\boldsymbol{z}$ by a piecewise polynomial function.

Optimization Variables As optimization variables in the NLP we choose the system variable values at the collocation points, $\boldsymbol{z}_{i,k}$, the state variable values at the start of each element, $\boldsymbol{x}_{i,0}$, the free parameters, $\boldsymbol{p}$, the initial

condition values, $\boldsymbol{z}_{1,0} := \boldsymbol{z}(t_0)$, and t_0 and t_f if they are free. In other words, we let

$$\begin{aligned}\boldsymbol{Z} := (&\boldsymbol{z}_{1,0}, \boldsymbol{z}_{1,1}, \boldsymbol{z}_{1,2}, \ldots, \boldsymbol{z}_{1,n^c},\\ &\boldsymbol{x}_{2,0}, \boldsymbol{z}_{2,1}, \boldsymbol{z}_{2,2}, \ldots, \boldsymbol{z}_{2,n^c},\\ &\boldsymbol{x}_{3,0}, \boldsymbol{z}_{3,1}, \boldsymbol{z}_{3,2}, \ldots, \boldsymbol{z}_{3,n^c},\\ &\vdots,\\ &\boldsymbol{x}_{n^e,0}, \boldsymbol{z}_{n^e,1}, \boldsymbol{z}_{n^e,2}, \ldots, \boldsymbol{z}_{n^e,n^c},\\ &\boldsymbol{p}, t_0, t_f) \in \mathbb{R}^{n_Z}.\end{aligned} \tag{3.13}$$

be the vector containing all the NLP variables, where

$$n_Z = (1 + n^e n^c) n_z + (n^e - 1) n_x + n_p + 2. \tag{3.14}$$

Note that the actual order of the variables in the implemented framework is different to allow contiguous access for efficiency reasons [Rodriguez, 2014, Section 5.2.2].

NLP With Radau collocation and the above choice of optimization variables, the NLP—whose concepts and notation are discussed below—that results from the transcription of (3.5) is

$$\text{min.} \quad \phi(t_0, t_f, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}) + \sum_{i=1}^{n^e} h_i \cdot (t_f - t_0) \sum_{k=1}^{n^c} \omega_k L\left(t_{i,k}, \boldsymbol{z}_{i,k}, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}\right), \tag{3.15a}$$

$$\text{w.r.t.} \quad \boldsymbol{Z} \in \mathbb{R}^{n_Z},$$

$$\text{s.t.} \quad \boldsymbol{F}(t_{i,k}, \boldsymbol{z}_{i,k}, \boldsymbol{p}) = \boldsymbol{0}, \qquad \boldsymbol{F}_0(t_0, \boldsymbol{z}_{1,0}, \boldsymbol{p}) = \boldsymbol{0}, \tag{3.15b}$$

$$\boldsymbol{u}_{1,0} - \sum_{k=1}^{n^c} \boldsymbol{u}_{1,k} \cdot \ell_k(0) = \boldsymbol{0}, \tag{3.15c}$$

$$\boldsymbol{g}_e(t_0, t_f, t_{i,k}, \boldsymbol{z}_{i,k}, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}) = \boldsymbol{0}, \quad \boldsymbol{g}_i(t_0, t_f, t_{i,k}, \boldsymbol{z}_{i,k}, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}) \le \boldsymbol{0}, \tag{3.15d}$$

$$\boldsymbol{G}_e(t_0, t_f, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}) = \boldsymbol{0}, \quad \boldsymbol{G}_i(t_0, t_f, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}) \le \boldsymbol{0}, \tag{3.15e}$$

$$\boldsymbol{z}_L \le \boldsymbol{z}_{i,k} \le \boldsymbol{z}_U, \qquad \boldsymbol{p}_L \le \boldsymbol{p} \le \boldsymbol{p}_U, \tag{3.15f}$$

$$t_{0,L} \le t_0 \le t_{0,U}, \qquad t_{f,L} \le t_f \le t_{f,U}, \tag{3.15g}$$

$$\forall (i,k) \in ([1..n^e] \times [1..n^c]) \cup \{(1,0)\},$$

$$\dot{\boldsymbol{x}}_{i,k} = \frac{1}{h_i \cdot (t_f - t_0)} \sum_{m=0}^{n^c} \boldsymbol{x}_{i,m} \cdot \frac{\mathrm{d}\tilde{\ell}_m}{\mathrm{d}\tau}(\tau_k), \quad \forall (i,k) \in [1..n^e] \times [1..n^c], \tag{3.15h}$$

$$\boldsymbol{x}_{n,n^c} = \boldsymbol{x}_{n+1,0}, \quad \forall n \in [1..n^e - 1], \tag{3.15i}$$

where $t_{i,k} := \tilde{t}_i(\tau_k)$ denotes the unnormalized collocation point k in element i.

Timed Variables There are two approaches in the treatment of the timed variables $\boldsymbol{z}_T$ during the transcription. The first is to approximate $\boldsymbol{z}(T_j)$ by the value of its corresponding collocation polynomial, that is, $\boldsymbol{z}_i(\tau(T_j))$. A less general approach is to assume that every time point T_j coincides with some collocation point $t_{i,k}$; that is, that there exists a map

$$\boldsymbol{\Gamma} : [1..n_T] \to [1..n^e] \times [1..n^c] \tag{3.16}$$

such that $T_j = t_{\Gamma(j)}$. We can then proceed to transcribe $\boldsymbol{z}_T$, defined analogously to (3.4), into

$$\tilde{\boldsymbol{z}}_T := \begin{bmatrix} \boldsymbol{z}_{\boldsymbol{\Gamma}(1)} & \boldsymbol{z}_{\boldsymbol{\Gamma}(2)} & \dots & \boldsymbol{z}_{\boldsymbol{\Gamma}(n_T)} \end{bmatrix}. \tag{3.17}$$

The former approach is more general, as it does not assume the existence of $\boldsymbol{\Gamma}$. It is also more user-friendly, since it does not force the user to align the element mesh with the time points T_j. On the other hand, the latter approach is more efficient for large n_T, which is typical for parameter estimation problems. Henceforth we adopt the latter approach, which assumes the existence of $\boldsymbol{\Gamma}$.

Objective Given the assumed existence of $\boldsymbol{\Gamma}$, the Mayer term of the objective (3.15a) is straightforward to transcribe as

$$\phi(t_0, t_f, \boldsymbol{z}_T, \boldsymbol{p}) \doteq \phi(t_0, t_f, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}), \tag{3.18}$$

where $a \doteq b$ denotes that b, which belongs to (3.15), is the corresponding transcription of a, which belongs to (3.5). The transcription of the Lagrange term is more involved and utilizes Gauss-Radau quadrature within each element:

$$\begin{aligned} \int_{t_0}^{t_f} L(t, \boldsymbol{z}(t), \boldsymbol{z}_T, \boldsymbol{p})\,\mathrm{d}t &= \sum_{i=1}^{n^e} \int_{t_{i-1}}^{t_i} L(t, \boldsymbol{z}(t), \boldsymbol{z}_T, \boldsymbol{p})\,\mathrm{d}t \\ &\approx \sum_{i=1}^{n^e} h_i \cdot (t_f - t_0) \sum_{k=1}^{n^c} \omega_k L\left(t_{i,k}, \boldsymbol{z}(t_{i,k}), \boldsymbol{z}_T, \boldsymbol{p}\right) \\ &\doteq \sum_{i=1}^{n^e} h_i \cdot (t_f - t_0) \sum_{k=1}^{n^c} \omega_k L\left(t_{i,k}, \boldsymbol{z}_{i,k}, \tilde{\boldsymbol{z}}_T, \boldsymbol{p}\right), \end{aligned} \tag{3.19}$$

where the quadrature weights ω_k are given by

$$\omega_k := \int_0^1 \ell_k(\tau)\,\mathrm{d}\tau. \tag{3.20}$$

The objective (3.5a) is thus transcribed into (3.15a).

DAE and Initial Conditions The essence of direct collocation is in the transcription of the system dynamics (3.5b). Instead of enforcing the DAE system for all times $t \in [t_0, t_f]$, it is only enforced at the collocation points. Thus

$$\begin{aligned} & \boldsymbol{F}(t, \boldsymbol{z}(t), \boldsymbol{p}) = \boldsymbol{0}, \quad \forall t \in [t_0, t_f] \\ \doteq\ & \boldsymbol{F}(t_{i,k}, \boldsymbol{z}_{i,k}, \boldsymbol{p}) = \boldsymbol{0}, \quad \forall i \in [1..n^e], \quad \forall k \in [1..n^c]. \end{aligned} \tag{3.21}$$

Due to the introduction of the NLP variable $\boldsymbol{z}_{1,0}$, the initial conditions (3.5b) are seemingly straightforward to transcribe into $\boldsymbol{F}_0(t_0, \boldsymbol{z}_{1,0}, \boldsymbol{p}) = \boldsymbol{0}$. However, this approach sometimes fails. Consider the case when $\boldsymbol{F}_0$ only depends on t_0, $\boldsymbol{x}(t_0)$, and $\boldsymbol{p}$, making it similar to the explicit initial conditions (2.6). The value of $\boldsymbol{u}_{1,0}$ will then have no effect on neither $\boldsymbol{x}(t_0)$ nor $\boldsymbol{u}_1$, and consequently has no effect on the trajectories or objective. Thus, the resulting NLP is ill-posed and has no unique solution; the KKT conditions are singular. The approach taken in JModelica.org follows [Magnusson and Åkesson, 2015]. Since $\boldsymbol{u}_{1,0}$ is not used to parametrize the collocation polynomial $\boldsymbol{u}_1$, its value is already determined by the collocation point values $\boldsymbol{u}_{1,k}$. The transcription of the initial conditions consequently also gives rise to the extrapolation constraint (3.15c). While this way of determining $\boldsymbol{u}_{1,0}$ allows the NLP to have a unique solution, the value is still largely inconsequential, although still useful for visualization purposes.

However, the above assumes that the initial conditions are akin to (2.6). We will in Section 3.3.3 discuss potential failure of the above approach in the general case.

Bounds as Well as Path and Point Constraints Given the assumed existence of $\boldsymbol{\Gamma}$, the point constraints (3.5d) can be transcribed into (3.15e). In the same approximative manner that we only enforced the DAE system at the collocation points, the path constraints (3.5c) and the bounds (3.5e) as well as (3.5f) are straightforwardly transcribed into (3.15d), (3.15f), and (3.15g), respectively.

State Variable To preserve the inherent coupling of $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$, which is implicit in the dynamic setting, we enforce (3.12) at all the collocation points, giving us the additional constraints (3.15h). These are not enforced at the start time t_0, where the state derivative $\dot{\boldsymbol{x}}$ instead is determined by the DAE system and initial conditions. Finally, to get a continuous trajectory for the state variable $\boldsymbol{x}$, we add the constraints (3.15i).

NLP Solution By solving the resulting NLP (3.15), we may obtain an approximate local optimum to the DOP (3.5). An important question is under what conditions the solution of (3.15) converges to the solution of (3.5) as either n^e or n^c tend to infinity. This question is largely unanswered. [Kameswaran and Biegler, 2008] shows convergence of Radau collocation for a very simple, but nevertheless important, case of (3.5) where the objective

is just a standard Mayer term, the system dynamics are given by an explicit ODE and explicit state initial conditions, with no further constraints (and some further technical assumptions). This result is based on the classical optimality conditions for an ODE-constrained optimal control problem given by Pontryagin's maximum principle [Liberzon, 2012]. Extending these results to DAEs would require a corresponding theory for DAEs. Such a theory has only been recently developed [Kunkel and Mehrmann, 2008], with no accompanying convergence results for direct collocation. Consequently, for many problems of interest, we do not know whether the solution of (3.15) approximates the solution of (3.2). However, this usually seems to be the case in practice, and so we take a leap of faith when solving the problems of Chapter 5.

Solving (3.15) using Newton's method requires all expressions of (3.15) to be twice continuously differentiable with respect to $\boldsymbol{Z}$. As stated earlier, this means that ϕ, L, $\boldsymbol{F}$, $\boldsymbol{F}_0$, $\boldsymbol{g}_e$, $\boldsymbol{g}_i$, $\boldsymbol{G}_e$, and $\boldsymbol{G}_i$ all need to be twice continuously differentiable, with one important exception: They do not need to be even continuous with respect to the time t. That is, L, $\boldsymbol{F}$ do not need to be continuous with respect to their first arguments and $\boldsymbol{g}_e$ as well as $\boldsymbol{g}_i$ do not need to be continuous with respect to their third argument. This allows for the treatment of a simple, but important, class of hybrid systems, which only involves time events; that is, all switchings occur at predetermined points in time.

Gauss and Lobatto Collocation The presented transcription is specialized for Radau collocation, which always places a collocation point at the end of each element, and the rest are chosen in a manner that maximizes numerical accuracy [Biegler, 2010, Theorem 10.1]. Lobatto collocation instead places a collocation point at both the beginning and end of each element, and Gauss collocation has no collocation points at the element boundaries.

The adaptation of this section to Gauss collocation only requires a few additional points of consideration, caused by the lack of collocation points at the mesh points. While Gauss collocation theoretically has a higher order of convergence, it often performs worse than Radau in practice [Bausa and Tsatsaronis, 2001; Biegler, 2007]. One reason for this is the superior stability properties for Radau [Kameswaran and Biegler, 2008; Biegler, 2010], especially for high-index DAEs, which is important in the presence of high-index inequality constraints, even if the DAE itself is low-index.

On the other hand, Lobatto collocation gives rise to new complications already during implementation. An approach based on (3.9a) will lead to overdetermined equality constraints in the NLP, resulting from having both DAE and continuity constraints for the state variables at the start of each element. This can be partially resolved by instead constructing polynomials for the state derivative and integrating them to obtain the collocation poly-

nomials for the states. However, the Lobatto method still potentially suffers from divergence of the costates (dual variables of the DOP), yielding inaccurate primal solutions [Garg et al., 2010]. For these reasons, this thesis focuses on the use of Radau collocation, rather than Gauss or Lobatto.

3.3.3 Transcription of Initial Conditions

The extrapolation constraint (3.15c) was introduced to get a unique solution of $\boldsymbol{z}_{1,0}$ when transcribing the initial conditions in (3.5b). This approach was designed with initial conditions that are similar to the explicit initial conditions (2.6) in mind, and works well in such situations.

However, for general implicit initial conditions it can sometimes yield unnecessarily inaccurate solutions. Consider the problem

$$\text{minimize} \quad (x(t_0) - 1)^2 + \int_0^5 \left(x^2(t) + u^2(t)\right) \mathrm{d}t, \tag{3.22a}$$

$$\text{subject to} \quad \dot{x}(t) = x(t) + u(t), \tag{3.22b}$$

$$\dot{x}(t_0) = 0. \tag{3.22c}$$

The solution of (3.15) with $n^e = 20$ and $n^c = 1$ (yielding a piecewise constant input) is shown in Figure 3.2. The initial input $u(t_0)$ is used both to determine the optimal initial state $x(t_0)$ and to control the system during the time span of the first element, forcing the system to remain in the steady state in the first element, which is clearly suboptimal. While the solution still converges to the correct one as n^e approach infinity, the convergence order is reduced because of the incorrect treatment of the first element.

The problem is that (3.15c) enforces $u_0 = u_1$, even though we would like to have $u_0 \neq u_1$ to let u_0 determine the optimal initial state and u_1 to determine the optimal initial input. The solution is to simply remove the constraint (3.15c).

The question is then whether or not to introduce (3.15c). As discussed earlier, the omission of (3.15c) can cause ill-posed problems, whereas its inclusion can cause reduced order of convergence. The answer is that (3.15c) should be introduced for a (not necessarily unique) subset of $\boldsymbol{u}$ of maximal cardinality that is not needed to parametrize the solution manifold for the initial state $\boldsymbol{x}(t_0)$ of the index-reduced initial equations (2.7). Unfortunately, determining such a parametrization in the general case requires analytic solution of the initial equations, which in general is not possible.

We could devise a symbolic method that only relies on structural information to find a suitable parametrization $\boldsymbol{x}(t_0)$ and does not need to solve any equations. By performing a causality analysis using a block-triangular ordering similar to the ones we will use in Chapter 4, we can determine which initial states depend on which components of $\boldsymbol{u}(t_0)$, and then select a suitable subset of $\boldsymbol{u}(t_0)$ to be constrained by $\boldsymbol{u}_1$ via (3.15c). However, such

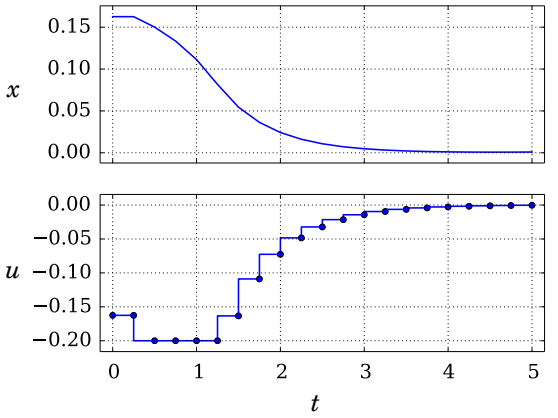

Figure 3.2 Solution of (3.22) obtained with deficient transcription of implicit initial conditions. The transcription leads to a suboptimal input in the first element.

a method is surprisingly complicated in the general case, and also will not be able to deal with nonstructural singularities that arise from the neglect of numerical information. Furthermore, this entire issue seems highly academic in nature, as the author has yet to encounter a real problem where the initial equations could not be transformed to only depend on t_0, $\boldsymbol{x}(t_0)$, and $\boldsymbol{p}$ (possibly involving offline computations prior to optimization to, for example, compute a desired steady state). Hence, we will not further consider the details of such a symbolic method.

The theoretical issue of order reduction resulting from the overzealous introduction of (3.15c) has a very pragmatic solution: Choose the length of the first element h_1 to be sufficiently small to make the error resulting from the incorrect treatment of the first element negligible. This is similar to how multistep methods for numerical integration [Brenan et al., 1996; Hairer and Wanner, 1996]—such as the popular Backward Difference Formula (BDF)—often solve the issue of obtaining a history of trajectory values needed to perform an integration step: Use a low-order method with tiny step size until sufficiently many values have been computed.

3.4 Implementation

The presented collocation framework is implemented in Python and distributed with JModelica.org under the GNU General Public License. All user interaction takes place in Python, and is centered around the Python

interface to CasADi Interface, which serves as a three-way interface between the user, the DOP, and the collocation framework. An overview is illustrated in Figure 3.3, which begins where the compilation toolchain ends, see Figure 3.1.

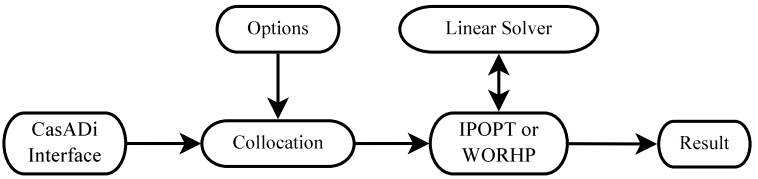

Figure 3.3 The framework surrounding the implemented collocation framework in JModelica.org. The framework starts with a representation of the DOP in CasADi Interface generated by the compiler. The DOP is then discretized by the collocation algorithm into an NLP, utilizing CasADi to perform the symbolic operations of the transcription. The NLP is finally solved by either IPOPT or WORHP.

The user can then call upon the collocation framework to solve the DOP, which will transcribe it into an NLP and then solve it using either IPOPT or WORHP, as decided by the user. The user provides options to the collocation framework using a dictionary-like class, specifying things such as discretization scheme, which NLP and linear solver to use, and additional features such as those presented in Section 3.5. The NLP and linear solver options are also provided directly to the collocation framework. The communication with these solvers is handled by the collocation framework through CasADi's NLP solver interface, so the user never has to interact with these solvers directly. The complete workflow in Python will be demonstrated in Section 3.6.

The result is stored in a textual format compliant with Dymola [Dassault Systèmes, 2016]. This result is then loaded into Python, allowing for convenient extraction of the trajectories. Because the hardest part of nonconvex optimization is usually to find a suitable initial guess, it is important that initial guesses can be conveniently provided from different sources. To this end, the same result format is used to provide initial guesses, making it convenient to provide initial guesses in the form of optimization or simulation results generated by JModelica.org or Dymola. There have been efforts within the Modelica community to standardize the result format [Pfeiffer et al., 2012], which certainly would improve the utility of the framework if they were to come to fruition.

The framework is designed to be symbolic in nature for maximal transparency and interactivity, with CasADi being the foundation of the symbolic engine. After the JModelica.org compiler has transferred the DOP to CasADi

Interface, a symbolic representation of the DOP is presented to the user. The DOP can be modified using Python and CasADi, to for example incorporate constructs that are more conveniently scripted rather than encoded in Modelica or Optimica. The NLP resulting from the direct collocation is also represented symbolically and is accessible by the user. This allows advanced users to diagnose the behavior of the NLP solver and also to trace the NLP variables and constraints back to the original DOP.

3.5 Additional Features

Additional considerations are needed to satisfactorily solve (3.5) in general. This section discusses the most important ones.

3.5.1 Initialization and Scaling

Solving large-scale nonconvex optimization problems requires accurate initial guesses of the solution to the problem for several reasons. The initial guess must lie within the method's region of convergence, meaning that it is sufficiently close to a local optimum, in order for the solver to succeed. Furthermore, for problems with multiple local minima, the initial guess must lie in a suitable region in order to converge to a desirable local minimum. Finally, automatic numerical scaling of the optimization problem is often done based on the initial guess. This approach achieves good scaling in the vicinity of the initial guess, but as the solver moves away from the initial guess, the scaling may deteriorate because of nonlinearities.

Initialization Many problems can not be solved without user-specified initial guesses. There are many ways of generating initial guesses for DOPs [Safdarnejad et al., 2015]. Since we are targeting Modelica models, we have some special needs to keep in mind. Because the user often does not construct their model from scratch, but instead relies on model libraries developed by others, they will not have complete understanding of all the model variables. Hence, they can not be expected to provide initial guesses or bounds for all of the variables, which is a common approach in tools for dynamic optimization. And even if they can, doing so is tedious because of the size of typical Modelica models.

It is however reasonable to expect the user to be familiar with the degrees of freedom of the problem, that is, $\boldsymbol{u}$, $\boldsymbol{p}$, t_0 and t_f. By forcing the user to provide initial guesses of these, the system can then be simulated to generate initial guesses for all of the variables. This approach has been found to be convenient and work well in practice. It also has the added benefit of generating initial guesses in the form of complete trajectories, rather than constant values.

Scaling The performance of numerical optimizers relies on the problem being reasonably well scaled numerically. Although Newton's method and quasi-Newton methods such as SR1 and BFGS are scale-invariant [Nocedal and Wright, 2006], this only holds when employing exact arithmetic. Hence, poor scaling can cause floating-point implementations to suffer from decreased convergence speeds or even divergence. There are many approaches to scaling problems, all with the goal of achieving close to unitary magnitude for relevant quantities, such as variables, functions, and condition numbers [Betts, 2010, Section 1.16]. There is no way to achieve perfect scaling, so the procedure is based on heuristics. For direct collocation, scaling techniques can be applied either directly to the NLP (3.15), or to the original DOP (3.5) (which then will propagate to the NLP during the transcription).

The scaling procedure can largely be automated, based on user-specified variable initial guesses (some tools also utilize variable bounds to perform scaling, but this would have limited use in Modelica). The automated scaling in the implemented framework focuses on variable scaling, that is, it exchanges the NLP variable Z_j for the scaled variable $\tilde{Z}_j$ according to $Z_j = d_j\tilde{Z}_j + e_j$. By an appropriate choice of d_j and e_j, the new variable $\tilde{Z}_j$ will have magnitude one. There are three strategies available in the framework for choosing d_j and e_j, with the possibility of applying different strategies for each individual DOP variable:

Time-Invariant Linear Scaling: The first approach is applied on the level of the dynamic problem (3.5), by setting $e_j = 0$ and d_j to the nominal value of the DOP variable corresponding to Z_j. This nominal value is defined either by the absolute value of the `nominal` attribute of the DOP variable, which typically is set by the user, or computed based on the initial guess. If the initial guess is given as a trajectory, the nominal value is chosen as the maximum absolute value of the trajectory over time. Otherwise, the nominal value is simply the absolute value of the constant initial guess.

Time-Invariant Affine Scaling: The second strategy is also applied on the level of (3.5) and requires an initial guesses in the form of a trajectory for the corresponding DOP variable. The idea is to choose d_j and e_j such that the scaled trajectory has a minimum value of 0 and a maximum value of 1. Let $Z_j^{\max}$ and $Z_j^{\min}$ denote the maximum and minimum value, respectively, for the initial-guess trajectory of the DOP variable corresponding to Z_j. The scaling factors are then chosen as

$$d_j = Z_j^{\max} - Z_j^{\min}, \quad e_j = Z_j^{\min}. \tag{3.23}$$

Time-Variant Linear Scaling: The third and final strategy is applied on the level of the NLP. It simply sets $e_j = 0$ and d_j to be the absolute value of the initial guess for Z_j. Thus, it is only different from the time-invariant

linear scaling when initial guesses for the DOP variables are provided in the form of trajectories rather than constant values.

Additional caution is needed in the choice of d_j for all of the strategies, since $d_j = 0$ does not work and values relatively close to zero are prone to make matters worse unless chosen with great care. The framework attempts to detect these cases and fall back to more conservative scaling strategies for the problematic variables.

The default scaling strategy is the time-invariant linear scaling, regardless of the form of the initial guess. It is preferred over the time-invariant affine scaling for its simplicity and over the time-variant linear scaling because the time-variant scaling requires more accurate initial guesses than typically are available to work better and is also more computationally expensive because each collocation point is treated individually.

As mentioned before, it is not only the variables that need to be scaled, but also other quantities, primarily the constraints. Unlike variable scaling, numerical optimizers (including IPOPT and WORHP) usually implement their own strategies for constraint scaling, most of which are also based on the user-provided initial guess. The current framework relies on external solvers to perform constraint scaling. This could however be improved upon, by exploiting the temporal structure of the problem. The constraint scaling of NLP solvers is akin to the time-variant linear scaling, in that it computes an individual scaling factor for each constraint. Just like the time-variant variable scaling, this puts high requirements on the initial guess to not generate bad scaling factors. IPOPT however is less aggressive in its computation of scaling factors, reducing them by a factor with a default value of 100. It may be preferable to instead do aggressive time-invariant constraint scaling.

3.5.2 Input Discretization

Sometimes further restrictions on the input $\boldsymbol{u}$ are desirable in the DOP (3.5), in particular constraining it to be piecewise constant. This is necessary to for example take into account that modern controllers usually are digital, which is especially important when using model predictive control (MPC), where the input signals are kept constant between each sample. This is supported in the framework by optionally enforcing $\boldsymbol{u}_i$ to be constant for all i and also possibly equal to $\boldsymbol{u}_{i+1}$; that is, only allowing changes in the input at a user-specified subset of the element boundaries defined by blocking factors, which may differ for different inputs. It is then also possible to add penalties or constraints on the difference in the input values between the element boundaries. This corresponds to penalizing or constraining the input derivative in the case that the input is not enforced to be piecewise constant.

3.5.3 Algorithmic Differentiation Graphs

Using the framework to solve large-scale problems is computationally expensive, both in terms of memory and computation time. The most memory is usually used during the computation of the Hessian of the NLP Lagrangian by CasADi's algorithmic differentiation (AD), which often requires memory in the order of gigabytes. The framework implements collocation based on either CasADi's SX or MX graphs (see Section 3.2.3), or a mixture of both, allowing the user to conveniently perform a tradeoff between memory use and execution time by choosing which graph types to use. The mixture employs SX graphs to build up the expressions for the constraints and objective in the DOP (3.5), which then are used to construct MX graphs for the NLP (3.15) by function calls in each collocation point.

3.5.4 Further Extensions

The collocation framework of this thesis has been extended with several other features, many of which are primarily due to the work of others. We briefly describe the notable ones here.

[Rodriguez, 2014] interfaced the framework with a prototypical parallelized NLP solver [Word et al., 2014] via C-code generation of the NLP functions and their derivatives. An important part of this is a new way of constructing the AD graphs similar to what is discussed in Section 3.5.3, but instead using MX graphs to construct a hierarchy of functions calls, which not only allows for generation of more efficient C code, but also reduces memory usage (at the cost of function evaluation times).

[Axelsson et al., 2015] made a high-level framework for MPC on top of the collocation framework, which was further extended in [Ekström, 2015] with real-time capabilities and code generation. In a similar fashion, high-level frameworks have also been made for grey-box identification [Palmkvist, 2014] and moving horizon estimation [Larsson, 2015].

[Magnusson et al., 2015] extended the framework with capabilities for solving problems involving parametric sensitivities, in particular for solving optimal experimental design problems. [Fouquet et al., 2016] extended the framework with capabilities for handling a special class of hybrid systems, in which the switchings depend explicitly on the system input, based on an extension of the sum up rounding method [Sager, 2005].

3.6 Example

To demonstrate how the framework is used, we will present the full code for solving (2.16). We will go through the full procedure of first simulating the system to generate an initial guess, then solve the DOP for different values

of r, and then simulate the system again using the optimal inputs with adaptive step sizes in the discretization to control the numerical accuracy.

A Modelica implementation of the VDP oscillator was given in Listing 2.1. Listing 2.3 extended the VDP model with the optimization formulation corresponding to (2.16) using Optimica. This Modelica and Optimica code can be used to solve the problem using the Python and JModelica.org code in Listing 3.1.

Listing 3.1 Python code for optimal control of the VDP oscillator using JModelica.org. The code first simulates the system to generate an initial guess, then finds the optimal input, and then simulates again using the optimal input and adaptive step sizes. The Modelica and Optimica code of Listings 2.1 and 2.3 are assumed to have been collected in a file `vdp.mop`.

```
# Import JModelica.org methods
from pymodelica import compile_fmu
from pyfmi import load_fmu
from pyjmi import transfer_optimization_problem

# Compile and simulate model
fmu = compile_fmu('VDP', 'vdp.mop')
model = load_fmu(fmu)
sim_res = model.simulate(final_time=10.)

# Compile DOP and transfer to CasADi Interface
dop = transfer_optimization_problem('VDP_DOP', 'vdp.mop')

# Set solver options
opts = dop.solve_options()
opts['n_e'] = 100 # Number of elements
opts['init_traj'] = sim_res # Initial guess trajectories
opts['IPOPT_options']['linear_solver'] = "ma27"

# Solve for different values of r
for r in [0.1, 1.0, 10.0]:
    dop.set('r', r)
    dop_res = dop.solve(options=opts)

    # Simulate with optimal inputs to verify discretization
    model.reset() # Reset state
    sim_res = model.simulate(
            final_time=10., input=dop_res.get_opt_input(),
            options={'CVode_options': {'rtol': 1e-6}})
```

We first import the compilation methods from JModelica.org and then compile and simulate the model to generate an initial guess. By not specifying the input values, it defaults to zero, which is sufficient for this simple problem. Next we compile the dynamic optimization problem and solve it

for $r \in \{0.1, 1, 10\}$ after setting some solver options. Finally, we simulate the model again, but this time with the different optimal inputs. By comparing the direct collocation trajectories with the simulated trajectories, it can be ascertained that a sufficiently fine discretization has been used in the direct collocation. The simulated trajectories were previously shown in Figure 2.2. The code for generating plots, which uses matplotlib [Hunter, 2007], is omitted.

This example demonstrates the flexibility and modularity of the framework. The modeling process is cleanly separated from the solution procedure and the same model is conveniently used to simulate the system to generate initial-guess trajectories, to solve the optimal control problem, and finally to verify the fixed-step collocation discretization by simulating the optimal input. It also hints at the interactivity offered by Python scripting, which allows us to easily solve and modify the problem formulation repeatedly in an interactive manner.

3.7 Conclusion

We have presented the framework for dynamic optimization in the open-source platform JModelica.org. The framework solves problems formulated using the modeling language Modelica and its extension Optimica. The framework implements a method utilizing direct collocation, of which the details have been discussed based on the Radau scheme. The implementation uses CasADi to construct the nonlinear program to efficiently obtain derivative information using algorithmic differentiation, and also to get convenient access to state-of-the-art nonlinear programming numerical solvers. The use of the framework has been demonstrated on a simple optimal control problem.

While no rigorous benchmark has been made to compare the efficiency of the framework with other tools for general-purpose dynamic optimization, the author believes the framework to be highly competitive in terms of online computational times (but probably not in terms of memory usage). This belief is corroborated by the following work.

- [Magnusson and Åkesson, 2012] found the framework of this chapter to be almost an order of magnitude faster than the old JMI-based framework in JModelica.org for optimal control of a large-scale power plant.

- [Lazutkin et al., 2014] found JModelica.org to be on par with other CasADi-based implementations of different flavors of multiple shooting for a small-scale satellite control problem.

- [Magnusson et al., 2015] found JModelica.org to be faster than gPROMS [Process Systems Enterprise, 2016]—state of the art of solving optimal experimental design problems—and a custom MATLAB framework for optimal experimental design of a small-scale fed-batch reactor.
- [Parini, 2015] found JModelica.org to be almost two orders of magnitude faster than OpenModelica [Ruge et al., 2014] for optimal control of a large-scale power plant (albeit with worse convergence properties, which we address in Chapter 4).

4

Symbolic Elimination Based on Block-Triangular Ordering

A problem solved, is a problem caused.

Karl Pilkington

The DAE (3.5b) is the main source of computational expense in the solution of (3.15). This holds especially true for Modelica models, where simple models can contain hundreds of variables, which turn into thousands of variables after simultaneous discretization. When dynamically simulating DAE systems, this is commonly dealt with by transforming the DAE to an ODE using techniques including index reduction, causalization, and tearing [Cellier and Kofman, 2006]. This transformation is (when possible) done in a symbolic preprocessing step, prior to numerical solution, which essentially hides the algebraic variables from the numerical solver, exposing only a minimal set of equations and variables: the state variables.

On the other hand, direct collocation is conventionally applied on the full DAE [Biegler, 2010; Betts, 2010]. In this chapter we consider the use of ODE transformation techniques on dynamic optimization problems. But rather than completing the transformation all the way to an explicit ODE, we will try to find the middle ground between the full implicit DAE and an equivalent ODE that maximizes computational efficiency and convergence robustness.

While these techniques can be applied in combination with any of the standard numerical methods for solving DAE-constrained optimization problems that were discussed in Section 2.3.4, the performance of some methods will be affected more than others. We will focus on the use of direct collocation and how it is affected by these transformation techniques.

Modelica models tend to be large, sparse, and have considerably more algebraic than differential variables, making them a prime target for the considered methods. We will however in Chapter 6 see that the presented methods hold merit also for more typical textbook DAEs.

4.1 Illustrative Example

To demonstrate the main ideas of the symbolic eliminations that we will employ, we will first go through a simple example. If some steps are unclear to the reader, they should be revisited after Section 4.3, where the general case is treated in detail. Consider the simple time-invariant DAE

$$\dot{x} + y_1 + y_2 - y_3 = 0, \tag{4.1a}$$
$$xy_3 + y_2 - \sqrt{x} - 2 = 0, \tag{4.1b}$$
$$2y_1y_2y_4 - \sqrt{x} = 0, \tag{4.1c}$$
$$y_1y_4 + \sqrt{y_3} - x - y_4 = 0, \tag{4.1d}$$
$$y_4 - \sqrt{y_5} = 0, \tag{4.1e}$$
$$y_5^2 - x = 0. \tag{4.1f}$$

A structural incidence matrix for (4.1) is (4.2).

	$\dot{x}$	y_1	y_2	y_3	y_4	y_5
(4.1a)	1	1	1	1	0	0
(4.1b)	0	0	1	1	0	0
(4.1c)	0	∗	∗	0	∗	0
(4.1d)	0	∗	0	∗	∗	0
(4.1e)	0	0	0	0	1	∗
(4.1f)	0	0	0	0	0	∗

(4.2)

where zeros indicate independence, ones indicate linear dependence, and asterisks indicate nonlinear dependence. Inspection of (4.1) reveals that we can eliminate y_4 using (4.1e), so we use

$$y_4 = \sqrt{y_5} \tag{4.3}$$

to substitute y_4 with $\sqrt{y_5}$, yielding

$$\dot{x} + y_1 + y_2 - y_3 = 0, \tag{4.4a}$$
$$xy_3 + y_2 - \sqrt{x} - 2 = 0, \tag{4.4b}$$
$$2y_1y_2\sqrt{y_5} - \sqrt{x} = 0, \tag{4.4c}$$
$$y_1\sqrt{y_5} + \sqrt{y_3} - x - \sqrt{y_5} = 0, \tag{4.4d}$$
$$y_5^2 - x = 0. \tag{4.4e}$$

Such eliminations can be identified by permuting (4.2) to a block-triangular matrix with blocks along the diagonal—henceforth referred to as diagonal blocks, despite them not being diagonal in general—of minimal size. Permuting (4.2) to such a form yields (4.5).

	y_5	y_4	y_1	y_2	y_3	$\dot{x}$
(4.1f)	$*$	0	0	0	0	0
(4.1e)	$*$	1	0	0	0	0
(4.1b)	0	0	0	1	1	0
(4.1c)	0	$*$	$*$	$*$	0	0
(4.1d)	0	$*$	1	0	$*$	0
(4.1a)	0	0	$*$	$*$	$*$	1

(4.5)

Note that some previously linear incidences have been reclassified as nonlinear and vice versa; we are no longer interested in the linearity of incidences outside of the diagonal blocks, and hence mark them all with asterisks. Within the diagonal blocks, we only care about linearity with respect to the variables in the block. For example, the incidence of the equation–variable pair $((4.1d), y_1)$ that was previously considered to be nonlinear is now considered to be linear. An elimination such as (4.3) can be found by identifying the diagonal blocks that are scalar and linear. The only algebraic variable that can be eliminated with this approach in this example is y_4.

We can however go further in our elimination procedure by eliminating y_1 and y_2 from (4.1d) and (4.1b), respectively. That is, we identify

$$y_1 = \frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}}, \tag{4.6a}$$

$$y_2 = 2 + \sqrt{x} - xy_3, \tag{4.6b}$$

and through substitution obtain the DAE

$$\dot{x} + \frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}} + 2 + \sqrt{x} - xy_3 - y_3 = 0, \tag{4.7a}$$

$$2\frac{x + \sqrt{y_5} - \sqrt{y_3}}{\sqrt{y_5}}(2 + \sqrt{x} - xy_3)\sqrt{y_5} - \sqrt{x} = 0, \tag{4.7b}$$

$$y_5^2 - x = 0. \tag{4.7c}$$

The eliminations (4.6) can be found by tearing the 3×3 diagonal block. Selecting y_3 as tearing variable and (4.1c) as tearing residual, we get the

torn incidence matrix (4.8).

	y_5	y_4	y_1	y_2	y_3	$\dot{x}$
(4.1f)	$*$	0	0	0	0	0
(4.1e)	$*$	1	0	0	0	0
(4.1d)	0	$*$	1	0	$*$	0
(4.1b)	0	0	0	1	1	0
(4.1c)	0	$*$	$*$	$*$	0	0
(4.1a)	0	0	$*$	$*$	$*$	1

(4.8)

The eliminations (4.6) are then identified as the diagonal incidences in the upper left subblock of the torn block, which are feasible since this subblock is triangular (in this case it is even diagonal) and each element along the diagonal is linear. Comparing (4.7) with (4.1), we have only 2 instead of 5 algebraic variables, but the residuals are more complicated.

4.2 Related Work

The standard approach of discretizing the full DAE relies on fill-reducing orderings, such as nested dissection [George, 1973], for efficient numerical linear algebra and numerical pivoting for stability [Duff et al., 1986]. While our proposed approach of using causalization techniques has been used in the context of dynamic optimization before [Franke, 2002; Bachmann et al., 2012; Pfeiffer, 2012; Franke et al., 2015], in this thesis we consider the effects they have on the solution procedure. We also do not complete the causalization all the way to an explicit ODE, but rather choose to keep some algebraic variables and implicit equations for efficiency.

[Safdarnejad et al., 2015] consider the use of block-triangular decompositions for the purpose of more efficiently generating initial guesses to (3.2) by solving square problems (with fixed degrees of freedom), and in particular identifying infeasibilities. They do however not consider block-triangular decompositions for the actual solution of (3.2). [Fletcher, 1998] considers the use of block-triangular ordering and tearing for efficient and sparsity-preserving orderings for implicit LU factorization tailored for linear programming.

The main purpose of causalization is the efficient treatment of algebraic equations. There are other approaches to achieving this. One technique that is common when using direct multiple shooting is the elimination of all algebraic variables in the shooting nodes by linearizing the consistency conditions (algebraic equations) in each iteration [Diehl et al., 2002], which is possible for semi-explicit index-one DAEs. Another loosely related concept is the use of reduced-space methods [Cervantes et al., 2000] which primarily work in the null space of the NLP equality constraints, which can be

beneficial when there are few degrees of freedom, as is typically the case in dynamic optimization problems. Since these techniques are applied to the NLP rather than the dynamic problem (3.5), they could be combined with the ideas presented in this chapter with potential benefits, but such possibilities are not considered further in this thesis. Reduced-space methods are also related to the concept of condensing quadratic programs arising in MPC with linear(ized) dynamic equations, allowing for elimination of the state variables. [Axehill, 2015] considers sparsity preservation in this context through partial condensation, which exploits temporal sparsity rather than the sparsity of the dynamics. Another possibility of exploiting this temporal sparsity when employing direct local collocation is the use of parallelization by exploiting the arrowhead structure of the KKT system that arises due to the temporal decoupling of the finite elements [Word et al., 2014]. This has great potential speedups when there are significantly more algebraic than differential variables, a sufficiently large amount of finite elements is used, and a large number of processor cores are available.

4.3 Causalization, Tearing, and Pivot Selection

In Section 3.2.3 we listed five important symbolic transformation steps performed by the JModelica.org compiler. In this section we review how the two final steps, causalization and tearing, are commonly employed when solving DAE simulation problems. We also describe well-established techniques for selecting pivot elements in direct, sparse linear solvers, which serve as inspiration for the sparsity-preservation techniques that we devise.

4.3.1 Causalization

Conceptually, an explicit ODE allows the computation of $\dot{\boldsymbol{x}}(t)$ given the current known variables $t, \boldsymbol{x}(t), \boldsymbol{u}(t)$, and $\boldsymbol{p}$. After having performed index reduction as discussed in Section 2.2.3, the remaining steps of the ODE transformation can thus be considered equivalent to solving the square system

$$\boldsymbol{F}(\boldsymbol{z}; t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) = \boldsymbol{0}, \tag{4.9}$$

where $\boldsymbol{z} := (\dot{\boldsymbol{x}}, \boldsymbol{y})$. Note that unlike Chapter 3, $\boldsymbol{z}$ no longer includes $\boldsymbol{x}$ and $\boldsymbol{u}$. Also note that we have now dropped the dependence on time for $\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{y}$, and $\boldsymbol{u}$, since we with this perspective only consider them as elements of $\mathbb{R}^n$ rather than functions.

While solving (4.9) can be done in a straightforward numerical manner by applying, for example, Newton's method, such an approach may be inefficient and may also be practically challenging because of the need of a having a sufficiently good initial guess of the solution for convergence. The idea of

causalization and this chapter is to apply symbolic transformations to the problem to allow for more efficient subsequent numerical solution. Note that it is dubious whether the application of numerical methods to (4.9) really can be considered to result in an explicit ODE, since a closed-form expression for $\boldsymbol{f}$ has not been found (in fact, one may not even exist). However, typical numerical ODE solvers will not be able to tell the difference as long as $\boldsymbol{f}$ can be evaluated.

Causalization can be divided into two steps: matching and block-lower triangular (BLT) ordering. The first step is to find a perfect matching between each scalar component of $\boldsymbol{F}$ and each scalar component of $\boldsymbol{z}$, meaning that each equation is matched to a single variable and vice versa. Such a matching exists if and only if the structural Jacobian (2.13) is structurally nonsingular, as guaranteed by JModelica.org's index reduction. Finding perfect matchings is a well-studied graph-theoretic problem with several available efficient algorithms. JModelica.org uses the Hopcroft-Karp [Hopcroft and Karp, 1973] algorithm, which has the best known worst case performance and often performs acceptably in practice [Setubal, 1993].

To construct a desired BLT ordering, we define the structural incidence matrix of the DAE, whose elements are defined by

$$\operatorname{struct} F_{i,j} := \begin{cases} 0 \text{ or } *, & \text{if } \nabla_{z_j} F_i \equiv 0, \\ 1 \text{ or } *, & \text{if } \nabla_{z_j} F_i \not\equiv 0 \text{ and } \nabla^2_{\boldsymbol{z},z_j} F_i \equiv \boldsymbol{0}, \\ *, & \text{otherwise,} \end{cases} \tag{4.10}$$

that is, 0 denotes incidences that do not depend on unknowns, 1 denotes incidences that depend affinely—henceforth called linearly—on unknowns, and $*$ denotes any kind of incidence. The reason for the incomplete definition of struct $F_{i,j}$ regarding the occurrence of $*$ was elucidated in Section 4.1, where we noted that certain elements of the incidence matrix are of no consequence to us. The main step of the causalization procedure is to permute the DAE structural incidence matrix to a BLT form. The result is that the equations and variables of the DAE have been sorted so that the DAE can be described by

$$\boldsymbol{F}^1(\boldsymbol{z}^1; \boldsymbol{v}^1) = \boldsymbol{0}, \tag{4.11a}$$

$$\boldsymbol{F}^2(\boldsymbol{z}^2; \boldsymbol{v}^2) = \boldsymbol{0}, \tag{4.11b}$$

$$\vdots$$

$$\boldsymbol{F}^m(\boldsymbol{z}^m; \boldsymbol{v}^m) = \boldsymbol{0}, \tag{4.11c}$$

where $\boldsymbol{F}^i$ corresponds to a diagonal block (colloquially known as an algebraic loop if it is not scalar valued), m is the number of such blocks, $\boldsymbol{z}^i$ is the

unknown variables of $\boldsymbol{F}^i$, and

$$\boldsymbol{v}^i := (\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^{i-1}, t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \tag{4.12}$$

is the unknown variables of preceding blocks and the known variables. In other words, we have found permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ such that

$$\mathbf{P}\,\mathrm{struct}(\boldsymbol{F})\mathbf{Q} = \begin{bmatrix} \mathbf{F}^{1,1} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{F}^{2,1} & \mathbf{F}^{2,2} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{F}^{3,1} & \mathbf{F}^{3,2} & \mathbf{F}^{3,3} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{F}^{m,1} & \mathbf{F}^{m,2} & \mathbf{F}^{m,3} & \cdots & \mathbf{F}^{m,m} \end{bmatrix}, \tag{4.13}$$

where $\mathbf{F}^{i,j} := \mathrm{struct}\,\boldsymbol{F}^i(\boldsymbol{z}^j; \boldsymbol{z}^1, \ldots, \boldsymbol{z}^{j-1}, \boldsymbol{z}^{j+1}, \ldots, \boldsymbol{z}^i, t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$.

This form allows the sequential treatment of each diagonal block, allowing us to solve multiple small systems rather than a single large. Furthermore, it enables specialized treatment of each diagonal block, allowing the exploitation of equation structure that may exist within a diagonal block but not in the full system $\boldsymbol{F}$, such as linearity.

Consequently, we are interested in a BLT decomposition that has the maximal number of diagonal blocks, or equivalently, diagonal blocks of minimal size. This form turns out to be unique in the sense that the number of diagonal blocks and their respective sizes and the variables and equations within a diagonal block are unique, but the ordering of diagonal blocks and orderings of variables and equations within the blocks are in general not unique. A trivial consequence of this form is that the diagonal blocks are irreducible.

Finding a BLT ordering that is optimal in this sense is equivalent to finding the strongly connected components of the directed graph defined by the equation–variable matching as follows. For each matched equation–variable pair, create a vertex. For each nonzero struct $F_{i,j}$, create an edge from the vertex corresponding to equation i to the vertex corresponding to variable j. This will create loops on all of the vertices, which may be removed.

Tarjan's algorithm [Tarjan, 1972] is widely regarded as the most efficient algorithm for finding the strongly connected components of a directed graph [Cellier and Kofman, 2006], with linear complexity in the number of vertices and edges. It also has the added benefit of not only identifying the strongly connected components, but also topologically sorting them so that all edges out of a component lead to preceding components, giving us the sought BLT ordering.

Block-triangular orderings are also useful in contexts other than DAE systems, such as solving sparse, unsymmetric linear equations [Duff and Reid, 1996].

4.3.2 Tearing

All that remains in the computation of $\dot{\boldsymbol{x}}$ is the solution of each $\boldsymbol{F}^i$. In general this will require iterative numerical methods. We will however not proceed down this route at this stage. We will instead only rely on symbolic techniques, without involving iterative methods, in our computation of $\dot{\boldsymbol{x}}$. Consequently, we will not go all the way to an explicit ODE. The details of this are discussed in Section 4.4. An important part of this procedure is the use of tearing [Kron, 1963; Duff et al., 1986; Elmqvist and Otter, 1994; Baharev et al., 2016a]. Tearing is a method for solving sparse systems of equations that lack significant structure, such as being banded or block diagonal. The idea is to order the variables and equations to get a partitioning of the system

$$\mathbf{0} = \boldsymbol{F}^i(\boldsymbol{z}^i; \boldsymbol{v}^i) = \begin{bmatrix} \bar{\boldsymbol{F}}^i(\bar{\boldsymbol{z}}^i; \hat{\boldsymbol{z}}^i, \boldsymbol{v}^i) \\ \hat{\boldsymbol{F}}^i(\hat{\boldsymbol{z}}^i; \bar{\boldsymbol{z}}^i, \boldsymbol{v}^i) \end{bmatrix} \tag{4.14}$$

such that the first partition $\bar{\boldsymbol{F}}^i$ is highly structured, allowing this part of the system to be solved efficiently on its own and then utilized in the solution of the full system by block elimination.

Tearing can be done in different ways, for different applications, with different goals in mind. We apply tearing to the diagonal blocks of the BLT form. We seek a partitioning such that one part of the diagonal block is triangular and linear along the diagonal. That is, we find permutations $\mathbf{P}^i$ and $\mathbf{Q}^i$ such that

$$\mathbf{P}^i \operatorname{struct}\left(\boldsymbol{F}^i\right) \mathbf{Q}^i = \begin{array}{c} \begin{matrix} \bar{\boldsymbol{z}}^i & & \hat{\boldsymbol{z}}^i \end{matrix} \\ \left[\begin{array}{ccc|c} 1 & & & \\ & \ddots & \mathbf{0} & * \\ * & & 1 & \\ \hline * & & & * \end{array}\right] \begin{matrix} \\ \\ \\ \bar{\boldsymbol{F}}^i \\ \\ \hat{\boldsymbol{F}}^i \end{matrix} \end{array}, \tag{4.15}$$

where $\bar{\boldsymbol{z}}^i$ is called the causalized variables, $\bar{\boldsymbol{F}}^i$ the causalized equations (because of the lower-triangular structure), $\hat{\boldsymbol{z}}^i$ the tearing variables, and $\hat{\boldsymbol{F}}^i$ the tearing residuals. This form is appealing because it allows for symbolic elimination of the causalized variables in terms of the tearing variables; that is,

$$\bar{\boldsymbol{z}}^i = \left(\bar{\boldsymbol{F}}^i\right)^{-1}(\mathbf{0}; \hat{\boldsymbol{z}}^i, \boldsymbol{v}^i) =: \bar{\boldsymbol{H}}^i(\hat{\boldsymbol{z}}^i, \boldsymbol{v}^i). \tag{4.16}$$

The fact that $\bar{\boldsymbol{F}}^i$ is triangular and linear along the diagonal allows for efficient and numerically (backward) stable [Trefethen and Bau, 1997] computation of $\left(\bar{\boldsymbol{F}}^i\right)^{-1}$ through forward substitution. The remaining equations

to be solved for $\hat{\boldsymbol{z}}^i$ numerically, typically using Newton's method, are given by

$$\hat{\boldsymbol{H}}^i(\hat{\boldsymbol{z}}^i) := \hat{\boldsymbol{F}}^i\left(\hat{\boldsymbol{z}}^i; \bar{\boldsymbol{H}}(\hat{\boldsymbol{z}}^i, \boldsymbol{v}^i), \boldsymbol{v}^i\right) = \boldsymbol{0}. \tag{4.17}$$

The benefits of this approach over solving the full diagonal block (4.14) numerically is that (4.17) has fewer variables and equations, equal to the number of tearing variables, often allowing it to be solved more efficiently. But a perhaps more important benefit is that a sufficiently good initial guess for convergence is only needed for $\hat{\boldsymbol{z}}^i$, whereas solving the full system would also require a sufficiently good guess for $\bar{\boldsymbol{z}}^i$. For these reasons, it is desirable to find a partitioning that minimizes the number of tearing variables and residuals. Unfortunately, this problem is NP-hard [Baharev et al., 2016c]. While in many cases it is tractable to solve the problem to optimality, it is common to instead apply heuristics to find near-optimal partitionings. The tearing algorithm used in JModelica.org, and in this thesis, is based on the heuristics described in [Meijer, 2011]. The core of these heuristics is to choose variables and equations that have a large number of incidences as tearing variables and residuals, making the remaining equations more sparse and therefore more likely to be causalizable.

The use of tearing is however a double-edged sword. While the symbolic solution of (4.16) and numerical solution of (4.17) are numerically stable, the tearing residuals (4.17) may be ill-conditioned even if the full system (4.14) is not. Furthermore, even if the tearing residuals and full system are well-conditioned, the numerical computation of $\bar{\boldsymbol{z}}^i$ through (4.16) after having computed the solution of (4.17) may be numerically unstable. Another potential drawback is that (4.17) is significantly more dense than (4.14), which may cause it to actually be more expensive to solve if sparsity is exploited in the computations. This topic is discussed further below. In conclusion, there is more to the choice of tearing variables and residuals than just minimizing their number.

4.3.3 Pivot Selection in Direct, Sparse Linear Solvers

When solving a nonlinear system of equations with Newton's method, the symbolic elimination of variables is computationally equivalent to a priori selection of pivot elements in the linear equation solver in each Newton iteration. Since pivots are selected based on numerical values to prevent numerical instability (caused by error growth because of numerical round-off), a priori selection based entirely on structural, as opposed to numerical, information can lead to numerical instability, as mentioned in Section 4.3.2 and discussed in [Duff et al., 1986].

Another potential issue of blindly eliminating variables is that the reduced system may be significantly more dense. As opposed to dense direct linear solvers, sparse direct linear solvers do not only select pivots to min-

imize error growth, but also to minimize fill-in. The typical approach is to select as pivot element the element that causes the least amount of estimated fill-in in the matrix factors while also being bigger (in magnitude) than a fraction of the largest element in the same column. This is called partial pivoting. The reason that only an estimate of the fill-in is used is because of the computational intractability of computing the fill-in caused on a global level (the final matrix factors). The most widely used estimate of fill-in is the Markowitz criterion [Markowitz, 1957]. Let $\operatorname{nnz}\mathbf{M}$ denote the number of nonzero elements of $\mathbf{M}$. When LU-factorizing a sparse (sub)matrix $\mathbf{M}$, the Markowitz criterion selects as the next pivot the element $M_{i,j}$ that minimizes

$$(-1 + \operatorname{nnz} \boldsymbol{M}_{i,:}) \cdot (-1 + \operatorname{nnz} \boldsymbol{M}_{:,j}), \tag{4.18}$$

typically out of those elements satisfying

$$|M_{i,j}| \geq p_{\text{tol}} \max_l |M_{l,j}|, \tag{4.19}$$

where $0 < p_{\text{tol}} \leq 1$ is the pivot tolerance. The red part of (4.18) corresponds to the number of dependencies used in the elimination of element $M_{i,j}$, with the incidence of $M_{i,j}$ itself being subtracted. The blue part of (4.18) is the number of equations in which $M_{i,j}$ will be substituted by its dependencies, with the equation used to eliminate $M_{i,j}$ being subtracted. A small value of p_{tol} allows for a small value of (4.18), leading to fast computations. On the other hand, a smaller value of p_{tol} leads to worse error bounds of the solution.

Another useful estimate is that of local minimum fill-in, also proposed in [Markowitz, 1957], which is a better but more expensive estimate. The difference is that local minimum fill-in computes the actual fill-in in each stage, whereas the Markowitz criterion estimates the fill-in by the amount of incidences in the pivot row and column; that is, it does not take into account that some of the incidences already occur in the remaining equations and hence do not cause additional fill-in. The sparsity-preservation techniques of Section 4.4.3 are related to these ideas.

4.4 Symbolic Elimination for Dynamic Optimization

In Section 4.3 we reviewed how implicit DAEs can be transformed to explicit ODEs. While almost all of the steps were symbolic in nature, in the end we were left with the algebraic loops $\boldsymbol{F}^i = 0$ (which can be further symbolically reduced through the use of tearing), which in general will require numerical iterative methods to solve. However, going all the way to an explicit ODE is not necessary when applying direct collocation, as collocation methods are

perfectly capable of dealing with low-index (and to some extent also high-index) implicit DAEs. In this section (and also in [Magnusson and Åkesson, 2016]) we describe how the causalization techniques can be applied to (3.5) to symbolically eliminate many of the algebraic variables.

We will describe various approaches to symbolic elimination which can be combined in different ways, giving rise to different *schemes* of symbolic elimination. In Chapter 6 we will compare the performance of these schemes to each other. All schemes start by applying the first three steps of the symbolic procedure of Section 3.2.3—alias elimination, variability propagation, and index reduction—to the DAE in (3.5b) in the standard manner. The first scheme is obtained by stopping the symbolic transformations at this point.

Scheme 0
Do not eliminate any algebraic variables (other than those eliminated by alias elimination and variability propagation). □

This scheme can be considered to be the standard approach when employing direct collocation to dynamic optimization problems, although many dynamic optimization tools do not perform any of the the first three symbolic steps. Our lofty goal is to devise a scheme that *always* outperforms Scheme 0 in terms of computation time, which requires us to strike a balance between the benefits and drawbacks of the techniques discussed in Section 4.3

4.4.1 Elimination in Scalar, Linear Diagonal Blocks

We proceed by applying the standard causalization procedure described in Section 4.3.1. We next identify the diagonal blocks $\boldsymbol{F}^i$ of the BLT form that are both scalar and linear; that is, $\operatorname{struct} \boldsymbol{F}^i = 1$. The corresponding variables $\boldsymbol{z}^i$ of these blocks, which are scalar, are considered for elimination. By elimination we mean that all occurrences of $\boldsymbol{z}^i$ in (3.5) are substituted by $\left(\boldsymbol{F}^i\right)^{-1}\left(\boldsymbol{0}; \boldsymbol{v}^i\right)$ using (4.11). Note that the computation of $\left(\boldsymbol{F}^i\right)^{-1}$ in this case is just division by a single, scalar, closed-form expression[1]. After this elimination step, the variable $\boldsymbol{z}^i$ and the equation to which it has been matched are removed from the problem.

In a first effort, we choose to eliminate all such variables (whose matched equations are scalar and linear with respect to the variable) that are unbounded, that is, those for which the corresponding elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ are $-\infty$ and $+\infty$, respectively. No other variables are eliminated.

[1] In the general case, care is needed to ensure that this scalar expression is nonzero [Meijer, 2011; Baharev et al., 2016b]. However, since this is always the case for the problems of Chapter 5, we henceforth ignore this issue for the sake of simplicity.

Scheme 1
Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, and whose incidences in the diagonal blocks are linear. □

A consequence of eliminating bounded variables is that the linear bound is transformed into a nonlinear inequality constraint (it is moved from the first part of (3.5e) to the second part of (3.5c)). Since IPOPT transforms (2.18) to (2.19), any bounded variable that we eliminated will get reintroduced again as a slack variable, leading to no difference in the dimension of the KKT system that is solved. So there is no obvious benefit in eliminating bounded variables. But while the size of the KKT system is unaffected, the structure is not. One situation when the change in structure is potentially harmful is when the NLP functions are undefined if the algebraic variable bound is not satisfied. Since IPOPT iterates always are feasible with respect to the variable bounds, but not necessarily the nonlinear inequality constraints, eliminating bounded algebraic variables can cause evaluation errors which otherwise would not have occurred. For these reasons, we do not consider the elimination of bounded variables.

There is a problem with the approach of only eliminating unbounded variables. Unbounded variables are in JModelica.org identified by their lack of finite values of their `min` and `max` attributes. As pointed out in Section 2.3.3, these attributes are already defined by Modelica with a slightly different meaning. Many variables in Modelica models are thus assigned `min` and `max` values that are not originally intended to be used as constraints in an optimization problem. For example, all variables declared as temperatures inherit a lower bound of 0 [K], even though these bounds will never be active at feasible points. Because of this, we force the user to specify which variables are expected to be actively bounded, and consider all other variables as unbounded for the purpose of elimination, even if they have finite `min` and `max` values.

We only consider the elimination of algebraic variables, and not state variables. While the techniques we use to find closed-form expressions for computing algebraic variables also can be used to find closed-form expressions for the state derivatives (which indeed is what is done when doing full causalization to obtain an explicit ODE), finding these expressions is not sufficient for the elimination of the state derivatives. In order to eliminate the state derivatives, we would also have to eliminate the corresponding state variables, which would require the solution of differential equations. Finding closed-form solutions to differential equations is rarely possible for the cases of interest, so this possibility is not considered further in this thesis.

There is however another potential use of the closed-form expressions for the state derivatives, that does not involve their elimination. All occur-

rences of a state derivative, except the ones in the matched diagonal block equations, can be substituted by the closed-form expression. While this will never affect NLP size, it will affect the NLP sparsity in a way that depends on the simultaneous discretization method. The collocation framework of Chapter 3 creates NLP variables for all state derivatives, see (3.15h). So from a sparsity perspective it is never good to substitute the differential variable derivatives in JModelica.org. However, another common approach to direct collocation is to eliminate the state derivatives by using (3.15h). In this case it is not obvious whether such substitutions would be beneficial. Comparing the number of incidences from a sparsity perspective would be easy enough, but since the closed-form expressions found via the causalization tend to be highly nonlinear whereas the collocation equations are linear, a pure sparsity perspective is probably too narrow to be useful.

While it is known that performing eliminations along the lines of Scheme 1 can be beneficial for dynamic optimization, tools usually require such eliminations to be manually identified and performed by the user. Such work can be tedious and, depending on the modeling language, lead to convoluted and model code and inefficient computations. The techniques of Section 4.3.1 enable the automation of Scheme 1.

4.4.2 Elimination in Nonscalar Diagonal Blocks

We next consider the elimination of variables in nonscalar diagonal blocks. If the diagonal blocks are linear—that is, struct $\boldsymbol{F}^i \in \{0, 1\}^{n^i \times n^i}$, where n^i is the dimension of $\boldsymbol{z}^i$—a conceivable approach would be to invoke a numerical (rather than symbolical) factorization algorithm to solve for $\boldsymbol{z}^i$. This approach however requires the use of MX graphs already at the level of the DAE, see Section 3.5.3. While this has been implemented in JModelica.org, the resulting NLP function evaluation times are exceedingly slow. This can be circumvented in two ways. Either a sophisticated mixture of SX and MX graphs to represent the DAE in JModelica.org could be used, or CasADi's SX graphs could be extended to allow function calls. Both of these possibilities are beyond the scope of this thesis. Hence, we do not further consider the embedding of numerical methods in the DAE.

Another possibility of treating linear, nonscalar blocks is symbolic factorization. While this is readily supported by the implemented framework, experiments on the problems that will be presented in Chapter 5 have shown that symbolic QR factorization all too often leads to numerical issues caused by instability because of the lack of numerical pivoting. This often prohibits its practical use, and so is not considered further in this thesis.

There is however another approach that allows us to eliminate some of the variables in nonscalar diagonal blocks, which even extends to nonlinear

blocks: tearing. By tearing the diagonal blocks as described in Section 4.3.2, we can then symbolically eliminate the causalized variables in the torn blocks by forward substitution.

There are two issues with the choice of causalized variables by the JModelica.org compiler. The first is that it can choose to causalize state derivatives, which is useful when the causalization goal is to get an explicit ODE. But as per the discussion in Section 4.4.1, we are not interested in eliminating state variables or their derivatives. Likewise, we are not interested in eliminating (actively) bounded variables. Hence, our approach is to, besides using the tearing variables and residuals selected by the compiler, add all state derivatives and bounded variables as tearing variables and their respectively matched equations as tearing residuals. It would have been better to force the compiler to make these choices, allowing it to potentially make better choices in choosing the remaining tearing variables and residuals, but this has not been implemented.

Scheme 2
Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, and whose incidences in the diagonal blocks are linear. Also eliminate causalized, unbounded algebraic variables in torn, nonscalar diagonal blocks. □

4.4.3 Sparsity-Preserving Elimination

In our attempt to eliminate as many variables as possible using only symbolic techniques, we may actually end up eliminating too many in terms of computational efficiency. As briefly mentioned before, although the elimination of variables leads to smaller systems, they also tend to be more dense. The increased density can cause crippling slowdowns when performing sparse computations, in particular when solving the sparse KKT system (2.25).

Therefore, we propose to avoid some of the eliminations performed by Scheme 1 and 2, by analyzing the effect each elimination has on the DAE structural incidence matrix in a sixth step of the symbolic transformations presented in Section 3.2.3. We define a density measure μ of a causalized block variable $\bar{\boldsymbol{z}}_j^i$ which measures how much denser the resulting NLP will become if it is eliminated using (4.16). To facilitate uniform treatment of scalar, linear diagonal blocks and torn diagonal blocks, we consider a scalar, linear diagonal block equation and its corresponding variable to be causalized. Unfortunately, any useful measure will depend on which other eliminations are performed. This fact makes it intractable to consider all possible combinations, just like it is not tractable for direct, sparse linear solvers to find the pivot sequence with minimal fill-in. However, due to the causalization, we can consider the blocks and the variables within the

blocks in sequence to devise a greedy algorithm which only considers the current block. Hence, for a given block, whether to perform the eliminations in preceding blocks has already been decided, and we only consider the situation in which no eliminations are performed in succeeding blocks, consequently disregarding how eliminations in succeeding blocks will be affected by the eliminations in the current block. This simplified approach is analogous to how linear solvers use the Markowitz criterion or local minimum fill-in as discussed in Section 4.3.3.

Hence, we want to estimate how much the number of nonzero elements in the KKT system will increase if $\bar{\boldsymbol{z}}^i_j$ is eliminated. We perform the elimination if

$$\mu(i,j) \leq \mu_{\text{tol}}, \tag{4.20}$$

where μ_{tol} is a user-provided tolerance. Rather than considering the sparsity of the KKT matrix, we instead only consider the number of nonzeros in the structural incidence matrix of the DAE—which is the dominant part of the KKT system of a typical dynamic optimization problem—for simplicity and computational efficiency during symbolic preprocessing. An estimative measure for this is

$$\mu(i,j) = \left(-2 + \sum_{\boldsymbol{\alpha}\in\{\dot{\boldsymbol{x}},\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{p}\}} \sum_{k=1}^{n_\alpha} \mathcal{I}(\nabla_{\alpha_k} \bar{F}^i_j)\cdot d(\alpha_k)\right)\cdot\left(-1+\operatorname{nnz} \nabla_{\bar{z}^i_j} \boldsymbol{F}\right), \tag{4.21}$$

where $\mathcal{I}$ is the indicator function (mapping zero functions to 0 and all other functions to 1), and $d(\alpha_k)$ is the number of post-elimination dependencies of α_k. A recursive expression for d is

$$d(\alpha_k) := \begin{cases} 1, & \text{if } \alpha_k \text{ is not eliminated,} \\ \displaystyle\sum_{\boldsymbol{\beta}\in\{\dot{\boldsymbol{x}},\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{p}\}} \sum_{l=1}^{n_\beta} \mathcal{I}(\nabla_{\beta_l} \bar{h}_{\alpha_k}) \cdot d(\beta_l), & \text{otherwise,} \end{cases} \tag{4.22}$$

where $\bar{h}_{\bar{z}^i_j} := \bar{H}^i_j$ (see (4.16)). Note that $d(\alpha_k)$ not only depends on α_k but also which variables have been chosen for elimination, which changes over the course of the sparsity analysis. The measure (4.21) is similar to the Markowitz criterion (4.18) in that the red and blue parts correspond to each other. A difference is that (4.21) considers the whole system simultaneously rather than a single stage of Gaussian elimination, and also takes into account that the full system not only depends on $\boldsymbol{z}$ but also $\boldsymbol{v}$. Another important difference is that the Markowitz criterion estimates fill-in, whereas (4.21) estimates the increase in number of nonzeros, hence the subtraction of 2 rather than 1 in the red part. For example, the elimination of an alias variable can cause a lot of fill-in, but will not increase the number of nonzeros in the structural incidence matrix of the DAE. Thus, the value given by

(4.18) of an alias variable will be equal to the number of incidences in the subsequent equations, whereas the value given by (4.21) will be 0.

To demonstrate (4.20)–(4.22), consider the torn incidence (4.8) of the previous example with $\mu_{\text{tol}} = 3$. The variables that are eligible for elimination are y_4, y_1, and y_2. At the start of the procedure, we have $d(\alpha) = 1$ for all α. Since the first block $\boldsymbol{F}^1$ has no eligible variables, we move on to $\boldsymbol{F}^2$ where we find the pair $((4.1e), y_4)$—noting that $\bar{F}_1^2$ is the residual for (4.1e)—for which we compute

$$\begin{aligned}
\mu(2,1) &= \Big(-2 + \sum_{\boldsymbol{\alpha} \in \{\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{y}\}} \sum_{k=1}^{n_\alpha} \mathcal{I}(\nabla_{\alpha_k} \bar{F}_1^2) \cdot d(\alpha_k) \Big) \cdot \big(-1 + \operatorname{nnz} \nabla_{\tilde{z}_1^2} \boldsymbol{F} \big) \\
&= \Big(-2 + \sum_{\alpha \in \{\dot{x}_1, x_1, y_1, y_2, y_3, y_4, y_5\}} \mathcal{I}(\nabla_{\alpha} \bar{F}_1^2) \cdot d(\alpha) \Big) \cdot \big(-1 + \operatorname{nnz} \nabla_{y_4} \boldsymbol{F} \big) \\
&= \Big(-2 + \mathcal{I}(1) \cdot d(y_4) + \mathcal{I}\Big(-\frac{1}{2\sqrt{y_5}} \Big) \cdot d(y_5) \Big) \\
&\qquad \cdot \big(-1 + \operatorname{nnz}(0, 0, 2y_1 y_2, y_1 - 1, 1, 0) \big), \\
&= (-2 + 2) \cdot (-1 + 3) = 0 \leq \mu_{\text{tol}} = 3,
\end{aligned} \tag{4.23}$$

and hence proceed to eliminate y_4 and compute $d(y_4) = 1$. In the next block we find the eligible pairs $((4.1d), y_1)$ and $((4.1b), y_2)$. For the first pair we compute

$$\mu(3,1) = (-2 + 4) \cdot (3 - 1) = 4 > \mu_{\text{tol}}, \tag{4.24}$$

and consequently do not eliminate y_1. Moving on to the second and final pair, we compute

$$\mu(3,2) = (-2 + 3) \cdot (3 - 1) = 2 \leq \mu_{\text{tol}}, \tag{4.25}$$

and hence eliminate y_2, for which we compute $d(y_2) = 2$.

The measure (4.21) is locally suboptimal in the same sense as the Markowitz criterion: It does not take into account that not all incidences will cause fill-in; it is a worst case estimate. To address this, we can instead of (4.21) use the measure

$$\begin{aligned}
\mu(i,j) = \sum_{\iota=i}^{m} \sum_{\jmath=\begin{cases} j+1, & \text{if } \iota = i, \\ 1, & \text{otherwise} \end{cases}}^{n^\iota} \mathcal{I}(\nabla_{\tilde{z}_\jmath^\iota} F_\jmath^\iota) \Big(-1 \\
+ \sum_{\boldsymbol{\alpha} \in \{\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{p}\}} \sum_{k=1}^{n_\alpha} \big(1 - \mathcal{I}(\nabla_{\alpha_k} F_\jmath^\iota) \big) \mathcal{I}(\nabla_{\alpha_k} F_j^i) d(\alpha_k) \Big),
\end{aligned} \tag{4.26}$$

where F_j^i is element j in block i with the ordering obtained after tearing; that is,

$$F_j^i := \begin{cases} \bar{F}_j^i, & \text{if } j \leq \bar{n}_i, \\ \hat{F}_{j-\bar{n}_i}^i, & \text{if } j > \bar{n}_i. \end{cases} \tag{4.27}$$

This measure is analogous to local minimum fill-in in the same way as (4.21) is analogous to (4.18).

As discussed in Section 4.3.3, the Markowitz criterion is generally regarded as the best for general-purpose pivot selection, with local minimum fill-in being a lot more computationally expensive while usually only yielding slightly better results. However, for our purposes, we only perform the sparsity-preservation analysis once offline. Furthermore, the computation times of both (4.21) and (4.26) are negligible compared to the other offline computations (model compilation, BLT analysis, collocation discretization, algorithmic differentiation graph construction, and so on). So while both measures have been implemented in JModelica.org, we will henceforth only consider the use of (4.26) because of its slightly better expected performance.

Sparsity preservation can be used both with and without tearing, yielding two new families of schemes, parametrized by the density tolerance μ_{tol}.

Scheme $3_{\mu_{\text{tol}}}$
Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, whose incidences in the diagonal blocks are linear, and whose density measures are smaller than or equal to μ_{tol}. □

Scheme $4_{\mu_{\text{tol}}}$
Eliminate those algebraic variables that are unbounded, belong to scalar diagonal blocks, whose incidence in the diagonal blocks are linear, and whose density measures are smaller than or equal to μ_{tol}. Also eliminate causalized, unbounded algebraic variables in torn, nonscalar diagonal blocks whose density measures are smaller than or equal to μ_{tol}. □

The choice of density tolerance is important and nonobvious. A tolerance of 0 means that we only perform eliminations that do not increase the number of nonzeros in the other equations. Such variables are similar to alias variables. A tolerance of $-\infty$ means that no eliminations are performed at all, and so we preserve the original DAE. A tolerance of ∞ means that we eliminate all causalized variables. Hence, we get

$$\text{Scheme } 0 = \text{Scheme } 3_{-\infty} = \text{Scheme } 4_{-\infty}, \tag{4.28a}$$

$$\text{Scheme } 1 = \text{Scheme } 3_{\infty}, \tag{4.28b}$$

$$\text{Scheme } 2 = \text{Scheme } 4_{\infty}. \tag{4.28c}$$

4.5 Conclusion

DAE-constrained optimization problems are usually solved by exposing the full DAE to a discretization method. We considered ways of preprocessing the DAE by symbolically eliminating most of the algebraic variables using techniques based on block-triangular orderings, tearing, and sparsity preservation, resulting in 4 different schemes which are summarized in Table 4.1. These schemes have been implemented in JModelica.org and their relative performances will be evaluated in Chapter 6.

Table 4.1 Summary of techniques used in schemes.

Scheme	BLT	Tearing	Sparsity preservation
0			
1	✓		
2	✓	✓	
$3_{\mu_{\text{tol}}}$	✓		✓
$4_{\mu_{\text{tol}}}$	✓	✓	✓

The proposed schemes are more relevant for simultaneous (rather than sequential, see Section 2.3.4) approaches to dynamic optimization where they have a major impact on the size and structure of the NLP, whereas they often only affect the NLP function evaluation times when using sequential approaches. Nevertheless, these methods certainly have the potential to be beneficial also when using sequential approaches, since the DAE simulation in shooting methods is often the computational bottleneck and a source of numerical problems.

While block-triangular ordering and sparsity preservation guarantees preservation of numerical stability, tearing does not. This is a well-known drawback of tearing, which is difficult to address under the typical circumstance of having no reliable a priori numerical information regarding the solution. However, in the context of dynamic optimization a decent initial guess of the full solution is often required to solve the problem. Utilizing this initial guess, it might be tractable to design a numerical tearing algorithm which guarantees numerical stability, under the assumption that the initial guess is sufficiently close to the solution, using techniques such as those considered by Westerberg et al. [Westerberg and Edie, 1971; Gupta et al., 1974].

The sparsity preservation approach is based on the DAE incidence matrix. The system whose sparsity we really are interested in is the KKT system. However, the sparsity of the DAE plays a major role in the sparsity of the KKT matrix via the Jacobian of the equality constraints. An important part of the KKT matrix that we have neglected is the Hessian of the

Lagrangian, which however often plays a minor role in the sparsity of the KKT matrix compared to the Jacobian when employing direct collocation.

The used tearing algorithm first selects tearing variables and residuals using the JModelica.org compiler to obtain causalized equations that are triangular and linear along the diagonal, and then selects additional tearing variables and residuals to not eliminate variables that are bounded, differential, or cause too much fill-in. Considering all of these criteria simultaneously, rather than sequentially, would enable fewer tearing variables and residuals to be selected. Simultaneously minimizing the number of tearing variables and the amount of fill-in would however require more sophisticated heuristics than (4.20), because of their complex interaction.

5

Problem Suite

> Inside of every problem lies an opportunity.
>
> Robert Kiyosaki

To demonstrate the capabilities of the framework in Chapter 3 and evaluate the schemes proposed in Chapter 4, we need a suite of test problems. Unfortunately, there is little in the way of an established suite of test problems for dynamic optimization. [Hedengren, 2008] is a collection of DAEs implemented in MATLAB that are suitable for dynamic optimization. But most of the models are of modest size and not representative of typical Modelica models. Furthermore, there are no accompanying optimization formulations, only models. The popular test suite for nonlinear programming CUTEr [Gould et al., 2003] contains discretized dynamic optimization problems, but these are of little value to us in their discretized form.

Hence, we present our own small suite of dynamic optimization problems that is suitable for evaluating the methods of Chapters 3 and 4. The suite consists of five different optimal control problems that are computationally challenging and have industrial relevance. The systems in the five problems are respectively a:

- Car
- Combined-cycle power plant (CCPP)
- Double pendulum
- Closed kinematic loop with four bars
- Distillation column

All of the problems are encoded in Modelica and Optimica and are available as a part of JModelica.org's collection of examples. For each problem, we

first present the system, then the considered optimization problem, and finally optimal trajectories corresponding to the optimization problem.

The problems have been chosen to together span a large part of all models of interest for dynamic optimization. The systems originate from various engineering domains and the DAEs have properties suitable for comparing the different schemes of Chapter 4. The DAE sizes range from moderate (36 equations) to large (1125 equations), to demonstrate how the implementation fares on problems of different sizes. Three out of the five problems have been developed by others for the sake of application-oriented research. The remaining two problems are based on examples from the Modelica Standard Library (MSL), to show that the optimization framework of the thesis to some extent can deal with general-purpose Modelica models that have not been developed specifically for optimization purposes.

The notation in each section is independent from the rest of the thesis.

5.1 The Many Colors of Block-Triangular Decompositions

For most of the models, we show a BLT decomposition of the DAE obtained with Scheme 4_{30} (see Chapter 4). In these figures, linear incidences are marked by green dots, and nonlinear incidences are marked by red dots. Since we do not distinguish between linear and nonlinear incidences outside of the diagonal blocks, such incidences are marked by black dots. Torn blocks are marked by red edges. Variables, and their respective matched equations, that have been user-specified as actively bounded (and hence are not eliminated) are marked by orange edges. State variable derivatives (which are not eliminated) and their respective matched equations are marked by blue edges. Variable–equation pairs along the diagonal that are not sparsity preserving—that is, do not satisfy (4.20)—are marked by yellow edges. The remaining variable–equation pairs along the diagonal are the ones used for elimination, which are marked by green edges.

5.2 Car

The first problem is a minimum-time problem, where a car maneuvers a 90-degree turn. Variations of this problem and suitable models and methodology have been investigated in [Berntorp et al., 2014; Berntorp and Magnusson, 2015] and several other publications by the same group of authors. The motivation for studying these problems is both to understand the limitations of system performance and to design automated vehicle control and safety systems.

5.2.1 Model

The model we use in the thesis has a nonlinear, single-track chassis model [Rajamani, 2006], where the two wheels on each axle are lumped together, see Figure 5.1. The chassis model has three degrees of freedom: two translational and one rotational. The model is described by

$$\dot{v}^X - v^Y\dot{\psi} = \frac{1}{m}(F_f^x\cos(\delta) + F_r^x - F_f^y\sin(\delta)), \tag{5.1a}$$

$$\dot{v}^Y + v^X\dot{\psi} = \frac{1}{m}(F_f^y\cos(\delta) + F_r^y + F_f^x\sin(\delta)), \tag{5.1b}$$

$$I_{ZZ}\ddot{\psi} = l_f F_f^y\cos(\delta) - l_r F_r^y + l_f F_f^x\sin(\delta), \tag{5.1c}$$

where m is the vehicle mass, I_{ZZ} is the vehicle inertia about the Z-axis, ψ is the yaw, δ is the steer angle, v^A is the velocity at the center of gravity along axis $A \in \{X, Y\}$, l_i is the distance from the mass center to wheel $i \in \{f, r\}$ (front or rear), and F_i^a is the tire force for wheel i along axis $a \in \{x, y\}$ in the coordinate system of the wheel.

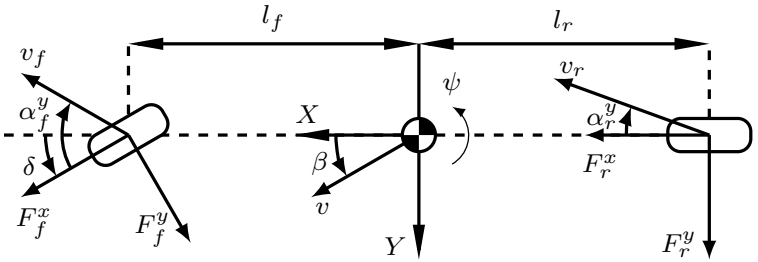

Figure 5.1 The single-track chassis model.

The nominal tire forces $F_{i,0}^a$ in wheel i along axis a under pure slip conditions are computed with the Magic formula [Pacejka, 2006], given by

$$F_{i,0}^a = \mu_i^a F_i^z \sin\Big(C_i^a \arctan\big(B_i^a\alpha_i^a - E_i^a(B_i^a\alpha_i^a - \arctan B_i^a\alpha_i^a)\big)\Big), \tag{5.2}$$

where F_i^z is the tire normal force, μ_i^a is a friction coefficient, B_i^a, C_i^a, as well as E_i^a are parameters depending on the physical properties of the tires and road, and α_i^y and α_i^x are the lateral and longitudinal slip, respectively, of wheel i defined as in [Pacejka, 2006]. The actual tire forces F_i^a under combined longitudinal and lateral slip are modeled by scaling the nominal forces $F_{i,0}^a$ with a weighting function G_i^a [Pacejka, 2006]. The relations are

$$\begin{aligned} F_i^a &= F_{i,0}^a G_i^a, \\ G_i^a &= \cos(C_{i,1}^a \arctan(H_i^a\alpha_i^a)), \\ H_i^a &= B_{i,1}^a \cos(\arctan(B_{i,2}^a\alpha_i^b)), \end{aligned} \tag{5.3}$$

where $B^a_{i,1}$, $B^a_{i,2}$, as well as $C^a_{i,1}$ are physical parameters and b is the axis opposing a; that is,

$$b := \begin{cases} x, & \text{if } a = y, \\ y, & \text{if } a = x. \end{cases} \tag{5.4}$$

The wheel dynamics are modeled by

$$\tau_i = I_i \dot{\omega}_i - R_i F^x_i, \tag{5.5}$$

where I_i is the wheel inertia, R_i is the wheel radius, ω_i is the wheel angular velocity, and τ_i is the applied torque.

The resulting DAE is index 1 and has 13 state variables, 23 algebraic variables, and 3 inputs. The inputs are the steer angle δ and the torque reference τ_i^{ref} on each wheel (related to the torque τ_i via linear dynamics).

A BLT decomposition for the DAE is shown in Figure 5.2. Since all diagonal blocks are linear and scalar, the DAE can be transformed to an explicit ODE on closed form (despite all the nonlinear expressions in (5.1)–(5.3)), although we do not do this since four algebraic variables have bounds as discussed in the next subsection.

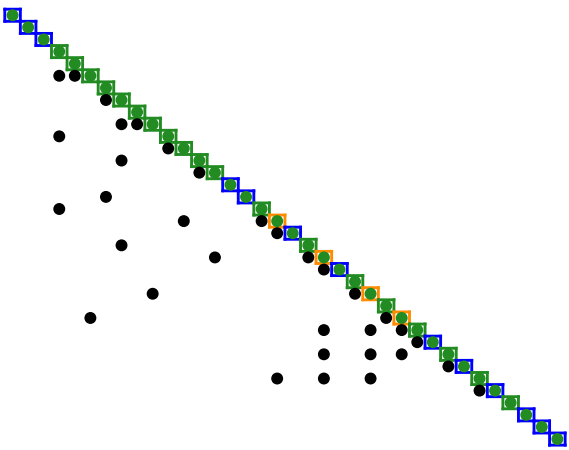

Figure 5.2 BLT decomposition of Car with Scheme 4_{30}, see Section 5.1. All diagonal blocks are linear and scalar. Furthermore, the system is sufficiently sparse to not require sparsity preservation. Consequently, all schemes—except 0—yield the same result for $\mu_{\text{tol}} = 30$.

5.2.2 Optimization Problem

The problem is to find the time-minimal maneuver for the car in a 90-degree turn on dry asphalt and with the initial velocity 25 m/s. This initial velocity

makes it hard simply to stay on the road and this maneuver is often used in the evaluation of active safety systems. The road boundary is described by two hyperellipses with exponent 8. The constraint of staying inside the road is thus a high-index, nonlinear path inequality constraint.

The objective is to minimize the time needed to go from a fixed point on one end of the curve to a fixed point on the other end of curve without leaving the road. The absolute values of the three system inputs are bounded. There are also theoretical upper bounds on the nominal tire forces $F^a_{i,0}$—from (5.2) we see that $|F^a_{i,0}| \leq \mu^a_i F^z_i$—which in this thesis have been incorrectly considered as potentially active, leading to the four orange incidences in Figure 5.2.

The input is constrained to be piecewise constant using blocking factors (see Section 3.5.2), with 30 different values of δ and 15 values of τ^{ref}_i. The absolute change in steer angle is further bounded by 2 degrees in each step. From a physical perspective, this use of blocking factors is rather nonsensical for two reasons. First, the update frequency depends on the optimal t_f, since the time horizon is free and the frequency is coupled to the fixed collocation element distribution. Second, as a result of the optimal t_f, the update frequencies are artificially low. The purpose of this use of blocking factors is more to demonstrate their use rather than adhering to physics.

This optimization problem is similar to the high-level trajectory generation problem considered in [Berntorp and Magnusson, 2015], with the only difference being some parameter values and the use of blocking factors.

5.2.3 Solution

The problem is solved with $n^e = 60$ elements and $n^c = 3$ Radau points per element. The initial guess is generated by simulating the system using linear state feedback together with feedforward to control the system. The closed-loop controller tracks the middle of the road, and a predefined velocity profile is fed forward. The details of this controller are described in [Berntorp et al., 2014]. Since this controller has no chance of generating a feasible trajectory for the high initial velocity $v(t_0) = 25$ m/s, the initial guess is instead generated for the case $v(t_0) = 10$ m/s, resulting in $t_f \approx 11.0$ s. The optimal solution instead has $t_f \approx 4.0$, with the position trajectory being shown in Figure 5.3 and optimal inputs in Figure 5.4.

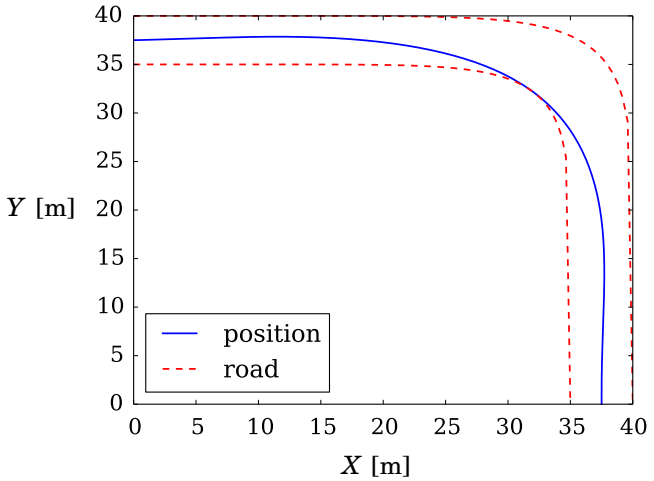

Figure 5.3 Optimal geometric path for Car. The car starts in the lower right corner and reaches the upper left corner in minimal time.

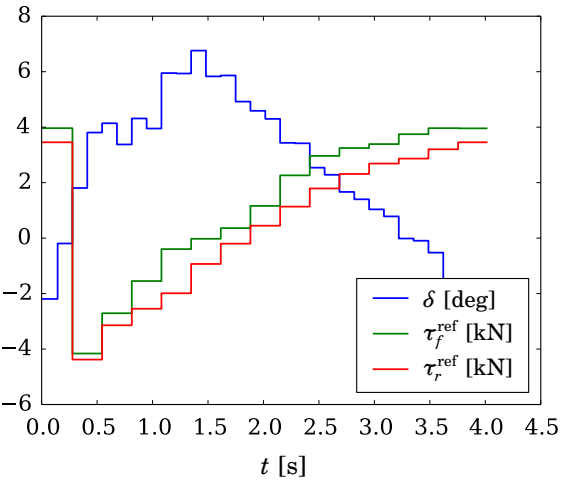

Figure 5.4 Optimal inputs corresponding to Figure 5.3. The torques are actively bounded above in the beginning and end, and the change in δ is also actively bounded.

5.3 Combined-Cycle Power Plant

The next case considers warm startup of a combined-cycle power plant (CCPP), where hot exhaust gas is used to heat and evaporate cold water to

drive a steam turbine that generates electricity. The problem of optimizing power plant startup has recently become highly industrially relevant, because of an increasing need to improve power-generation flexibility caused by the unpredictable output of renewable energy sources such as solar and wind power. Hence, there is a need to be able to efficiently start and shut down other power plants in order to balance the power grid.

5.3.1 Model

The model we use was first developed in [Casella et al., 2011], but the problem has been further studied in [Larsson et al., 2013; Parini, 2015]. Its object diagram is shown in Figure 5.5. The system has one input, which is the percentual load u [1] of the gas turbine. This load determines both exhaust gas flow from the turbine and the gas temperature. The two relationships between load and gas flow as well as temperature are accurately modeled as being piecewise affine with two pieces and the junction being at $u = 0.5$. Such a model is however not continuously differentiable, and is hence smoothly approximated using the shifted sigmoid step function

$$\frac{\arctan(100(u - 0.5))}{\pi} + 0.5, \tag{5.6}$$

which is used to connect the two affine pieces of each relation.

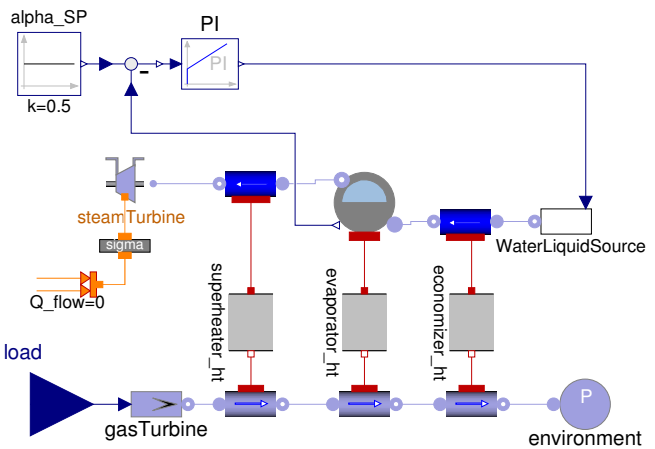

Figure 5.5 Dymola object diagram of CCPP. The exhaust from the gas turbine heats and evaporates cold, liquid water. Each heat exchanger has a component for the water side and the gas side. The superheated vapor then drives the steam turbine to generate electricity.

The exhaust gases then heat liquid water through three countercurrent heat exchangers: a superheater, evaporator, and economizer. With the exception of the water side of the evaporator, the gas and water sides of the heat exchangers are modeled by dynamic energy balances, accounting for the energy stored in the gas and for the heat transfer, while neglecting changes in mass storage and pressure losses.

To model the phase change of water from liquid to vapor in the evaporator, the water side of the evaporator also needs an equation for dynamic mass balance, with the thermodynamic properties of the water being approximated by low-order polynomials. The water flow is controlled by a PI controller to keep the volumetric fraction of liquid and vapor in the evaporator at 0.5. The superheated vapor finally generates electricity by driving the rotor of the steam turbine.

The resulting DAE has 10 state variables, 123 algebraic variables, and 1 input. Its BLT decomposition is shown in Figure 5.6, where we see that all diagonal incidences are linear, but there are two linear 2-by-2 algebraic loops. One of the loops has been torn, whereas the other remains untouched because it only involves state derivatives. Hence, we can eliminate all algebraic variables except the tearing variable in the first algebraic loop and another one on which we impose a bound according to the discussion in the next subsection.

5.3.2 Optimization Problem

The goal is to perform a quick warm start of the power plant. The plant is running at full capacity when $u = 1$ and the evaporator pressure p has reached its reference value. The objective is therefore a quadratic Lagrange function penalizing the deviation of u and p from their respective constant references. The problem is solved over a fixed time horizon of $t_f = 4000$ s.

Starting the plant too quickly, for example using $u \equiv 1$, will lead to large thermal stress σ in the steam turbine shaft. This drastically reduces the lifetime of the steam turbine, which is one of the most expensive components of the system. Hence, an upper bound is added on the algebraic variable σ. Furthermore, the gas turbine load has a physical upper bound on the rate of change and is also forbidden to be negative during startup.

This problem is the closest thing to a standard problem of dynamic optimization in the Modelica community[1], having been used several times for benchmark purposes [Andersson et al., 2011; Word et al., 2014; Lazutkin et al., 2015; Axelsson et al., 2015; Parini, 2015].

[1] Another often studied dynamic optimization problem in the Modelica community is the drum boiler in [Krüger et al., 2004], which however has hybrid dynamics and hence is out of scope for this thesis.

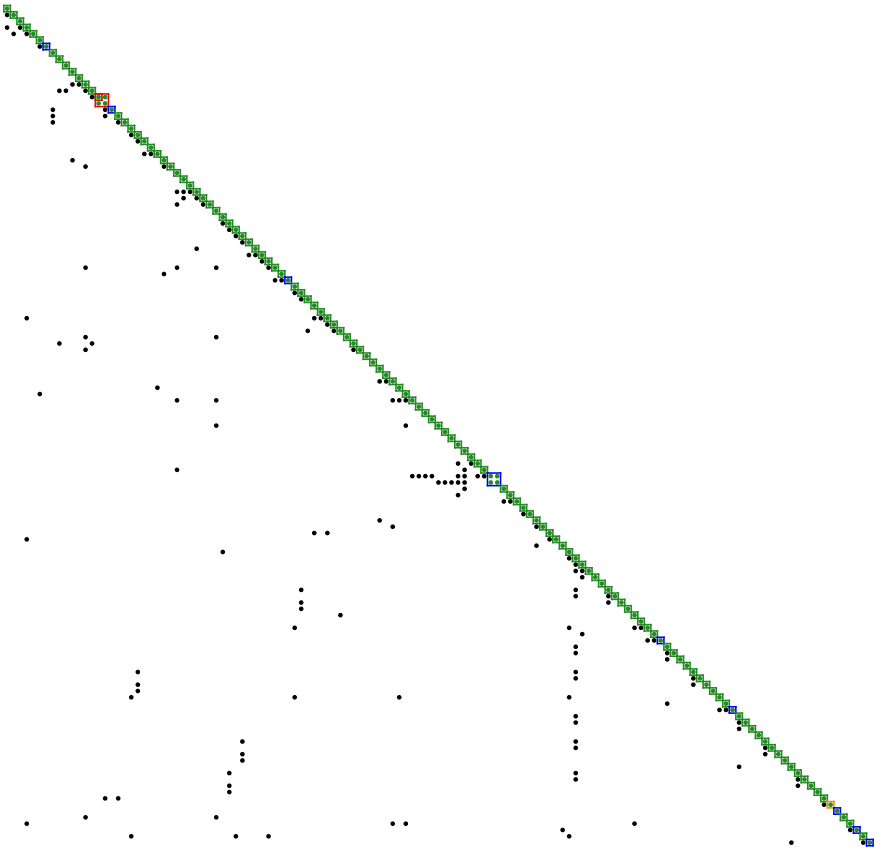

Figure 5.6 BLT decomposition of CCPP with SCHEME 4_{30}. All diagonal blocks are linear and sparsity preserving. All blocks except two are scalar.

5.3.3 Solution

The problem is solved with $n^e = 40$ elements and $n^c = 4$ points per element. The initial guess is generated by simulating the system using an open-loop input that is an affine function of time that avoids any constraint violations, but is far from completing the startup within the fixed t_f. The optimal solution is shown in Figure 5.7.

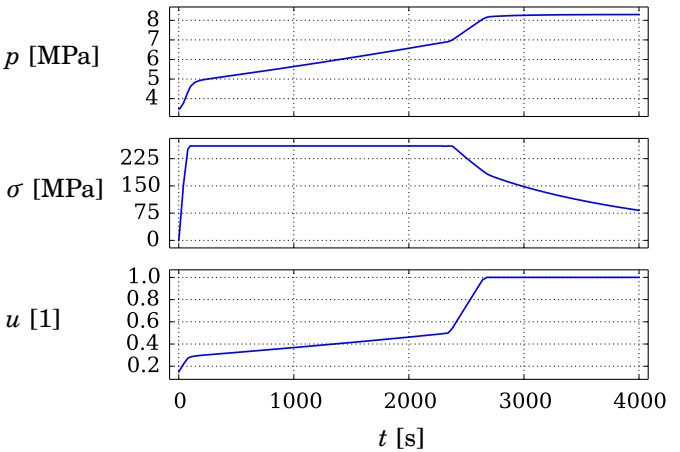

Figure 5.7 Optimal CCPP startup. The load u and evaporator pressure p reach their reference values in approximately 1 hour. During most of the startup, either the upper bound on σ or $\dot{u}$ is active.

5.4 Double Pendulum

Since no PhD thesis in automatic control is complete without an inverted pendulum, we next consider a double pendulum. The system is unstable and chaotic, so numerical methods based on single shooting are unsuitable for this system.

5.4.1 Model

One of the elementary examples of multibody mechanics in the Modelica Standard Library (MSL) is a damped double pendulum: `Modelica.Mechanics.Examples.Elementary.DoublePendulum`. This example's object diagram is shown in Figure 5.8. Since the model is unactuated in its original form, we add a torque on the first revolute joint `revolute1`, yielding the sole system input u [Nm].

A schematic of the considered double pendulum is shown in Figure 5.9. While this double pendulum is a conceptually simple fourth-order system, the MSL components for rigid bodies and revolute joints contain equations for computing forces and torques in various frames and transformation matrices from one frame to another. So the resulting DAE, which is of index 3, has 4 state variables and 124 algebraic variables after index reduction. The state variable vector is

$$\boldsymbol{x} = \begin{bmatrix} \phi_1 & \dot{\phi}_1 & \phi_2 & \dot{\phi}_2 \end{bmatrix}, \tag{5.7}$$

where ϕ_i [rad] is the angle of revolute joint i. A BLT decomposition of the DAE is shown in Figure 5.10.

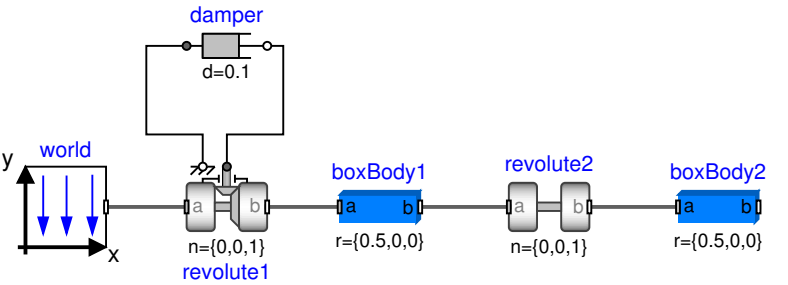

Figure 5.8 Dymola object diagram of the damped double pendulum from MSL. A rigid body is attached to a damped revolute joint in one end, and in the other end is another revolute joint with another rigid body.

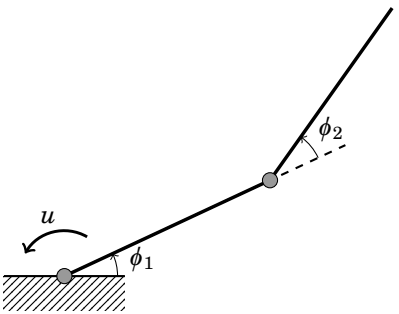

Figure 5.9 Double pendulum schematic. The torque u is used to control the angles ϕ_1 and ϕ_2.

5.4.2 Optimization Problem

The goal is to drive the pendulum from $\boldsymbol{x}(0) = -0.25\pi \cdot \mathbf{1}$ to

$$\boldsymbol{x}_f = \begin{bmatrix} 0.5\pi & 0 & 0 & 0 \end{bmatrix}^T, \tag{5.8}$$

which corresponds to the unstable equilibrium where both bodies are inverted. We do this over the fixed time horizon $t_f = 3$ s with the quadratic Lagrange objective

$$\int_0^3 \left((\boldsymbol{x}(t) - \boldsymbol{x}_f)^T \mathbf{Q} (\boldsymbol{x}(t) - \boldsymbol{x}_f) + Ru(t)^2 \right) \mathrm{d}t, \tag{5.9}$$

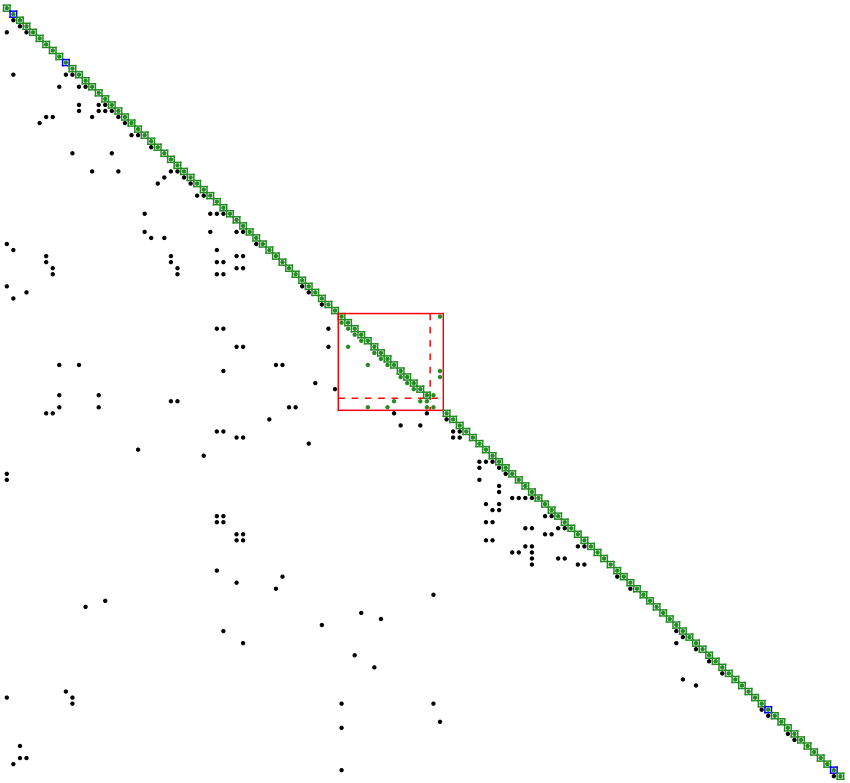

Figure 5.10 BLT decomposition of Double Pendulum with SCHEME 4_{30}. All diagonal blocks are linear and sparsity preserving. All but one block are scalar. The nonscalar block has 16 variables from both joints and bodies, many of which are dummy derivatives and a few being temporary variables introduced by the compiler. The block is torn with 2 tearing variables.

where

$$\mathbf{Q} = \operatorname{diag}\begin{bmatrix}1 & 5 & 0.1 & 0.1\end{bmatrix}, \quad R = 0.02. \tag{5.10}$$

We also impose the input bound $|u| \leq 500$ Nm (which may sound like a lot, but the two bodies each have a length of 0.5 m and are made of steel).

5.4.3 Solution

The problem is solved with $n^e = 100$ elements and $n^c = 3$ points per element. The initial guess is generated by simulating the system using linear state feedback to determine u. Only ϕ_1 and $\dot{\phi}_1$ are fed back and used to stabilize the first body of the pendulum in the inverted position, while

the second body runs amok. The initial guess also violates the upper bound on u. The optimal solution is shown in Figure 5.11.

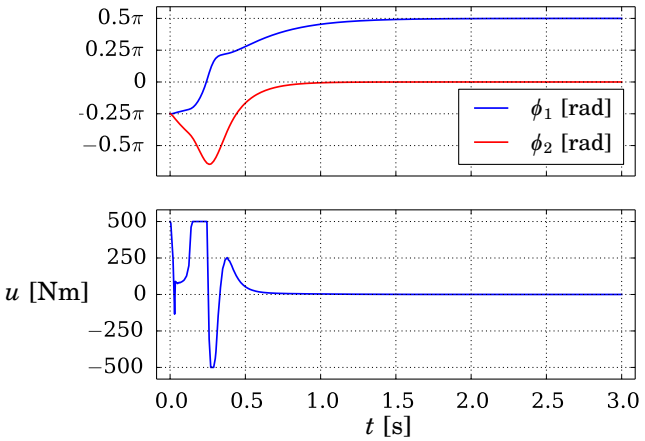

Figure 5.11 Optimal Double Pendulum swing-up with bounded torque u.

A benefit of using MSL's multibody mechanics library is that Dymola implements three-dimensional animations for many components. The web version of the thesis contains an appendix with an animation of the optimal trajectory. The appendix is not available in the printed version of the thesis.

5.5 Fourbar1

Next we consider another conceptually simple multibody mechanical system: a fourbar. Four rigid bodies are connected by 6 revolute joints and 1 prismatic joint. The resulting closed kinematic loop gives rise to a nonlinear algebraic loop.

5.5.1 Model

Once again the model is based on an example from MSL: `Modelica.Mechanics.Examples.Loops.Fourbar1`. The block diagram of the model was previously shown in Figure 2.1. To actuate the system, we add a torque on the first revolute joint, yielding the sole system input u [Nm].

The index-reduced DAE has 2 state variables and 452 algebraic variables. The state variables are the angle ϕ [rad] of the joint driven by the torque u and the corresponding angular velocity. A BLT decomposition of the DAE is shown in Figure 5.12.

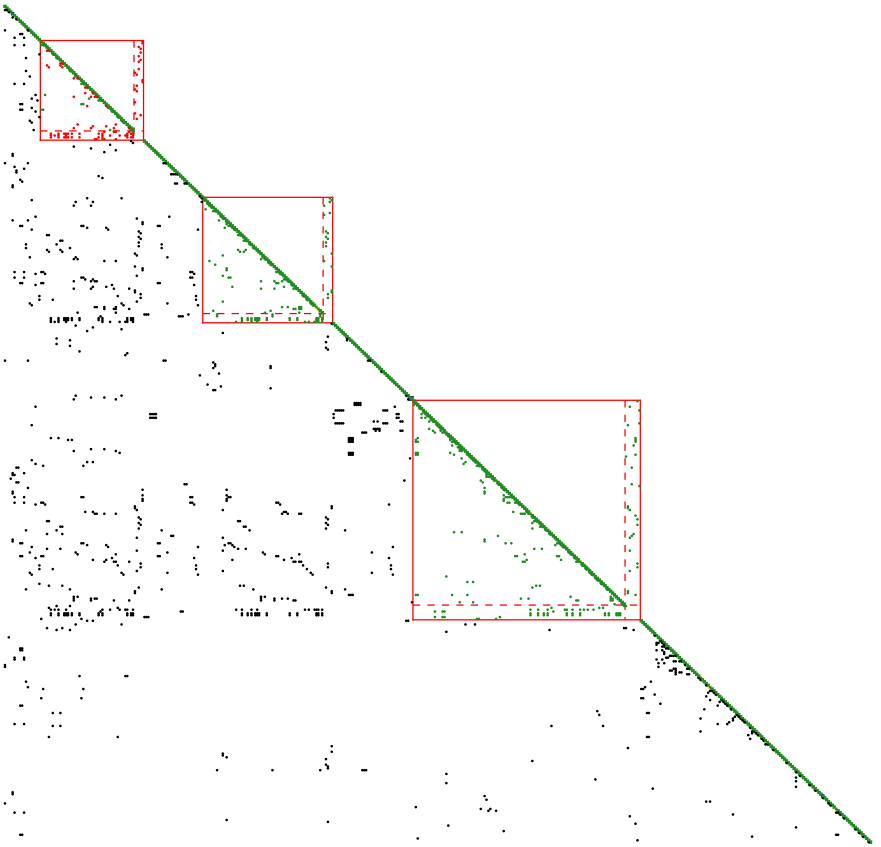

Figure 5.12 BLT decomposition of Fourbar1 with SCHEME 4_{30}. Almost all of the diagonal blocks are scalar and diagonal, but there are three large algebraic loops of dimensions 54, 68, and 119. Each loop is torn using a handful of variables. The smallest of the loops is nonlinear. 7 variables are retained to preserve sparsity.

It is noteworthy that this system can be modeled more efficiently by modeling the joints differently, as done in `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar2`, which only has a single algebraic loop, and `Modelica.Mechanics.MultiBody.Examples.Loops.Fourbar_analytic`, whose only algebraic loop is also linear. However, since the purpose of this problem is not to efficiently solve an actual problem, but rather to get something that is computationally challenging, we choose to work with Fourbar1.

5.5.2 Optimization Problem

We consider the problem of controlling the translation s [m] along the prismatic joint `j2` in one end of the mechanism by the torque u applied in the other end. We move it from $s(0) \approx 0.43$, which corresponds to the stable equilibrium of the system, to $s(t_f) = 0.35$ with $t_f = 1$. We impose the bound $|u| \leq u_{\max} = 90$ Nm and use the Lagrange objective

$$\int_0^1 \left((s(t) - 0.35)^2 + 10^{-8}(u(t) - 54.52)^2 \right) \mathrm{d}t, \tag{5.11}$$

where $u(t) \approx 54.52$ is the steady-state input yielding $s(t_f) = 0.35$.

Although this problem is silly, not only because of its artificiality, but also because it is adequately (albeit suboptimally) solved by a PI controller, it is still computationally challenging and the model itself contains fundamental building blocks of multibody mechanics.

5.5.3 Solution

The problem is solved with $n^e = 60$ elements and $n^c = 3$ points per element. This problem requires a very good initial guess (at least when using Scheme 0). The initial guess is generated by solving a sequence of optimization problems with increasing values of $u_{\max}$. The value is increased from 0 in increments of 10 until the desired value of 90 is reached. The solution is shown in Figure 5.13. The web version of the thesis contains an appendix with a three-dimensional animation of the optimal trajectories.

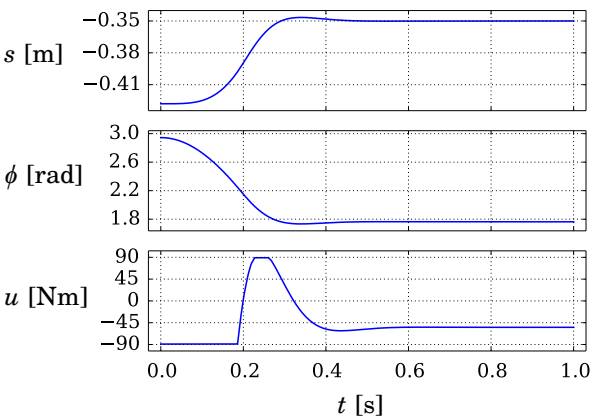

Figure 5.13 Optimal Fourbar1 repositioning with bounded torque u.

5.6 Distillation Column

The final problem concerns optimal control of a high-purity binary distillation column, which separates methanol from n-propanol and has 40 trays. Distillation columns are widely used in chemical engineering for the purpose of separating compounds into its separate parts.

5.6.1 Model

A schematic of a binary distillation column with only 8 trays is shown in Figure 5.14. The liquid mixture of the two components is boiled at the bottom. The resulting vapor rises through the column to the condenser, and the resulting liquid flows back down the column through each tray. Only the most volatile vapor reaches the top of the column, thus separating the most volatile of the components, methanol, from the other, n-propanol. Liquid mixture is fed at the middle tray with constant flow, temperature, and concentration. Pure liquid is extracted from the reflux drum and reboiler after separation. The system inputs are the reboiler heat Q [kW] and reflux flow rate L_{vol} [l/h].

We use a model that was developed in [Nagy et al., 2000; Diehl, 2002] and its Modelica implementation is based on the MATLAB implementation from [Hedengren, 2008].

Let X_i denote the liquid mole fraction of methanol (consequently, the liquid mole fraction of n-propanol is $1 - X_i$) in tray $i \in [1..42]$, numbered from top to bottom, where tray 1 is the condenser and tray 42 is the reboiler for notational convenience. Assuming constant molar holdup n in each tray, componentwise mass conservation yields

$$\dot{X}_i = \frac{1}{n}(Y_{i+1}V_{i+1} + X_{i-1}L_{i-1} - Y_iV_i - X_iL_i), \tag{5.12}$$

where in each tray i, Y_i is the vapor mole fraction of methanol and L_i as well as V_i are the outgoing liquid and vapor molar flux, respectively.

The total pressure P_i on each tray is assumed constant with a constant pressure increase from each tray to the next. The tray temperatures T_i [°C] are implicitly defined by the assumption that the sum of the partial pressures P_j^s for each component $j \in \{1, 2\}$ equals the total pressure on each tray, yielding

$$P_i = P_1^s(T_i)X_i - P_2^s(T_i)(1 - X_i), \tag{5.13}$$

where the partial pressures are computed according to the Antoine Equation

$$P_j^s(T) = \exp\left(A_j - \frac{B_j}{T + C_j}\right), \tag{5.14}$$

where A_j, B_j, and C_j are component-specific parameters.

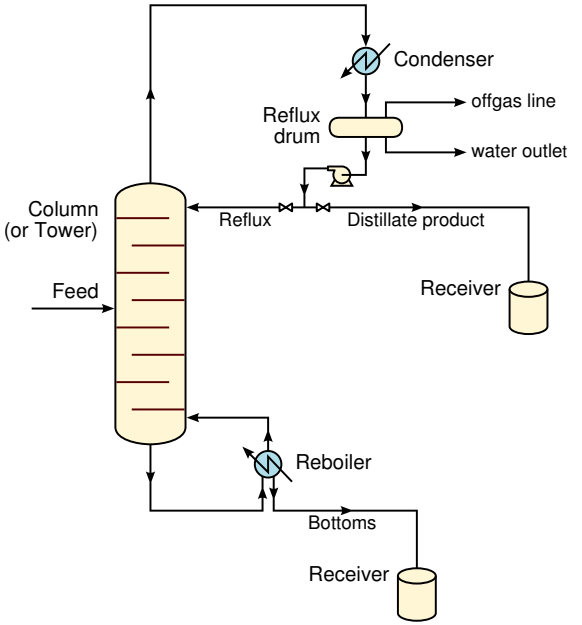

Figure 5.14 Schematic of binary distillation column, courtesy of [Sponk, 2010]. Liquid is boiled at the bottom, and the rising vapor is condensed at the top and then flows back down to the bottom through trays. This separates a binary mixture into the most volatile part, extracted at the top, and the least volatile part, extracted at the bottom.

The vapor molar flux is in each tray governed by the energy balance

$$\frac{\partial h_i^L}{\partial X_i}\dot{X}_i + \frac{\partial h_i^L}{\partial T_i}dT_i = \frac{1}{n}\left(h_{i+1}^V V_{i+1} + h_{i-1}^L L_{i-1} - h_i^V V_i - h_i^L L_i\right), \qquad (5.15)$$

where h_i^L and h_i^V is the liquid and vapor stream enthalpy, respectively, and dT_i is the dummy derivative of T_i, implicitly determined by the differentiation of (5.13).

Note that (5.12) and (5.15) are slightly different for the condenser, feed, and reboiler trays 1, 22, 42, due to the inlets and outlets.

The resulting index-one DAE has 42 state variables—the liquid mole fraction of each tray—1083 algebraic variables, and two system inputs—Q and L_{vol}. For each tray, there is a 3-by-3 nonlinear algebraic loop corresponding to (5.13) and (5.14) in the BLT decomposition of the DAE. These are torn using (5.13) as residual and T_i as variable. Then there is a large, linear diagonal block of size 205, torn using the tearing variables V_i and $\dot{X}_i$.

Eight additional variables are retained by SCHEME 4_{30} for sparsity preservation. The remainder of the algebraic variables belong to linear, scalar diagonal blocks and are hence eliminated. The MATLAB implementation from [Hedengren, 2008] has manually eliminated all the algebraic variables except for those chosen as tearing variables by the JModelica.org compiler, with the slight difference that the compiler chooses the additional tearing variable L_{41}.

The Modelica implementation has been previously used for benchmarking purposes in [Magnusson and Åkesson, 2015; Lazutkin et al., 2015], albeit with a faulty implementation. The old implementation had incorrect versions of (5.13) and (5.15) involving $\dot{T}_i$ and $\dot{V}_i$, turning them into state variables.

5.6.2 Optimization Problem

We consider the short reflux breakdown scenario from [Diehl, 2002], where the reflux flow rate is reduced by nearly 90% for 5 minutes, causing the system to drift from the desired steady state. The objective is to steer back to the desired high-purity steady state after the breakdown.

The aim is to control the purities X_1 and X_{42} in the condenser and reboiler, respectively. This is best achieved by instead controlling the tray concentrations X_{14} and X_{28}, which are more sensitive to input disturbances. Since concentrations are difficult to measure, the temperatures T_{14} and T_{28} are instead controlled, which implicitly determine the corresponding concentrations.

The problem is solved over the fixed time horizon $t_f = 5000$ s, using quadratic Lagrange costs on the normalized deviation of the two tray temperatures and input signals from the high-purity steady state. The normalized state penalties are weighted by identity, and the inputs Q and L_{vol} by 0.02 and 0.015, respectively. There are nonnegativity bounds on the flux out of the condenser and reboiler, which are algebraic variables. These bounds put implicit upper limits on the two system inputs.

5.6.3 Solution

The problem is solved with $n^e = 20$ elements and $n^c = 3$ points per element. The initial guess is generated by simulating the system with constant input values equal to the reference values. The optimal solution is shown in Figure 5.15.

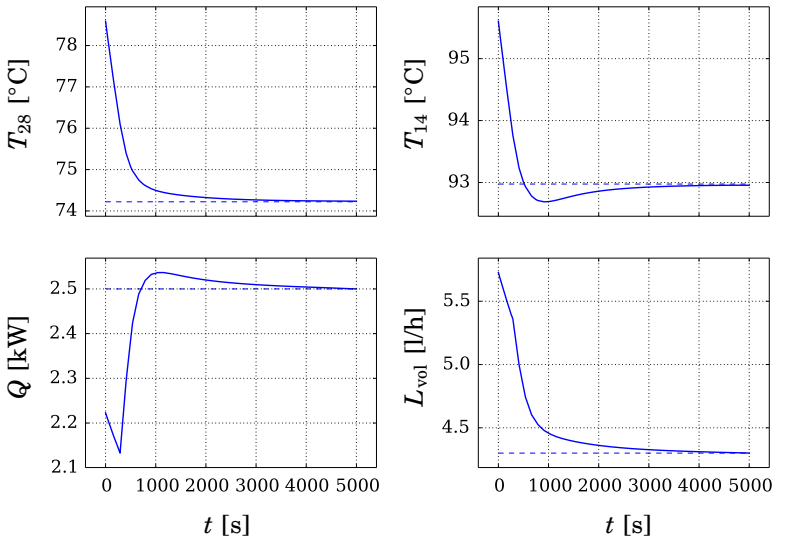

Figure 5.15 Optimal Distillation Column control to recover from a short reflux breakdown. The dashed lines correspond to the desired steady-state values, which are used to form the quadratic objective. The reflux flow L_{vol} is implicitly constrained roughly the first 300 seconds by the nonnegativity bound on the flow out of the condenser.

5.7 Conclusion

Five different optimal control problems have been presented, which we in the next chapter will use to evaluate the methods of Chapter 3 and Chapter 4. Two of the problems, Double Pendulum and Fourbar1, have been chosen to represent general-purpose Modelica models. The remaining three problems have been developed as part of research done by others for actual application.

The Modelica code for two of the problems, Car and Distillation Column, are atypical in their being flat, rather than hierarchically object-oriented, leading to them consisting of relatively few lines of code in their entirety and no trivial algebraic equations resulting from component connections. Hence, they will serve to demonstrate the effects of symbolic elimination on more typical representations of DAEs, rather than the verbose equations that result when using Modelica model libraries.

The remaining problem, CCPP, is constructed using an object-oriented Modelica model library developed specifically for optimization purposes.

The Modelica and Optimica implementation of all the problems are distributed with JModelica.org's example suite, as detailed in Table 5.1.

Table 5.1 Problems distributed with JModelica.org.

Problem	Main file
Car	`pyjmi.examples.vehicle_turn.py`
CCPP	`pyjmi.examples.ccpp.py`
Double Pendulum	`pyjmi.examples.double_pendulum.py`
Fourbar1	`pyjmi.examples.fourbar1.py`
Distillation Column	`pyjmi.examples.distillation4_opt.py`

5.8 Heat Recovery Steam Generator

Finally, we present one additional problem concerning the startup of a Heat Recovery Steam Generator (HRSG). The model has been developed and studied as part of industrial research projects and master's theses [Runvik, 2014; Thelander Andrén and Wedding, 2015; Åberg, 2016]. The model is proprietary, and consequently not a part of the publically available problem suite presented in this chapter. The problem is however an important use case and highly industrially relevant, so we will use this problem in the benchmark of Chapter 6.

5.8.1 Model

The system is similar to the CCPP of Section 5.3 and serves a similar purpose, but it has a few different components and is modeled in greater detail. It is also similar to the CCPP in the regard that the Modelica implementation is hierarchical and based on model components that have been developed specifically for optimization purposes.

The block diagram of the model is shown in Figure 5.16. Flue gas comes from the gas source, a boiler, with constant flow and a temperature determined by the firing power, whose derivative `uFP_der` is one of the system inputs. The flue gas is used to evaporate the liquid water. The vapor then goes through the first and second superheater, being further heated by the flue gas which goes through the superheaters in reverse order. After the second superheater, the vapor is collected in a header, where it is important to keep the thermal stress sufficiently low. The pressure and temperature in the header is controllable by the superheater valve input, whose derivative `uSH_der` is another system input.

If the vapor has sufficiently high pressure and temperature after being collected in the superheater header, it drives a turbine to generate electric-

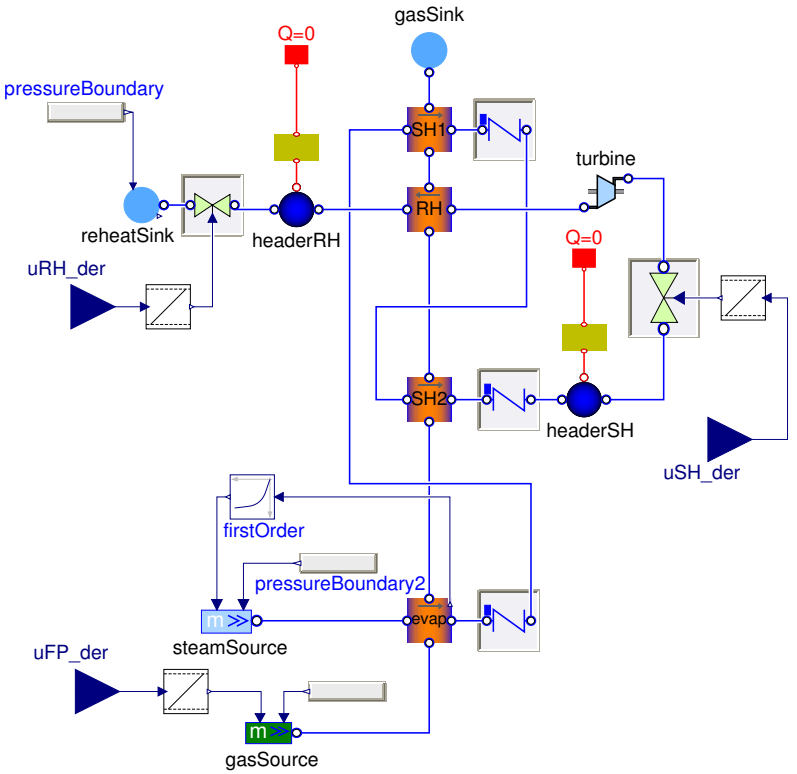

Figure 5.16 Dymola object diagram of the HRSG. Water is evaporated and superheated. If the vapor has sufficient temperature and pressure, it drives a turbine to generate electricity.

ity. After the turbine, the vapor is reheated and once again collected in a header whose thermal stress should be kept sufficiently low. At the end the water is recirculated to the evaporator after being condensed, here modeled by boundary conditions.

While the model is moderate in size, it contains high-fidelity modeling of thermodynamic properties of media—unlike the CCPP model, which uses low-order polynomials—causing high NLP function evaluation times. The standard way of modeling thermodynamic properties involves state events as the phase of the media changes, leading to nondifferentiable equation residuals, making such models unsuitable for the optimization methods of this thesis. This model instead uses high-order piecewise polynomials which

are twice continuously differentiable at the junction points, as described in [Åberg, 2016]. Unlike the CCPP model, the system model also takes into account the water pressure drop between the heat exchangers.

The resulting index-1 DAE has 18 states, 84 algebraic variables, and 3 inputs. Its BLT decomposition is shown in Figure 5.17.

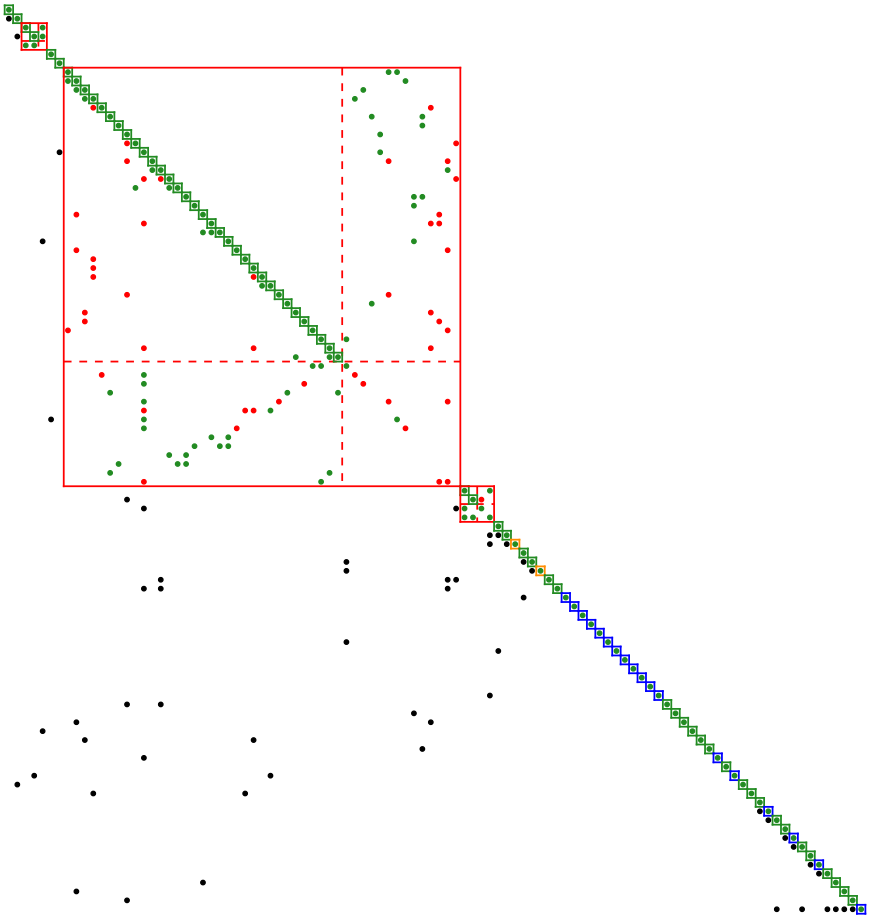

Figure 5.17 BLT decomposition of HRSG with SCHEME 4_{30}. Most of the diagonal blocks are scalar and linear, except for two small algebraic loops and a large one of dimension 47, which is torn with 14 tearing variables.

5.8.2 Optimization Problem

The goal is to control the system from a point where the boiler is running at low load to a point where the vapor pressure and temperature are sufficiently high to connect the turbines. The model is thus slightly artificial in its use of the turbine, as we connect the turbine already from the start. The model was developed to test the optimization-friendly high-fidelity media property models developed in [Åberg, 2016]. So the artificial use of the turbine was introduced to test additional media models, in particular the modeling of specific enthalpy and entropy.

The objective is a quadratic Lagrange function penalizing the deviation of the vapor pressures out of the the reheater and second superheater as well as the vapor temperature out of the second superheater from their respective reference values. The reference values correspond to when the turbines can be connected to generate electricity, at which point the next phase of the startup can begin. There are also quadratic penalties on the three system inputs (the derivatives of the firing power and the two header valve positions).

Besides penalties on the inputs, their absolute values are also bounded. Furthermore, the thermal stresses in the headers are constrained by bounding the temperature gradients in the spatially discretized header walls.

5.8.3 Solution

The problem is solved with $n^e = 25$ elements and $n^c = 5$ points per element. The initial guess is generated by simulating the system using a constant firing power derivative, constant superheater valve position, and a closed-loop integral controller for the reheater valve position to keep the reheater pressure at its reference. The optimal solution is shown in Figure 5.18.

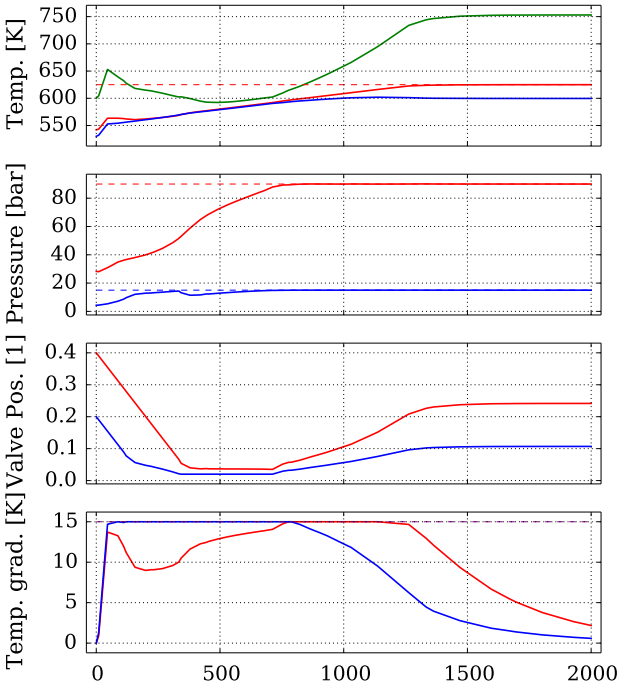

Figure 5.18 Optimal HRSG startup. The green curve is the boiler temperature. The red solid curves are the quantities in the second superheater and the blue solid curves are the quantities in the reheater. The dashed curves are the corresponding reference values used in the objective function, except for the temperature gradients, where the dashed curve is the upper bound for both heat exchanger headers.

6

Scheme Benchmarks

分かる必要はない、ただ知ればいい。
ーアンチスパイラル

In this chapter we evaluate the various proposed schemes of Chapter 4 when combined with the collocation algorithm of Chapter 3 on the problems from Chapter 5. We look at the average online solution time, offline pre-processing time, and convergence robustness. Since six problems is too small a test suite to draw any general conclusions regarding computational speed, and in particular convergence robustness, we will generate a larger number of problem instances by using randomized initial states. We will primarily focus on direct local collocation, but will in the end of this chapter also present a small benchmark on how the schemes perform when using direct global collocation.

Python scripts (which depend on JModelica.org) for reproducing the results of this chapter are available in [Magnusson, 2016].

6.1 Benchmark Setup

To generate a large test suite, we start with the solution to each of the six *nominal* problems described in Chapter 5. We then randomly perturb the initial state $\boldsymbol{x}_0$ and solve the perturbed problem using the nominal solution as initial guess. The purpose of this approach is to emulate the setting of Model Predictive Control (MPC), without actually doing MPC, which would have had the drawback of introducing correlation between the problems solved in each sample point. Hence, the information gained from solving a single DOP would be reduced, leading to a waste of computational effort.

The nominal value of each initial state variable $x_{0,i}$ is multiplicatively perturbed by independently, identically, normally distributed random variables with standard deviation σ to yield the new initial state variable $\bar{x}_{0,i}$;

that is

$$\bar{x}_{0,i} := \nu_i \cdot x_{0,i}, \quad \nu_i \sim \mathcal{N}(1, \sigma^2), \quad i \in [1..n_x]. \tag{6.1}$$

The standard deviation σ is handpicked for each problem to make the corresponding instances suitably difficult with high probability, although never higher than 0.3 to stay within the realm of reason. The resulting perturbed problem, henceforth referred to as an *instance*, may be infeasible. To counteract this, we make sure that the initial state satisfies all the problem constraints enforced at the initial time t_0. To satisfy the bounds on state variables, we project $\bar{x}_{0,i}$ inside of its bounds. We therefore modify (6.1) according to

$$\bar{x}_{0,i} = \max\Big(\min\big(\nu_i \cdot x_{0,i}, x_{0,i} + 0.9(x_{U,i} - x_{0,i})\big), x_{0,i} - 0.9(x_{0,i} - x_{L,i})\Big), \tag{6.2}$$

where $x_{U,i}$ is the element of the bounds $\boldsymbol{z}_U$ that corresponds to x_i. The factor 0.9 in (6.2) is used instead of 1.0 to project strictly inside the interior of the bounds, which is needed for (3.15b) to satisfy the linear independence constraint qualification (see Section 2.4.2). Projecting inside the feasible region of general path inequality constraints and algebraic variable bounds is more difficult. To satisfy these, we use JModelica.org's FMI-based DAE initialization algorithm to compute the value of $\bar{\boldsymbol{y}}_0$ that corresponds to $\bar{\boldsymbol{x}}_0$ and then check if they satisfy the constraints with a safety margin analogous to the one in (6.2); that is, the projected distance to the boundary should decrease by no more than 90%. If they do not, we discard the problem instance and generate a new one and repeat.

We solve each instance with all schemes and use the values $\{5, 10, 20, 30, 40\}$ for μ_{tol}. The problem sizes are detailed in Tables 6.1 and 6.2. Each scheme is only allowed a lax CPU time $t_{\max}$ [s] for each problem instance, after which we regard the scheme as having failed. The time $t_{\max}$ has been chosen as approximately the average observed solution time of the slowest scheme for a given problem plus 5 estimated standard deviations. For some problems, some schemes yield identical results, as indicated in the table. For example, Car has no algebraic loops, and so Scheme 1 and Scheme 2 are the same for this problem.

All problems—except Distillation Column—are solved with JModelica.org revision [8915], IPOPT 3.12.5, and the linear solver MA57 [HSL, 2016] with ordering by MeTiS [Karypis and Kumar, 1998] using a common, contemporary desktop computer. Distillation Column is instead solved with JModelica.org revision [9153] and IPOPT 3.12.6, in order to use the correct model implementation discussed in Section 5.6.1. The acceptable NLP tolerance in IPOPT is set equal to the NLP tolerance of 10^{-8}. To put a focus on convergence robustness rather than speed, the automatic scaling of MA57 is enabled and the pivot tolerance (see (4.19)) is increased from 10^{-8} to 10^{-4}.

Table 6.1 The first four benchmark problems and the considered schemes for each problem, where $t_{\max}$ [s] is the allotted CPU time for each instance, n_y the number of noneliminated algebraic variables, n the number of NLP variables, $\operatorname{nnz} J$ the number of nonzero elements in the NLP constraint Jacobian, $\operatorname{nnz} H$ the number of nonzero elements in the Hessian of the NLP Lagrangian. The details of the two remaining benchmark problems are shown in Table 6.2.

Problem	σ	$t_{\max}$	Schemes	n_y	$\frac{n}{1000}$	$\frac{\operatorname{nnz} J}{1000}$	$\frac{\operatorname{nnz} H}{1000}$
Car	0.1	30	0	23	9.7	37	9
			$1, 2, 3_{20}, 3_{30}, 3_{40}, 4_{20}, 4_{30}, 4_{40}$	4	6.3	30	10
			$3_5, 3_{10}, 4_5, 4_{10}$	5	6.4	30	9
CCPP	0.3	40	0	123	23.6	73	11.6
			$1, 3_{10}, 3_{20}, 3_{30}, 3_{40}$	3	4.3	19	4
			$2, 4_{10}, 4_{20}, 4_{30}, 4_{40}$	2	4.1	19	4
			3_5	6	4.7	21	4
			4_5	5	4.6	20	4
Dbl. Pend.	0.3	50	0	124	40.4	123	24
			$1, 3_{10}, 3_{20}, 3_{30}, 3_{40}$	16	7.9	25	5
			$2, 4_{20}, 4_{30}, 3_{40}$	2	3.7	13	5
			3_5	17	8.2	27	5
			4_{10}	3	4.0	15	5
			4_5	5	4.6	17	5
Fourbar1	0.03	30	0	452	82.8	334	192
			$1, 3_{30}, 3_{40}$	246	45.5	215	160
			2	23	5.2	50	24
			3_{20}	247	45.7	216	161
			3_{10}	249	46.1	217	161
			3_5	255	47.2	220	162
			4_{40}	29	6.3	63	29
			4_{30}	30	6.5	63	32
			4_{20}	46	9.3	88	58
			4_{10}	85	16.4	127	107
			4_5	114	21.7	142	128

The IPOPT barrier parameter selection strategy is changed to adaptive (see Section 2.4.4) to avoid the issue of selecting the initial value of the barrier parameter and also because of the improved performance perceived by the author. Other than the above exceptions, all the default values of the options of all the algorithms are used.

Table 6.2 The last two benchmark problems and the considered schemes for each problem. The notation is the same as in Table 6.2, where the details of the first four benchmark problems are shown.

Problem	σ	$t_{\max}$	Schemes	n_y	$\frac{n}{1000}$	$\frac{\text{nnz}\, J}{1000}$	$\frac{\text{nnz}\, H}{1000}$
Dist. Col.	0.3	40	0	1083	72.1	258	96
			1	291	23.8	223	153
			2	85	11.2	269	238
			3_{40}	294	24.0	148	75
			3_{30}	296	24.1	143	68
			3_{20}	298	24.2	136	62
			3_{10}	305	24.6	132	57
			3_5	331	26.2	130	53
			4_{40}	90	11.5	128	83
			4_{30}	93	11.7	114	66
			4_{20}	97	12.0	102	55
			4_{10}	108	12.6	69	38
			4_5	138	14.4	86	33
HRSG	0.3	60	0	84	15.9	56	6
			1, 3_5, 3_{10}, 3_{20}, 3_{30}, 3_{40}	56	12.4	47	6
			2, 4_{30}, 4_{40}	19	7.7	42	7
			4_{20}	20	7.9	41	6
			4_{10}	22	8.1	39	5
			4_5	23	8.2	39	5

6.2 Problem Results

This section presents the results of the benchmark described in Section 6.1. We generate 1000 instances whose initial states are in the interior of the constraints for each problem. We will use the same table headers throughout this section, with the following meanings. A scheme is considered to have *succeeded* on a problem instance if it converges to within tolerance of a local solution within the maximum CPU time. A problem instance is considered *valid* if at least one scheme succeeds on it. Success [1] is the ratio of success of a scheme on the valid instances. For the instances on which all schemes succeed, Time [s] is the average solution CPU time, σ_t [s] is the sample standard deviation of the solution time, and Iter [1] is the average number of iterations needed by a scheme.

6.2.1 Car

The results for the valid instances of Car are shown in Table 6.3. On 68.0% of the instances, all schemes succeed. 18.6% of the instances are invalid. The most significant difference between the schemes is that SCHEME 0 is approximately 35% slower than the others and slightly less robust. The dominant reason for failure is maximum CPU time, with some cases of convergence to a point of local infeasibility (meaning that there is no search direction decreasing the primal infeasibility). A likely reason for the high percentage of invalid instances is that the nominal solution is close to the boundaries of feasibility, because of the high initial car velocity. The perturbed problems can hence end up being infeasible with significant probability, which is also why we chose the relatively small value of $\sigma = 0.1$.

Table 6.3 Scheme performances on Car.

Schemes	Success	Time	σ_t	Iter
0	89.8%	8.9	4.5	104.0
1, 2, 3_{20}, 3_{30}, 3_{40}, 4_{20}, 4_{30}, 4_{40}	97.2%	6.8	4.9	94.3
3_5, 3_{10}, 4_5, 4_{10}	95.2%	6.4	5.0	103.9

6.2.2 Combined-Cycle Power Plant

The results for CCPP are shown in Table 6.4. On 70.1% of the instances, all schemes succeed. 20.9% of the instances are invalid. We see that SCHEME 0 is an order of magnitude slower and also sometimes fails unlike the other schemes. There is little difference between the other schemes, which is to be expected, since tearing and sparsity preservation only makes minor changes to the problem as seen in Table 6.1. On almost all of the instances in which all schemes fail, all schemes except 0 report local infeasibility. SCHEME 0 only fails because of maximum CPU time.

Table 6.4 Scheme performances on CCPP.

Schemes	Success	Time	σ_t	Iter
0	88.6%	13.6	5.3	101.9
1, 3_{10}, 3_{20}, 3_{30}, 3_{40}	100.0%	0.9	0.6	46.3
2, 4_{10}, 4_{20}, 4_{30}, 4_{40}	100.0%	0.9	0.5	46.4
3_5	100.0%	1.0	0.7	46.4
4_5	99.9%	1.0	0.6	46.5

6.2.3 Double Pendulum

The results for Double Pendulum are shown in Table 6.5. On 99.6% of the instances, all schemes succeed. All instances are valid. We see that Scheme 1 reduces the solution time by an order of magnitude compared to Scheme 0, which is further halved by Scheme 2. The overzealous sparsity preservation that is needed to have an impact on the number of variables is not beneficial.

Table 6.5 Scheme performances on Double Pendulum.

Schemes	Success	Time	σ_t	Iter
0	99.6%	15.5	7.0	99.9
1, 3_{10}, 3_{20}, 3_{30}, 3_{40}	100.0%	2.1	1.0	72.4
2, 4_{20}, 4_{30}, 3_{40}	100.0%	0.8	0.4	58.3
3_5	100.0%	2.2	1.0	71.1
4_{10}	100.0%	0.9	0.4	58.7
4_5	100.0%	1.0	0.5	58.9

6.2.4 Fourbar1

The results for Fourbar1 are shown in Table 6.6. On 56.1% of the instances, all schemes succeed. 34.0% of the instances are invalid. The dominant reasons for failure are restoration failure in IPOPT and maximum CPU time. We see that Scheme 2 performs significantly better than all the other schemes, and that despite significant density in the problem after applying tearing, sparsity preservation actually does more harm than good. One might have suspected this already when inspecting Table 6.1, as Fourbar1 is the only problem where sparsity preservation significantly increases $\text{nnz}\, J$ rather than decrease it. Also, the Lagrangian Hessian is relatively denser for this problem than the others, which we have neglected in the sparsity preservation.

6.2.5 Distillation Column

The state variable $\boldsymbol{x}$ for this problem consists entirely of molar fractions. Although $\boldsymbol{x}$ is formally unbounded, each component lies in the interval $[0, 1]$ by definition. For the purposes of (6.2), we hence impose the bounds $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$ to adhere to physics. These bounds are not enforced during the numerical solution of the optimization problem. Scheme 0 is unable to solve a single instance of Distillation Column. The results for the remaining schemes for Distillation Column are shown in Table 6.7.

On 97.2% of the instances, all remaining schemes succeed. All of the instances are valid. On the valid instances, the sole reason for failure is

Table 6.6 Scheme performances on Fourbar1.

Schemes	Success	Time	σ_t	Iter
0	86.8%	8.9	2.4	15.6
1, 3_{30},3_{40}	87.1%	4.6	1.3	15.4
2	99.8%	1.4	0.4	15.2
3_{20}	87.1%	4.6	1.3	15.4
3_{10}	87.1%	4.6	1.3	15.4
3_5	86.8%	5.0	1.4	15.4
4_{40}	93.5%	1.6	0.5	15.3
4_{30}	95.2%	1.5	0.4	15.2
4_{20}	90.8%	1.7	0.5	15.4
4_{10}	85.8%	2.3	0.7	15.4
4_5	95.6%	3.2	0.9	15.4

Table 6.7 Scheme performances on Distillation Column. SCHEME 0 fails all generated instances.

Schemes	Success	Time	σ_t	Iter
1	98.8%	10.2	1.5	16.3
2	99.9%	10.7	1.0	14.3
3_{40}	99.5%	4.3	1.2	16.5
3_{30}	99.4%	4.7	0.5	16.2
3_{20}	99.4%	6.0	1.0	16.3
3_{10}	99.3%	4.1	1.1	16.3
3_5	99.6%	4.8	0.6	15.8
4_{40}	100.0%	3.6	0.3	14.4
4_{30}	100.0%	3.0	0.3	14.5
4_{20}	100.0%	2.6	0.2	14.4
4_{10}	99.9%	2.2	0.2	14.4
4_5	100.0%	3.4	0.3	14.2

maximum CPU time. While tearing slightly improves robustness, it also slightly increases the solution time in the absence of sparsity preservation. Sparsity preservation however greatly reduces the solution time, especially in combination with tearing.

6.2.6 Heat Recovery Steam Generator

The results for HRSG are shown in Table 6.8. On 72.2% of the instances, all schemes succeed. 23.1% of the instances are invalid. The dominant reasons for failure are local infeasibility and maximum CPU time. We see that SCHEME 1 only offers a slight improvement over SCHEME 0, but the use of

tearing and sparsity preservation improves both robustness and solution times.

Table 6.8 Scheme performances on HRSG.

Schemes	Success	Time	σ_t	Iter
0	96.0%	13.5	8.2	66.0
1, 3_5, 3_{10}, 3_{20}, 3_{30}, 3_{40}	96.9%	11.1	7.2	68.8
2, 4_{30}, 4_{40}	99.3%	8.9	4.1	48.5
4_{20}	99.2%	7.7	4.4	49.8
4_{10}	99.1%	6.7	4.0	49.3
4_5	99.2%	6.8	4.2	49.7

6.3 Performance Profiles

In Section 6.2 we saw that Scheme 1 on average outperforms Scheme 0 for each considered problem, and that Schemes 2, $3_{\mu_{\text{tol}}}$, and $4_{\mu_{\text{tol}}}$ usually yield further improvements. To illustrate the aggregated results, Figure 6.1 shows the performance profile [Dolan and Moré, 2002] for all schemes on the valid portion of the 6000 problem instances. The performance $\rho_s(\tau)$ of a scheme s is defined as the ratio of instances in which s solves the problem no slower than a factor τ of the fastest scheme of that instance. In particular, $\rho_s(1)$ is the ratio of instances in which s was the fastest scheme, and $\rho_s(\infty)$ is the ratio of instances in which s succeeds.

There is however no clear best value for μ_{tol} in Figure 6.1, and average performance as a function of μ_{tol} is far from convex, suggesting that a larger suite than that of Chapter 5 is needed to pinpoint a suitable default value for μ_{tol}. Suitable values however seem to lie in the range of $[10, 30]$, and so 15 is chosen as the default value of JModelica.org.

For a clearer comparison of the various schemes without excessive focus on μ_{tol}, we generate a new performance profile where we only consider $\mu_{\text{tol}} = 30$, which was the density tolerance yielding the best robustness, that is, $\rho_s(\infty)$. The result is shown in Figure 6.2. Fourbar1 was the only problem where sparsity preservation significantly deteriorated robustness, which leads to Scheme 2 being the most robust scheme.

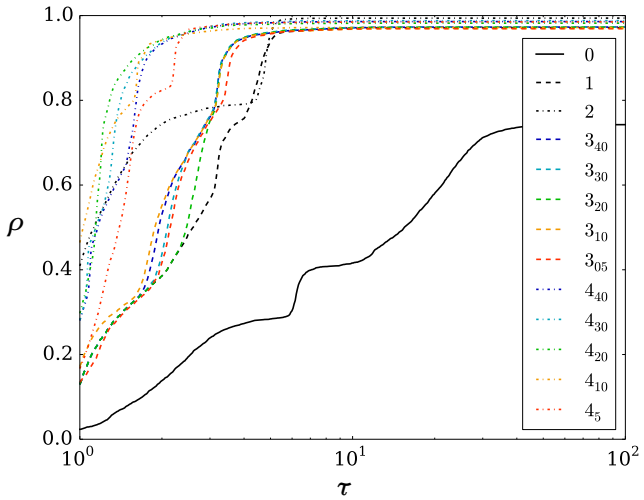

Figure 6.1 Performance profiles for the valid portion of 6000 problem instances of all considered schemes with 5 different values of μ_{tol}. Scheme 0 is outperformed by Scheme 1, which is outperformed by both Scheme 2 and Scheme $3_{\mu_{\text{tol}}}$, both of which are outperformed by Scheme $4_{\mu_{\text{tol}}}$.

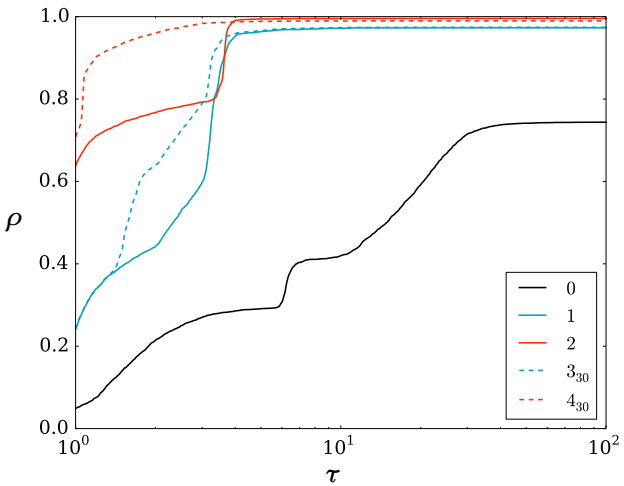

Figure 6.2 Performance profile for the valid portion of 6000 problem instances using only $\mu_{\text{tol}} = 30$. Scheme 4_{30} is on average superior both in terms of speed and robustness, although Scheme 2 has slightly better robustness. In particular, Scheme 4_{30} is an order of magnitude faster than Scheme 0 in approximately half of the instances.

In Chapter 4 we stated the goal of devising a scheme that always outperforms Scheme 0. To see whether we have succeeded, we generate a performance profile for only Scheme 0 and 4_{30}, the most promising scheme. The result is shown in Figure 6.3. Our goal translates to $\rho_{4_{30}} \equiv 1$ for this profile. Although we fall short of this goal, both in terms of speed and robustness, we are quite close.

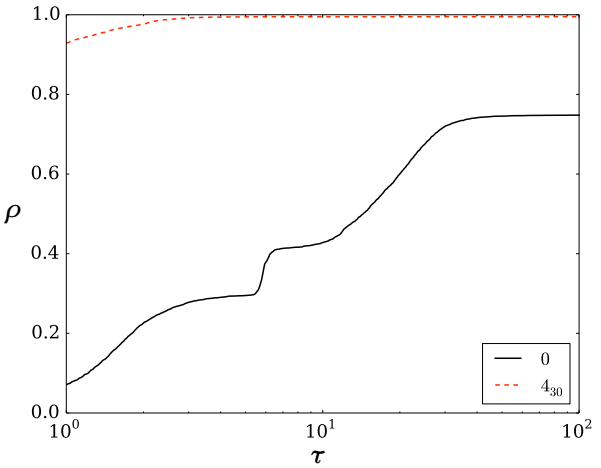

Figure 6.3 Performance profile of the conventional scheme and the most promising scheme.

6.4 Computation Times

Regarding computation times, we have so far focused on the respective solution times of the schemes. For many applications, such as online MPC, these are the only times that matter. For other applications, the computation time of the full toolchain is important, in which case the proposed techniques add additional computational steps during preprocessing compared to Scheme 0. Excepting Scheme 0, there is little difference in the offline computations times between the remaining schemes, and so this section focuses on the comparison of Scheme 0 and Scheme 4_{30}.

The main parts of the offline computation times are model compilation, BLT analysis, collocation discretization, and algorithmic differentiation graph construction (including first- and second-order derivatives). Another significant part of offline computation times is starting the java virtual machine of the compiler, which takes approximately 1 second. We however neglect this, since it only needs to be performed once per Python session and

is entirely independent of the considered optimization problem. Note that each of these offline steps only has to be performed once in for example the benchmark of Section 6.2 for each problem, rather than for each instance, since the only thing to change is numerical parameter values; the problem structure remains unaltered. The main parts of the online computation times are NLP function evaluations and KKT matrix factorization. While Scheme 4_{30} adds computation time compared to Scheme 0 in the form of BLT analysis, it also results in a smaller DAE which may reduce computation in subsequent offline steps, allowing the additional computation time to be regained before we even reach the online computations.

Table 6.9 compares the offline and online computation times of Scheme 0 and Scheme 4_{30} for the six benchmark problems. The online times are the averages observed in Section 6.2. We see that for all except one problem, Scheme 4_{30} has a lower total time. We also see that the offline times for Scheme 4_{30} become significantly larger for problems with many DAE variables. Since the implementation of Scheme 4_{30} is just a prototype with little effort spent on efficient offline computations, its implementation can probably be optimized to scale better. The bottleneck for Distillation Column lies in the construction of the structural incidence matrix: Identifying the nonzero incidences and determining which of those are linear, which affects all schemes except 0. These computations are not inherently expensive; the JModelica.org compiler performs very similar computations for larger systems much faster.

Table 6.9 Offline and online computation times [s].

Problem	Scheme	Offline	Online	Total
Car	0	2.7	8.9	11.6
	4_{30}	2.7	6.8	9.5
CCPP	0	3.5	13.6	17.1
	4_{30}	3.6	0.9	4.5
Double Pendulum	0	7.8	15.5	23.3
	4_{30}	7.1	0.8	7.9
Fourbar1	0	31.3	8.9	40.2
	4_{30}	43.4	1.5	44.9
Distillation Column	0	14.7	∞	∞
	4_{30}	119.2	3.0	122.2
HRSG	0	10.7	13.5	24.2
	4_{30}	12.5	8.9	21.4

6.5 Global Collocation

We have so far only evaluated the schemes when combined with local collocation. Global collocation methods, where a large number of collocation points and a small number of elements are used instead of vice versa, have become increasingly popular the last decade. Hence, we will also consider these methods, but only briefly to avoid the inadvertent comparison between the performances of the two flavors of collocation that a more thorough investigation would have entailed. One important difference between local and global collocation to keep in mind is that the NLPs resulting from global collocation tend to be smaller, owing to their potential spectral (exponential) convergence rate, but also more dense, because of their global temporal coupling.

We will limit our evaluation with global collocation to the Distillation Column problem. Instead of using $n^e = 20$ and $n^c = 3$, we use $n^e = 1$ and $n^c = 25$. Just as we observed for local collocation, Scheme 0 is unable to solve the problem. The results for the remaining schemes are shown in Table 6.10. All of the instances are valid. The results are similar to those obtained with local collocation, see Table 6.7. Two notable differences are that every scheme (except 0) solves every instance and that smaller values of μ_{tol} are preferable. This indicates that it is crucial to preserve what little sparsity there is in an NLP resulting from global collocation. However, if the problem dimensions allow it, it may be more efficient to combine Scheme 2 with dense rather than sparse numerical linear algebra, which is a possibility we do not consider further in this thesis.

Table 6.10 Scheme performances on Distillation Column with global collocation.

Scheme	Success	Time	σ_t	Iter
1	100.0%	9.4	1.4	9.1
2	100.0%	8.9	1.1	8.6
3_{40}	100.0%	3.1	0.5	9.1
3_{30}	100.0%	3.5	0.6	9.1
3_{20}	100.0%	2.6	0.4	9.1
3_{10}	100.0%	2.4	0.4	9.1
3_{5}	100.0%	2.7	0.4	9.1
4_{40}	100.0%	6.0	0.8	8.6
4_{30}	100.0%	3.6	0.5	8.6
4_{20}	100.0%	2.6	0.4	8.6
4_{10}	100.0%	3.8	0.5	8.6
4_{5}	100.0%	2.0	0.2	8.6

6.6 Conclusion

We evaluated the schemes of Chapter 4 on the problems of Chapter 5 using the collocation algorithm from Chapter 3. We found that the scheme that utilizes all of the proposed techniques performed the best on average, often being an order of magnitude faster than the conventional scheme of exposing the full DAE to the discretization method.

We found that a suitable value of the density tolerance μ_{tol} seems to be 15 when employing local collocation, although it is problem dependent and more data is needed to pinpoint a suitable default value. It also seems that smaller tolerances are preferable when there are more collocation points per element.

We observed that the techniques of Chapter 4 lend themselves well to DAEs resulting from the hierarchical modeling of Modelica, but that they also show potential for flat DAEs.

Sparsity preservation was beneficial for the problems where it made a significant difference, excepting Fourbar1. A possible and tractable refinement of the proposed sparsity preservation procedure that may remedy this is to not only consider the sparsity of the NLP Jacobian, but also the Hessian of the NLP Lagrangian.

While block-triangular ordering and sparsity preservation guarantees preservation of numerical stability, tearing does not, as discussed in Chapter 4. Although numerical instability caused by tearing does not appear to have been an issue for the considered problems, the author is confident that there are industrially relevant problems where JModelica.org selects numerically troublesome tearing variables and residuals.

7

Conclusion

This chapter concludes the thesis by a summary and directions for future research.

7.1 Summary

We have considered methods and software for numerical solution of DAE-constrained optimization problems. The presented software framework is integrated in a Modelica-based toolchain in the open-source platform JModelica.org. The framework implements a method using direct collocation and relies heavily upon the open-source third-party software packages CasADi for symbolic operations and algorithmic differentiation and IPOPT for solving the resulting nonlinear programs.

Models from typical Modelica libraries are often developed for general purposes and high-fidelity simulation, making them ill-suited for optimization purposes. Even if they are sufficiently differentiable—for example, do not involve hybrid dynamics—they sometimes model phenomena which may have no relevance for a particular application and also involve a multitude of algebraic equations that result from acausal, hierarchical modeling. Hence, the models may end up being overly complicated for the purposes of dynamic optimization. This causes the conventional approach of discretizing the full DAE using direct collocation to often lead to convergence failure, which is difficult to troubleshoot. This thesis proposed methods for overcoming this problem by eliminating algebraic variables using techniques based on block-triangular ordering, tearing, and sparsity preservation. Methods for automated initialization and scaling of the many variables based on dynamic simulation were also discussed. These help to streamline the workflow when dealing with large-scale, heterogeneous models.

A suite of computationally challenging optimal control problems was presented and used to evaluate the methods of the thesis. The proposed elimination methods were shown to improve both convergence robustness

and also average computational speed. In particular, the proposed methods are an order of magnitude faster than the conventional approach of discretizing the full DAE for approximately half of the considered problems. While the thesis focuses on DAEs described by Modelica code, it was demonstrated that the proposed methods are also beneficial when applied on conventional DAEs not originating from hierarchical modeling.

7.2 Directions for Future Research

There is always room for extensions of the framework for dynamic optimization of JModelica.org. Notable missing features are mesh refinement, support for multiphase and mixed-integer problems, and shooting-based algorithms. A particularly important research direction is the dynamic optimization of systems described by hybrid Modelica models. There are several possible approaches to treating hybrid dynamics in dynamic optimization, for example the use of complementarity constraints [Baumrucker and Biegler, 2009], mixed-integer nonlinear programming based on branch-and-bound [Belotti et al., 2009] or sum-up-rounding [Fouquet et al., 2016; Kirches and Lenders, 2016], multiple phases for predetermined switching sequences [Betts, 2010], and widely applicable derivative-free methods such as genetic algorithms [Thieriot et al., 2011]. The various approaches each have their respective strengths and weaknesses. Identifying suitable strategies for the diverse hybrid dynamics of Modelica should prove challenging.

The problem suite of this thesis would benefit from enlargement. However, constructing a suite for public use is not a one-man job; the perspectives of multiple experts of any gender are necessary. And before a coordinated effort can be made to establish a suite of dynamic optimization problems involving Modelica, the Modelica community needs to standardize the formulation of optimization problems. While there have been efforts in this direction, they have yet to see fruition.

One reason for enlarging the problem suite is that the proposed elimination techniques have, to some extent, been adapted to the considered problem suite. The performance profiles may tell a different story when applying the schemes on other problems. In particular, few measures are taken to prevent numerical instability caused by tearing or division by zero. Such measures however tend to be conservative, so there is a trade-off to be made between reliability and performance. Another possible improvement of the elimination techniques is to coordinate them, rather than applying them in a sequential manner.

Bibliography

Åberg, M. (2016). *Optimisation-Friendly Modelling of Thermodynamic Properties of Media*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Åkesson, J. (2007). *Languages and Tools for Optimization of Large-Scale Systems*. Ph.D. thesis. Department of Automatic Control, Lund University, Sweden.

Åkesson, J. (2008). "Optimica—an extension of Modelica supporting dynamic optimization". In: *Proceedings of the 6th International Modelica Conference*. Bielefeld, Germany, pp. 57–66.

Åkesson, J., K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit (2010a). "Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems". *Computers & Chemical Engineering* **34**, pp. 1737–1749.

Åkesson, J., T. Ekman, and G. Hedin (2010b). "Implementation of a Modelica compiler using JastAdd attribute grammars". *Science of Computer Programming* **75**, pp. 21–38.

Allgöwer, F., T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright (1999). "Nonlinear predictive control and moving horizon estimation—an introductory overview". In: Frank, P. M. (Ed.). *Advances in Control: Highlights of ECC '99*. Springer, Berlin, Germany, pp. 391–449.

Andersson, C., J. Åkesson, and C. Führer (2016). *PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*. Tech. rep. LUTFNA-5008-2016. Centre for Mathematical Sciences, Lund University, Sweden.

Andersson, C., C. Führer, and J. Åkesson (2015). "Assimulo: A unified framework for ODE solvers". *Mathematics and Computers in Simulation* **116**, pp. 26–43.

Andersson, J. (2013). *A General-Purpose Software Framework for Dynamic Optimization*. Ph.D. thesis. Arenberg Doctoral School, KU Leuven, Belgium.

Andersson, J., J. Åkesson, F. Casella, and M. Diehl (2011). "Integration of CasADi and JModelica.org". In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, pp. 218–231.

Axehill, D. (2015). "Controlling the level of sparsity in mpc". *Systems & Control Letters* **76**, pp. 1–7.

Axelsson, M., F. Magnusson, and T. Henningsson (2015). "A framework for nonlinear model predictive control in JModelica.org". In: *Proceedings of the 11th International Modelica Conference*. Paris, France, pp. 301–310.

Bachmann, B., L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson (2012). "Parallel multiple-shooting and collocation optimization with OpenModelica". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 659–668.

Baharev, A., H. Schichl, and A. Neumaier (2016a). "Decomposition methods for solving sparse nonlinear systems of equations". Submitted for publication. Available online: `http://reliablecomputing.eu/baharev_tearing_survey.pdf`.

Baharev, A., H. Schichl, and A. Neumaier (2016b). "Ordering matrices to bordered lower triangular form with minimal border width". Submitted for publication. Available online: `http://reliablecomputing.eu/baharev_tearing_exact_algorithm.pdf`.

Baharev, A., H. Schichl, A. Neumaier, and T. Achterberg (2016c). "An exact method for the minimum feedback arc set problem". Submitted for publication. Available online: `http://reliablecomputing.eu/baharev_minimum_feedback_arc_set.pdf`.

Baltes, M., R. Schneider, C. Sturm, and M. Reuss (1994). "Optimal experimental design for parameter estimation in unstructured growth models". *Biotechnology Progress* **10**:5, pp. 480–488.

Baumgarte, J. (1972). "Stabilization of constraints and integrals of motion in dynamical systems". *Computer Methods in Applied Mechanics and Engineering* **1**:1, pp. 1–16.

Baumrucker, B. T. and L. T. Biegler (2009). "MPEC strategies for optimization of a class of hybrid dynamic systems". *Journal of Process Control* **19**:8, pp. 1248–1256.

Bausa, J. and G. Tsatsaronis (2001). "Dynamic optimization of startup and load-increasing processes in power plants—Part I: Method". *Journal of Engineering for Gas Turbines and Power* **123**:1, pp. 246–250.

Becerra, V. M. (2010). “Solving complex optimal control problems at no cost with PSOPT”. In: *2010 IEEE International Symposium on Computer-Aided Control System Design*. Yokohama, Japan, pp. 1391–1396.

Belkhir, F., D. K. Cabo, F. Feigner, and G. Frey (2015). “Optimal startup control of a steam power plant using the JModelica platform”. In: *Proceedings of the 8th Vienna International Conference on Mathematical Modelling*. Vienna, Austria, pp. 204–209.

Belotti, P., J. Lee, L. Liberti, F. Margot, and A. Wächter (2009). “Branching and bounds tighteningtechniques for non-convex MINLP”. *Optimization Methods & Software* **24**:4–5, pp. 597–634.

Berntorp, K. and F. Magnusson (2015). “Hierarchical predictive control for ground-vehicle maneuvering”. In: *2015 American Control Conference*. Chicago, IL, pp. 2771–2776.

Berntorp, K., B. Olofsson, K. Lundahl, and L. Nielsen (2014). “Models and methodology for optimal trajectory generation in safety-critical road–vehicle manoeuvres”. *Vehicle System Dynamics* **52**:10, pp. 1304–1332.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*. 3rd edition. Vol. 1. Athena Scientific, Belmont, MA.

Betts, J. T. (1998). “Survey of numerical methods for trajectory optimization”. *Journal of Guidance, Control, and Dynamics* **21**, pp. 193–207.

Betts, J. T. (2010). *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. 2nd edition. SIAM, Philadelphia, PA.

Betts, J. T. and W. P. Huffman (1998). “Mesh refinement in direct transcription methods for optimal control”. *Optimal Control Applications and Methods* **19**, pp. 1–21.

Biegler, L. T. (2007). “An overview of simultaneous strategies for dynamic optimization”. *Chemical Engineering and Processing: Process Intensification* **46**:11, pp. 1043–1053.

Biegler, L. T. (2010). *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM, Philadelphia, PA.

Blochwitz, T., M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel (2012). “Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models”. In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 173–184.

Blochwitz, T., M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf (2011). “The Functional Mockup Interface for tool independent exchange of simulation models”. In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, pp. 105–114.

Brenan, K., S. Campbell, and L. Petzold (1996). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Vol. 14. Classics in Applied Mathematics. SIAM, Philadelphia, PA.

Brooke, A., D. A. Kendrick, A. Meeraus, and R. E. Rosenthal (1988). *GAMS: A User's Guide*. Scientific Press, Redwood City, CA.

Büskens, C. and D. Wassel (2013). "The ESA NLP solver WORHP". In: Fasano, G. et al. (Eds.). *Modeling and Optimization in Space Engineering*. Vol. 73. Springer, New York, NY, pp. 85–110.

Campbell, S. L. (1985). "The numerical solution of higher index linear time varying singular systems of differential equations". *SIAM Journal on Scientific and Statistical Computing* **6**:2, pp. 334–348.

Campbell, S. L. and J. T. Betts (2016). "Comments on direct transcription solution of DAE constrained optimal control problems with two discretization approaches". *Numerical Algorithms*. In press. Available online: http://dx.doi.org/10.1007/s11075-016-0119-6.

Campbell, S. L. and C. W. Gear (1995). "The index of general nonlinear DAEs". *Numerische Mathematik* **72**:2, pp. 173–196.

Casella, F., F. Donida, and J. Åkesson (2011). "Object-oriented modeling and optimal control: A case study in power plant start-up". In: *Proceedings of the 18th IFAC World Congress*. Milano, Italy, pp. 9549–9554.

Cellier, F. E. and E. Kofman (2006). *Continuous System Simulation*. Springer, New York, NY.

Cervantes, A. M., A. Wächter, R. H. Tütüncü, and L. T. Biegler (2000). "A reduced space interior point strategy for optimization of differential algebraic systems". *Computers & Chemical Engineering* **24**:1, pp. 39–51.

Dassault Systèmes (2016). *Dymola*. URL: `http://www.dymola.com` (visited on 2016-08-15).

Diehl, M. (2002). *Real-Time Optimization for Large Scale Nonlinear Processes*. Ph.D. thesis. Heidelberg University, Germany.

Diehl, M., H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer (2002). "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations". *Journal of Process Control* **12**:4, pp. 577–585.

Dolan, E. D. and J. J. Moré (2002). "Benchmarking optimization software with performance profiles". *Mathematical Programming* **91**:2, pp. 201–213.

Duff, I. S., A. Erisman, and J. K. Reid (1986). *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, United Kingdom.

Duff, I. S. and J. K. Reid (1996). "The design of MA48: A code for the direct solution of sparse unsymmetric linear systems of equations". *ACM Transactions on Mathematical Software* **22**:2, pp. 187–226.

Ekman, T. and G. Hedin (2007). "The JastAdd system—modular extensible compiler construction". *Science of Computer Programming* **69**, pp. 14–26.

Ekström, S. (2015). *Real Time Model Predictive Control in JModelica.org*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Elmqvist, H. (1978). *A structured model language for large continuous systems*. Ph.D. thesis. Department of Automatic Control, Lund Institute of Technology, Sweden.

Elmqvist, H. and M. Otter (1994). "Methods for tearing systems of equations in object-oriented modeling". In: *Proceedings of the 1994 European Simulation Multiconference*. Barcelona, Spain, pp. 326–332.

Elsheikh, A. and W. Wiechert (2008). "Automatic sensitivity analysis of DAE-systems generated from equation-based modeling languages". In: Bischof, C. H. et al. (Eds.). *Advances in Automatic Differentiation*. Springer-Verlag, Berlin Heidelberg, Germany, pp. 235–246.

Fletcher, R. (1998). "Block triangular orderings and factors for sparse matrices in LP". In: Griffiths, D. et al. (Eds.). *Numerical Analysis 1997*. Longman, Harlow, United Kingdom, pp. 91–110.

Fouquet, M., F. Magnusson, H. Guéguen, S. Velut, D. Faille, D. Dumur, and T. Henningsson (2016). "Hybrid dynamic optimization of power plants based on physical models and the collocation method". *Journal of Process Control*. Accepted subject to major revision.

Fourer, R., D. M. Gay, and B. W. Kernighan (2003). *AMPL: A Modeling Language for Mathematical Programming*. Thomson Brooks/Cole, Pacific Grove, CA.

Franke, R. (2002). "Formulation of dynamic optimization problems using Modelica and their efficient solution". In: *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, pp. 315–322.

Franke, R., M. Walther, N. Worschech, W. Braun, and B. Bachmann (2015). "Model-based control with FMI and a C++ runtime for Modelica". In: *Proceedings of the 11th International Modelica Conference*. Paris, France, pp. 339–347.

Fritzson, P. (2015). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3*. 2nd edition. Wiley-IEEE Press, Piscataway, NJ.

Garg, D., M. Patterson, W. W. Hager, A. V. Rao, D. A. Benson, and G. T. Huntington (2010). "A unified framework for the numerical solution of

optimal control problems using pseudospectral methods". *Automatica* **46**:11, pp. 1843–1851.

George, A. (1973). "Nested dissection of a regular finite element mesh". *SIAM Journal on Numerical Analysis* **10**:2, pp. 345–363.

Ghazaei Ardakani, M. M. and F. Magnusson (2016). "Ball and finger system: modeling and optimal trajectories". *Robotics and Autonomous Systems*. To be submitted.

Gill, P. E., W. Murray, and M. H. Wright (1981). *Practical Optimization*. Academic Press.

Giselsson, P., J. Åkesson, and A. Robersson (2009). "Optimization of a pendulum system using Optimica and Modelica". In: *Proceedings of the 7th International Modelica Conference*. Como, Italy, pp. 480–489.

Gould, N. I., D. Orban, and P. L. Toint (2003). "CUTEr and SifDec: A constrained and unconstrained testing environment, revisited". *ACM Transactions on Mathematical Software* **29**:4, pp. 373–394.

Griewank, A. and A. Walther (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd edition. SIAM, Philadelphia, PA.

Gupta, P. K., A. W. Westerberg, J. E. Hendry, and R. R. Hughes (1974). "Assigning output variables to equations using linear programming". *AIChE Journal* **20**:2, pp. 397–399.

Hairer, E. and G. Wanner (1996). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2nd edition. Springer-Verlag, Berlin, Germany.

Hart, W. E., C. Laird, J.-P. Watson, and D. L. Woodruff (2012). *Pyomo – Optimization Modeling in Python*. Springer, Berlin, Germany.

Hedengren, J. D. (2008). "A nonlinear model library for dynamics and control". URL: `http://www.hedengren.net/research/Publications/Cache_2008/NonlinearModelLibrary.pdf` (visited on 2016-08-15).

Hedengren, J. D., R. A. Shishavan, K. M. Powell, and T. F. Edgar (2014). "Nonlinear modeling, estimation and predictive control in APMonitor". *Computers & Chemical Engineering* **70**, pp. 133–148.

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". *ACM Transactions on Mathematical Software* **31**:3, pp. 363–396.

Holmqvist, A. and F. Magnusson (2016). "Open-loop optimal control of batch chromatographic separation processes using direct collocation". *Journal of Process Control* **46**, pp. 55–74.

Holmström, K. (1999). "The TOMLAB optimization environment in Matlab". *Advanced Modeling and Optimization* **1**, pp. 47–69.

Hopcroft, J. E. and R. M. Karp (1973). "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs". *SIAM Journal on Computing* **2**:4, pp. 225–231.

Houska, B., H. J. Ferreau, and M. Diehl (2011). "ACADO toolkit—An open-source framework for automatic control and dynamic optimization". *Optimal Control Applications and Methods* **32**, pp. 298–312.

HSL (2016). *A collection of Fortran codes for large scale scientific computation*. Software available at `http://www.hsl.rl.ac.uk`.

Hunter, J. D. (2007). "Matplotlib: a 2D graphics environment". *Computing in Science & Engineering* **9**:3, pp. 90–95.

ITEA (2015). *Model driven physical systems operation*. URL: `https://itea3.org/project/modrio.html` (visited on 2015-06-16).

Kameswaran, S. and L. T. Biegler (2008). "Convergence rates for direct transcription of optimal control problems using collocation at radau points". *Computational Optimization and Applications* **41**:1, pp. 81–126.

Karypis, G. and V. Kumar (1998). "A fast and high quality multilevel scheme for partitioning irregular graphs". *SIAM Journal on Scientific Computing* **20**:1, pp. 359–392.

Kirches, C. and F. Lenders (2016). "Approximation properties and tight bounds for constrained mixed-integer optimal control". *Mathematical Programming*. Submitted for publication. Available online: `http://www.optimization-online.org/DB_FILE/2016/04/5404.pdf`.

Klenk, M., D. G. Bobrow, J. De Kleer, and B. Janssen (2014). "Making Modelica applicable for formal methods". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 205–211.

Kron, G. (1963). *Diakoptics: The Piecewise Solution of Large-Scale Systems*. Vol. 2. MacDonald, London, United Kingdom.

Krüger, K., R. Franke, and M. Rode (2004). "Optimization of boiler start-up using a nonlinear boiler model and hard constraints". *Energy* **29**:12, pp. 2239–2251.

Kunkel, P. and V. Mehrmann (2008). "Optimal control for unstructured nonlinear differential-algebraic equations of arbitrary index". *Mathematics of Control, Signals, and Systems* **20**:3, pp. 227–269.

Larsson, P.-O. (2011). *Optimization of low-level controllers and high-level polymer grade changes*. Ph.D. thesis. Department of Automatic Control, Lund University, Sweden.

Larsson, P.-O., F. Casella, F. Magnusson, J. Andersson, M. Diehl, and J. Åkesson (2013). "A framework for nonlinear model-predictive control using object-oriented modeling with a case study in power plant start-up". In: *Proceedings of the 2013 IEEE Multi-Conference on Systems and Control*. Hyderabad, India, pp. 346–351.

Larsson, T. (2015). *Moving Horizon Estimation for JModelica.org*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Lazutkin, E., A. Geletu, S. Hopfgarten, and P. Li (2014). "Modified multiple shooting combined with collocation method in JModelica.org with symbolic calculations". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 999–1006.

Lazutkin, E., A. Geletu, S. Hopfgarten, and P. Li (2015). "An analytical hessian and parallel-computing approach for efficient dynamic optimization based on control-variable correlation analysis". *Industrial & Engineering Chemistry Research* **54**:48, pp. 12086–12095.

Lennernäs, B. (2013). *A CasADi Based Toolchain for JModelica.org*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Liberzon, D. (2012). *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, Princeton, NJ.

Maciejowski, J. M. (2002). *Predictive Control with Constraints*. Prentice-Hall, Upper Saddle River, NJ.

Magnusson, F. (2012). *Collocation Methods in JModelica.org*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Magnusson, F. (2016). *Symbolic elimination performance profiles*. GitHub repository. URL: `https://github.com/Bleevoe/blt_performance_profiles` (visited on 2016-10-05).

Magnusson, F. and J. Åkesson (2012). "Collocation methods for optimization in a Modelica environment". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 649–658.

Magnusson, F. and J. Åkesson (2015). "Dynamic optimization in JModelica.org". *Processes* **3**:2, pp. 471–496.

Magnusson, F. and J. Åkesson (2016). "Symbolic elimination in dynamic optimization based on block-triangular ordering". *Optimization Methods and Software*. Submitted for publication.

Magnusson, F., K. Palmer, L. Han, and G. Bollas (2015). "Dynamic parametric sensitivity optimization using simultaneous discretization in JModelica.org". In: *2015 International Conference on Complex Systems Engineering*, pp. 37–42.

Maree, J. P. and L. Imsland (2016). "Combined economic and regulatory predictive control". *Automatica* **69**, pp. 342–347.

Markowitz, H. M. (1957). "The elimination form of the inverse and its application to linear programming". *Management Science* **3**:3, pp. 255–269.

MathWorks (2016). *Simulink*. URL: `http://mathworks.com/products/simulink` (visited on 2016-09-15).

Mattsson, S. E., H. Elmqvist, and M. Otter (1998). "Physical system modeling with Modelica". *Control Engineering Practice* **6**:4, pp. 501–510.

Mattsson, S. E., H. Olsson, and H. Elmqvist (2000). "Dynamic selection of states in Dymola". In: *Modelica Workshop 2000 Proceedings*. Lund, Sweden, pp. 61–67.

Mattsson, S. E. and G. Söderlind (1993). "Index reduction in differential-algebraic equations using dummy derivatives". *SIAM Journal on Scientific Computing* **14**:3, pp. 677–692.

Meijer, P. (2011). *Tearing Differential Algebraic Equations*. M.Sc. thesis. Centre for Mathematical Sciences, Lund University, Sweden.

Modelica Association (2016a). *List of Modelica libraries*. URL: `https://www.modelica.org/libraries` (visited on 2016-08-31).

Modelica Association (2016b). *Modelica home page*. URL: `http://www.modelica.org` (visited on 2016-08-01).

Modelon AB (2016). *JModelica.org User Guide*. URL: `http://www.jmodelica.org/page/236` (visited on 2016-08-30).

Nagel, L. W. and D. Pederson (1973). *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Tech. rep. UCB/ERL M382. EECS Department, University of California, Berkeley, CA.

Nagy, Z., R. Findeisen, M. Diehl, F. Allgöwer, H. G. Bock, S. Agachi, J. P. Schlöder, and D. Leineweber (2000). "Real-time feasibility of nonlinear predictive control for large scale processes - a case study". In: *2000 American Control Conference*. Chicago, IL, pp. 4249–4253.

Nocedal, J., A. Wächter, and R. A. Waltz (2009). "Adaptive barrier update strategies for nonlinear interior methods". *SIAM Journal on Optimization* **19**:4, pp. 1674–1693.

Nocedal, J. and S. Wright (2006). *Numerical optimization*. 2nd edition. Springer, New York, NY.

Norén, C. (2013). *Path Planning for Autonomous Heavy Duty Vehicles Using Nonlinear Model Predictive Control*. M.Sc. thesis. Department of Electrical Engineering, Automatic Control, Linköping University, Sweden.

Pacejka, H. B. (2006). *Tire and Vehicle Dynamics*. 2nd edition. Butterworth-Heinemann, Oxford, United Kingdom.

Palmkvist, E. (2014). *Implementation of Grey-Box Identification in JModelica.org*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Pantelides, C. C. (1988). "The consistent initialization of differential-algebraic systems". *SIAM Journal on Scientific and Statistical Computing* **9**:2, pp. 213–231.

Parini, P. (2015). *Object Oriented Modeling and Dynamic optimization of Energy Systems with Application to Combined Cycle Power Plant Start-Up*. M.Sc. thesis. Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy.

Parrotto, R., J. Åkesson, and F. Casella (2010). "An XML representation of DAE systems obtained from continuous-time Modelica models". In: *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Oslo, Norway, pp. 91–98.

Pettersson, J., U. Persson, T. Lindberg, L. Ledung, and X. Zhang (2005). "On-line pulp mill production optimization". In: *Proceedings of the 16th IFAC World Congress*. Prague, Czech Republic, pp. 443–448.

Pfeiffer, A. (2012). "Optimization library for interactive multi-criteria optimization tasks". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 669–680.

Pfeiffer, A., I. Bausch-Gall, and M. Otter (2012). "Proposal for a standard time series file format in HDF5". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 495–506.

Piela, P. C., T. Epperly, K. Westerberg, and A. W. Westerberg (1991). "ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language". *Computers & Chemical Engineering* **15**:1, pp. 53–72.

Prata, A., J. Oldenburg, A. Kroll, and W. Marquardt (2008). "Integrated scheduling and dynamic optimization of grade transitions for a continuous polymerization reactor". *Computers & Chemical Engineering* **32**, pp. 463–476.

Process Systems Enterprise (2016). *gPROMS*. URL: `https://www.psenterprise.com/products/gproms` (visited on 2016-08-15).

Pytlak, R., T. Tarnawski, B. Fajdek, and M. Stachura (2014). "Interactive dynamic optimization server – connecting one modelling language with many solvers". *Optimization Methods and Software* **29**, pp. 1118–1138.

Rajamani, R. (2006). *Vehicle Dynamics and Control*. Springer-Verlag, Berlin Heidelberg, Germany.

Rao, A. V. (2009). "A survey of numerical methods for optimal control". In: *Proceedings of the 2009 AAS/AIAA Astrodynamics Specialists Conference*. Pittsburgh, PA.

Rodriguez, J. S. (2014). *Large-scale Dynamic Optimization Using Code Generation and Parallel Computing*. M.Sc. thesis. Department of Mathematics, KTH Royal Institute of Technology, Sweden.

Ruge, V., W. Braun, B. Bachmann, A. Walther, and K. Kulshreshtha (2014). "Efficient implementation of collocation methods for optimization using OpenModelica and ADOL-C". In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 1017–1025.

Runvik, H. (2014). *Modelling and Start-Up Optimization of a Coal-Fired Power Plant*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Safdarnejad, S. M., J. D. Hedengren, N. R. Lewis, and E. L. Haseltine (2015). "Initialization strategies for optimization of dynamic systems". *Computers & Chemical Engineering* **78**, pp. 39–50.

Sager, S. (2005). *Numerical Methods for Mixed-Integer Optimal Control Problems*. Ph.D. thesis. Heidelberg University, Germany.

Setubal, J. C. (1993). "New experimental results for bipartite matching". In: *Proceedings of* NETFLOW93. Pisa, Italty, pp. 211–216.

Sponk (2010). *Continuous binary fractional distillation*. Licensed under CC BY-SA 3.0 `http://creativecommons.org/licenses/by-sa/3.0`. URL: `https://commons.wikimedia.org/wiki/File%3AContinuous_Binary_Fractional_Distillation_EN.svg` (visited on 2016-08-27).

Tarjan, R. E. (1972). "Depth-first search and linear graph algorithms". *SIAM Journal on Computing* **1**:2, pp. 146–160.

Thelander Andrén, M. and C. Wedding (2015). *Development of a Solution for Start-up Optimization of a Thermal Power Plant*. M.Sc. thesis. Department of Automatic Control, Lund University, Sweden.

Thieriot, H., M. Nemura, M. Torabzadeh-Tari, P. Fritzson, R. Singh, and J. J. Kocherry (2011). "Towards design optimization with OpenModelica emphasizing parameter optimization with genetic algorithms". In: *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, pp. 756–762.

Thümmel, M., G. Looye, M. Kurze, M. Otter, and J. Bals (2005). "Nonlinear inverse models for control". In: *Proceedings of the 4th International Modelica Conference*. Hamburg, Germany, pp. 267–279.

Tomlab Optimization (2016). *PROPT*. URL: `http://tomdyn.com` (visited on 2016-08-15).

Trefethen, L. N. and D. Bau (1997). *Numerical Linear Algebra*. SIAM, Philadelphia, PA.

Vasantharajan, S. and L. T. Biegler (1990). “Simultaneous strategies for optimization of differential-algebraic systems with enforcement of error criteria”. *Computers & Chemical Engineering* **14**, pp. 1083–1100.

Wächter, A. and L. T. Biegler (2006). “On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming”. *Mathematical Programming* **106**, pp. 25–57.

Westerberg, A. W. and F. C. Edie (1971). “Computer-aided design, part 1: Enhancing convergence properties by the choice of output variable assignments in the solution of sparse equation sets”. *Chemical Engineering Journal* **2**:1, pp. 9–16.

Word, D. P., J. Kang, J. Åkesson, and C. D. Laird (2014). “Efficient parallel solution of large-scale nonlinear dynamic optimization problems”. *Computational Optimization and Applications* **59**:3, pp. 667–688.

Yang, D., B. Jacobson, M. Jonasson, and T. J. Gordon (2014). “Closed-loop controller for post-impact vehicle dynamics using individual wheel braking and front axle steering”. *International Journal of Vehicle Autonomous Systems* **12**:2, pp. 158–179.

Zavala, V. M. (2014). “Inference of building occupancy signals using moving horizon estimation and Fourier regularization”. *Journal of Process Control* **24**, pp. 714–722.

Zimmer, D., M. Otter, and H. Elmqvist (2014). “Custom annotations: Handling meta-information in Modelica”. In: *Proceedings of the 10th International Modelica Conference*. Lund, Sweden, pp. 174–182.