Inaugural - Dissertation
zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich-Mathematischen Gesamtfakultät
der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Diplom-Mathematiker Jan Albersmeyer
aus Hamburg

Tag der mündlichen Prüfung: 23. Dezember 2010

Adjoint-based algorithms and numerical methods for sensitivity generation and optimization of large scale dynamic systems

Gutachter: Prof. Dr. Dr. h.c. Hans Georg Bock
Prof. Dr. Moritz Diehl

Contents

Abstract IX

Zusammenfassung X

0 Introduction 1

1 Optimal control problems 9
1.1 Problem formulation . . . 9
1.2 Solution approaches . . . 12
1.2.1 Dynamic programming . . . 12
1.2.2 Indirect methods . . . 13
1.2.3 Direct methods . . . 14

2 Derivative generation 21
2.1 Symbolic differentiation . . . 21
2.2 Finite differences . . . 22
2.3 Complex step derivative approximation . . . 23
2.4 Automatic differentiation . . . 26
2.4.1 Basic concepts . . . 26
2.4.2 First order derivatives . . . 29
2.4.3 Higher-order derivative generation . . . 37
2.4.4 AD implementations . . . 55
2.5 Generation of sparse derivatives . . . 57

3 Nonlinear Programming 61
3.1 Notation and definitions . . . 61
3.2 Local optimality conditions . . . 65
3.3 Newton-type methods for NLP solution . . . 69
3.3.1 The Sequential Quadratic Programming (SQP) framework . . . 70
3.3.2 Full step exact-Hessian SQP . . . 72
3.3.3 Full step constrained Gauss-Newton . . . 73
3.3.4 Other SQP variants . . . 74
3.3.5 Globalization strategies . . . 75

4 Lifted methods for Nonlinear Programming **77**
4.1 The Lifted Newton Method . . . 78
4.1.1 An algorithmic trick for efficient computation of reduced quantities . . . 81
4.1.2 Simple practical implementation . . . 82
4.2 Application to Optimization . . . 86
4.2.1 A lifted Gauss-Newton method . . . 86
4.2.2 Nonlinear optimization via the lifted Newton method . . . 87
4.2.3 A lifted SQP method . . . 89
4.2.4 Equivalence of lifted SQP and full-space iterations in the constrained case . 91
4.3 Local Convergence Analysis of Lifted Newton Methods . . . 94
4.3.1 Local Convergence of the Non-Lifted Newton Method . . . 95
4.3.2 Local Convergence of the Lifted Newton Method . . . 96
4.3.3 Comparison of Lifted and Non-Lifted Newton Method . . . 97
4.4 A tutorial root finding example . . . 100

5 Solution of IVPs for ODEs and index 1 DAEs **105**
5.1 Basic DAE theory . . . 105
5.1.1 Notation and definitions . . . 105
5.1.2 Existence and uniqueness of solutions . . . 108
5.2 Numerical solution of IVPs for DAEs of index 1 . . . 109
5.2.1 Linear multistep methods on equidistant grids . . . 110
5.2.2 LMMs on variable grids . . . 117
5.2.3 BDF methods . . . 119
5.2.4 BDF methods for index 1 DAEs . . . 120
5.3 Strategies used in `DAESOL-II` . . . 122
5.3.1 Representation of the interpolation polynomials . . . 122
5.3.2 Error estimation . . . 127
5.3.3 Solution of the nonlinear corrector equation . . . 130
5.3.4 Stepsize and order control . . . 139
5.3.5 Start-up of the BDF method . . . 142
5.3.6 Computation of consistent initial values . . . 143
5.3.7 Relaxed formulation of the algebraic equations . . . 144
5.3.8 Scaling . . . 144
5.3.9 Continuous representation of the solutions . . . 145

6 Sensitivity generation **147**
6.1 Problem formulation . . . 148
6.2 Variational DAEs . . . 149
6.3 The principle of internal numerical differentiation . . . 153
6.4 First order sensitivity generation for DAEs . . . 155
6.4.1 Forward sensitivity generation . . . 156
6.4.2 Adjoint sensitivity generation . . . 161

6.5 Generation of sensitivities of arbitrary order . . . 168
6.5.1 Higher-order forward schemes . . . 170
6.5.2 Higher-order forward/adjoint schemes . . . 174
6.6 Comparison of the different IND-based schemes . . . 176
6.6.1 Computational effort . . . 176
6.6.2 Memory usage . . . 179
6.7 Other sensitivity related topics . . . 181
6.7.1 Exact interpolation . . . 181
6.7.2 Tape management . . . 181
6.7.3 Continuous representation of sensitivities . . . 182
6.7.4 Sensitivity injection . . . 182
6.7.5 Error control of forward sensitivities . . . 185
6.7.6 Global error estimation . . . 188
6.7.7 Time and control transformations of function and derivatives . . . 189
6.7.8 Sensitivity propagation across switching events . . . 191

7 A lifted exact-Hessian SQP method for OCPs with DAEs 203
7.1 The fundamentals of the method . . . 203
7.1.1 The structure of the QP subproblem . . . 204
7.1.2 Partial reduction technique for DAEs . . . 206
7.1.3 Computing the condensed QP using lifting and IND-TC propagation . . . 209
7.1.4 The basic L-PRSQP algorithm . . . 212
7.2 Further aspects . . . 213
7.2.1 Termination criterion . . . 213
7.2.2 Trust region globalization . . . 213
7.2.3 Infeasible subproblems . . . 214
7.2.4 Treatment of node bounds . . . 214
7.2.5 Problem scaling . . . 215
7.2.6 Free initial values . . . 215
7.2.7 Other cost functionals . . . 215
7.2.8 Multistage problems . . . 216
7.3 Comparison with a classical condensing approach . . . 216

8 Numerical examples for lifting-based optimization 219
8.1 A Gauss-Newton toy example . . . 219
8.2 An SQP toy example . . . 221
8.3 Lifting a simulation code for the shallow water equation . . . 222

9 Numerical examples for sensitivity related strategies 225
9.1 Comparison of the different IND-based strategies . . . 225
9.1.1 The SMB model . . . 225
9.1.2 The test setup . . . 226

9.2 Adjoint IND versus solution of adjoint variational equation 232
9.2.1 The HIRES problem 232
9.2.2 The Pleiades problem 236
9.2.3 The chemical Akzo Nobel problem 239
9.2.4 Test summary 242
9.3 Error control for forward sensitivities 242
9.4 Global error estimation 245

10 Optimal control of a distillation column **249**
10.1 Description of the distillation column 249
10.2 Description of the optimal control problem 255
10.3 Numerical results 256

11 Summary and Outlook **261**
11.1 Summary 261
11.2 Outlook and future work 264

Notation **267**

Acronyms **268**

List of figures **269**

List of tables **271**

List of algorithms **273**

Acknowledgments **275**

Bibliography **277**

Abstract

This thesis presents advances in numerical methods for the solution of optimal control problems. In particular, the new ideas and methods presented in this thesis contribute to the research fields of structure-exploiting Newton-type methods for large scale nonlinear programming and sensitivity generation for Initial Value Problems (IVPs) for ordinary differential equations and differential algebraic equations. Based on these contributions, a new lifted adjoint-based partially reduced exact-Hessian SQP (L-PRSQP) method for nonlinear multistage constrained optimization problems with large scale differential algebraic process models is proposed. It is particularly well suited for optimization problems which involve many state variables in the dynamic process but only few degrees of freedom, i.e., controls, parameter or free initial values. This L-PRSQP method can be understood as an extension of the work of Schäfer [Sch05] to the case of exact-Hessian SQP methods, making use of directional forward/adjoint sensitivities of second order. It stands hence in the tradition of the direct multiple shooting approaches for differential algebraic equations of index 1 of Bock and co-workers [BP84, Boc87, Sch88, BES88, Lei99]. To the novelties that are presented in this thesis further belong

- the generalization of the direct multiple shooting idea to structure-exploiting algorithms for Nonlinear Programs (NLPs) with an internal chain structure of the problem functions,
- an algorithmic trick that allows these so-called lifted methods to compute the condensed subproblems directly based on minor modifications to the user given problem functions and without further knowledge on the internal structure of the problem,
- a lifted adjoint-based exact-Hessian SQP method that is shown to be equivalent to a full-space approach, but only has the complexity of an unlifted/single shooting approach per iteration,
- new adjoint schemes for sensitivity generation based on Internal Numerical Differentiation (IND) for implicit Linear Multistep Methods (LMMs) using the example of Backward Differentiation Formulas (BDF),
- the combination of univariate Taylor Coefficient (TC) propagation and IND, resulting in IND-TC schemes which allow for the first time the efficient computation of directional forward and forward/adjoint sensitivities of arbitrary order,
- a strategy to propagate directional sensitivities of arbitrary order across switching events in the integration,
- a local error control strategy for sensitivities and a heuristic global error estimation strategy for IVP solutions in connection with IND schemes,
- the software packages `DAESOL-II` and `SolvIND`, implementing the ideas related to IVP solution and sensitivity generation, as well as the software packages `LiftOpt` and `DynamicLiftOpt` that implement the lifted Newton-type methods for general NLP problems and the L-PRSQP method in the optimal control context, respectively.

The performance of the presented approaches is demonstrated by the practical application of our codes to a series of numerical test problems and by comparison to the performance of alternative state-of-the-art approaches, if applicable. In particular, the new lifted adjoint-based partially reduced exact-Hessian SQP method allows the efficient and successful solution of a practical optimal control problem for a binary distillation column, for which the solution using a direct multiple shooting SQP method with an exact-Hessian would have been prohibitively expensive until now.

Keywords

Large Scale Nonlinear Programming, Optimal Control, Parameter Estimation, Partially Reduced Newton-Type Methods, Lifted Methods, Constrained Optimization, Gauss-Newton, Exact-Hessian SQP, Direct Multiple Shooting, Internal Numerical Differentiation, Automatic Differentiation, Taylor Coefficient Propagation, Initial Value Problem Solution, Directional Sensitivity Generation, Ordinary Differential Equations, Differential Algebraic Equations

Zusammenfassung

Die vorliegende Arbeit präsentiert Fortschritte in numerischen Methoden zur Lösung von Optimalsteuerungsproblemen. Insbesondere tragen die präsentierten Ideen und Methoden zur Forschung auf den Gebieten der strukturausnutzenden Newton-ähnlichen Verfahren für hochdimensionale nichtlineare Optimierungsprobleme und der Sensitivitätserzeugung für Lösungen von Anfangswertproblemen (AWP) von gewöhnlichen Differentialgleichungen und differentiell-algebraischen Gleichungen bei. Basierend auf diesen Beiträgen wird ein neues, sogenanntes geliftetes, auf adjungierten Sensitivitäten beruhendes, partiell reduziertes SQP-Verfahren mit exakter Hessematrix (L-PRSQP) zur Behandlung mehrstufiger, beschränkter Optimierungsprobleme mit hochdimensionalen Modellen aus differentiell-algebraischen Gleichungen vorgestellt. Dieses eignet sich besonders für Optimierungsprobleme, deren dynamisches Modell viele Zustandsvariablen enthält, und die nur wenige Freiheitsgrade wie Steuerungen, Parameter oder freie Anfangswerte besitzen. Sie kann als eine Erweiterung des Ansatzes von Schäfer [Sch05] auf den Fall von SQP-Verfahren mit exakter Hessematrix verstanden werden, die von kombinierten vorwärts/rückwärts Richtungssensitivitäten zweiter Ordnung Gebrauch macht. Das Verfahren steht damit in der Tradition der von Bock und Mitarbeitern [BP84, Boc87, Sch88, BES88, Lei99] entwickelten direkten Mehrfachschießverfahren für differentiell-algebraische Modelle vom Index 1. Zu den Neuheiten, die in dieser Arbeit vorgestellt werden, gehören weiterhin

- die Verallgemeinerung der Idee des direkten Mehrfachschießverfahrens auf strukturausnutzende Algorithmen zur Lösung allgemeiner nichtlinearer Optimierungsprobleme, deren Problemfunktionen eine kettenartige innere Struktur aufweisen,
- ein algorithmischer Trick, der es diesen sogenannten gelifteten Verfahren erlaubt, die kondensierten Subprobleme nach kleiner Modifikation der vom Benutzer bereitgestellten Problemfunktionen und ohne weitergehende Kenntnis der inneren Struktur des Problems, direkt zu berechnen,
- ein effizientes, geliftetes, auf Adjungierten basierendes SQP-Verfahren mit exakter Hessematrix, von dem die Äquivalenz zur Vollraummethode bewiesen wird, welches aber pro Iteration nur die Komplexität eines ungelifteten Verfahrens/Einfachschießverfahrens aufweist,
- neue adjungierte Schemata zur Sensitivitätserzeugung basierend auf Interner Numerischer Differentiation (IND) für implizite lineare Mehrschrittverfahren am Beispiel von BDF-Methoden,
- die Kombination von univariater Taylorkoeffizientenpropagation und IND, welche in IND-TC Schemata resultiert, die erstmals die effiziente Berechnung von Richtungssensitivitäten beliebiger Ordnung erlauben,
- eine Strategie zur Propagation von Richtungssensitivitäten beliebiger Ordnung durch Schaltpunkte in der Integration hindurch,
- eine lokale Fehlerkontrolle für Sensitivitäten und einen heuristischen globalen Fehlerschätzer für die AWP-Lösung, basierend auf den IND Schemata,
- die Softwarepakete `DAESOL-II` und `SolvIND`, in welchen die Strategien zur AWP-Lösung und Sensitivitätserzeugung implementiert sind, sowie die Pakete `LiftOpt` und `DynamicLiftOpt`, welche die gelifteten Newton-ähnlichen Verfahren für allgemeine NLPs beziehungsweise das L-PRSQP-Verfahren im Optimalsteuerungskontext beinhalten.

Die Effizienz der vorgestellten Ansätze wird mittels der praktischen Anwendung unserer Softwarepakete auf eine Reihe von numerischen Testproblemen und, wenn möglich, eines Vergleichs mit alternativen state-of-the-art Verfahren demonstriert. Insbesondere erlaubt das neue L-PRSQP-Verfahren eine effiziente und erfolgreiche Behandlung eines praktischen Optimalsteuerungsproblems für eine binäre Destillationskolonne, bei dem der Aufwand der Lösung mittels eines Mehrfachschießverfahrens mit exakter Hessematrix bislang unvertretbar hoch wäre.

Schlagworte

Hochdimensionale nichtlineare Optimierungsprobleme, Optimale Steuerung, Parameterschätzung, Partiell reduzierte Newton-ähnliche Verfahren, Geliftete Verfahren, Beschränkte Optimierung, Gauss-Newton, SQP-Verfahren mit exakter Hessematrix, Direktes Mehrfachschießverfahren, Interne Numerische Differentiation, Automatisches Differenzieren, Taylorkoeffizientenpropagation, Lösung von Anfangswertproblemen, Erzeugung von Richtungssensitivitäten, Gewöhnliche Differentialgleichungen, Differentiell-algebraische Gleichungen

0 Introduction

Over the last decades mathematical methods for modeling, simulation and optimization have gained not only continuously more and more importance in industrial applications, but have also an ever increasing impact on our everyday life. Processes from nearly all fields, such as engineering, chemistry, biology, medicine, physics and economics are now often translated into a mathematical model and afterwards analyzed, simulated, and optimized using mathematical methods. The results of this development can be observed in our daily life in many different forms. Examples are the planning of surgeries and radiation therapies based on an individual model of the patient, improvements of the operation of industrial plants regarding to quality, safety, throughput, the use of raw materials, energy consumption, etc., the fast determination of the optimal route to a target by a GPS-based navigation system in our car or mobile phone (possibly even accounting for the actual traffic situation) or the automated trading software agents performing a large number of deals every fraction of a second autonomously in the stock markets.

To the same extent to which the understanding and the modeling of the processes gets more and more detailed, the complexity and usually also the size of the resulting mathematical models increases. This in return drives the need for an increased performance in the solution of the related mathematical simulation and optimization problems.

This is one of the motivations for performing research on the mathematical methods for the simulation, analysis and optimization of these processes, and hence also for this work. This is even more the case, as advances in the methods often give rise to enormous speedups and regularly open up the possibility to treat completely new classes of problems.

Many (dynamic) processes can be described mathematically by a system of nonlinear differential and algebraic equations involving so-called control functions, which together with the initial system state determine the development of the process in time. The optimization of this kind of processes, possibly subject to constraints as costs, time, safety margins, limited ressources, etc., is commonly denoted by optimal control.

Optimal Control Problems (OCPs) often occur as "offline" problems, i.e., they are solved once and the resulting optimal controls are then applied to the process. But they also occur in the context of "online" optimization, e.g., in the framework of Nonlinear Model Predictive Control (NMPC). Here the optimal control problem is solved repeatedly after every feedback from the process (e.g., in form of measurements of the system state), which allows to react to occurring disturbances based on the recomputed and adapted optimal control profile (see, e.g., [RMM94, ABQ+99]). In

any case, to solve these problems in practice, fast and reliable numerical methods are needed.

The three fundamental approaches to solve the infinite-dimensional OCPs are given by dynamic programming, which is based on Bellman's principle of optimality, the indirect approach based on Pontryagin's maximum principle and the direct approach, based on the transcription of the optimal control problem into a finite-dimensional Nonlinear Program (NLP) which is afterwards solved, e.g., by a tailored Sequential Quadratic Programming (SQP) method. The transcription is achieved by discretizing the control functions and possibly also the states on a suitable grid.

For the solution of constrained OCPs, methods based on the direct approach have been proven to be particular successful for practical problems. They can roughly be separated into sequential and simultaneous approaches. In sequential approaches like direct single shooting ([HR71, SS78, Kra85, MS86]), the dynamic model is considered as black box, and the optimization algorithm is based only on the input-output relationship between the discretized control functions and the values of cost functional and constraints. Simultaneous or all-at-once approaches like direct collocation ([THE75, Bär83, Bie84, CB89, LB92, Sch96]) or direct multiple shooting ([Pli81, BP84, BES88, Lei95, Lei99, LBBS03, LSBS03, TB95, TB96, PRG$^+$97, GJL$^+$00, BP04, Sch05, Rie06, Sag06]) also parametrize or discretize the dynamic of the process, and add the resulting variables as additional degrees of freedom to the NLP, which means that the optimization and the solution of model dynamics occur simultaneously.

The simultaneous approaches like direct multiple shooting seem to have the disadvantage of a larger number of variables, but this can be overcome to a large extent by exploiting the resulting specific structure in the Newton-type NLP method. Their advantages include a better possibility of introducing a priori knowledge of the solution and better convergence properties (see, e.g., [BP84, Boc87]). For multiple shooting, the structure exploitation can be performed in different ways. The first possibility is to build the full-space subproblems and to perform afterwards condensing steps to reduce the size of the QP subproblem to that of the QP in the comparable single shooting problem. However, this would need the building of the complete sensitivity matrices of the states, which is, already for moderately large problems, by far the most expensive task in the optimization algorithm. And it is prohibitively expensive or even impossible in case of large scale problems that arise, e.g., from spatial discretizations of instationary Partial Differential Equations (PDEs).

Alternatively, the condensed problem can be computed directly by using directional sensitivities in a suitable way, which avoids the need to form the complete state sensitivities, and instead needs essentially only sensitivities with respect to the discretized control functions, similar to single shooting. This is favorable in the case where we have only a few control functions and an, in comparison, large number of states of the dynamic system. This reduced approach has been developed first for constrained Gauss-Newton methods [Sch88] and later for update-based SQP methods in the direct multiple shooting context [Sch05, GJL$^+$00]. However, until now this idea cannot be used in connection with exact-Hessian SQP methods. In case of a comparatively large number of control functions and few system states, like it occurs often in methods for mixed-integer optimal

control, other means like complementary condensing [KBSS11, Ste95] and the direct exploitation of the QP sparsity are of advantage.

Until now simultaneous approaches usually lead to a higher implementational burden and the algorithms are tailored to the specific structure of the underlying problem, which explains why direct single shooting is still widely used in practice despite its drawbacks.

Even if by the choice of a reduction approach the number of needed directional sensitivities can be decreased, sensitivity generation remains, together with the solution of Initial Value Problems (IVPs), the most expensive subtask in many modern optimization algorithms for large scale problems. Efficient sensitivity generation is, however, not only of interest in the solution of dynamic optimization problems, but also, e.g., for the analysis of dynamical systems in general or for model reduction [LSF08].

Most algorithms for sensitivity generation are based on the solution of the associated forward or adjoint variational differential equation with a discretization grid that is different from the one of the solution of the nominal IVP. This approach has the advantage that it is relatively easy to implement, once an IVP solver is at hand. However, drawbacks are that the formulation of the corresponding variational differential equation often has to be done manually, which might be cumbersome and error-prone especially for large scale systems. Furthermore, these methods do compute an approximation of the derivative of the exact (analytical) IVP solution, but this approximation has in general no or only a limited connection to the derivative of the numerical IVP solution produced by the integrator. This, however, is important for the use in adaptive direct optimization methods, where it is important that the computed derivatives and the function evaluations are properly related. Another drawback is that the internal structure arising from the relationship of the nominal IVP and the variational IVP is either not exploited, or this exploitation has to be done manually.

An approach to overcome these problems is the principle of Internal Numerical Differentiation (IND), invented by Bock [Boc81]. Simply speaking, using IND means to derive the numerical scheme of the integrator with frozen adaptive components, e.g., using the techniques of Automatic Differentiation (AD) (see, e.g., [Gri00]). The result is a numerical scheme for the approximation of the sensitivities that delivers always the exact derivative of the numerical IVP solution. Furthermore, IND allows to reuse a lot of information from the nominal integration for the sensitivity computation and hence leads to an automatic structure exploitation. Over the years, several IND-based codes have been developed and used with great success. However, with few exceptions all of them only allow the computation of forward sensitivities of first or at most second order. For implicit multistep methods, that can be applied to stiff IVPs, until now no IND schemes exist for the computation of adjoint sensitivities, even though they are of great interest, e.g., in the context of SQP methods with inexact constraint Jacobians [JS97, HV01, GW02, BDK04, DWBK09], whenever gradient-type information is needed or if many control functions and parameter are present. Also the computation of second order sensitivities, which are needed for exact-Hessian

approaches, robust optimization [KKBS04, DBK06] and Optimal Experimental Design (OED) [Bau99, BBKS00, Kör02, KKBS04], in a pure forward way is very inefficient for medium to large scale systems. Hence, the possibility to combine a forward and adjoint mode of IND for the computation of second order sensitivities, analogously to the similar concept in the context of AD for the computation of second order derivatives of ordinary functions is highly desirable and motivates the research presented in this work.

Goals of this thesis

This thesis aims to make contributions to two major research fields that are strongly connected to the efficient solution of large scale dynamic optimization problems in general and the solution of optimal control problems in particular: the field of structure-exploiting Newton-type methods for nonlinear programming and the field of sensitivity generation for solutions of IVPs for Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs).

On the one hand, we will explain how the idea of direct multiple shooting and the advantages of this simultaneous approach can be transferred to the more general context of nonlinear programming problems, in which the problem function evaluation possesses a certain internal structure in form of intermediate values. These intermediate variables are then, as in multiple shooting, added as degrees of freedom to the problem – thus "lifting" the problem into a space with more variables – together with corresponding constraints to ensure equivalence of the solution with the original problem. We then develop lifted algorithms that are able to exploit the internal structure of the augmented (lifted) problems automatically, without requiring much detailed information on the problem structure. This also significantly reduces the implementation overhead normally associated with this approach as well as the dependency of the algorithm on a specific problem structure, which makes them usable for a broader audience. Furthermore, we will investigate – carrying on the tradition of other structure-exploiting methods developed by Schlöder [Sch88] and Schäfer [Sch05] in the multiple shooting context – how a lifted exact-Hessian SQP method can be constructed that possesses the same order of complexity in terms of needed derivatives per nonlinear iteration as a corresponding unlifted/sequential/single shooting method. This is contrary to the existing exact-Hessian SQP methods which become prohibitively expensive for large scale problems regarding both run-time and memory demands.

On the other hand, we will present new methods for the efficient generation of (directional) sensitivities of IVP solutions of stiff systems, a task that is in many cases essential for the construction of efficient algorithms for the solution of dynamic optimization problems. We want to benefit from both the favorable properties of the adjoint mode of AD and of the principle of IND to obtain new first order adjoint IND schemes for sensitivity generation in a Backward Differentiation Formula (BDF) method. In addition, we want to explore the possibilities of transferring the approach of univariate Taylor Coefficient (TC) propagation to the IND context to develop IND-TC schemes that allow for the first time the generation of directional sensitivities of arbitrary order. This is a new capability that is very important not only for exact-Hessian methods for optimal control problems, but also in the context of robust optimization and OED, where higher-

order directional sensitivities are needed regularly. Furthermore, several new strategies related to sensitivity generation are analyzed, e.g., for the propagation of sensitivities across switching events.

To bring these newly developed ideas immediately to a practical application we finally like to show, how in connection with a partial reduction technique for DAEs a lifted partially reduced exact-Hessian SQP (L-PRSQP) method in the framework of direct multiple shooting can be constructed. The method shows a complexity that is independent of the number of states of the dynamic system, which is a significant improvement compared to existing approaches.

Among the novelties that we will present in this thesis are

- the generalization of the direct multiple shooting idea to structure-exploiting algorithms for general NLPs with intermediate values in their function evaluations, the so-called lifted methods,
- an algorithmic trick that allows these lifted methods to compute condensed subproblems directly, based on only minor modification to the given user functions and without further knowledge on the internal structure of the problem,
- an efficient adjoint-based lifted exact-Hessian SQP method that is shown to be equivalent to a full-space approach, but only has the complexity of an unlifted/single shooting approach per iteration,
- new adjoint IND schemes for implicit Linear Multistep Methods (LMMs) at the example of BDF methods,
- the combination of univariate TC propagation and IND resulting in the first IND-TC schemes, which allow for the first time the efficient computation of directional forward and forward/adjoint sensitivities of arbitrary order,
- a strategy to propagate arbitrary order directional sensitivities across switching events in the integration,
- a local error control strategy for sensitivities and a heuristic for global error estimation for the IVP solution in connection with IND schemes,
- a lifted partially reduced exact-Hessian SQP (L-PRSQP) method tailored to DAE optimal control problems in the direct multiple shooting framework, that allows the treatment of large scale problems that otherwise would be too expensive to solve with an exact-Hessian SQP method,
- the software packages `DAESOL-II` and `SolvIND`, implementing the ideas related to IVP solution and sensitivity generation, as well as the software packages `LiftOpt` and `DynamicLiftOpt` that implement the lifted Newton-type methods for general NLP problems and the L-PRSQP method in the optimal control context, respectively.

Thesis overview

This thesis is organized as follows. In Chapter 1 we present a class of optimal control problems. We first give a basic definition of optimal control problems for index 1 DAE models and then discuss possible extensions. Afterwards, we give a brief overview of the commonly used solution strategies with an emphasis on direct methods. Here we explain shortly the principles behind direct single shooting, direct collocation and direct multiple shooting methods and discuss the individual advantages and disadvantages of the different approaches.

As the computation of derivatives of functions such as cost functionals, constraints or ODE/DAE model functions will play an important role in most of the later chapters of this thesis we give in Chapter 2 an overview of different techniques for derivative generation. We present the commonly known approaches with their advantages and shortcomings. Afterwards, we give a detailed introduction to the basic concept of the derivative generation using AD, as we will use this technique regularly throughout this thesis. We put a special emphasis here on the generation of higher-order directional derivatives using univariate TC propagation, as this topic is not so commonly known but essential to understand the methods we develop in this thesis. At the end, we discuss briefly how sparsity can be exploited in the computation of derivatives.

Chapter 3 is dedicated to constrained nonlinear programming. We give a general formulation of a constrained NLP, as well as some special important subclasses along with the required notation and definitions. Then we address necessary and sufficient conditions for local optimal solutions of a NLP. In Section 3.3 we explain the solution of NLPs using Newton-type optimization methods. In particular, we address here the framework of SQP methods and discuss two important members of the SQP family in more detail. Afterwards, we give a short overview of other SQP variants not treated in this thesis. The chapter ends with a short presentation on strategies to ensure global convergence of the algorithms.

In Chapter 4 we present the lifting idea for the solution of NLPs and develop algorithms for their solution that solve the augmented (lifted) system by a structure-exploiting Newton-type method, yet do not require any additional knowledge about the structure of the problem functions or the meaning of the intermediate variables. We explain in Section 4.1 the basic idea at the example of Newton's method for a root finding problem and derive a "lifted" Newton algorithm for the efficient solution of the problem. In Section 4.2 we discuss the application of the lifting approach to optimization and derive a lifted Gauss-Newton method and a lifted exact-Hessian SQP method for equality and inequality constrained NLPs that is based on adjoint gradient computations. We prove the equivalence of this last method with the iterations obtained by a full-space SQP method. Afterwards, we discuss in Section 4.3 under which circumstances "lifted" approaches converge faster than non-lifted ones, and give a proof in a simplified setting. We illustrate the potential of the developed method with the help of a tutorial example in Section 4.4 and confirm here for the numerical solution with our software package `LiftOpt` also the convergence properties that we derived theoretically.

Chapter 5 treats the efficient numerical solution of (stiff) IVPs for ODEs and DAEs of index 1, which occur as subproblems, e.g., in the solution of optimal control problems. In Section 5.1 we state some important definitions and results from DAE theory. Afterwards, we explain the properties of LMMs and the subclass of BDF methods, which form the basis of our numerical integration code `DAESOL-II`. The more specific strategies related to IVP solution that are used in our code `DAESOL-II` are presented in Section 5.3. The different subtasks occurring in the numerical scheme for IVP solution are discussed in more detail, as they are required for the deeper understanding of the strategies for sensitivity generation that are developed in the following chapter. Topics presented here include, e.g., the representation of predictor and corrector polynomials, the monitor strategy for the Newton-like method used for the solution of the implicit corrector equation as well as the continuous representation of the IVP solution.

In Chapter 6 we address the computation of sensitivities, i.e., of the derivatives of solutions of IVPs with respect to initial values and parameter. We start here with the general problem formulation and show that forward and adjoint sensitivities can be represented as solutions of the so-called forward or adjoint variational ODE/DAE, which could be used to compute the sensitivities analytically (or also numerically). Afterwards, we explain the principle of IND and show in the following how, based on the IND principle and in connection with AD, efficient numerical structure-exploiting schemes for the computation of sensitivities can be derived. We present here new adjoint IND schemes for implicit LMMs using the example of BDF methods. In Section 6.5 we present the combination of the principle of IND and of univariate TC propagation which enables us to derive the first schemes capable of computing directional forward and forward/adjoint sensitivities of arbitrary order. In particular, the new possibility to compute reduced Hessian sensitivities based on an IND approach is essential for the lifted exact-Hessian SQP method presented in the following chapter. We give then a comparison of the different IND-based schemes for sensitivity generation and present in Section 6.7 several more specific strategies related to sensitivity generation in general and their implementation in connection with `DAESOL-II` and `SolvIND`. We address here among others the continuous representation of the sensitivities, the computation of adjoint derivatives of functions defined on states on an arbitrary timegrid by adjoint sensitivity injection and the local error control for forward sensitivities of arbitrary order. Furthermore we give a proposal for an IND-based a posteriori error estimator for the global error of the ODE/DAE solution. Finally, we develop for the first time a strategy for the propagation of directional sensitivities of arbitrary order across switching events, which makes the computation of higher-order sensitivities in this context now computationally feasible.

The combination of the developed lifted exact-Hessian SQP method and the new sensitivity generation schemes in the framework of direct multiple shooting for optimal control problems for DAEs is presented in Chapter 7. In Section 7.1 we present the fundamentals of our algorithm. We explain the basic problem structure resulting from the application of the direct multiple shooting approach to the optimal control problem and how the classical condensing approach is working in this case. In this context, we address also the partial reduction technique for DAEs. We show that this approach can be used in connection with our lifted optimization methods to enable them also

for the use in the context of DAE problems. Furthermore, we explain how (IND-)TC propagation can be used in the context of lifting to compute here the condensed QP subproblem directly and give a basic form of our algorithm. Section 7.2 covers further practical aspects in connection with our algorithm, such as the termination criterion and the treatment of infeasibilities. We end this chapter with Section 7.3, where we give a comparison of our lifted partially reduced exact-Hessian SQP (L-PRSQP) algorithm with an exact-Hessian SQP algorithm based on classical condensing to illustrate the advantages of our approach.

Numerical examples of the application of our code `LiftOpt`, in which the lifting idea presented in Chapter 4 is implemented, are given in Chapter 8. Sections 8.1 and 8.2 illustrate the benefits of the presented lifted optimization methods compared to unlifted methods on a small toy example, while Section 8.3 treats a very large scale example. The latter one, a parameter estimation problem for hyperbolic PDEs, demonstrates also the practical feasibility of lifting a large, user given simulation code with `LiftOpt`.

In Chapter 9 we present numerical tests for the sensitivity related strategies implemented in our packages `DAESOL-II` and `SolvIND`. We analyze in Section 9.1 the costs of different integration and sensitivity generation tasks on a scalable test problem given by a model for an SMB process. In Section 9.2 we compare our newly developed adjoint IND scheme against an alternative approach based on the solution of the adjoint variational ODE/DAE, implemented in the `SUNDIALS` suite. The comparison is based on several test problems from the IVP testset of the University of Bari and shows the efficiency of our IND-based schemes. In Section 9.3 we demonstrate that the proposed strategy for error control of forward sensitivities works and leads to only slightly larger computational costs. Section 9.4 confirms numerically that the intermediate adjoint IND quantities can be used to create an efficient a posteriori error estimator for the global error of the IVP solution, also in case of index 1 DAEs.

In Chapter 10 we show that our L-PRSQP algorithm, which is implemented in the package `DynamicLiftOpt`, is able to efficiently solve practical DAE optimal control problems. We demonstrate this for the example of an optimal control problem for a binary distillation column. In Section 10.1 we describe briefly the model of the distillation column. Afterwards, we explain in Section 10.2 the setup of the optimal control problem and present in Section 10.3 the numerical results.

Chapter 11 finally contains a brief summary of the most important ideas and results of this thesis and addresses several topics that might be interesting as directions of further research.

1 Optimal control problems

In this chapter we present a class of continuous optimal control problems. We define the basic problem class and discuss possible extensions. Afterwards, we address shortly the different fundamental solution strategies, i.e., dynamic programming, indirect methods and direct methods. We then focus on direct methods and have a closer look at their main ideas, as we will refer to some of them in the derivation and description of our optimization algorithms in Chapters 4 and 7.

1.1 Problem formulation

In the following, we first give a basic formulation of an optimal control problem for dynamic processes described by Differential Algebraic Equations (DAEs).

Definition 1.1 (Continuous optimal control problem)
A continuous Optimal Control Problem (OCP) for DAEs is a constrained optimization problem of the form

$$\min_{\mathbf{u}(\cdot),\mathbf{x}(\cdot),\mathbf{z}(\cdot),\mathbf{p}} \quad c\big(\mathbf{x}(\cdot),\mathbf{z}(\cdot),\mathbf{u}(\cdot),\mathbf{p}\big) \tag{1.1a}$$

s.t.

$$\begin{aligned}
\mathbf{A}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p})\cdot\dot{\mathbf{x}}(t) &= \mathbf{f}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}), \qquad t\in T=[t_s,t_f] && \text{(1.1b)}\\
\mathbf{0} &= \mathbf{g}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}) && \text{(1.1c)}\\
\mathbf{0} &\leq \mathbf{h}^{\text{cont}}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}) && \text{(1.1d)}\\
\mathbf{x}(t_s) &= \mathbf{x_0}. && \text{(1.1e)}
\end{aligned}$$

The different components of the problem are given by

- $T=[t_s,t_f]\subset\mathbb{R}$ *represents the fixed time horizon on which the problem is formulated and the time variable is denoted by* t.
- $\mathbf{u}: T\to\mathbb{R}^{n_u}$ *is the control function which is to be determined. In the general case,* u *is assumed to be a measurable function.*
- $\mathbf{x}: T\to\mathbb{R}^{n_x}, \mathbf{z}: T\to\mathbb{R}^{n_z}$ *are the differential and algebraic states of the dynamic process under consideration. They must fulfill the DAE model of the process and hence are assumed to be at least continuously differentiable.* $\mathbf{x_0}\in\mathbb{R}^{n_x}$ *describes the initial differential state of the process.*

- $\mathbf{A} : T \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x \times n_x}, \mathbf{f} : T \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ *and* $\mathbf{g} : T \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_z}$ *are the DAE model functions describing the time development of the underlying dynamic process. They are in general assumed to be twice continuously differentiable. Throughout this thesis, we consider only linearly implicit DAEs of index 1, as they suffice to describe dynamic processes that occur in a broad variety of application fields. Furthermore, their intrinsic structure can be efficiently exploited by DAE solvers. We assume that* $\frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ *and* $\mathbf{A}$ *are regular along the DAE solution trajectory to guarantee the index 1 assumption. As a result, the algebraic states are fully determined by the differential states, the control and the parameter. For more details on the properties and the efficient numerical integration of the DAE model we refer to Chapter 5.*

- $\mathbf{p} \in \mathbb{R}^{n_p}$ *is a parameter vector that contains all degrees of freedom of the problem that are not time-dependent, e.g., the parameter of the DAE model of the dynamic process.*

- *c is a cost functional defined on the control function, the trajectories of the differential and algebraic states as well as the parameter. A common type of cost functional is the Bolza cost functional. It consists of two parts: a Lagrange term, i.e., the integral over a Lagrange objective function* $l(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p})$*, and a Mayer term, i.e., an end-point contribution* $m(t_f, \mathbf{x_f}, \mathbf{z_f}, \mathbf{p})$*. Hence a Bolza objective can be written in the form*

$$c\big(\mathbf{x}(\cdot), \mathbf{z}(\cdot), \mathbf{u}(\cdot), \mathbf{p}\big) = \int_{t_s}^{t_f} l(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p}) dt + m(t_f, \mathbf{x}(t_f), \mathbf{z}(t_f), \mathbf{p}). \tag{1.2}$$

  We assume in general that the cost functional is twice continuously differentiable .

- $\mathbf{h}^{\text{cont}} : T \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{\text{dec}}}$ *is a twice continuously differentiable function representing possibly mixed state and control constraints on the time horizon. As they do not combine the values of states or controls at different points in time they are also called decoupled constraints.*

In addition to this basic problem formulation there exist several possible extensions and specializations that are important in practice. These include:

- Variable time horizon, e.g., free end time:

  The above OCP is formulated on the fixed time horizon $T = [t_s, t_f]$. The end time or equivalent the length of the time horizon can be made a degree of freedom by performing a time transformation. If we define

$$t(\tau) := t_s + p_h \tau \quad \text{and} \quad p_h := t_f - t_s, \tag{1.3}$$

  we can add p_h as degree of freedom to the parameter vector $\mathbf{p}$ and reformulate the OCP in the "normalized" time τ on the time horizon $\tau \in [0, 1]$. The differential right hand side of the DAE (1.1b) then transforms to (assuming $\mathbf{A} \equiv \mathbb{I}$ here for readability)

$$\dot{\mathbf{x}}(\tau) = p_h \cdot \mathbf{f}(t(\tau), \mathbf{x}(t(\tau)), \mathbf{z}(t(\tau)), \mathbf{u}(t(\tau)), \mathbf{p}), \qquad \tau \in T = [0, 1] \tag{1.4}$$

  and all function evaluations have to be performed for the "physical" time $t(\tau)$.

- Other constraint types:

  There are further possibilities to impose constraints to the OCP. Boundary constraints only involve the initial and final states and hence have the form

  $$\mathbf{0} \leq \mathbf{h}^{\text{bnd}}(\mathbf{x}(t_s), \mathbf{z}(t_s), \mathbf{x}(t_f), \mathbf{z}(t_f)). \tag{1.5}$$

  Interior (multi-)point constraints may depend on the states at one (or more) time points within the time horizon as well as on the parameter

  $$\mathbf{0} \leq \mathbf{h}^{\text{ip}}(\mathbf{x}(t_1), \mathbf{z}(t_1), \ldots, \mathbf{x}(t_i), \mathbf{z}(t_i), \ldots, \mathbf{x}(t_n), \mathbf{z}(t_n), \mathbf{p}), \quad t_i \in [t_s, t_f], \quad 1 \leq i \leq n \tag{1.6}$$

  We also distinguish between coupled constraints, which may depend beside the parameter also on the state and the control values at different timepoints, and decoupled constraints, which depend beside the parameter only on quantities at one time point. An example for the latter are the continuous path and control constraints in the basic OCP formulation. Note that in principle all these constraint types can also contain equality constraints, e.g., in the case of boundary constraints to describe periodicity conditions.

- Least-squares objective:

  An important subtype of cost functionals is given by the so-called least-squares functionals. They have the form

  $$c\big(\mathbf{x}(\cdot), \mathbf{z}(\cdot), \mathbf{u}(\cdot), \mathbf{p}\big) = \int_{t_s}^{t_f} \|\mathbf{r}(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p})\|_2^2 \mathrm{d}t, \tag{1.7}$$

  where $\mathbf{r} : T \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{\text{res}}}$ is the least-squares residual function. Least-squares functionals often occur in tracking problems, where the deviation of the states from a prescribed path is penalized, and in parameter estimation problems. Furthermore, they can be used to regularize the controls $\mathbf{u}$ numerically.

- Multistage problems:

  The OCP (1.1) was defined based on one specific dynamical model with a fixed number of states, parameter and controls. In practice, however, it can be desirable and sometimes even necessary to allow changes in the underlying process dynamics or other transitions that can be modeled by a multistage optimal control problem. Practical examples for this can be found, e.g., in [Lei99, DLS+02]. A multistage formulation with n_{mos} model stages can be formed by defining on each model stage an OCP of type (1.1) with possibly different dimensions, dynamic models and constraints. The overall cost functional is then defined as the sum of the cost functionals on the different model stages. In this formulation each model stage is connected with the next model stage by a transition constraint of the form

  $$\mathbf{x_{k+1}}(t_{s,k+1}) = \mathbf{h}_{\mathbf{k}}^{\text{trans}}(t_{f,k}, \mathbf{x_k}(t_{f,k}), \mathbf{z_k}(t_{f,k}), \mathbf{u_k}(t_{f,k}), \mathbf{p_k}), \qquad 1 \leq k \leq n_{\text{mos}} - 1, \tag{1.8}$$

  where the subindices k and $k+1$ denote to which model stage the quantities belong.

- Mixed-integer problems:

  It is also possible to restrict some of the controls and/or the parameter to a set of distinct values. This leads to mixed-integer optimal control problems which require a special treatment. For more information, we refer to [Sag05, KSBS10] .

We refrain here from stating and discussing the optimality conditions for the continuous OCP as well as the structure of optimal solutions. Necessary optimality conditions are usually derived using Pontryagin's maximum principle and can be found (for the basic OCP with ODE dynamics), e.g., in [PBGM62, BH75]. For a derivation of sufficient optimality conditions for certain problem classes we refer to [MO04] and references therein.

1.2 Solution approaches

Over the years, several different approaches have been developed to (numerically) solve OCPs of type (1.1). They can be distinguished, e.g., by the space in which the optimization of the problem occurs, the discretization approach and the order in which discretization and optimization take place. Depending on the chosen approach, they differ in the run-time and memory demands for the numerical solution, the achievable accuracy of the solution, the possibility to compute global optima, the effort for their implementation and, as a result, also in their applicability to certain problem classes and problem sizes. In general, the solution approaches can be divided into three larger classes: Dynamic programming, indirect and direct methods.

1.2.1 Dynamic programming

Dynamic programming is an approach that is based on Bellman's principle of optimality [Bel57]. Roughly speaking, it says that any subarc of a given optimal solution of the OCP on the whole time horizon is also optimal on the corresponding part of the time horizon.

Theorem 1.2 (Bellman's principle of optimality)
If $(\mathbf{u}^*(\cdot), \mathbf{x}^*(\cdot))$ *is an optimal solution of the OCP* (1.1) *(for notational ease we consider in the following only the ODE case with a Bolza type cost functional) on the time horizon* $[t_s, t_f]$*, then for any given time point* $\tilde{t} \in [t_s, t_f]$ *holds, that* $(\mathbf{u}^*(\cdot), \mathbf{x}^*(\cdot))$ *is also an optimal solution of the problem for the time horizon* $[\tilde{t}, t_f]$ *and the initial value* $\mathbf{x}(\tilde{t}) = \mathbf{x}^*(\tilde{t})$*.*

The vehicle to make practical use of this idea is the cost-to-go function

Definition 1.3 (Optimal-cost-to-go function)
We define the optimal-cost-to-go function for the OCP (1.1) *on the interval* $[\tilde{t}, t_f]$ *as*

$$\mathrm{cost}(\tilde{t}, \tilde{\mathbf{x}}) := \min_{\mathbf{u}(\cdot), \mathbf{x}(\cdot)} \int_{\tilde{t}}^{t_f} l(\mathbf{x}(t), \mathbf{u}(t)) dt + m(\mathbf{x}(t_f)), \tag{1.9}$$

where the condition $\mathbf{x}(\tilde{t}) = \tilde{\mathbf{x}}$ *as well as the constraints of the OCP are imposed.*

Dynamic programming is normally used for discrete time systems. Assuming a timegrid $t_s = t_1 < t_2 < \ldots < t_n = t_f$ the optimal-cost-to-go function at a gridpoint $1 \leq j < n$ can be defined recursively by

$$\text{cost}(t_j, \mathbf{x_j}) := \min_{\mathbf{u}(\cdot),\mathbf{x}(\cdot)} \int_{t_j}^{t_{j+1}} l(\mathbf{x}(t), \mathbf{u}(t))\mathrm{d}t + \text{cost}(t_{j+1}, \mathbf{x}(t_{j+1})), \tag{1.10}$$

subject to $\mathbf{x}(t_j) = \mathbf{x_j}$ and the OCP constraints.

Dynamic programming algorithms use this recursive version of the optimal-cost-to-go function to compute an optimal solution backwards from the end of the time horizon to the beginning. Starting with $c(t_m, \mathbf{x_m}) = m(\mathbf{x}(t_m))$ in t_m, the problem (1.10) has to be solved on every subinterval $[t_j, t_{j+1}]$ for $j = m-1, \ldots, 1$ for each value $\mathbf{x}$ fulfilling the constraints. For continuous variables $\mathbf{x}$ this requires also a discretization of the state space. For each of the solved subproblems, the obtained cost function values and the corresponding solutions are stored for the use in the computation of the subproblem on the preceding time interval. This storing is called the "tabulation in state space".

Dynamic programming has one unique advantage compared to the other approaches presented later: As the search for the solution occurs in the entire state space, a global optimal solution of the OCP is found. Another property that is of special use in the context of feedback control algorithms for online optimization is that the tabulation of the subproblem solution corresponds to a precomputation of the optimal control moves for any given system state. Hence, the optimal control feedback can be obtained immediately after the measurement of the system state by a simple table look-up.
However, these advantages come at a very high price in terms of memory demand and run-time complexity of the approach which suffers from the so-called "curse of dimensionality": The run-time for a dynamic programming algorithm grows exponentially with the number of states and hence becomes prohibitively large already for medium size problems. Therefore dynamic programming is of practical use only for quite small systems.

If the size of the subintervals $[t_j, t_{j+1}]$ tends to zero, this leads to the Hamilton-Jacobi-Bellman equation. This is a partial differential equation that can be used to determine the optimal solution for continuous time systems. It shares in principle the advantages and disadvantages of the dynamic programming approach. For more information on dynamic programming and the Hamilton-Jacobi-Bellman equation we refer to [Bel57] and in the context of optimal control also to [Ber05, Ber07].

1.2.2 Indirect methods

The class of indirect methods for the solution of optimal control problems is based on Pontryagin's maximum principle and can be characterized as "first optimize - then discretize". In this classical approach the necessary conditions of optimality in the infinite-dimensional function space are used

to transform the OCP into a multipoint boundary value problem that has no degrees of freedom any more. The latter is then solved by suitable numerical methods, as for example multiple shooting [Osb69, Bul71, Deu74, Boc78a, Boc78b]. No discretization occurs until the boundary value problem is solved numerically. For some special (small) cases, also an analytical solution might be possible.
Indirect methods are able to compute the optimal control with a very high accuracy, as no approximation of the control occurs before the optimization step and the infinite-dimensional problem is solved. Since the transformation to a boundary value problem eliminates the degrees of freedom in the controls, indirect method offer some advantages for problems with many control functions compared to the number of states. Thus, the size of the problems scales linearly with the number of states, and hence the complexity is much better than in the case of direct programming or the use of the Hamilton-Jacobi-Bellman equation.
However, there are a number of drawbacks associated with the indirect approach. The derivation of the optimality conditions and the transformation into a boundary value problem require knowledge on the specific problem and often a lot of manual work. Furthermore, in the presence of general path and control constraints or interior point constraints, the solution structure is usually unknown in advance (see, e.g., in [Boc78a, Pes94, HSV95, Sag05] for more details). Even if the structure of the optimal solution has been determined once, it is very sensitive to small changes in the problem formulation, e.g., the addition of a constraint, but also to changes in the initial conditions or parameter values. Hence, in these cases the possibly lengthy derivation has to be repeated quite often. For systems with a larger number of states and controls the derivation may simply be impossible in practice.
For the successful numerical solution of the resulting boundary value problems not only a priori knowledge about the structure of the optimal solution is of importance. Also suitable initial values for all variables have to be available, such that the initial guesses lie inside the convergence region of the numerical method (usually a Newton-type method). This is often difficult, especially for the adjoint variables. Hence in some algorithms direct methods (see the following Section 1.2.3) are used as "starter" for indirect methods to generate the needed initial values for the variables [BNPvS91].
Note also that the solutions computed by indirect methods are, other than for dynamic programming, not necessarily global optima. Usually, they are only locally optimal.
Summarizing, it can be said that the solution of optimal control problems by indirect methods is in general an interactive process that requires knowledge about the specific problem such that it cannot be performed in a fully automated manner. As a result, indirect methods are usually not chosen as foundation of multiple purpose algorithms but are mostly applied if a high accuracy solution for a specific problem class is needed.

1.2.3 Direct methods

Direct methods, contrary to indirect methods, can be characterized by the expression "first discretize - then optimize". In direct methods the infinite-dimensional optimal control problem is transformed into a finite-dimensional Nonlinear Program (NLP). This resulting optimization problem is then solved using a suitable numerical algorithm for finite-dimensional optimization

as, e.g., the ones presented in the subsequent chapters. To perform the transformation, all direct methods employ a discretization of the control function $\mathbf{u}(\cdot)$. The different classes of direct methods are then distinguished based on their treatment of the system states $\mathbf{x}(\cdot)$ and $\mathbf{z}(\cdot)$, i.e., whether or not the system states are discretized. For a detailed comparison of the different classes of direct approaches for the solution of optimal control problems we also refer to [BBB+01].

Direct single shooting

In direct single shooting [HR71, SS78, Kra85] the control function $\mathbf{u}(\cdot)$ in (1.1) is replaced by a finite-dimensional discretization. This discretization can, e.g., be performed by first choosing a time grid $t_s = t_0 < t_1 < \ldots < t_{n_{\text{grid}}} = t_f$ on the horizon $[t_s, t_f]$. Then, on the subintervals defined by this grid, the control function is approximated piecewise by

$$\mathbf{u}(t) = \boldsymbol{\psi}_\mathbf{i}(t, \mathbf{u_i}) \quad \text{for } t \in [t_i, t_{i+1}], \tag{1.11}$$

where $\mathbf{u_i}$ is a finite-dimensional control parameter vector and $\boldsymbol{\psi}_\mathbf{i}$ is chosen typically as constant or linear in each component. At this level, the system states are not discretized and interpreted as variables that depend on the values of the controls (and of course on the model parameter and the initial values). To determine the states as a function of them the corresponding DAE Initial Value Problem (IVP) has to be solved. Thus, direct single shooting is called a sequential approach.

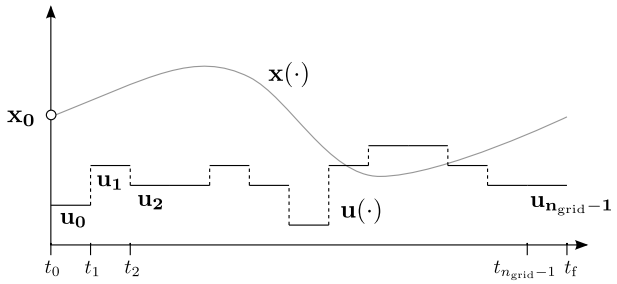

Figure 1.1: Illustration of a direct single shooting discretization of the optimal control problem for ordinary differential equations. Here, a piecewise constant discretization of the controls on an equidistant grid is used.

Based on the control discretization and the representation of the system states as a function of the control parameter, the original optimal control problem can be written as an NLP in the unknowns

$$(\mathbf{u_0}, \mathbf{u_1}, \ldots, \mathbf{u_{n_{\text{grid}}-1}}),$$

provided that no free parameter are present and that the initial value is fixed. The infinite-dimensional path and control constraints of the OCP are either discretized, and thus enforced only at the discretization points, or added to the cost functional in form of penalty terms. Other constraints such as interior point constraints can be transferred straightforward to the NLP context.

The advantages of the direct single shooting approach are that it leads to relatively small NLPs, as the only degrees of freedom are the control parameter, and that it is quite easy to implement, provided an NLP solver and a suitable numerical integration code are available. Furthermore, state-of-the-art adaptive DAE solvers can be used in this context, cf. Chapters 5 and 6, and easily exchanged to handle specific properties of the dynamic model.
On the other hand, there are a number of disadvantages of the direct single shooting approach. Depending on the nonlinearity and the stability of the dynamics, the initial guess for the control parameter might have to be chosen very close to the solution in order to guarantee the existence of a (numerical) solution of the DAE-IVP. It is possible, e.g., for unstable DAEs, that for many initial guesses for the control parameter no solution of DAE-IVP can be computed, either because of a singularity or due to numerical instabilities that lead to an explosion of the error during the integration of the IVP. In these cases, direct single shooting cannot be applied successfully in practice. Furthermore, no a priori knowledge on the state trajectory of the solution (e.g., in tracking or parameter estimation problems) can be used in solution process, because only the control parameter enter the problem and the state values are regarded as completely determined by them. The convergence rate of the NLP problem is often determined by the nonlinear dependence of the DAE solution at the end of the time horizon on the controls at the beginning, i.e., the nonlinearity "accumulated" over the whole time horizon (see also the discussion on the local convergence of unlifted methods in Section 4.3). This is different (and usually worse) than, e.g., in the case of direct collocation and direct multiple shooting, where this nonlinearity might be "distributed" and hence is effectively reduced.

Direct Collocation

Direct collocation for the solution of optimal control problems has first been proposed in [THE75] and was further developed, extended and applied, e.g., in [Bär83, Bie84, Str93, SBS98, KB06, KB08] and references therein. Here, not only the control $\mathbf{u}(\cdot)$ is discretized but also the system states $\mathbf{x}(\cdot)$ and $\mathbf{z}(\cdot)$. The approximation is in general based on polynomials and is done on a common fine grid with n_{grid} subintervals, where each subinterval contains n_{col} collocation points. The DAE in the OCP is then replaced by an n_{col}-point collocation formula on each subinterval which usually means that n_x+n_z (nonlinear) equality constraints are introduced in each collocation point on every interval. These conditions can usually be interpreted as a kind of numerical integration scheme on the intervals that ensures in any case that in the NLP solution the DAE is fulfilled in the collocation points. As the state trajectory is determined in parallel to the optimization, direct collocation is called a simultaneous or also an all-at-once approach. Continuous path and control constraints can be discretized analogously and point constraints can be added straightforward to the NLP.
Denoting the state values in interval i and collocation point j with $\mathbf{w}^{\mathbf{x}}_{\mathbf{i,j}}$ and $\mathbf{w}^{\mathbf{z}}_{\mathbf{i,j}}$ and analogously the control parameter values with $\mathbf{u}_{\mathbf{i,j}}$ we obtain a NLP in the variables

$$\left(\mathbf{w}^{\mathbf{x}}_{\mathbf{0,0}}, \mathbf{w}^{\mathbf{z}}_{\mathbf{0,0}}, \mathbf{u}_{\mathbf{0,0}}, \mathbf{w}^{\mathbf{x}}_{\mathbf{0,1}}, \mathbf{w}^{\mathbf{z}}_{\mathbf{0,1}}, \mathbf{u}_{\mathbf{0,1}}, \ldots, \mathbf{w}^{\mathbf{x}}_{\mathbf{n_{grid}-1,n_{col}-1}}, \mathbf{w}^{\mathbf{z}}_{\mathbf{n_{grid}-1,n_{col}-1}}, \mathbf{u}_{\mathbf{n_{grid}-1,n_{col}-1}}, \mathbf{w}^{\mathbf{x}}_{\mathbf{n_{grid},0}}, \mathbf{w}^{\mathbf{z}}_{\mathbf{n_{grid},0}}\right),$$

if no free parameter are present. Hence, the resulting NLP is usually huge, but also very sparse and structured. Despite its size it can be solved quite efficiently by sparsity exploiting NLP solvers,

see, e.g., [BGHBW03, WB06].

By the introduction of the discretized state values as variables to the NLP, a priori knowledge on the state trajectory of the optimal solution can be used. Furthermore, direct collocation is more robust against nonlinearities and numerical instabilities than direct single shooting as the propagation of errors over the whole time horizon is damped or even cut-off by the tolerance in the collocation matching conditions. Hence, it is also possible to treat unstable systems for which the OCP itself is well-posed. In a similar way this decoupling leads to a better distribution of the nonlinearity of the problem.
The main drawback of direct collocation methods is that they do not allow an adaptivity in time, i.e., in the process of the DAE solution, or at least not in a straightforward manner. The problem here is that each change in the time-stepping scheme leads to a change in the resulting NLP, usually even to changes of the dimensions of the NLP. However, a proper solution of highly nonlinear and stiff problems usually needs a very fine resolution in time at least on some parts of the time horizon. These regions are in general not known in advance, and hence very many gridpoints might be needed since in this case a fine discretization in time must be used on the whole time horizon. For large problems or long time horizons this might even not be possible in practice.
Furthermore, if the collocation scheme itself shall be changed, e.g., to "upgrade" it or to treat a different model class, this leads to a change in the underlying NLP structure, probably requiring manual adaptations to further ensure a proper structure exploitation in the algorithm. Therefore, the change of the scheme is usually significantly more difficult than changing the numerical integration method in direct single or multiple shooting.

Direct multiple shooting

Direct multiple shooting can be understood as a kind of hybrid approach between direct single shooting and direct collocation. In direct multiple shooting the controls are discretized and the state trajectories are parameterized, which is explained in the following. Direct multiple shooting goes back to the diploma thesis of Plitt [Pli81], supervised by H.G. Bock, and was first published in [BP84]. Extensions and applications of the idea in different fields can, e.g., be found in [FMT02, LBBS03, BP04, TBK04, Sch05, Sag05, Rie06]. An efficient implementation of the idea for the solution of optimal control problems is given by the code `MUSCOD-II` [DLS01].

In direct multiple shooting the control discretization is performed similar to direct single shooting (cf. (1.11)) by

$$\mathbf{u}(t) = \boldsymbol{\psi}_{\mathbf{i}}(t, \mathbf{u_i}) \quad \text{for } t \in [t_i, t_{i+1}]. \tag{1.12}$$

The choice that the finite-dimensional control parameter $\mathbf{u_i}$ influences the control approximation only locally on the corresponding subinterval contributes to the separability of the problem and the favorable overall structure of the resulting NLP.

Contrary to direct single shooting and different from direct collocation the state trajectories $\mathbf{x}(\cdot)$ and $\mathbf{z}(\cdot)$ are parameterized. This is done by introducing another grid, the so-called multiple

shooting grid. It can be chosen independently from the discretization grid of the controls and its gridpoints are called multiple shooting nodes. For notational convenience we assume here that both grids coincide. In each multiple shooting node new variables $\mathbf{w_i^x}$ and $\mathbf{w_i^z}$, the so-called node values, are introduced that represent the values of the differential and algebraic states $\mathbf{x}(t_i)$ and $\mathbf{z}(t_i)$ at the corresponding multiple shooting node. The value of the states at time points between the shooting nodes is obtained by the solution of a DAE-IVP on each multiple shooting interval, where the initial values are given by the corresponding node values $\mathbf{w_i^x}$ and $\mathbf{w_i^z}$ at the beginning of the interval.
The formulation of the algebraic equations is usually relaxed (cf.[BES88, SBS98, Lei99] and Section 5.3.7) to allow the integration of the DAE-IVP with inconsistent initial values for the algebraic variables. In exchange, the algebraic equations in the shooting nodes are added as equality constraints to the optimization problem to ensure consistency in the NLP solution. Furthermore, matching conditions

$$\mathbf{w_{i+1}^x} - \mathbf{x}(t_{i+1}; t_i, \mathbf{w_i^x}, \mathbf{w_i^z}) = 0, \quad 0 \leq i \leq n_{\text{grid}} - 1, \tag{1.13}$$

have to be added to the NLP to guarantee continuity of the differential state trajectories in the NLP solution. Here, $\mathbf{x}(t_{i+1}; t_i, \mathbf{w_i^x}, \mathbf{w_i^z})$ stands for the differential state at the time t_{i+1} of the solution of the DAE-IVP on the multiple shooting interval $[t_i, t_{i+1}]$ with initial values $\mathbf{w_i^x}$ and $\mathbf{w_i^z}$. Hence, like direct collocation, direct multiple shooting is an all-at-once approach that solves the dynamic model in parallel to the optimization. Note that the algebraic state trajectories are not necessarily continuous on the interval borders, even in the NLP solution. This is the case, e.g., if the algebraic equations depend on the controls and the discretized controls themselves are not continuous.

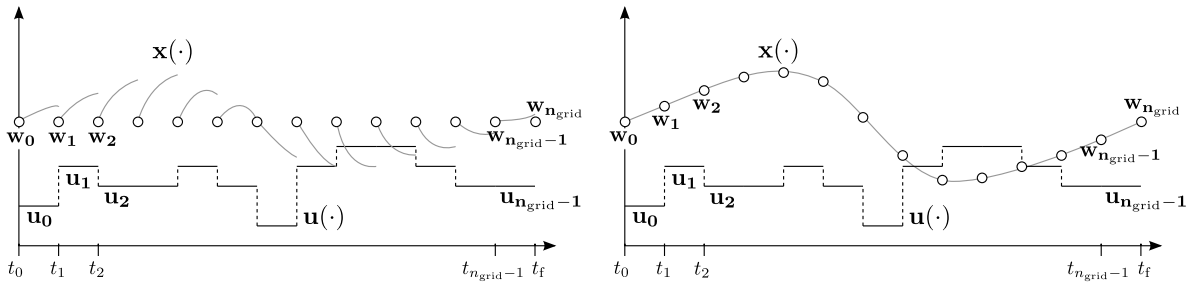

Figure 1.2: Illustration of a direct multiple shooting discretization of the optimal control problem for ordinary differential equations. Here a piecewise constant discretization of the controls on an equidistant grid is used. The left image shows the initialization of the problem where all node values are chosen identically and the resulting IVP solutions do not fulfill the matching (continuity) conditions. The right image shows the solution of NLP problem, where the matching conditions are fulfilled.

The continuous path constraints are discretized and added to the NLP. Usually, this is done on the multiple shooting grid and thus the constraints are enforced only in the multiple shooting nodes. In practice, however, this is normally already sufficient to ensure that the constraints are satisfied

on the whole time horizon, once an NLP solution has been found. If this is not the case, it can be overcome, e.g., by choosing a finer grid for the discretization of the controls. An alternative which makes use of semi-infinite programming techniques to track and eliminate violations of the constraints within the shooting intervals is presented in [PBS09]. Point constraints can also be transferred quite simply to the NLP formulation, even easier if the timepoints in which they are formulated are part of the multiple shooting grid.

The application of direct multiple shooting to the OCP then leads to an NLP in the variables

$$(\mathbf{w}_{\mathbf{0}}^{\mathbf{x}}, \mathbf{w}_{\mathbf{0}}^{\mathbf{z}}, \mathbf{u}_{\mathbf{0}}, \ldots, \mathbf{w}_{\mathbf{n}_{\mathrm{grid}}-\mathbf{1}}^{\mathbf{x}}, \mathbf{w}_{\mathbf{n}_{\mathrm{grid}}-\mathbf{1}}^{\mathbf{z}}, \mathbf{u}_{\mathbf{n}_{\mathrm{grid}}-\mathbf{1}}, \mathbf{w}_{\mathbf{n}_{\mathrm{grid}}}^{\mathbf{x}}, \mathbf{w}_{\mathbf{n}_{\mathrm{grid}}}^{\mathbf{z}}),$$

which is structured and sparse. This looks similar to the case of direct collocation, however, the number of gridpoints n_{grid} and hence the number of NLP variables is much smaller.

The resulting NLP can be solved efficiently by a suitable finite-dimensional NLP solver, e.g., a structure exploiting SQP algorithm like the one presented in Chapter 7. Structure exploitation can be performed on several algorithmic levels. On the level of the QP subproblems, structure exploitation can be done, e.g., on the basis of the condensing algorithm presented in [Pli81, BP84], which efficiently reduces the size of the quadratic subproblem to be solved in each step to the size of the QP in the single shooting case. This is preferable if the number of control parameter is relatively small in comparison to the number of nodes (compare the description of condensing in the case of lifted methods in Chapters 4 and 7).
Another possibility is to directly exploit the sparsity structure of the SQP subproblem for its solution as described in [Ste95, Ste02, KBSS11]. This is preferable if the number of control parameter is relatively large compared to the number of node variables and/or for special problem classes arising from mixed integer optimal control.
Further developments of structure exploitation (based on the condensing approach) include, e.g., the projection onto an invariant manifold [SBS98], the usage of point constraints to eliminate degrees of freedom from the problem [Sch88, Sch05, AD10] as well as parallelization of the approach based on the natural decoupling of the problem on the individual multiple shooting intervals [GB94, Rie01].

Due to the additional degrees of freedom introduced by the state parametrization, direct multiple shooting shares the favorable properties of direct collocation: The resulting algorithms are stable and able to treat unstable systems as well as highly nonlinear problems. A priori knowledge about the states in the optimal solution can be used for the initialization of the node values. Contrary to direct single shooting, the convergence of the NLP solver is not governed by the "accumulated" nonlinearity over the whole time horizon but more by the "maximum" of the nonlinearities on the individual multiple shooting intervals (see also the discussion on local convergence of lifted methods in Section 4.3). By this "distribution" of the nonlinearity the "effective" nonlinearity of the NLP is often reduced.
A big advantage compared to direct collocation is the possibility to use adaptive state-of-the-art integrators for the solution of the DAE-IVP on the shooting intervals. These integrators can also be exchanged easily in case of improvements or according to the specific dynamic model class.

2 Derivative generation

In all theoretical strategies and numerical algorithms presented in this thesis, derivative information of the involved functions play a central role. The efficient generation of function derivatives is hence a very important issue. This is not only the case for the presented optimization algorithms but also for the presented numerical methods for the solution of Initial Value Problems (IVPs) for Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs) as well as the computation of sensitivities of these solutions. The derivative information needed throughout this thesis includes directional derivatives, dense and sparse Jacobian matrices, gradients as well as higher-order (directional) derivatives. This chapter explains different strategies to obtain these derivatives and compares advantages and disadvantages as well as the efficiency of the different approaches.

2.1 Symbolic differentiation

Symbolic differentiation, also called analytical differentiation, assumes that a symbolic expression of the function to be differentiated, i.e., an explicit formula describing the output of the function in terms of the function's inputs, is available. Based on this expression differentiation rules like product, quotient and chain rule are applied stepwise to manipulate the expression and to finally obtain a symbolic expression for the derivative. This procedure can be iterated to obtain expressions for higher-order derivatives. The whole process can be done manually, or by a computer-algebra software like Maple [Map09] or Mathematica [Res08]. Afterwards, the resulting expression has to be implemented in a programming language to be evaluated by the computer. Some computer algebra programs also offer the possibility to directly export a source code for the symbolic derivative expression. The strength of this approach is that, provided it is executed correctly, it leads to derivative values without truncation errors, i.e., they are only subject to round-off errors. Furthermore, one obtains an expression for the derivative mapping, not only the evaluation of the derivative in a certain point. There are however some drawbacks: First, there needs to exist a symbolic expression of the function. In practice it might be the case that the function which has to be differentiated is only given as a potentially large piece of computer code, from which a symbolic expression is difficult to obtain. Even if a symbolic expression is available, a manual manipulation to obtain the derivative of a large and complex function is tedious and error-prone, while using a computer algebra software may lead to correct but not efficient expressions for the derivative. A classical example to illustrate the latter is the following product function of Speelpenning [Spe80].

Example 2.1 (Speelpenning's function)
We define the scalar valued function $f : \mathbb{R}^n \to \mathbb{R}$ as

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i. \tag{2.1}$$

By using symbolic differentiation we obtain the gradient $\nabla \mathbf{f}(x_1 \ldots, x_n) \in \mathbb{R}^n$

$$\nabla \mathbf{f}(x_1 \ldots, x_n) = \left(\prod_{i \neq j} x_i \right)_{j=1,\ldots,n} = \begin{pmatrix} x_2 \cdot x_3 \cdots x_n \\ x_1 \cdot x_3 \cdots x_n \\ \vdots \\ x_1 \cdot x_2 \cdots x_{n-1} \end{pmatrix}. \tag{2.2}$$

This symbolic expression for the gradient contains a lot of common subexpressions. They are usually not automatically exploited when the gradient is implemented based on this formula. Hence the evaluation of the gradient will not be efficient. It takes about $n - 1$ times the effort of the function evaluation of f itself. A similar effort is needed to compute one directional derivative of f. For higher-order derivatives this problem also exists, e.g., the symbolic computation of the Hessian of f yields the expression

$$\begin{aligned} \mathbf{H} &= \left(\frac{\partial^2}{\partial x_i \partial x_j} \right)_{i=1,\ldots,n, j=1,\ldots,n} \\ &= \begin{pmatrix} 0 & \prod_{i\neq 1, i\neq 2} x_i & \prod_{i\neq 1, i\neq 3} x_i & \cdots & \prod_{i\neq 1, i\neq n} x_i \\ \prod_{i\neq 2, i\neq 1} x_i & 0 & \prod_{i\neq 2, i\neq 3} x_i & \cdots & \prod_{i\neq 2, i\neq n} x_i \\ \vdots & & \ddots & & \vdots \\ \prod_{i\neq n, i\neq 1} x_i & \cdots & & \prod_{i\neq n, i\neq n-1} x_i & 0 \end{pmatrix} \end{aligned} \tag{2.3}$$

Again, there are a lot of common subexpressions which are usually not automatically exploited in an implementation. The computation of the Hessian based on this expression takes $\frac{(n^2-n)(n-3)}{2}$ multiplications and therefore approximately $\frac{n(n-3)}{2}$ times the effort of the function evaluation of f. Note that we already exploited the symmetry of the Hessian in this example.

Summarizing, we can say that the computation of derivatives using symbolic differentiation, e.g., using a computer algebra package, is a reliable method to compute derivatives with a high accuracy, provided a symbolic expression for the function is accessible. To obtain an efficient implementation, however, often a postprocessing and optimization of the obtained expressions would be necessary.

2.2 Finite differences

The idea to approximate the derivative of a given scalar function $f(x) : \mathbb{R} \to \mathbb{R}$ at a point $x_0 \in \mathbb{R}$ by finite differences is based on the Taylor expansion of f around x_0

$$f(x_0 + h) = f(x_0) + \frac{\partial f}{\partial x}(x_0)h + \mathcal{O}(h^2). \tag{2.4}$$

This leads to the approximation of the derivative by one-sided finite differences

$$\frac{\partial f}{\partial x}(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + \mathcal{O}(h). \tag{2.5}$$

Developing f twice around x_0 with increments h and $-h$ leads to the central finite difference scheme

$$\frac{\partial f}{\partial x}(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \mathcal{O}(h^2). \tag{2.6}$$

Both schemes can be transferred directly to compute an approximation of a directional derivative of a function $\mathbf{g} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ at a point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$ in direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ by

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x_0}) \cdot \mathbf{d} = \frac{\mathbf{g}(\mathbf{x_0} + h\mathbf{d}) - \mathbf{g}(\mathbf{x_0})}{h} + \mathcal{O}(h) \tag{2.7}$$

or

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x_0}) \cdot \mathbf{d} = \frac{\mathbf{g}(\mathbf{x_0} + h\mathbf{d}) - \mathbf{g}(\mathbf{x_0} - h\mathbf{d})}{2h} + \mathcal{O}(h^2). \tag{2.8}$$

One big advantage of the finite differences approach to compute derivative approximations is that it is very easy to implement. Furthermore, no knowledge about the inner structure of f is needed, because a black-box procedure to evaluate f (or $\mathbf{g}$) is sufficient. Also the computation of a directional derivative is very cheap, as only 2 function evaluations are needed. Hence, the Jacobian is available at the expense of $n_{\text{indep}} + 1$ function evaluations using the one-sided scheme and $2n_{\text{indep}}$ function evaluations using the central scheme, respectively. On the other hand, this approach suffers from a lack of accuracy of the computed derivative approximations. Furthermore, the accuracy depends crucially on the choice of the increment h. If h is chosen large, the higher-order terms become significant and accuracy is lost due to truncation errors. If h is chosen small, the cancellation errors increase. Even an optimal choice of h, which itself depends strongly on the function, will typically lead to derivative approximations with an accuracy of only about $\frac{1}{2}$ (one-sided scheme) or $\frac{2}{3}$ (central scheme) of the significant digits of the underlying function evaluation. This is depicted in Figure 2.1 for the example of $f(x) = e^x$ and $x_0 = 1$ with double precision arithmetic.
By comparison with the Taylor series one can derive more accurate (in terms of the truncation error) approximations, but they require more function evaluations and suffer in principle from the same problem. This is also true for finite difference schemes for second and higher-order derivatives. In this case the accuracy problem increases further.

2.3 Complex step derivative approximation

The idea of using complex arithmetic to compute numerical approximations of first and higher-order derivatives of real functions can first be found in Lyness [Lyn67] and Lyness and Moler [LM67]. By comparison with the Taylor series using a complex increment Squire and Trapp [ST98] gave a simple formula to approximate a directional derivative of a function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ at

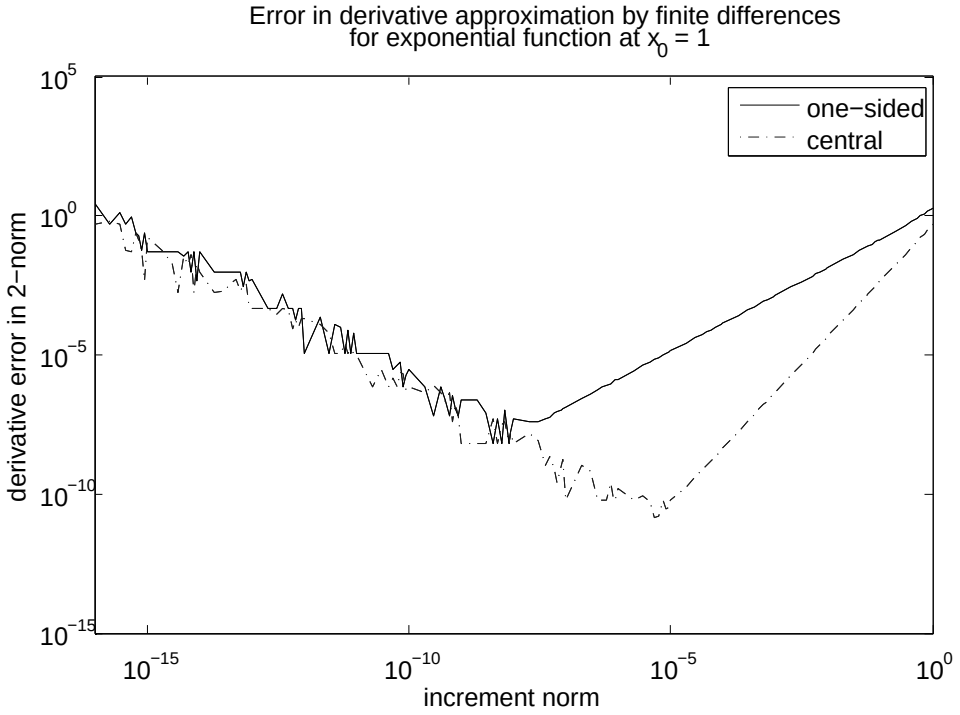

Figure 2.1: The error made in the derivative approximation of $f(x) = e^x$ at $x_0 = 1$ using the finite differences approach. Depicted is the Euclidean norm of the error for different choices of the increment h for both the one-sided (2.5) and the central (2.6) finite difference scheme using double precision arithmetic. Even for the optimal choice of h about half respectively one-third of the significant digits are lost.

a point $\mathbf{x_0}$ in direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x_0}) \cdot \mathbf{d} = \Im\left(\frac{\mathbf{f}(\mathbf{x_0} + ih\mathbf{d})}{h}\right) + \mathcal{O}(h^2). \tag{2.9}$$

Usually this approach is referred to as complex step method. At first sight it seems somewhat similar to the finite differences approach presented above. However this approach does not suffer from cancellation errors for small h. Hence h can and should be chosen very small, e.g., $h = 2^{-256} \approx 10^{-77}$. A comparison of the errors made in the evaluation of the derivative of the exponential function by this approach and one-sided finite differences is given in Figure 2.2. While for small h the finite difference approach looses accuracy due to cancellation errors, the complex approach returns for $h < 10^{-8}$ the derivative within machine precision. Martins et al. [MSA03] and Newman et al. [NAW98] noted that the complex step method is in the end equivalent to the first order forward mode of automatic differentiation explained in the next section. The complex step method can be straightforward extended to the computation of second order derivatives, but then suffers from cancellation errors like the finite difference approach. Recent works of Lai [Lai06] and Ridout [Rid09] give suggestions for improved second order derivative approximations based on the complex step method. Overall, the complex step method gives a simple possibility to obtain accurate derivative information, even if the function is given only as a black-box algorithm. A

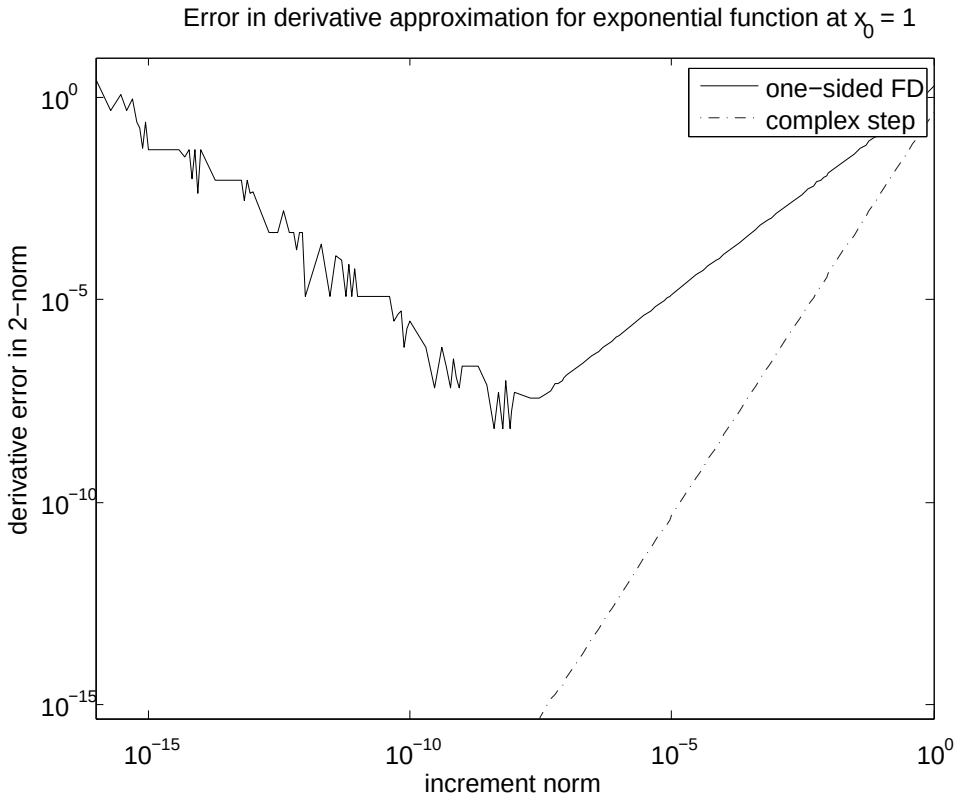

Figure 2.2: The error made in the derivative approximation of $f(x) = e^x$ at $x_0 = 1$ using the complex step method (2.9) and the one-sided finite differences approach (2.5) in double precision arithmetic. Depicted is the Euclidean norm of the error for different choices of the increment h. While the finite differences approach is subject to cancellation errors for small h, the complex step method is not and delivers for $h < 10^{-8}$ a derivative approximation within machine precision.

prerequisite is, of course, that the platform where the complex step methods is to be implemented supports complex arithmetic. Also all operations made during the evaluation of the function $\mathbf{f}$ have to support complex arguments. This might sometimes not be the case if, e.g., the evaluation of $\mathbf{f}$ involves calls to the `abs` or `max` functions, linear algebra packages, integration routines or in general third-party software. If this prerequisite is fulfilled, the complex step method delivers a directional derivative at the cost of one function evaluation of $\mathbf{f}$ with complex argument and in complex arithmetic. The efficiency depends strongly on the implementation of complex arithmetic on the target platform, e.g., if hardware support is present or a software emulation of complex operations is necessary. For some implementations there exist also problems if h is chosen too small, as was pointed out by Martins et al. [MSA03]. If the implementation of complex arithmetic is efficient and reliable, the complex step approach will be nearly as fast as the finite differences approach and can be recommended as an easy-to-use mean of computing at least first order directional derivatives.

2.4 Automatic differentiation

In this section we describe the powerful technique of automatic differentiation (AD). As we use its ideas and concepts on several other occasions in the different algorithms and strategies developed in this thesis, it is described here in detail. First the basic ideas and concepts of AD are presented. Then we discuss algorithmic schemes for first order derivative generation using the so-called forward and reverse modes of AD. Afterwards we give an introduction to the not so commonly known concept of Taylor coefficient (TC) propagation for the efficient computation of higher-order derivatives. At the end we discuss possibilities how the presented ideas can be implemented in practice and give an overview on the approaches of generating sparse derivative matrices in the case of large but structured systems. The presentation and notation in this section is mainly inspired by the book of Griewank [Gri00], but sometimes modified and extended where needed.

2.4.1 Basic concepts

The underlying idea of automatic differentiation (sometimes also referred to as algorithmic differentiation or computational differentiation) is the decomposition of the function to be differentiated into a sequence of elemental functions and the systematic application of the chain rule known from basic calculus. Other than in the approach of symbolic differentiation presented in Section 2.1 here the chain rule is not applied to manipulate symbolic expressions but works on numerical values. As the chain rule is known since Leibniz and Newton and hence is a part of the basic calculus curriculum since many years, AD has been rediscovered and reinvented several times in different contexts and applications. Here the works of Wengert [Wen64] and Kedem [Ked80] are sometimes cited as pioneer works in the field. For a more complete overview of AD history we refer to [Iri91, Gri00].
In this section we will consider functions of type $\mathbf{f} : D \subset \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$, $\mathbf{y} = \mathbf{f}(\mathbf{x})$, mapping the "independent" variables $\mathbf{x}$ to the "dependent" variables $\mathbf{y}$. We assume further that the evaluation of $\mathbf{f}$ can be decomposed into a sequence of elemental functions, leading from the independent variables via intermediate variables to the dependent variables. This is usually true for most functions whose evaluation can be coded as a computer program. To formulate this property mathematically we use the following definition of a factorable function, which is slightly modified compared to the one given in [Ked80].

Definition 2.2 (Factorable function)
Let $\mathcal{L}$ be a set of real valued functions taking one or more real arguments, the so-called elemental functions. A function $\mathbf{f} : D \subset \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$, $\mathbf{y} = \mathbf{f}(\mathbf{x})$, $\mathbf{x} = (x_1, \ldots, x_{n_{\text{indep}}})$, $\mathbf{y} = (y_1, \ldots, y_{n_{\text{dep}}})$ is a factorable function if there exists a finite sequence of real valued functions $\varphi_{1-n_{\text{indep}}}, \ldots, \varphi_k$, such that the following conditions are satisfied:

- $\varphi_{i-n_{\text{indep}}} \equiv \pi_i^{n_{\text{indep}}}$, *for* $1 \leq i \leq n_{\text{indep}}$,
- $\varphi_{k-n_{\text{dep}}+i} \equiv \pi_i^{n_{\text{dep}}} \circ \mathbf{f}$, *for* $1 \leq i \leq n_{\text{dep}}$,

- *for all* $1 \leq j \leq k - n_{\text{dep}}$ *the function* φ_j *is either a constant function or a composition of an elemental function with one or more functions* φ_l *with* $1 - n_{\text{indep}} \leq l \leq j - 1$*, i.e., appearing earlier in the sequence.*

Here $\pi_i^n : \mathbb{R}^n \to \mathbb{R}, (z_1, \ldots, z_n) \mapsto z_i$ *means the projection onto the* i*-th component of a vector. If* $\mathbf{f}$ *is factorable, we call the sequence* $\varphi_{1-n_{\text{indep}}}, \ldots, \varphi_k$ *an elemental representation of* $\mathbf{f}$*.* $\mathcal{L}$ *is then called the elemental library.*

Note that there is mathematically no fundamental difference if the elemental functions would have been defined as vector valued. This is sometimes of advantage in the practical setup, e.g., to treat linear algebra operations such as matrix-vector products on a higher level. But for the development of the theory we stick for notational simplicity to scalar valued elemental functions. For a factorable function $\mathbf{f}$ we can write down a sequence of instructions describing the evaluation of the function for a given input $\mathbf{x}$ using elemental functions via intermediate values. To simplify the notation we define the following dependency relation.

Definition 2.3 (Dependency relation)
Assume that $\mathbf{f}$ *is factorable and* $\varphi_{1-n_{\text{indep}}}, \ldots, \varphi_k$ *is an elemental representation of* f*. We denote by* v_i *the intermediate quantity that occurs as output of the elemental function* φ_i *during the evaluation of* $\mathbf{f}$*. This means that* v_i *is computed by* φ_i *from a set of arguments containing the intermediate value* v_j *with* $j < i$*.*
The dependency relation $\prec$ *is then defined as*

$$\begin{aligned} j \prec i \quad &\Leftrightarrow \quad v_i \text{ depends directly on } v_j \\ &\Leftrightarrow \quad v_j \text{ is an argument of } \varphi_i. \end{aligned} \tag{2.10}$$

Using the dependency relation we can write the general evaluation procedure, also called a (zero order) forward sweep, for a factorable function $\mathbf{f}$ with input $\mathbf{x}$ and output $\mathbf{y}$ in the form of Table 2.1.

$v_{i-n_{\text{indep}}}$	$=$	x_i	$i = 1, \ldots, n_{\text{indep}}$
v_i	$=$	$\varphi_i(v_j)_{j \prec i}$	$i = 1, \ldots, k$
$y_{n_{\text{dep}}-i}$	$=$	v_{k-i}	$i = n_{\text{dep}} - 1, \ldots, 0$

Table 2.1: General evaluation procedure ("zero order forward sweep") for a factorable function with independent variables $\mathbf{x}$, intermediate variables $\mathbf{v}$ and dependent variables $\mathbf{y}$.

Another useful possibility to describe the evaluation of $\mathbf{f}$ is an acyclic graph visualizing the dependency relations in the function evaluation, the so-called computational graph. We elaborate on this with the following example.

Example 2.4 (Evaluation procedure and computational graph)
Consider the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^2$ defined as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}. \tag{2.11}$$

Then $\mathbf{f}$ is factorable and the evaluation procedure can be written as given in Table 2.2:

v_{-2}	$=$	x_1
v_{-1}	$=$	x_2
v_0	$=$	x_3
v_1	$=$	$e^{(v_{-2})}$
v_2	$=$	$v_{-1} + v_0$
v_3	$=$	$\sqrt{v_2}$
v_4	$=$	$v_1 v_2$
v_5	$=$	$\sin(v_{-1})$
v_6	$=$	$v_4 + v_5$
v_7	$=$	$v_5 - v_3$
y_1	$=$	v_6
y_2	$=$	v_7

Table 2.2: The evaluation procedure for the evaluation of the function $\mathbf{f}$ given in (2.11) using a chain of elemental functions.

The corresponding computational graph of $\mathbf{f}$ for this elemental representation is depicted in Figure 2.3.

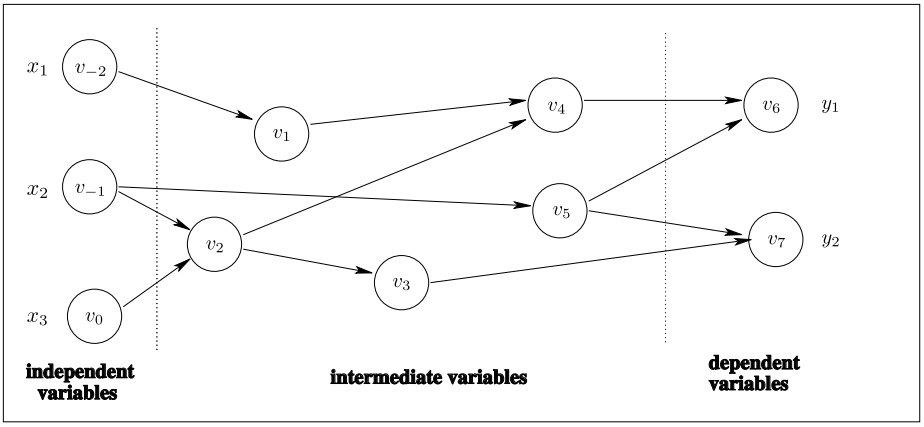

Figure 2.3: The computational graph for the evaluation of the function $\mathbf{f}$ given in (2.11) using its elemental representation given in Table 2.2.

A numerical evaluation of $\mathbf{f}$ using the representation in Table 2.2, e.g., at the point $\mathbf{x} = (0, 0, 1)^T$ takes the operations given in Table 2.3.

Now that we can write down the evaluation of $\mathbf{f}$ systematically in terms of elemental functions we consider the question how to differentiate $\mathbf{f}$ efficiently. First, we define the property that assures the differentiability of the used elemental functions.

v_{-2}	$=$	0
v_{-1}	$=$	0
v_0	$=$	1
v_1	$=$	e^0
v_2	$=$	$0+1$
v_3	$=$	$\sqrt{1}$
v_4	$=$	$1 \cdot 1$
v_5	$=$	$\sin(0)$
v_6	$=$	$1+0$
v_7	$=$	$0-1$
y_1	$=$	1
y_2	$=$	-1

Table 2.3: The evaluation of the function $\mathbf{f}$ given in (2.11) at point $\mathbf{x} = (0,0,1)^T$ using the elemental representation given by Table 2.2.

Definition 2.5 (Elemental differentiability)
We say that the elemental library $\mathcal{L}$ fulfills the assumption of elemental differentiability (ED) of order k if and only if all elemental functions $\varphi_i \in \mathcal{L}$ are k times continuously differentiable on their open domains D_i, i.e., $\varphi_i \in \mathcal{C}^k(D_i, \mathbb{R})$, $0 \leq k \leq \infty$.

Based on this definition we obtain immediately the following result regarding the differentiability of $\mathbf{f}$. For a proof, we refer to [Gri00].

Proposition 2.6
Let $\mathbf{f}$ be a factorable function with a representation of elements out of the elemental library $\mathcal{L}$. Assume that $\mathcal{L}$ fulfills the assumption ED of order k. Then the set D of points $\mathbf{x} \in D$ for which the function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is well defined by the evaluation procedure given in Table 2.1 forms an open subset of $\mathbb{R}^{n_{\text{indep}}}$ and $\mathbf{f} \in \mathcal{C}^k(D, \mathbb{R}^{n_{\text{dep}}})$.

If not indicated otherwise we assume in the following that the functions are factorable and that the elemental library $\mathcal{L}$ fulfills the assumption ED with an order $k \geq 1$. This means that, e.g., $\{+, -, *, /, \exp, \sin, \cos\} \subset \mathcal{L}$, but $\{\max, \min, \text{abs}\} \cap \mathcal{L} = \emptyset$.

2.4.2 First order derivatives

Based on the elemental representation of $\mathbf{f}$ we now derive algorithmic schemes to compute first order derivatives of $\mathbf{f}$. This is done by differentiating the general evaluation procedure in Table 2.1 and applying the chain rule.
Depending on whether the application of the chain rule is done in the direction of the function evaluation procedure or opposite to it, we speak of the forward or the reverse mode of AD.

Forward mode

The forward mode of AD is used to compute for a given evaluation point $\mathbf{x}$ and a given direction $\dot{\mathbf{x}} \in \mathbb{R}^{n_{\text{indep}}}$ the directional derivative

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \dot{\mathbf{x}}. \tag{2.12}$$

It should be noted that usually, in contrast to symbolic differentiation, the Jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$ is not built explicitly and afterwards multiplied by $\dot{\mathbf{x}}$ when AD is used for the computation of a directional derivative. This would only be efficient if a larger number of directional derivatives at the same point $\mathbf{x}$ is needed. Instead by applying the general tangent procedure, also referred to as first order forward sweep, given in Table 2.4, we directly compute the directional derivative.

$[v_{i-n_{\text{indep}}}, \dot{v}_{i-n_{\text{indep}}}]$	$=$	$[x_i, \dot{x}_i]$	$\mid$	$i = 1, \ldots, n_{\text{indep}}$
$[v_i, \dot{v}_i]$	$=$	$[\varphi_i(v_j)_{j \prec i}\ ,\ \sum_{j \prec i} \frac{\partial}{\partial v_j} \varphi_i \dot{v}_j]$	$\mid$	$i = 1, \ldots, k$
$[y_{n_{\text{dep}}-i}, \dot{y}_{n_{\text{dep}}-i}]$	$=$	$[v_{k-i}, \dot{v}_{k-i}]$	$\mid$	$i = n_{\text{dep}} - 1, \ldots, 0$

Table 2.4: General tangent procedure ("first order forward sweep") for a factorable function with independent variables $\mathbf{x}$, intermediate variables $\mathbf{v}$ and dependent variables $\mathbf{y}$. Here $\dot{\mathbf{x}}$ describes the derivative direction, $\dot{v}_i$ the derivatives of the intermediate quantities and $\dot{\mathbf{y}}$ the directional derivative.

The forward sweep is initialized with the independent variables $\mathbf{x}$ and the derivative direction $\dot{\mathbf{x}}$. Then step by step the derivative is computed simultaneously to the evaluation of the function itself. This is achieved by accumulating in every intermediate quantity $\dot{v}_i$ the derivative information from all quantities that have directly contributed to v_i, until the end of the evaluation is reached. This means that after the accumulation of an intermediate variable v_i is completed, $\dot{v}_i$ holds the directional derivative of v_i in direction $\dot{\mathbf{x}}$. The simultaneous treatment of function and derivative evaluation is of advantage because usually, except in case that φ_i is constant or linear, for the evaluation of the partial derivatives $c_{ij} := \frac{\partial \varphi_i}{\partial v_j}$ the values of either v_i or v_j are needed. If the evaluation of the derivative is not done simultaneously to that of the function, we call this a pure (first order) forward derivative sweep. Here either the intermediate variables have to be recalculated or some of the intermediate variables (alternatively the c_{ij}) that have been stored somewhere during a prior function evaluation, have to be retrieved. The first order forward sweep can easily be extended to compute several directional derivatives simultaneously with the function. The overall effort of computing p first order directional derivatives at once using the forward mode of AD can be theoretically bounded from above by $1+1.5p$ times the effort of a function evaluation (cf. [Gri00]). Compared to the finite differences approach this is a slightly higher effort. On the other side, the AD approach is only subject to round-off errors and hence produces derivative approximations within machine precision. We give now an example of a practical application of the forward mode by computing the Jacobian of the function of Example 2.4.

Example 2.7 (Jacobian computation using the forward mode)
Consider the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^2$ defined as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}.$$

We now want to evaluate the Jacobian $\mathbf{J}$ of $\mathbf{f}$

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} e^{x_1}(x_2 + x_3) & e^{x_1} + \cos(x_2) & \mathrm{e}^{x_1} \\ 0 & \cos(x_2) - \frac{1}{2\sqrt{x_2+x_3}} & -\frac{1}{2\sqrt{x_2+x_3}} \end{pmatrix} \tag{2.13}$$

at point $\mathbf{x} = (0, 0, 1)^T$ using the forward mode of AD. We achieve this by computing the $n_{\text{indep}} = 3$ directional derivatives along the Cartesian coordinates, each one yielding one column of the Jacobian, i.e., we choose $\dot{\mathbf{x}}_1 = (1, 0, 0)^T$, $\dot{\mathbf{x}}_2 = (0, 1, 0)^T$ and $\dot{\mathbf{x}}_3 = (0, 0, 1)^T$. The forward sweep is then performed abstractly by the following calculations for each derivative direction:

Function evaluation			Derivative evaluation		
v_{-2}	$=$	x_1	$\dot{v}_{-2}$	$=$	$\dot{x}_1$
v_{-1}	$=$	x_2	$\dot{v}_{-1}$	$=$	$\dot{x}_2$
v_0	$=$	x_3	$\dot{v}_0$	$=$	$\dot{x}_3$
v_1	$=$	$e^{v_{-2}}$	$\dot{v}_1$	$=$	$v_1\dot{v}_{-2}$
v_2	$=$	$v_{-1} + v_0$	$\dot{v}_2$	$=$	$\dot{v}_{-1} + \dot{v}_0$
v_3	$=$	$\sqrt{v_2}$	$\dot{v}_3$	$=$	$\dot{v}_2/(2v_3)$
v_4	$=$	$v_1 v_2$	$\dot{v}_4$	$=$	$\dot{v}_1 v_2 + v_1\dot{v}_2$
v_5	$=$	$\sin(v_{-1})$	$\dot{v}_5$	$=$	$\cos(v_{-1})\dot{v}_{-1}$
v_6	$=$	$v_4 + v_5$	$\dot{v}_6$	$=$	$\dot{v}_4 + \dot{v}_5$
v_7	$=$	$v_5 - v_3$	$\dot{v}_7$	$=$	$\dot{v}_5 - \dot{v}_3$
y_1	$=$	v_6	$\dot{y}_1$	$=$	$\dot{v}_6$
y_2	$=$	v_7	$\dot{y}_2$	$=$	$\dot{v}_7$

In practice only the actual numerical values are propagated and this is done for all directions simultaneously. The superscripts in the follwing computations denote the direction to which a quantity belongs.

Function evaluation			Der. eval. dir. 1			Der. eval. dir. 2			Der. eval. dir. 3		
v_{-2}	=	0	$\dot{v}_{-2}^1$	=	1	$\dot{v}_{-2}^2$	=	0	$\dot{v}_{-2}^3$	=	0
v_{-1}	=	0	$\dot{v}_{-1}^1$	=	0	$\dot{v}_{-1}^2$	=	1	$\dot{v}_{-1}^3$	=	0
v_0	=	1	$\dot{v}_0^1$	=	0	$\dot{v}_0^2$	=	0	$\dot{v}_0^3$	=	1
v_1	=	e^0	$\dot{v}_1^1$	=	$1 \cdot 1$	$\dot{v}_1^2$	=	$1 \cdot 0$	$\dot{v}_1^3$	=	$1 \cdot 0$
v_2	=	$0+1$	$\dot{v}_2^1$	=	$0+0$	$\dot{v}_2^2$	=	$1+0$	$\dot{v}_2^3$	=	$0+1$
v_3	=	$\sqrt{1}$	$\dot{v}_3^1$	=	$0/(2 \cdot 1)$	$\dot{v}_3^2$	=	$1/(2 \cdot 1)$	$\dot{v}_3^3$	=	$1/(2 \cdot 1)$
v_4	=	$1 \cdot 1$	$\dot{v}_4^1$	=	$1 \cdot 1 + 1 \cdot 0$	$\dot{v}_4^2$	=	$0 \cdot 1 + 1 \cdot 1$	$\dot{v}_4^3$	=	$0 \cdot 1 + 1 \cdot 1$
v_5	=	$\sin(0)$	$\dot{v}_5^1$	=	$\cos(0) \cdot 0$	$\dot{v}_5^2$	=	$\cos(0) \cdot 1$	$\dot{v}_5^3$	=	$\cos(0) \cdot 0$
v_6	=	$1+0$	$\dot{v}_6^1$	=	$1+0$	$\dot{v}_6^2$	=	$1+1$	$\dot{v}_6^3$	=	$1+0$
v_7	=	$0-1$	$\dot{v}_7^1$	=	$0-0$	$\dot{v}_7^2$	=	$1-1/2$	$\dot{v}_7^3$	=	$0-1/2$
y_1	=	1	$\dot{y}_1^1$	=	1	$\dot{y}_1^2$	=	2	$\dot{y}_1^3$	=	1
y_2	=	-1	$\dot{y}_2^1$	=	0	$\dot{y}_2^2$	=	$1/2$	$\dot{y}_2^3$	=	$-1/2$

In the end, we have computed along with the function value $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (1, -1)^T$ the correct Jacobian $J(\mathbf{x})$ at point $\mathbf{x} = (0, 0, 1)^T$

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} 1 & 2 & 1 \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix} \tag{2.14}$$

using $n_{\text{indep}} = 3$ directional derivatives. It should be noted that the intermediate variables v_i of the function evaluation and also the partial derivatives of the φ_i only had to be computed once for the whole set of directions.

Summarizing, the forward mode of AD allows the efficient computation of directional derivatives of a given function with slightly higher effort than the finite differences approach. As in all AD-based approaches the function cannot be treated as a black box. At least a piece of computer code describing the evaluation must be available. However, the forward mode computes derivatives within machine precision and does not rely on the support and the efficiency of complex arithmetic on the specific platform or the algorithms used. The computation of the Jacobian of $\mathbf{f}$ using the forward mode requires n_{indep} directional derivatives of $\mathbf{f}$, like the finite differences approach and the complex step method.

Reverse mode

The reverse mode of AD is used to compute for a given evaluation point $\mathbf{x}$ and a given so-called adjoint direction $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$ the so-called adjoint directional derivative

$$\bar{\mathbf{x}}^T = \bar{\mathbf{y}}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}). \tag{2.15}$$

Again, this is done through differentiation of the general evaluation procedure in Table 2.1 using the chain rule. However, other than in the forward mode the order in which the chain rule

is applied is reverse to the evaluation procedure, i.e., starting at the dependent variables and going back until the independent variables are reached. We call this a first order reverse sweep. Once again, for the computation of the partial derivatives of some elemental functions φ_i the values of the corresponding intermediate variables v_i are needed. More specifically, the arguments of nonlinear elemental functions and the results of the power and the exponential function are needed during a reverse sweep. Hence in general a reverse sweep is preceded by a zero order forward sweep computing the values of the intermediate variables in the function evaluation and storing the ones needed later on a so-called tape. Alternatively, if the evaluation point $\mathbf{x}$ has not changed, also the taped intermediate values of an earlier forward sweep can be used. Table 2.5 shows the computations for a general reverse sweep in the variant with a preceding zero order forward sweep. We show here the incremental version where the accumulation in the intermediate variable $\bar{v}_i$ is done stepwise for each operation occurring in the function evaluation. The reverse sweep could also be formulated nonincrementally, i.e., by grouping the accumulation operations by variables and summing up directly all contributions to a single $\bar{v}_i$.

$\bar{v}_i$	$=$	0		$i = 1 - n_{\text{indep}}, \ldots, k - n_{\text{dep}}$
$v_{i-n_{\text{indep}}}$	$=$	x_i		$i = 1, \ldots, n_{\text{indep}}$
v_i	$=$	$\varphi_i(v_j)_{j \prec i}$		$i = 1, \ldots, k$
$y_{n_{\text{dep}}-i}$	$=$	v_{k-i}		$i = n_{\text{dep}} - 1, \ldots, 0$
$\bar{v}_{k-i}$	$=$	$\bar{y}_{n_{\text{dep}}-i}$		$i = 0, \ldots, n_{\text{dep}} - 1$
$\bar{v}_j$	$+=$	$\bar{v}_i \frac{\partial}{\partial v_j} \varphi_i$	$\forall j \prec i$	$i = k, \ldots, 1$
$\bar{x}_i$	$=$	$\bar{v}_{i-n_{\text{indep}}}$		$i = n_{\text{indep}}, \ldots, 1$

Table 2.5: General incremental first order reverse sweep for a factorable function with independent variables $\mathbf{x}$, intermediate variables $\mathbf{v}$ and dependent variables $\mathbf{y}$. Here $\bar{\mathbf{y}}$ describes the adjoint derivative direction, $\bar{v}_i$ the intermediate adjoint quantities and $\bar{\mathbf{x}}$ the adjoint derivative. The reverse sweep is preceded by a zero order forward sweep for the computation of the intermediate values $\mathbf{v}$ of the function evaluation that are needed during the reverse sweep. At the very beginning, the intermediate adjoint variables $\bar{v}_i$ are initialized to zero. Note that the $+=$ here stands for the add-assign operation: $u+= v$ is equivalent to $u = u + v$.

The reverse sweep is initialized with the adjoint direction $\bar{\mathbf{y}}$. During the reverse sweep for each intermediate quantity $\bar{v}_j$ derivative information from all values v_i to which v_j has contributed to is accumulated incrementally until, at the end of the procedure, $\bar{\mathbf{x}}$ contains the value of the adjoint derivative. Like for the forward sweep, the procedure can be easily extended to compute several adjoint directional derivatives simultaneously, thereby making efficient use of common terms. For the computation of p adjoint directional derivatives including the preceding zero order forward sweep, an upper complexity bound of $1.5 + 2.5p$ times the cost of the function evaluation itself can be shown (cf. [Gri00]). In this estimate the cost of accessing the tape is neglected, which is a reasonable assumption if the whole tape can be stored in the main memory or even in the cache. If this is not the case, other strategies like checkpointing (cf. Walther [Wal00]) may be advisable to obtain more efficient schemes. Apart from this, it is very important to note that this bound does

not depend on the number of independent variables n_{indep}. This makes the reverse mode of AD extremely useful for the computation of gradients or Jacobians of functions with $n_{\text{dep}} \ll n_{\text{indep}}$. We illustrate now the reverse mode in detail using the well-known function of Example 2.4 and show afterwards the full power of the reverse mode for gradient computation using the example of Speelpenning's function (2.1).

Example 2.8 (Jacobian computation using the reverse mode)
Consider again the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^2$ defined as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}.$$

We now want to compute the Jacobian $\mathbf{J}$ of $\mathbf{f}$ at point $\mathbf{x} = (0, 0, 1)^T$ using the reverse mode of AD. We do this by computing the $n_{\text{dep}} = 2$ adjoint directional derivatives along the Cartesian coordinates in the target space, each one yielding one row of the Jacobian, i.e., we choose $\bar{\mathbf{y}}_1 = (1, 0)^T$ and $\bar{\mathbf{y}}_2 = (0, 1)^T$. The reverse sweep (with preceding forward sweep) for $\mathbf{f}$ is then abstractly performed by the calculations given in Table 2.6 for each adjoint derivative direction $\bar{\mathbf{y}}$. Note that we assume that the $\bar{v}_i$ are initialized with zero and that first the instructions on the left side are executed (forward sweep) and afterwards the ones on the right (reverse sweep). In the

Forward sweep			Reverse sweep		
v_{-2}	$=$	x_1	$\bar{v}_7$	$=$	$\bar{y}_2$
v_{-1}	$=$	x_2	$\bar{v}_6$	$=$	$\bar{y}_1$
v_0	$=$	x_3	$\bar{v}_5$	$+=$	$\bar{v}_7$
			$\bar{v}_3$	$+=$	$-\bar{v}_7$
v_1	$=$	$e^{v_{-2}}$	$\bar{v}_4$	$+=$	$\bar{v}_6$
v_2	$=$	$v_{-1} + v_0$	$\bar{v}_5$	$+=$	$\bar{v}_6$
v_3	$=$	$\sqrt{v_2}$	$\bar{v}_{-1}$	$+=$	$\bar{v}_5 \cos(v_{-1})$
v_4	$=$	$v_1 v_2$	$\bar{v}_1$	$+=$	$\bar{v}_4 v_2$
v_5	$=$	$\sin(v_{-1})$	$\bar{v}_2$	$+=$	$\bar{v}_4 v_1$
v_6	$=$	$v_4 + v_5$	$\bar{v}_2$	$+=$	$\bar{v}_3/(2v_3)$
v_7	$=$	$v_5 - v_3$	$\bar{v}_{-1}$	$+=$	$\bar{v}_2$
			$\bar{v}_0$	$+=$	$\bar{v}_2$
			$\bar{v}_{-2}$	$+=$	$\bar{v}_1 v_1$
			$\bar{x}_3$	$=$	$\bar{v}_0$
y_1	$=$	v_6	$\bar{x}_2$	$=$	$\bar{v}_{-1}$
y_2	$=$	v_7	$\bar{x}_1$	$=$	$\bar{v}_{-2}$

Table 2.6: Abstract forward and reverse sweep for $\mathbf{f}$ of Example 2.8. We see that in the reverse sweep only intermediate values v_i are needed that have been arguments of nonlinear elementals or that were the results of power or exponential function.

actual computation only the numerical values are propagated and this is done for all directions simultaneously. The superscripts in the following computations denote the adjoint direction to which a quantity belongs. We again assume that the intermediate adjoint quantities $\bar{v}_i^j$ have been initialized to zero. In the end we have computed along with the function value $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (1, -1)^T$

Forward sweep			Rev. sweep dir. 1			Rev. sweep dir. 2		
v_{-2}	=	0	$\bar{v}_7^1$	=	1	$\bar{v}_7^2$	=	0
v_{-1}	=	0	$\bar{v}_6^1$	=	0	$\bar{v}_6^2$	=	1
v_0	=	1	$\bar{v}_5^1$	+=	1	$\bar{v}_5^2$	+=	0
			$\bar{v}_3^1$	+=	-1	$\bar{v}_3^2$	+=	0
v_1	=	e^0	$\bar{v}_4^1$	+=	0	$\bar{v}_4^2$	+=	1
v_2	=	$0+1$	$\bar{v}_5^1$	+=	0	$\bar{v}_5^2$	+=	1
v_3	=	$\sqrt{1}$	$\bar{v}_{-1}^1$	+=	$1 \cdot 1$	$\bar{v}_{-1}^2$	+=	$1 \cdot 1$
v_4	=	$1 \cdot 1$	$\bar{v}_1^1$	+=	$0 \cdot 1$	$\bar{v}_1^2$	+=	$1 \cdot 1$
v_5	=	$\sin(0)$	$\bar{v}_2^1$	+=	$0 \cdot 1$	$\bar{v}_2^2$	+=	$1 \cdot 1$
v_6	=	$1+0$	$\bar{v}_2^1$	+=	$-1/(2 \cdot 1)$	$\bar{v}_2^2$	+=	$0/(2 \cdot 1)$
v_7	=	$0-1$	$\bar{v}_{-1}^1$	+=	$-1/2$	$\bar{v}_{-1}^2$	+=	1
			$\bar{v}_0^1$	+=	$-1/2$	$\bar{v}_0^2$	+=	1
			$\bar{v}_{-2}^1$	+=	$0 \cdot 1$	$\bar{v}_{-2}^2$	+=	$1 \cdot 1$
			$\bar{x}_3^1$	=	$-1/2$	$\bar{x}_3^2$	=	1
y_1	=	1	$\bar{x}_2^1$	=	$1/2$	$\bar{x}_2^2$	=	2
y_2	=	-1	$\bar{x}_1^1$	=	0	$\bar{x}_1^2$	=	1

Table 2.7: Actual forward and reverse sweep for $\mathbf{f}$ of Example 2.8 using $n_{\text{dep}} = 2$ adjoint directions for the computation of the Jacobian at point $x = (0, 0, 1)^T$. We see that for each adjoint unit direction one row of the Jacobian is computed.

the Jacobian $J(\mathbf{x})$ at point $\mathbf{x} = (0, 0, 1)^T$ row-wise by $n_{\text{dep}} = 2$ adjoint directional derivatives. Like for the forward mode it should be noted that the intermediate values of the function evaluation and the partial derivatives of the φ_i only had to be computed once for the whole set of directions.

Example 2.9 (Gradient computation of Speelpennings's function)
Consider now Speelpenning's function from (2.1) that maps from $\mathbb{R}^n$ to $\mathbb{R}$:

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i.$$

To compute the gradient of f at point $\mathbf{x}$ we need one adjoint directional derivative with adjoint direction $\bar{y}_1 = 1$. The first order reverse sweep with preceding zero order forward sweep then has the form

Forward sweep			Reverse sweep		
v_{1-n}	$=$	x_1	$\bar{v}_{n-1}$	$=$	$\bar{y}_1$
	$\vdots$		$\bar{v}_{n-2}$	$+=$	$\bar{v}_{n-1}v_0$
v_0	$=$	x_n	$\bar{v}_0$	$+=$	$\bar{v}_{n-1}v_{n-2}$
v_1	$=$	$v_{2-n}v_{1-n}$	$\bar{v}_{n-3}$	$+=$	$\bar{v}_{n-2}v_{-1}$
v_2	$=$	v_1v_{3-n}	$\bar{v}_{-1}$	$+=$	$\bar{v}_{n-2}v_{-3}$
	$\vdots$			$\vdots$	
v_{n-1}	$=$	$v_{n-2}v_0$	$\bar{v}_{2-n}$	$+=$	$\bar{v}_1v_{1-n}$
			$\bar{v}_{1-n}$	$+=$	$\bar{v}_1v_{2-n}$
			$\bar{x}_n$	$=$	$\bar{v}_0$
				$\vdots$	
y_1	$=$	v_{n-1}	$\bar{x}_1$	$=$	$\bar{v}_{1-n}$

We observe that for the zero order forward sweep, i.e., the function evaluation, $n-1$ multiplications are needed, while the reverse sweep needs $2(n-1)$ multiplications. This means the effort of a gradient computation for Speelpenning's function using the reverse mode costs roughly 3 times a function evaluation, which is below the theoretical bound stated above. Furthermore, we see that this is true for any number of independent variables $n_{\text{indep}} = n$. The independence of the cost of a gradient computation with respect to the number of independent variables is an advantage that is unique to the reverse mode, because for all other derivative generation approaches the effort of a gradient computation grows at least linearly with n_{indep}.

Summarizing, the reverse mode of AD is the only derivative generation approach that allows the computation of first order adjoint directional derivatives, i.e., linear combinations of rows of the Jacobian. This allows the computation of a Jacobian with n_{dep} adjoint directional derivatives and the computation of the gradient of a scalar function using only one adjoint direction. Furthermore, the cost of the adjoint directional derivative can be bounded in terms of the function evaluation itself and is independent of the number of independent variables, which makes it very efficient to compute gradient type derivative information and, e.g., Jacobians if $n_{\text{dep}} \ll n_{\text{indep}}$. Like the forward mode, it is only subject to round-off errors and hence computes derivatives within machine precision. The drawback is that some of the intermediate quantities of the function evaluation have to be stored on a tape which leads to a larger memory consumption. If for very complex functions the tape does not fit into main memory any more, the reverse sweep might loose a lot of efficiency and checkpointing strategies might have to be considered. As in all approaches based on AD, at least a program code describing the evaluation of the function must be available to apply the reverse mode.

2.4.3 Higher-order derivative generation

So far we have described how to obtain first order directional derivatives using the idea of automatic differentiation. We now present two approaches for the efficient computation of higher-order derivatives based on Automatic Differentiation (AD). We first explain some shortcomings of an intuitive extension of first order schemes and introduce afterwards the approach of Taylor Coefficient (TC) propagation that allows the development of efficient arbitrary order schemes.

Repeated application of first order schemes

The first approach is based on the fact that the first order forward and reverse sweeps themselves can be viewed as factorable functions. Hence, AD could be applied to differentiate them in order to obtain schemes for higher-order derivatives. We explain this at the example of deriving a first order reverse sweep once more using the forward and the reverse mode, respectively, to obtain second order derivatives.

Example 2.10 (Second order derivatives by deriving reverse scheme)
Consider the general function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$. Applying the reverse mode once for the point $\mathbf{x}$ and the adjoint direction $\bar{\mathbf{y}}$ leads to a scheme that can be interpreted as a function $\bar{\mathbf{f}} : \mathbb{R}^{n_{\text{indep}}} \times \mathbb{R}^{n_{\text{dep}}} \to \mathbb{R}^{n_{\text{dep}}} \times \mathbb{R}^{n_{\text{indep}}}, (\mathbf{x}, \bar{\mathbf{y}}) \mapsto (\mathbf{y}, \bar{\mathbf{x}})$

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \tag{2.16a}$$

$$\bar{\mathbf{x}}^T = \bar{\mathbf{y}}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}). \tag{2.16b}$$

To this function, again the forward or the reverse mode can be applied. An application of the reverse mode for the point $(\mathbf{x}, \bar{\mathbf{y}})$ and the adjoint direction $(\tilde{\mathbf{y}}, \tilde{\bar{\mathbf{x}}})$ leads to a function $\bar{\bar{\mathbf{f}}} : \mathbb{R}^{n_{\text{indep}}+n_{\text{dep}}+n_{\text{dep}}+n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}+n_{\text{indep}}+n_{\text{dep}}+n_{\text{indep}}}, (\mathbf{x}, \bar{\mathbf{y}}, \tilde{\mathbf{y}}, \tilde{\bar{\mathbf{x}}}) \mapsto (\mathbf{y}, \bar{\mathbf{x}}, \tilde{\bar{\mathbf{y}}}, \tilde{\mathbf{x}})$ which calculates besides (2.16) also

$$\tilde{\bar{\mathbf{y}}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) \tilde{\bar{\mathbf{x}}} \tag{2.17a}$$

$$\tilde{\mathbf{x}}^T = \bar{\mathbf{y}}^T \frac{\partial^2 \mathbf{f}}{\partial \mathbf{x}^2}(\mathbf{x}) \tilde{\bar{\mathbf{x}}} + \tilde{\mathbf{y}}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}). \tag{2.17b}$$

On the other hand, an application of the forward mode to $\bar{\mathbf{f}}$ for the point $(\mathbf{x}, \bar{\mathbf{y}})$ and the forward direction $(\dot{\mathbf{x}}, \dot{\bar{\mathbf{y}}})$ leads to a function $\dot{\bar{\mathbf{f}}} : \mathbb{R}^{n_{\text{indep}}+n_{\text{dep}}+n_{\text{indep}}+n_{\text{dep}}} \to \mathbb{R}^{n_{\text{dep}}+n_{\text{indep}}+n_{\text{dep}}+n_{\text{indep}}}, (\mathbf{x}, \bar{\mathbf{y}}, \dot{\mathbf{x}}, \dot{\bar{\mathbf{y}}}) \mapsto (\mathbf{y}, \bar{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\bar{\mathbf{x}}})$ that computes besides (2.16) also

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) \dot{\mathbf{x}} \tag{2.18a}$$

$$\dot{\bar{\mathbf{x}}}^T = \bar{\mathbf{y}}^T \frac{\partial^2 \mathbf{f}}{\partial \mathbf{x}^2}(\mathbf{x}) \dot{\mathbf{x}} + \dot{\bar{\mathbf{y}}}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}). \tag{2.18b}$$

From this example we learn two things that also hold for the other AD-based schemes for higher-order derivatives presented later. First, we see that for the computation of the Hessian of a scalar

function a complexity bound independent of n_{indep}, like for the gradient computation using the reverse mode, cannot be established. To compute a Hessian, in general at least n_{indep} calls of $\bar{\bar{\mathbf{f}}}$ or $\dot{\bar{\mathbf{f}}}$ are needed. Second, we observe that if we identify $\dot{\mathbf{x}} \equiv \bar{\tilde{\mathbf{x}}}$ and $\dot{\tilde{\mathbf{y}}} \equiv \tilde{\mathbf{y}}$ in (2.17) and (2.18) then we obtain $\bar{\tilde{\mathbf{y}}} = \dot{\mathbf{y}}$ and $\tilde{\mathbf{x}} = \dot{\bar{\mathbf{x}}}$. This means that the quantities obtained by a repeated application of the reverse sweep can also be obtained by a combination of a forward and a reverse sweep. Repeating the previous process for $\dot{\bar{\mathbf{f}}}$ and so on shows that this is also true for derivatives of order greater than 2. Note that every application of the reverse mode leads to another pair of a forward and reverse sweeps. The length of these sweeps increases with each use, as the forward sweep in the n-th order scheme consists of all computations made to obtain the results of the scheme with order $(n-1)$. This finally results in a complexity that is exponentially increasing with the derivative order of the scheme. As a consequence, a repeated application of the reverse mode should be avoided. While the derivation of an efficient second order forward/reverse scheme for $\dot{\bar{\mathbf{f}}}$ is still relatively easy, the derivation and implementation of efficient schemes for derivatives of arbitrary order based on the first order sweeps is complex and error-prone, sometimes even for a specific, explicitly given function. To overcome this difficulty we now discuss an alternative approach to compute higher-order derivatives. This approach is based on the propagation of a truncated Taylor series through the function that shall be differentiated and allows the development of schemes with quadratic complexity in the derivative order.

Taylor coefficient propagation

In this section we describe the approach of Taylor coefficient (or also Taylor polynomial) propagation that allows a compact formulation of efficient schemes for the computation of derivative tensors of arbitrary order, or specific parts of them. We explain here a variant that allows an efficient implementation due to very regular data access patterns. The first step on this way is the forward propagation of an univariate Taylor polynomial through a function evaluation, as already described, e.g., in [Moo66, Wan69, Ral81]. Based on this, higher-order (univariate) directional derivatives can be obtained by simple scaling of the coefficients of the propagated polynomials. As an extension a reverse propagation scheme for the computation of higher-order adjoint derivatives can be derived. To finally obtain elements of higher-order derivative tensors in the multivariate case, first a family of univariate Taylor polynomials is propagated through the evaluation of the function. From the propagated quantities the elements of the derivative tensors can be obtained by so-called exact interpolation, which we describe at the end of this section. Finally we present practical examples of the actual application of the higher-order forward and reverse schemes to facilitate a better understanding of the approach.

Univariate Taylor coefficient propagation The first step is the computation of univariate higher-order directional derivatives. Here we separate between pure forward and forward/adjoint directional derivatives.

Definition 2.11 (Higher-order directional derivatives)
Let $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ *be an at least* $k+1$ *times continuously differentiable function. We then*

define a univariate forward directional derivative of order k in a direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ by

$$\dot{\mathbf{f}}^{(\mathbf{k})}(\mathbf{x}, \mathbf{d}) := \frac{\partial^k \mathbf{f}}{\partial \mathbf{x}^k}(\mathbf{x})\mathbf{d}^k = \left.\frac{\partial^k}{\partial t^k}\mathbf{f}(\mathbf{x} + t\mathbf{d})\right|_{t=0} \in \mathbb{R}^{n_{\text{dep}}} \tag{2.19}$$

and a univariate forward/adjoint directional derivative of order $k+1$ in (forward) direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ and adjoint direction $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$ by

$$\begin{aligned} \dot{\bar{\mathbf{f}}}^{(\mathbf{k})}(\bar{\mathbf{y}}, \mathbf{x}, \mathbf{d}) &:= \bar{\mathbf{y}}^T \frac{\partial}{\partial \mathbf{x}} \dot{\mathbf{f}}^{(\mathbf{k})}(\mathbf{x}, \mathbf{d})(\mathbf{x}) \\ &= \bar{\mathbf{y}}^T \frac{\partial^{k+1}}{\partial \mathbf{x}^{k+1}} \mathbf{f}(\mathbf{x})\mathbf{d}^k \\ &= \left.\frac{\partial^k}{\partial t^k} \bar{\mathbf{y}}^T \frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x} + t\mathbf{d})\right|_{t=0} \in \mathbb{R}^{n_{\text{indep}}}. \end{aligned} \tag{2.20}$$

Here the multiplications with $\mathbf{d}^k$ are to be understood as k contractions with the direction $\mathbf{d}$ in the domain space of the derivative tensor. In the forward case, where the degree of the derivative tensor equals k, this leads to a vector in $\mathbb{R}^{n_{\text{dep}}}$. In the forward/adjoint case, where the degree of the derivative tensor is $k+1$, the combination with one contraction with $\bar{\mathbf{y}}$ in the range space leads to a vector in $\mathbb{R}^{n_{\text{indep}}}$.

We first address the issue of computing higher-order forward derivatives of type $\dot{\mathbf{f}}^{(\mathbf{k})}(\mathbf{x}, \mathbf{d})$ by propagating a Taylor polynomial $\mathbf{x}(t)$ of degree k

$$\mathbf{x}(t) = \mathbf{x_0} + \mathbf{x_1}t + \mathbf{x_2}t^2 + \ldots + \mathbf{x_k}t^k \in \mathbb{R}^{n_{\text{indep}}} \tag{2.21}$$

forward through the function $\mathbf{f}$, i.e., we want derive from $\mathbf{x}(t)$ and $\mathbf{f}$ a Taylor polynomial $\mathbf{y}(t)$ of degree k fulfilling

$$\mathbf{y}(t) = \mathbf{y_0} + \mathbf{y_1}t + \mathbf{y_2}t^2 + \ldots + \mathbf{y_k}t^k = \mathbf{f}(\mathbf{x}(t)) + \mathcal{O}(t^{k+1}) \in \mathbb{R}^{n_{\text{dep}}}. \tag{2.22}$$

Here the $\mathbf{x_i} \in \mathbb{R}^{n_{\text{indep}}}, \mathbf{y_i} \in \mathbb{R}^{n_{\text{dep}}}$ are called Taylor coefficients of degree i or also the i-th Taylor coefficients of $\mathbf{x}(t)$ and $\mathbf{y}(t)$. We will use the abbreviation TC synonymously also for the whole set of Taylor coefficients $\mathbf{x_0}, \ldots, \mathbf{x_k}$ that describes a Taylor polynomial, and for which we use the matrix-like notation $\mathbf{X} = [\mathbf{x_0}, \ldots, \mathbf{x_k}] \in \mathbb{R}^{n_{\text{indep}} \times (k+1)}$. We use $\mathbf{X}$ and $\mathbf{x}(t)$ synonymously, depending on the context. The Taylor coefficient of degree i of component j of $\mathbf{x}(t)$ is then indexed by $X_{j,i}$. We use the notation $\mathbf{Y} = \mathbf{f}(\mathbf{X})$ for the propagation of TCs through $\mathbf{f}$. The dependency of the TC $\mathbf{y_i}$ on the TC $\mathbf{x_j}$ can be expressed in functional form as follows.

Definition 2.12 (Taylor coefficient function)
Assuming elemental differentiability of order k we define the Taylor coefficient functions $\mathbf{f_i} : \mathbb{R}^{n_{\text{indep}} \times (i+1)} \to \mathbb{R}^{n_{\text{dep}}}$, $i \leq k$,

$$\mathbf{y_i} = \mathbf{f_i}(\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_i}) \tag{2.23}$$

as the relation of the corresponding quantities in (2.21) and (2.22). We also write shorter $\mathbf{y_i} = \mathbf{f_i}(\mathbf{X})$ and hence allow formally more arguments to be present that might not be used in the evaluation of $\mathbf{f_i}$. Note that due to the elemental differentiability of order k the Taylor coefficient function $\mathbf{f_i}$ is at least $k-i$-times continuously differentiable.

If we assume that we know how to evaluate the $\mathbf{f_i}$ for an input $\mathbf{X}$, i.e., how to propagate $\mathbf{x}(t)$ through $\mathbf{f}$ and compute $\mathbf{y}(t)$, then we can compute the higher-order directional derivative $\dot{\mathbf{f}}^{(\mathbf{k})}(\mathbf{x_0}, \mathbf{d})$ at a point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$ and in direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ simply by a rescaling of the k-th Taylor coefficient $\mathbf{y_k}$ of the propagated Taylor polynomial for the input $\mathbf{x}(t) = \mathbf{x_0} + t\mathbf{d}$.

Proposition 2.13 (Forward directional derivatives by univariate TC propagation)
Let $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ be an at least $k \geq 1$ times continuously differentiable function. Denote with $\mathbf{y}(t)$ the propagated Taylor polynomial fulfilling (2.22) for the input polynomial $\mathbf{x}(t)$ of degree k with Taylor coefficients $\mathbf{X}$, such that $\mathbf{y_i} = \mathbf{f_i}(\mathbf{X})$ for $0 \leq i \leq k$. Then the univariate forward directional derivative of order k at a point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$ and in direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ is obtained by

$$\dot{\mathbf{f}}^{(\mathbf{k})}(\mathbf{x_0}, \mathbf{d}) = k!\, \mathbf{y_k} = k!\, \mathbf{f_k}(\mathbf{X}), \tag{2.24}$$

where the TCs of the input polynomial are chosen as $\mathbf{X} = [\mathbf{x_0}, \mathbf{d}, \mathbf{0}, \ldots, \mathbf{0}] \in \mathbb{R}^{n_{\text{indep}} \times (k+1)}$.

Proof:

Using the input polynomial $\mathbf{x}(t) = \mathbf{x_0} + t\mathbf{d}$ and differentiating (2.22) k-times with respect to t leads to

$$k! \mathbf{y_k} = \frac{\partial^k \mathbf{f}}{\partial \mathbf{x}^k}(\mathbf{x_0} + t\mathbf{d})\, \mathbf{d}^k + \mathcal{O}(t), \tag{2.25}$$

where the multiplication with $\mathbf{d}^k$ is again to be understood as tensor contractions. Setting $t = 0$ then gives the desired result and completes the proof.

□

TC forward propagation rules To compute the Taylor coefficient functions $\mathbf{f_i}$ and the propagated coefficients $\mathbf{y_i}$, respectively, we have to propagate Taylor coefficients through the elemental functions respectively the corresponding intermediate variables. Hence we have to define Taylor polynomials also for the intermediate variables v.

Definition 2.14 (Taylor polynomials of intermediate variables)
For a given k-times continuously differentiable input $\mathbf{x}(t) : (-\epsilon, \epsilon) \to \mathbb{R}^{n_{\text{indep}}}$, $\epsilon > 0$, we denote the corresponding intermediate variables by $v(\mathbf{x}(t)) : (-\epsilon, \epsilon) \to \mathbb{R}$ and define its Taylor polynomial as

$$v(t) = v_0 + v_1 t + v_2 t^2 + \ldots + v_k t^k = v(\mathbf{x}(t)) + \mathcal{O}(t^k). \tag{2.26}$$

We denote the vector of Taylor coefficients corresponding to an intermediate variable v_i with $\mathbf{v_i} = [v_{i,0}, \ldots, v_{i,k}] \in \mathbb{R}^{1 \times (k+1)}$.

With the previous definitions we can now describe rules for the propagation of Taylor coefficients through elemental functions. In the following we restrict ourselves to scalar functions working on scalar arguments u and w. There are mainly two possibilities to obtain forward propagation rules for Taylor coefficients. For basic operations like addition, multiplication or division the rules can

be derived by (truncated) polynomial arithmetic. Consider for example the Taylor polynomials $u(t)$ and $w(t)$ of degree 2, then we obtain in polynomial arithmetic $v(t) = u(t)w(t)$ as

$$\begin{aligned} v(t) &= (u_0 + u_1 t + u_2 t^2)(w_0 + w_1 t + w_2 t^2) \\ &= u_0 w_0 + (u_1 + w_1)t + (u_0 w_2 + u_1 w_1 + u_2 w_0)t^2 + \mathcal{O}(t^3). \end{aligned} \tag{2.27}$$

In truncated polynomial arithmetic the terms with order greater than 2 are dropped and hence the Taylor coefficients of $v(t)$ can be expressed as $v_0 = u_0 w_0$, $v_1 = u_1 w_0 + u_0 w_1$ and $v_2 = u_0 w_2 + u_1 w_1 + u_2 w_0$. Note that the coefficient v_0 is the result of the corresponding "ordinary" multiplication on real arguments. This is true for all elemental operations, i.e., if $v(t) = \varphi(u(t), w(t))$ then $v_0 = \varphi(u_0, w_0)$. Also note that computations leading to the first order coefficient v_1 are equal to the computations in the first order forward sweep presented earlier. Also this is true in general, such that first order Taylor propagation is equivalent to a first order forward sweep.
To construct propagation rules for the univariate nonlinear elemental functions of interest, such as exp, sin and cos, we can use the interpretation of the elemental function as solution of a linear ODE. All elemental functions $v = \varphi(u)$ satisfy an equation of the type

$$b(u)\frac{\partial \varphi(u)}{\partial u} - a(u)\varphi(u) = c(u), \tag{2.28}$$

where the coefficients $a(u)$, $b(u)$ and $c(u)$ can be computed in terms of arithmetic operation and univariate elemental functions for which the propagation rules are already known. This means that the Taylor coefficients a_i, b_i, c_i of the elementals $a(u)$, $b(u)$ and $c(u)$ can be computed from the coefficients u_i of the input. Examples would be the exponential $\varphi(u) = e^u$ with coefficients $a(u) = b(u) = 1$ and $c(u) = 0$ or the power function $\varphi(u) = u^r$ for which $a(u) = r$, $b(u) = u$ and $c(u) = 0$ holds.
To obtain now the propagation rules, we differentiate $v(\mathbf{x}(t)) = \varphi(u(\mathbf{x}(t)))$ for a given input $\mathbf{x}(t)$ with respect to t and multiply with $b\big(u(\mathbf{x}(t))\big)$. This leads to

$$b\big(u(\mathbf{x}(t))\big)\frac{\mathrm{d}v(\mathbf{x}(t))}{\mathrm{d}t} = \Big(c\big(u(\mathbf{x}(t))\big) + a\big(u(\mathbf{x}(t))\big)v(\mathbf{x}(t))\Big)\frac{\mathrm{d}u(\mathbf{x}(t))}{\mathrm{d}t}. \tag{2.29}$$

If we assume ED up to order k then $u(\mathbf{x}(t))$ and $v(\mathbf{x}(t))$ are at least k-times continuously differentiable with the expansion for the derivative of $u(\mathbf{x}(t))$

$$\frac{\mathrm{d}u(\mathbf{x}(t))}{\mathrm{d}t} = u_1 + 2tu_2 + \ldots + kt^{(k-1)}u_k + \mathcal{O}(t^k) \tag{2.30}$$

and analogously for $v(\mathbf{x}(t))$. Substituting these into the equation above and comparing coefficients leads to the following proposition given in [Gri00].

Proposition 2.15 (Taylor polynomials of ODE solutions)
Under the condition that $b_0 \equiv b(u_0) \neq 0$ it holds that

$$v_i = \frac{1}{ib_0}\left(\sum_{j=1}^{i}(c_{i-j} + e_{i-j})ju_j - \sum_{j=1}^{i-1} b_{i-j}jv_j\right), \quad \text{for } i = 1, \ldots, k, \tag{2.31}$$

with

$$e_i := \sum_{j=0}^{i} a_j v_{i-j}, \quad \text{for } j = 1, \ldots, k-1. \tag{2.32}$$

The rules for the propagation of Taylor polynomials through the most common elemental functions are shown in Table 2.8. Note that the operation count for the propagation of Taylor coefficients of

Operation	Propagation rule for $i = 1, \ldots, k$
$v = u + cv,\ c \in \mathbb{R}$	$v_i = u_i + cw_i$
$v = uw$	$v_i = \sum_{j=0}^{i} u_j w_{i-j}$
$v = u/w$	$v_i = \frac{1}{w_0}\left(u_i - \sum_{j=0}^{i-1} v_j w_{i-j}\right)$
$v = \sqrt{u}$	$v_i = \frac{1}{2v_0}\left(u_i - \sum_{j=1}^{i-1} v_j v_{i-j}\right)$
$v = \log u$	$v_i = \frac{1}{iu_0}\left(iu_i - \sum_{j=1}^{i-1} u_{i-j} j v_j\right)$
$v = e^u$	$v_i = \frac{1}{i}\sum_{j=1}^{i} v_{i-j} j u_j$
$v = u^r$	$v_i = \frac{1}{iu_0}\left(r\sum_{j=1}^{i} v_{i-j} j u_j - \sum_{j=1}^{i-1} u_{i-j} j v_j\right)$
$s = \sin(u)$ $c = \cos(u)$	$s_i = \frac{1}{i}\sum_{j=1}^{i} c_{i-j} j u_j$ $c_i = -\frac{1}{i}\sum_{j=1}^{i} s_{i-j} j u_j$

Table 2.8: Propagation of Taylor coefficients through elemental functions. While the first four rules are easily derived by truncated polynomial arithmetic similar to (2.27), the last four are obtained via their interpretation as ODE solutions via Proposition 2.15.

degree k is in general $\mathcal{O}(k^2)$. Only scalar multiplications and additions are significantly cheaper with a complexity of $\mathcal{O}(k)$. Based on the presented propagation rules for elemental functions the forward propagation of TCs through an arbitrary factorable function $\mathbf{f}$ is simply done by concatenation of the elemental propagations according to the elemental representation of $\mathbf{f}$. We call the forward propagation of a Taylor polynomial of degree k through $\mathbf{f}$ also a k-th order forward sweep. Note further that with respect to addition and multiplication given in Table 2.8 the polynomials

$$\mathcal{P}_k := \left\{ v(t) = \sum_{j=0}^{k-1} v_j t^j, \quad v_j \in \mathbb{R} \right\} \tag{2.33}$$

form for every $k > 0$ a commutative ring which is in general not free of zero divisors. With the modulus defined by

$$|v(t)| := \sum_{j=0}^{k-1} |v_j| \in \mathbb{R} \tag{2.34}$$

the triangle inequality as well as the relation $|u(t)v(t)| \leq |u(t)||v(t)|$ hold in $\mathcal{P}_k$. Hence we can, like for real numbers, perform arithmetic on Taylor polynomials. Vectors of Taylor polynomials

out of $\mathcal{P}_k$ with n components form a free module over $\mathbb{R}$ which we denote with $\mathcal{P}_k^n$. On $\mathcal{P}_k^n$ we define the Euclidean norm

$$\left\| \begin{pmatrix} v^1(t) \\ \vdots \\ v^n(t) \end{pmatrix} \right\| := \sqrt{\sum_{j=1}^{n} |v^j(t)|^2}. \tag{2.35}$$

Then it follows from multivariate calculus that any $k-1$-times continuously differentiable function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ has a unique extension to a mapping $E_k(\mathbf{f}) : \mathcal{P}_k^{n_{\text{indep}}} \to \mathcal{P}_k^{n_{\text{dep}}}$. This means for $k > 0$ that there exists a linear extension operator

$$E_k : \mathcal{C}^{k-1}(\mathbb{R}^{n_{\text{indep}}}, \mathbb{R}^{n_{\text{dep}}}) \to \mathcal{C}^0(\mathcal{P}_k^{n_{\text{indep}}}, \mathcal{P}_k^{n_{\text{dep}}}). \tag{2.36}$$

This more abstract framework for TC propagation will be helpful in the next section for the derivation of reverse TC propagation schemes.

Reverse TC propagation schemes After discussing the propagation of TCs forward through a function evaluation and the computation of univariate forward directional derivatives of type (2.19) we now discuss how univariate forward/adjoint directional derivatives of type (2.20) can be computed by reverse TC propagation. The straightforward idea to develop a reverse TC propagation scheme is based on the application of the reverse mode of AD to the actual computations arising in the forward TC propagation. In this way corresponding reverse propagation formulas could be derived for all elemental functions, which could be used to obtain an adjoint derivative of a k-th order forward sweep. By rescaling, analogously to Proposition 2.13, then the forward/adjoint derivative could be obtained. However, instead of deriving and implementing reverse TC propagation formulas for all elemental functions it is more economical to use the reverse mode not on the instruction level of forward TC propagation, but on the level of Taylor arithmetic. To achieve this transition, we have to show that differentiation and the application of the extension operator E_k (2.36) commute, i.e., lead in the end to the same Taylor coefficients for the Jacobian mapping of the underlying function $\mathbf{f}$. Based on the elemental representation of $\mathbf{f}$ we can then perform an "ordinary" reverse sweep, which now simply has to be executed in Taylor arithmetic to propagate (adjoint) Taylor polynomials reverse to the evaluation procedure. The first step to show commutativity is the following relation between the partial Jacobians of the Taylor coefficient functions and the Taylor coefficients of the Jacobian mapping of a function.

Proposition 2.16 (Jacobians of Taylor functions)
Let $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ be k-times continuously differentiable in a neighborhood of a point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$. Then

$$\frac{\partial \mathbf{y_j}}{\partial \mathbf{x_i}} = \frac{\partial \mathbf{f_j}}{\partial \mathbf{x_i}} \equiv \mathbf{A_{j-i}}(\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{j-i}}), \quad \forall 0 \leq i \leq j < k, \tag{2.37}$$

holds for the partial Jacobians of the Taylor coefficient functions, where $\mathbf{A_i}$ is the i-th Taylor coefficient of the Jacobian mapping $\mathbf{J}$ of $\mathbf{f}$ at $\mathbf{x}(t)$, i.e.,

$$\mathbf{J}(\mathbf{x}(t)) = \sum_{i=0}^{k-1} \mathbf{A_i} t^i + o(t^{k-1}). \tag{2.38}$$

Proof:

Provided t is sufficiently small, the value $\mathbf{y}(\mathbf{x}(t)) = \mathbf{f}(\mathbf{x}(t))$ is k-times continuously differentiable with respect to t and $\mathbf{x_i}$. Hence differentiation with respect to t and $\mathbf{x_i}$ commutes and therefore we have for $j < k$:

$$\begin{aligned}
\frac{\partial \mathbf{y_j}}{\partial \mathbf{x_i}} &= \frac{1}{j!}\frac{\partial}{\partial \mathbf{x_i}}\left(\frac{\partial^j}{\partial t^j}\mathbf{y}(\mathbf{x}(t))\bigg|_{t=0}\right) = \frac{1}{j!}\frac{\partial^j}{\partial t^j}\frac{\partial}{\partial \mathbf{x_i}}\mathbf{y}(\mathbf{x}(t))\bigg|_{t=0} \\
&= \frac{1}{j!}\frac{\partial^j}{\partial t^j}\Big(\mathbf{J}(\mathbf{x}(t))t^i\Big)\bigg|_{t=0} && \text{by chain-rule} \\
&= \frac{i!}{j!}\binom{j}{i}\frac{\partial^{j-i}}{\partial t^{j-i}}\mathbf{J}(\mathbf{x}(t))\bigg|_{t=0} && \text{by Leibniz's rule} \\
&= \frac{1}{(j-i)!}\frac{\partial^{j-i}}{\partial t^{j-i}}\mathbf{J}(\mathbf{x}(t))\bigg|_{t=0} = \mathbf{A_{j-i}} && (2.39)
\end{aligned}$$

□

We also can express the derivative of the Taylor calculus using the $\mathbf{A_i}$ according to the following proposition.

Proposition 2.17 (Derivative of Taylor calculus)
Let $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ be k-times continuously differentiable in a neighborhood of a point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$ and let $\mathbf{\Delta x}(t) = \sum_{j=0}^{k-1} t^j \mathbf{\Delta x_j} \in \mathcal{P}_k^{n_{\text{indep}}}$ be any perturbation that is sufficiently small with respect to the Euclidean norm (2.35). Then the polynomial

$$\mathbf{\Delta y}(t) = \sum_{j=0}^{k-1} t^j \mathbf{\Delta y_j} = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t)) - \mathbf{y}(t) + o(t^{k-1}) \in \mathcal{P}_k^{n_{\text{dep}}} \tag{2.40}$$

satisfies

$$\mathbf{\Delta y_i} - \sum_{j=0}^{i} \mathbf{A_{i-j}} \mathbf{\Delta x_j} = o(\|\mathbf{\Delta x}(t)\|), \quad \forall 0 \le i < k, \tag{2.41}$$

respectively written as power series

$$\mathbf{\Delta y}(t) = \mathbf{A}(t)\mathbf{\Delta x}(t) = o(\|\mathbf{\Delta x}(t)\|), \tag{2.42}$$

with

$$\mathbf{A}(t) = \sum_{j=0}^{k-1} t^j \mathbf{A_j} = \mathbf{J}(\mathbf{x}(t)) + o(t^{k-1}) \in \mathcal{P}_k^{m \times n}. \tag{2.43}$$

Proof:

For each i holds that $\mathbf{y_i} + \mathbf{\Delta y_i}$ is a continuously differentiable function of the $\mathbf{\Delta x_j}$ with $\mathbf{x_j}$ fixed for $j = 0, \ldots, i$. From Proposition 2.16 we have that the Jacobians with respect to these real vectors

are the $\mathbf{A_{k-j}}$. Hence the result follows by ordinary multivariate calculus on real numbers.

□

As a consequence, we have shown that differentiation and the linear extension operator E_k commute. This means if we assume $\mathbf{f} \in \mathcal{C}^k(\mathbb{R}^{n_{\text{indep}}}, \mathbb{R}^{n_{\text{dep}}})$ it does not matter if we first develop a scheme in real arithmetic to compute the Jacobian mapping $\mathbf{J}$ of $\mathbf{f}$ and then apply E_k, i.e., replace the operations in real arithmetic by operations in Taylor arithmetic or if we first use E_k to switch from $\mathbf{f} \in \mathcal{C}^k(\mathbb{R}^{n_{\text{indep}}}, \mathbb{R}^{n_{\text{dep}}})$ to $E_k(\mathbf{f}) \in \mathcal{C}^1(\mathcal{P}_k^{n_{\text{indep}}}, \mathcal{P}_k^{n_{\text{dep}}})$ and then differentiate in module space. This result can now be used to compute forward/adjoint directional derivatives of type (2.20) by a reverse sweep in Taylor arithmetic.

Proposition 2.18 (Forward/adjoint directional derivatives by reverse TC propagation) *Let the function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ be at least $k+1$ times continuously differentiable ($k \geq 1$) in the point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$. Let $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ and $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$ be arbitrary forward and adjoint directions, respectively. Furthermore, denote with $\bar{\mathbf{x}}_\mathbf{k} \equiv \bar{\mathbf{f}}_\mathbf{k}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d}) \in \mathbb{R}^{n_{\text{indep}}}$ the k-th Taylor coefficient of the adjoint Taylor polynomial $\nabla(E_{k+1}(\bar{\mathbf{y}}^T\mathbf{f})) \in \mathcal{P}_{k+1}^{n_{\text{indep}}}$ for the input $\mathbf{x_0} + t\mathbf{d}$. Then the univariate forward/adjoint directional derivative of order $k+1$ at point $\mathbf{x_0} \in \mathbb{R}^{n_{\text{indep}}}$ and in direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ and adjoint direction $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$ is obtained by*

$$\dot{\bar{\mathbf{f}}}^{(\mathbf{k})}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d}) = k!\, \bar{\mathbf{x}}_\mathbf{k} \equiv k!\, \bar{\mathbf{f}}_\mathbf{k}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d}). \tag{2.44}$$

Here ∇ stands for the Jacobian operator in $\mathcal{P}_k$.

Proof:

Consider for fixed $\bar{\mathbf{y}}$ the real valued function $g(\mathbf{x}) := \bar{\mathbf{y}}^T\mathbf{x}$. Then $g \in \mathcal{C}^{k+1}(\mathbb{R}^{n_{\text{indep}}}, \mathbb{R})$ and we have for the adjoint Taylor polynomial with general Taylor coefficients $\bar{\mathbf{f}}_\mathbf{j}$

$$\nabla(E_{k+1}(g))(\mathbf{x}(t)) = \sum_{j=0}^{k} t^j \bar{\mathbf{f}}_\mathbf{j} = \frac{\partial}{\partial \mathbf{x}} g(\mathbf{x}(t)) + o(t^k) = \frac{\partial}{\partial \mathbf{x}} (\bar{\mathbf{y}}^T\mathbf{f})(\mathbf{x}(t)) + o(t^k), \tag{2.45}$$

and hence for the input $\mathbf{x}(t) = \mathbf{x_0} + t\mathbf{d}$ we obtain by differentiating k times and evaluating at $t = 0$

$$\begin{aligned} k!\bar{\mathbf{f}}_\mathbf{k}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d}) &= \frac{\partial^k}{\partial t^k} \frac{\partial}{\partial \mathbf{x}} (\bar{\mathbf{y}}^T\mathbf{f})(\mathbf{x_0} + t\mathbf{d}) \Big|_{t=0} \\ &= \bar{\mathbf{y}}^T \frac{\partial^{k+1}}{\partial \mathbf{x}^{k+1}} \mathbf{f}(\mathbf{x_0})\mathbf{d}^k \\ &= \dot{\bar{\mathbf{f}}}^{(\mathbf{k})}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d}). \end{aligned} \tag{2.46}$$

□

For the actual computation of the coefficients $\bar{\mathbf{f}}_\mathbf{k}(\bar{\mathbf{y}}, \mathbf{x_0}, \mathbf{d})$ of the adjoint Taylor polynomial we use the commutativity we have just shown: First we apply a first order reverse sweep to obtain

a computational scheme for $\bar{\mathbf{y}}^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ in real arithmetic. Then we apply E_{k+1} to get the Taylor coefficients of the adjoint Taylor polynomial. In the end this means that we execute the operations of the first order reverse scheme in k-th order Taylor arithmetic, which we call a $(k+1)$-th order reverse sweep. This can be performed using the normal TC propagation rules for elemental functions which we already know, i.e., there is no need to derive and implement special reverse TC propagation rules. Analogously to the first order reverse sweep a $(k+1)$-th order reverse sweep has to be preceded by a forward sweep of order k that computes the values of the intermediate TCs that are needed for the reverse sweep. The complete forward/reverse scheme to compute one forward/adjoint directional derivative is then given by the following steps:

- Initialize the forward TC propagation with input $\mathbf{X} = \begin{bmatrix} \mathbf{x_0} & \mathbf{d} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}$.
- Propagate the TCs forward through the evaluation of $\mathbf{f}$ by applying the TC propagation rules to the operations of the elemental representation of $\mathbf{f}$. Store the TCs needed for a reverse sweep.
- Set the initial adjoint TCs for the reverse sweep according to the adjoint direction to $\bar{\mathbf{Y}} = \begin{bmatrix} \bar{\mathbf{y}} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}$.
- Reverse propagate adjoint TCs by applying the TC propagation rules to the chain of operations resulting from a first order reverse sweep for $\mathbf{f}$.

Note that the cost of this forward/reverse sweep can be bounded in terms of the function evaluation of $\mathbf{f}$. If we neglect the cost of accessing the tape with the stored intermediate TCs, the cost is proportional to $(k+1)^2$ evaluations of the function $\mathbf{f}$.

Multivariate higher-order derivatives Based on univariate Taylor propagation we have presented efficient schemes for the computation of univariate forward and forward/adjoint directional derivatives. Unfortunately not all entries of higher-order derivative tensors can be directly computed this way. Consider for example the general function $f(x,y,z) : \mathbb{R}^3 \to \mathbb{R}$ and let e_i^n be the i-th unit vector in $\mathbb{R}^n$. Then, e.g., the tensor entry $f_{xxx} \equiv \frac{\partial^3 f}{\partial x^3}$ can be computed as third order univariate forward derivative in direction e_1^3, and $f_{yzz} \equiv \frac{\partial^3 f}{\partial y \partial z^2}$ can be obtained as third order forward/adjoint univariate directional derivative with adjoint direction 1 and forward direction e_3^3, but this is not the case for the mixed derivative $f_{xyz} \equiv \frac{\partial^3 f}{\partial x \partial y \partial z}$. In the remainder of this section we present strategies to compute also these entries of the derivative tensors. More general, we consider now the task to compute for a function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ derivative tensors of arbitrary order, restricted to a subspace of dimension $p \geq 1$ spanned by the columns of the matrix $\mathbf{S} \in \mathbb{R}^{n_{\text{indep}} \times p}$. Given the parametrization $\mathbf{x}(\mathbf{z}) = \mathbf{x}(0) + \mathbf{S}\mathbf{z}$, with $\mathbf{z} \in \mathbb{R}^p$ we can understand the intermediate variables v and the dependent variables as functions of the reduced variables $\mathbf{z}$ and denote their derivatives w.r.t. $\mathbf{z}$ at $\mathbf{z} = \mathbf{0}$ with $\nabla_{\mathbf{S}} v$.
To directly compute all entries of a derivative tensor in the multivariate case one could use a multivariate version of the chain rule and transfer most of the previous results for TC propagation to this context. Note that also multivariate Taylor polynomials (with bounded total degree) form algebras in which arithmetic can be performed. An example for this approach can be found, e.g.,

in [Nei92]. For a complexity analysis of this approach we consider the multiplication $u = vw$. Its propagation rules for the first two orders are then given by

$$\nabla_{\mathbf{S}} v = u \nabla_{\mathbf{S}} w + w \nabla_{\mathbf{S}} u \tag{2.47}$$

and

$$\nabla^2_{\mathbf{S}} v = u \nabla^2_{\mathbf{S}} w + \nabla_{\mathbf{S}} u (\nabla_{\mathbf{S}} w)^T + \nabla_{\mathbf{S}} w (\nabla_{\mathbf{S}} u)^T + w \nabla^2_{\mathbf{S}} u. \tag{2.48}$$

In general, the number of operations to perform this multivariate multiplication rule is given by

$$\binom{2p+k}{k} \approx \frac{(2p)^k}{k!}, \qquad \text{if } k \ll p, \tag{2.49}$$

where k is the derivative order. Like for univariate TC propagation, the cost for a division and other nonlinear elementals are roughly the same as for the multiplication. The problem of this direct approach is the storing and the manipulation of the tensor entries in memory. The k-th order derivative tensor $\nabla^k_{\mathbf{S}} v$ has p^k entries, of which only

$$\binom{p+k-1}{k} \approx \frac{p^k}{k!}, \qquad \text{if } k \ll p, \tag{2.50}$$

are distinct. This means a lot of memory is wasted if symmetry is not exploited and also the operation count is higher, as more elements have to be carried through the propagation process. On the other hand, the use of a symmetric storage scheme leads to additional effort for the computation of the memory locations of specific elements and to a more irregular data access. This irregular data access pattern might lead to cache misses and a significant loss of performance. As until now this problem has not been solved in a satisfying manner, we now present another approach that was developed by Bischof et al. [BCG93]. It tries to reconstruct the multivariate Taylor polynomial corresponding to the tensor $\nabla^k_{\mathbf{S}} v$ from the coefficients of as few propagated univariate Taylor polynomials as possible. The directions corresponding to this set of propagated Taylor polynomials are called rays and the process of reconstruction is called exact interpolation. As a motivating example let us again have a look at the computation of f_{xyz} from above. First note that f_{yz} cannot be computed directly by an univariate forward directional derivative. However it can be expressed as a linear combination of such directional derivatives, namely as

$$f_{yz} = \frac{1}{2}(f_{dd} - f_{yy} - f_{zz}), \quad \text{where } d = (0,1,1)^T. \tag{2.51}$$

As a result f_{xyz} can then be obtained by a linear combination of univariate forward/adjoint directional derivatives. Fortunately such a linear interpolation scheme can be obtained not only in this simple setup, but also for every tensor element of arbitrary order for an arbitrary vector valued function. This has been shown for the case of forward TC propagation in the paper of Griewank et al. [GUW00] and is formulated in the following proposition.

Proposition 2.19 (Exact interpolation of tensor elements using Taylor coefficients) *Let the function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ be at least k-times differentiable at a point $x \in \mathbb{R}^{n_{\text{indep}}}$. Denote with $\mathbf{f_i}(\mathbf{x}, \mathbf{d})$ the i-th Taylor coefficient of $\mathbf{f}(\mathbf{x} + t\mathbf{d})$ at $t = 0$ for a direction $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$. Then*

for any seed matrix of directions $\mathbf{S} = [\mathbf{d}_1, \ldots, \mathbf{d}_p] \in \mathbb{R}^{n_{\text{indep}} \times p}$ *and any multi-index* $\mathbf{m} \in \mathbb{N}^p$ *with* $|\mathbf{m}| \leq k$ *we have with* $\mathbf{z} \in \mathbb{R}^p$ *that*

$$\left.\frac{\partial^{|\mathbf{m}|}\mathbf{f}(\mathbf{x} + z_1\mathbf{d_1} + z_2\mathbf{d_2} + \ldots + z_p\mathbf{d_p})}{\partial z_1^{m_1}\partial z_2^{m_2}\ldots\partial z_p^{m_p}}\right|_{\mathbf{z}=\mathbf{0}} = \sum_{|\mathbf{j}|=k} \gamma_{\mathbf{mj}}\mathbf{f}_{|\mathbf{m}|}(\mathbf{x}, \mathbf{Sj}), \tag{2.52}$$

where the coefficients $\gamma_{\mathbf{mj}}$ *are given by*

$$\gamma_{\mathbf{mj}} = \sum_{0<\mathbf{l}\leq\mathbf{m}} (-1)^{|\mathbf{m}-\mathbf{l}|} \prod_{i=1}^{p} \binom{m_i}{l_i} \binom{k(l_i/|\mathbf{l}|)}{j_i} \binom{|\mathbf{l}|}{k}^{|\mathbf{m}|}. \tag{2.53}$$

The coefficients $\gamma_{\mathbf{mj}}$ *only depend on the derivative degree* k *and the number of directions* $p \leq n_{\text{indep}}$ *in the seed matrix and are therefore independent of the evaluation point* $\mathbf{x}$*, the directions* $\mathbf{S}$ *and the function* $\mathbf{f}$*.*

Proof:

For the long and detailed proof we refer to the paper of Griewank et al. [GUW00].

□

Note that the number of rays needed to reconstruct the complete derivative tensor $\nabla^k_{\mathbf{S}}\mathbf{f}$ by forward TC propagation equals the number of distinct elements in the tensor. This is somehow intuitive, as only one coefficient per univariate direction carries information about $\nabla^k_{\mathbf{S}}\mathbf{f}$. We also see that as a by-product all entries of the derivative tensors with order smaller k can as well be computed from the ray coefficients. Concerning the complexity of this approach it can be shown that usually the computation count for the propagation of the rays lies close to the cost for the multivariate propagation while allowing a very regular data access pattern. The cost of the final interpolation can usually be neglected. A drawback is the increase of required memory by a factor of roughly $pk/(p+k)$. If this should become a problem in practice, the propagation can be done sequentially for several subsets of the needed rays. Also note that despite the name the exact interpolation may be subject to cancellation errors for $k \geq 2$ if the entries of the derivative tensor have very different scales. The case where cancellation might occur can simply be detected, but even if detected not necessarily avoided. We describe now the practical application of this approach using the example of Hessian tensor calculation using a second order forward TC sweep.

Example 2.20 (Hessian computation using forward TC sweep)
Consider the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^2$ from Example 2.4 on page 27 defined as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}.$$

We now want to compute the complete second-order derivative tensor given by the Hessians of the function components

$$\mathbf{H}^1 := \frac{\partial^2 y_1}{\partial \mathbf{x}^2} = \begin{pmatrix} e^{x_1}(x_2 + x_3) & e^{x_1} & e^{x_1} \\ e^{x_1} & -\sin(x_2) & 0 \\ e^{x_1} & 0 & 0 \end{pmatrix} \tag{2.54}$$

and

$$\mathbf{H}^2 := \frac{\partial^2 y_2}{\partial \mathbf{x}^2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\sin(x_2) + \frac{1}{4}(x_2+x_3)^{-3/2} & \frac{1}{4}(x_2+x_3)^{-3/2} \\ 0 & \frac{1}{4}(x_2+x_3)^{-3/2} & \frac{1}{4}(x_2+x_3)^{-3/2} \end{pmatrix} \tag{2.55}$$

at the point $\mathbf{p} = (0,0,1)^T$ using forward TC propagation. This means we use Taylor polynomials of degree $k = 2$ and propagate a family of rays through the evaluation procedure of $\mathbf{f}$ given in Example 2.4. To compute the needed rays we first define the set of underlying directions as the unit directions of the $\mathbb{R}^3$, i.e., we use the direction matrix $\mathbf{S} = \mathbb{I}_3$. The multi-indices corresponding to the distinct elements constituting the tensor are then given by

$$\mathbf{m_1} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{m_2} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{m_3} = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \quad \mathbf{m_4} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \quad \mathbf{m_5} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{m_6} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}. \tag{2.56}$$

In this special case, where exactly the elements of the tensor with degree k shall be computed, it holds for the multi-indices $\mathbf{j_i}$ with $|\mathbf{j_i}| = k = 2$ corresponding to the rays that $\mathbf{j_i} = \mathbf{m_i}$, $1 \leq i \leq 6$. The interpolation coefficients γ_{mj} are computed from Equation (2.53) and are given by

$$\Gamma := (\gamma_{\mathbf{m_i j_l}})_{i=1,\dots,6}^{l=1,\dots,6} = \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ -0.25 & 1 & -0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ -0.25 & 0 & 0 & 1 & 0 & -0.25 \\ 0 & 0 & -0.25 & 0 & 1 & -0.25 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix}. \tag{2.57}$$

As there are no zero columns, all rays are needed for the computation of the requested derivatives. The Taylor coefficients of the rays to be propagated are given by $\mathbf{X^i} = [\mathbf{p}, \mathbf{S}\,\mathbf{j_i}, 0]$, i.e.,

$$\begin{aligned} \mathbf{X^1} &= \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{X^2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{X^3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \\ \mathbf{X^4} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{X^5} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{X^6} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 0 \end{bmatrix}. \end{aligned} \tag{2.58}$$

The propagation of a Taylor polynomial of degree 2 with TCs $\mathbf{X}$ through the evaluation of $\mathbf{f}$ is done in Taylor arithmetic by the following operations

$v_{-2,0}$	$=$	$X_{1,0}$	$v_{-2,1}$	$=$	$X_{1,1}$	$v_{-2,2}$	$=$	$X_{1,2}$
$v_{-1,0}$	$=$	$X_{2,0}$	$v_{-1,1}$	$=$	$X_{2,1}$	$v_{-1,2}$	$=$	$X_{2,2}$
$v_{0,0}$	$=$	$X_{3,0}$	$v_{0,1}$	$=$	$X_{3,1}$	$v_{0,2}$	$=$	$X_{3,2}$
$v_{1,0}$	$=$	$e^{v_{-2,0}}$	$v_{1,1}$	$=$	$v_{1,0}v_{-2,1}$	$v_{1,2}$	$=$	$(v_{1,1}v_{-2,1} + 2v_{1,0}v_{-2,2})/2$
$v_{2,0}$	$=$	$v_{-1,0} + v_{0,0}$	$v_{2,1}$	$=$	$v_{-1,1} + v_{0,1}$	$v_{2,2}$	$=$	$v_{-1,2} + v_{0,2}$
$v_{3,0}$	$=$	$\sqrt{v_{2,0}}$	$v_{3,1}$	$=$	$v_{2,1}/(2v_{3,0})$	$v_{3,2}$	$=$	$(v_{2,2} - v_{3,1}v_{3,1})/(2v_{3,0})$
$v_{4,0}$	$=$	$v_{1,0}v_{2,0}$	$v_{4,1}$	$=$	$v_{1,1}v_{2,0} + v_{1,0}v_{2,1}$	$v_{4,2}$	$=$	$v_{1,2}v_{2,0} + v_{1,1}v_{2,1} + v_{1,0}v_{2,2}$
$v_{5,0}$	$=$	$\sin(v_{-1,0})$	$v_{5,1}$	$=$	$\cos(v_{-1,0})v_{-1,1}$	$v_{5,2}$	$=$	$\cos(v_{-1,0})v_{-1,2} - v_{-1,1}^2 \sin(v_{-1,0})/2$
$v_{6,0}$	$=$	$v_{4,0} + v_{5,0}$	$v_{6,1}$	$=$	$v_{4,1} + v_{5,1}$	$v_{6,2}$	$=$	$v_{4,2} + v_{5,2}$
$v_{7,0}$	$=$	$v_{5,0} - v_{3,0}$	$v_{7,1}$	$=$	$v_{5,1} - v_{3,1}$	$v_{7,2}$	$=$	$v_{5,2} - v_{3,2}$
$Y_{1,0}$	$=$	$v_{6,0}$	$Y_{1,1}$	$=$	$v_{6,1}$	$Y_{1,2}$	$=$	$v_{6,2}$
$Y_{2,0}$	$=$	$v_{7,0}$	$Y_{2,1}$	$=$	$v_{7,1}$	$Y_{2,2}$	$=$	$v_{7,2}$

Note that here the first two columns represent the same calculations we made in the computation of the function value and the first derivative using the forward mode in Example 2.7.
By propagating the rays $\mathbf{X^i}$ we obtain the intermediate TCs

$\mathbf{v_i^j}$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=1$	$\begin{bmatrix}1 & 2 & 2\end{bmatrix}$	$\begin{bmatrix}1 & 1 & \frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 1 & \frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$
$i=2$	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$
$i=3$	$\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & \frac{1}{2} & -\frac{1}{8}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & -\frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}1 & \frac{1}{2} & -\frac{1}{8}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & -\frac{1}{2}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & -\frac{1}{2}\end{bmatrix}$
$i=4$	$\begin{bmatrix}1 & 2 & 2\end{bmatrix}$	$\begin{bmatrix}1 & 2 & \frac{3}{2}\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & \frac{3}{2}\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 0\end{bmatrix}$
$i=5$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 2 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 0\end{bmatrix}$

and finally

$$\mathbf{Y^1} = \begin{bmatrix}1 & 2 & 2\\ -1 & 0 & 0\end{bmatrix}, \quad \mathbf{Y^2} = \begin{bmatrix}1 & 3 & \frac{3}{2}\\ -1 & \frac{1}{2} & \frac{1}{8}\end{bmatrix}, \quad \mathbf{Y^3} = \begin{bmatrix}1 & 4 & 0\\ -1 & 1 & \frac{1}{2}\end{bmatrix},$$
$$\mathbf{Y^4} = \begin{bmatrix}1 & 2 & \frac{3}{2}\\ -1 & -\frac{1}{2} & \frac{1}{8}\end{bmatrix}, \quad \mathbf{Y^5} = \begin{bmatrix}1 & 3 & 0\\ -1 & 0 & \frac{1}{2}\end{bmatrix}, \quad \mathbf{Y^6} = \begin{bmatrix}1 & 2 & 0\\ -1 & -1 & \frac{1}{2}\end{bmatrix}. \tag{2.59}$$

Now we can use the exact interpolation, described in Proposition 2.19 on page 47, to obtain the entries of the derivative tensor:

$$\begin{pmatrix}H^1_{11}\\ H^2_{11}\end{pmatrix} = 0.5\,\mathbf{y_2^1} = \begin{pmatrix}1\\ 0\end{pmatrix}, \quad \begin{pmatrix}H^1_{22}\\ H^2_{22}\end{pmatrix} = 0.5\,\mathbf{y_2^3} = \begin{pmatrix}0\\ \frac{1}{4}\end{pmatrix}, \quad \begin{pmatrix}H^1_{33}\\ H^2_{33}\end{pmatrix} = 0.5\,\mathbf{y_2^6} = \begin{pmatrix}0\\ \frac{1}{4}\end{pmatrix},$$

$$\begin{pmatrix}H^1_{12}\\ H^2_{12}\end{pmatrix} = -0.25\,\mathbf{y_2^1} + \mathbf{y_2^2} - 0.25\,\mathbf{y_2^3} = \begin{pmatrix}1\\ 0\end{pmatrix}, \quad \begin{pmatrix}H^1_{13}\\ H^2_{13}\end{pmatrix} = -0.25\,\mathbf{y_2^1} + \mathbf{y_2^4} - 0.25\,\mathbf{y_2^6} = \begin{pmatrix}1\\ 0\end{pmatrix},$$

$$\begin{pmatrix} H^1_{23} \\ H^2_{23} \end{pmatrix} = -0.25\,\mathbf{y_2^3} + \mathbf{y_2^5} - 0.25\,\mathbf{y_2^6} = \begin{pmatrix} 0 \\ \frac{1}{4} \end{pmatrix}. \tag{2.60}$$

Hence we obtain the derivative tensor as

$$\mathbf{H}^1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \mathbf{H}^2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}. \tag{2.61}$$

After this example of univariate forward TC propagation and exact interpolation we now present how the idea of exact interpolation can also be used in connection with univariate forward/reverse TC sweeps to obtain elements of higher-order derivative tensors. For this, we extend the results of [GUW00] stated in Proposition 2.19 for the use of forward/adjoint directional derivatives.

Proposition 2.21 (Exact interpolation of higher-order adjoint derivatives)
Let the function $\mathbf{f} : \mathbb{R}^{n_{\text{indep}}} \to \mathbb{R}^{n_{\text{dep}}}$ *be at least* $k+1$*-times differentiable at a point* $x \in \mathbb{R}^{n_{\text{indep}}}$*. Denote with* $\bar{\mathbf{f}}_{\mathbf{i}}(\bar{\mathbf{y}}, \mathbf{x}, \mathbf{d})$ *the* i*-th Taylor coefficient of the adjoint polynomial for input* $\mathbf{x} + t\mathbf{d}$ *at* $t = 0$ *for a direction* $\mathbf{d} \in \mathbb{R}^{n_{\text{indep}}}$ *and an adjoint direction* $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$*. Then for any adjoint direction* $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\text{dep}}}$*, any seed matrix of directions* $\mathbf{S} = [\mathbf{d}_1, \ldots, \mathbf{d}_p] \in \mathbb{R}^{n_{\text{indep}} \times p}$ *and any multi-index* $\mathbf{m} \in \mathbb{N}^p$ *with* $|\mathbf{m}| \leq k$ *we have that with* $\mathbf{z} \in \mathbb{R}^p$

$$\bar{\mathbf{y}}^T \frac{\partial^{|\mathbf{m}|+1} \mathbf{f}(\mathbf{x} + z_1\mathbf{d_1} + z_2\mathbf{d_2} + \ldots + z_p\mathbf{d_p})}{\partial z_1^{m_1} \partial z_2^{m_2} \ldots \partial z_p^{m_p}}\bigg|_{\mathbf{z}=0} = \sum_{|\mathbf{j}|=k} \gamma_{\mathbf{mj}} \bar{\mathbf{f}}_{|\mathbf{m}|}(\bar{\mathbf{y}}, \mathbf{x}, S\mathbf{j}), \tag{2.62}$$

where the coefficients $\gamma_{\mathbf{mj}}$ *are given by* (2.53) *in Proposition 2.19 on page 47.*

Proof:

The proposition follows directly from Proposition 2.19. Because the coefficients $\gamma_{\mathbf{mj}}$ do not depend on the function $\mathbf{f}$ and the exact interpolation described in Proposition 2.19 is linear in the Taylor coefficients, adjoint differentiation and exact interpolation simply commute.

□

We see that the number of rays needed for the computation of a complete derivative tensor of order $k+1$ is smaller than in the pure forward case, namely equal to the number of distinct elements of the derivative tensor with order k. Also the degree of the Taylor polynomials to be propagated is only k. On the other hand we need to propagate a total of n_{dep} sets of these rays backwards. Hence like in the case of first order sweep it depends on the number of rays and the size of n_{dep} whether the application of reverse TC propagation will be more efficient for the computation of the whole tensor than a pure forward TC propagation. In the optimization context the forward/reverse TC propagation scheme is usually the method of choice, because here we usually need higher-order adjoint derivatives involving one adjoint direction only, namely the gradient of the Lagrangian w.r.t. the output values of $\mathbf{f}$. In the final example of this section we now present the practical application of the exact interpolation using the forward/reverse TC propagation for the computation of the Hessian tensor of our well-known example function.

Example 2.22 (Hessian computation using forward/reverse TC sweep)
Consider the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^2$ from Example 2.4 on page 27 defined as

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \equiv \mathbf{f}(x_1, x_2, x_3) = \begin{pmatrix} e^{x_1}(x_2 + x_3) + \sin(x_2) \\ \sin(x_2) - \sqrt{x_2 + x_3} \end{pmatrix}.$$

We again want to compute the complete second-order derivative tensor given by the Hessians of the function components

$$\mathbf{H}^1 := \frac{\partial^2 y_1}{\partial \mathbf{x}^2} = \begin{pmatrix} e^{x_1}(x_2 + x_3) & e^{x_1} & e^{x_1} \\ e^{x_1} & -\sin(x_2) & 0 \\ e^{x_1} & 0 & 0 \end{pmatrix} \tag{2.63}$$

and

$$\mathbf{H}^2 := \frac{\partial^2 y_2}{\partial \mathbf{x}^2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\sin(x_2) + \frac{1}{4}(x_2 + x_3)^{-3/2} & \frac{1}{4}(x_2 + x_3)^{-3/2} \\ 0 & \frac{1}{4}(x_2 + x_3)^{-3/2} & \frac{1}{4}(x_2 + x_3)^{-3/2} \end{pmatrix} \tag{2.64}$$

at the point $\mathbf{p} = (0, 0, 1)^T$ using forward/reverse TC propagation. This is done by propagating a family of rays of degree $k = 1$ forward through the evaluation of $\mathbf{f}$ given in Example 2.4. Afterwards we do reverse TC propagation along this set of rays for a number of adjoint directions. To compute the needed rays we first define the set of underlying directions as the unit directions of the $\mathbb{R}^3$, i.e., we use the direction matrix $\mathbf{S} = \mathbb{I}_3$. The multi-indices for the forward propagation, here corresponding to the unit directions, are then given by

$$\mathbf{m_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{m_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{m_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{2.65}$$

In this case, these are also the multi-indices of the rays for degree $k = 1$, i.e., $\mathbf{j_i} = \mathbf{m_i}$, $1 \leq i \leq 3$. The interpolation coefficients γ_{mj} are computed from (2.53) and are simply given by

$$\Gamma := (\gamma_{\mathbf{m_i j_l}})_{i=1,\ldots,3}^{l=1,\ldots,3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.66}$$

As there are no zero columns, all rays are needed for the computation of the requested derivatives. The Taylor coefficients of the rays to be propagated are given by $\mathbf{X^i} = [\mathbf{p}, \mathbf{S}\,\mathbf{j_i}]$, i.e.,

$$\mathbf{X^1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad \mathbf{X^2} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mathbf{X^3} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.67}$$

The propagation of a Taylor polynomial of degree 1 with TCs $\mathbf{X}$ through the evaluation of $\mathbf{f}$ is done in Taylor arithmetic by the following operations:

$v_{-2,0}$	$=$	$X_{1,0}$	$v_{-2,1}$	$=$	$X_{1,1}$
$v_{-1,0}$	$=$	$X_{2,0}$	$v_{-1,1}$	$=$	$X_{2,1}$
$v_{0,0}$	$=$	$X_{3,0}$	$v_{0,1}$	$=$	$X_{3,1}$
$v_{1,0}$	$=$	$e^{v_{-2,0}}$	$v_{1,1}$	$=$	$v_{1,0}v_{-2,1}$
$v_{2,0}$	$=$	$v_{-1,0} + v_{0,0}$	$v_{2,1}$	$=$	$v_{-1,1} + v_{0,1}$
$v_{3,0}$	$=$	$\sqrt{v_{2,0}}$	$v_{3,1}$	$=$	$v_{2,1}/(2v_{3,0})$
$v_{4,0}$	$=$	$v_{1,0}v_{2,0}$	$v_{4,1}$	$=$	$v_{1,1}v_{2,0} + v_{1,0}v_{2,1}$
$v_{5,0}$	$=$	$\sin(v_{-1,0})$	$v_{5,1}$	$=$	$\cos(v_{-1,0})v_{-1,1}$
$v_{6,0}$	$=$	$v_{4,0} + v_{5,0}$	$v_{6,1}$	$=$	$v_{4,1} + v_{5,1}$
$v_{7,0}$	$=$	$v_{5,0} - v_{3,0}$	$v_{7,1}$	$=$	$v_{5,1} - v_{3,1}$
$Y_{1,0}$	$=$	$v_{6,0}$	$Y_{1,1}$	$=$	$v_{6,1}$
$Y_{2,0}$	$=$	$v_{7,0}$	$Y_{2,1}$	$=$	$v_{7,1}$

By propagation of the rays $\mathbf{X^i}$ we obtain

$$\mathbf{v_1^1} = \begin{bmatrix}1 & 1\end{bmatrix}, \quad \mathbf{v_1^2} = \begin{bmatrix}1 & 0\end{bmatrix}, \quad \mathbf{v_1^3} = \begin{bmatrix}1 & 0\end{bmatrix}, \tag{2.68a}$$

$$\mathbf{v_2^1} = \begin{bmatrix}1 & 0\end{bmatrix}, \quad \mathbf{v_2^2} = \begin{bmatrix}1 & 1\end{bmatrix}, \quad \mathbf{v_2^3} = \begin{bmatrix}1 & 1\end{bmatrix}, \tag{2.68b}$$

$$\mathbf{v_3^1} = \begin{bmatrix}1 & 0\end{bmatrix}, \quad \mathbf{v_3^2} = \begin{bmatrix}1 & \frac{1}{2}\end{bmatrix}, \quad \mathbf{v_3^3} = \begin{bmatrix}1 & \frac{1}{2}\end{bmatrix}, \tag{2.68c}$$

$$\mathbf{v_4^1} = \begin{bmatrix}1 & 1\end{bmatrix}, \quad \mathbf{v_4^2} = \begin{bmatrix}1 & 1\end{bmatrix}, \quad \mathbf{v_4^3} = \begin{bmatrix}1 & 1\end{bmatrix}, \tag{2.68d}$$

$$\mathbf{v_5^1} = \begin{bmatrix}0 & 0\end{bmatrix}, \quad \mathbf{v_5^2} = \begin{bmatrix}0 & 1\end{bmatrix}, \quad \mathbf{v_5^3} = \begin{bmatrix}0 & 0\end{bmatrix}, \tag{2.68e}$$

$$\mathbf{Y^1} = \begin{bmatrix}1 & 1\\ -1 & 0\end{bmatrix}, \quad \mathbf{Y^2} = \begin{bmatrix}1 & 2\\ -1 & \frac{1}{2}\end{bmatrix}, \quad \mathbf{Y^3} = \begin{bmatrix}1 & 1\\ -1 & -\frac{1}{2}\end{bmatrix}. \tag{2.68f}$$

In the reverse sweep, we propagate backwards one adjoint TC for each ray and for each adjoint direction. As $n_{\text{dep}} = 2$ we need the two adjoint directions

$$\bar{\mathbf{y}}_\mathbf{1} = \begin{pmatrix}1\\0\end{pmatrix}, \qquad \bar{\mathbf{y}}_\mathbf{2} = \begin{pmatrix}0\\1\end{pmatrix}, \tag{2.69}$$

to compute the complete derivative tensor. This results in the following 6 adjoint TCs $\bar{\mathbf{Y}}^{\mathbf{i,j}}$ to propagate, where i denotes the corresponding ray and j the corresponding adjoint direction:

$$\bar{\mathbf{Y}}^{\mathbf{1,1}} = \bar{\mathbf{Y}}^{\mathbf{2,1}} = \bar{\mathbf{Y}}^{\mathbf{3,1}} = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}, \qquad \bar{\mathbf{Y}}^{\mathbf{1,2}} = \bar{\mathbf{Y}}^{\mathbf{2,2}} = \bar{\mathbf{Y}}^{\mathbf{3,2}} = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}. \tag{2.70}$$

As explained earlier, the reverse TC sweep is performed for each adjoint TC like the regular first order reverse sweep, given in Example 2.8 on page 34, only that now the operations are performed in Taylor arithmetic. This leads for each adjoint TC $\bar{\mathbf{Y}}$ to the following operations (already accumulated for every variable):

$$
\begin{array}{rcl|rcl}
\bar{v}_{5,0} &=& \bar{Y}_{2,0} + \bar{Y}_{1,0} & \bar{v}_{5,1} &=& \bar{Y}_{2,1} + \bar{Y}_{1,1} \\
\bar{v}_{4,0} &=& \bar{Y}_{1,0} & \bar{v}_{4,1} &=& \bar{Y}_{1,1} \\
\bar{v}_{3,0} &=& -\bar{Y}_{2,0} & \bar{v}_{3,1} &=& -\bar{Y}_{2,1} \\
\bar{v}_{2,0} &=& \bar{v}_{4,0} v_{1,0} & \bar{v}_{2,1} &=& \bar{v}_{4,0} v_{1,1} + \bar{v}_{4,1} v_{1,0} \\
&+& \bar{v}_{3,0}/(2v_{3,0}) & &+& (\bar{v}_{3,1} - (\bar{v}_{3,0}/v_{3,0}) v_{3,1})/(2v_{3,0}) \\
\bar{v}_{1,0} &=& \bar{v}_{4,0} v_{2,0} & \bar{v}_{1,1} &=& \bar{v}_{4,0} v_{2,1} + \bar{v}_{4,1} v_{2,0} \\
\bar{X}_{3,0} &=& \bar{v}_{2,0} & \bar{X}_{3,1} &=& \bar{v}_{2,1} \\
\bar{X}_{2,0} &=& \bar{v}_{5,0} \cos(v_{-1,0}) & \bar{X}_{2,1} &=& -\bar{v}_{5,0} \sin(v_{-1,1} v_{-1,1}) + \bar{v}_{5,1} \cos(v_{-1,0}) \\
&+& \bar{v}_{2,0} & &+& \bar{v}_{2,1} \\
\bar{X}_{1,0} &=& \bar{v}_{1,0} v_{1,0} & \bar{X}_{1,1} &=& \bar{v}_{1,0} v_{1,1} + \bar{v}_{1,1} v_{1,0}
\end{array}
$$

Hence we obtain for the propagation of the $\bar{Y}^{i,j}$ the intermediate adjoint TCs

$\bar{\mathbf{v}}_{\mathbf{i}}^{\mathbf{j,k}}$	$(j,k) = (1,1)$	$(2,1)$	$(3,1)$	$(1,2)$	$(2,2)$	$(3,2)$
$i = 5$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$
$i = 4$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} -\frac{1}{2} & 0 \end{bmatrix}$	$\begin{bmatrix} -\frac{1}{2} & \frac{1}{4} \end{bmatrix}$	$\begin{bmatrix} -\frac{1}{2} & \frac{1}{4} \end{bmatrix}$
$i = 3$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \end{bmatrix}$
$i = 2$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$
$i = 1$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \end{bmatrix}$

and the propagated adjoint TCs corresponding to the propagated adjoint Taylor polynomials

$$
\bar{\mathbf{X}}^{\mathbf{1,1}} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \bar{\mathbf{X}}^{\mathbf{2,1}} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1 & 0 \end{bmatrix}, \qquad \bar{\mathbf{X}}^{\mathbf{3,1}} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1 & 0 \end{bmatrix}, \tag{2.71}
$$

$$
\bar{\mathbf{X}}^{\mathbf{1,2}} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 \end{bmatrix}, \qquad \bar{\mathbf{X}}^{\mathbf{2,2}} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{4} \end{bmatrix}, \qquad \bar{\mathbf{X}}^{\mathbf{3,2}} = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{4} \end{bmatrix}. \tag{2.72}
$$

Finally, we use the adjoint version of exact interpolation described in Proposition 2.21 on page 51 to obtain the entries of the derivative tensor:

$$
\begin{pmatrix} H^1_{11} \\ H^1_{21} \\ H^1_{31} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{1,1}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \begin{pmatrix} H^1_{12} \\ H^1_{22} \\ H^1_{32} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{2,1}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} H^1_{13} \\ H^1_{23} \\ H^1_{33} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{3,1}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix},
$$

$$
\begin{pmatrix} H^2_{11} \\ H^2_{21} \\ H^2_{31} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{1,2}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} H^2_{12} \\ H^2_{22} \\ H^2_{32} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{2,2}} = \begin{pmatrix} 0 \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}, \qquad \begin{pmatrix} H^2_{13} \\ H^2_{23} \\ H^2_{33} \end{pmatrix} = \bar{\mathbf{x}}_{\mathbf{1}}^{\mathbf{3,2}} = \begin{pmatrix} 0 \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}. \tag{2.73}
$$

Hence we obtain the derivative tensor as

$$\mathbf{H}^1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \mathbf{H}^2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}. \tag{2.74}$$

Note that in the Hessian case the exact adjoint interpolation is always trivial and hence not subject to cancellation errors.
Summarized, we have seen in this section how the approach of univariate Taylor coefficient propagation is used to obtain efficient forward and reverse schemes for the computation of higher-order directional derivatives of arbitrary order. With the help of exact interpolation univariate TC propagation can also be used to generate all elements of a derivative tensor of arbitrary order by the propagation of a family of rays of the corresponding order. This can be done quite efficient due to very regular data access pattern. We rely on the approach of univariate Taylor coefficient propagation later in this thesis for the efficient computation of a reduced Hessian matrix in our lifted optimization algorithm as well as, in connection with the idea of Internal Numerical Differentiation (IND), for the computation of derivatives of arbitrary order of the solutions of IVPs for ODEs and DAEs in our numerical integration schemes.

2.4.4 AD implementations

In the previous part of the section we have derived various strategies for derivative generation based on the idea of automatic differentiation. While the derivation of the actual schemes and their evaluation can be done for a specific function by hand, as done in the presented examples, the real power of AD lies in the automatic derivation of the specific derivative generation scheme for a given function from its evaluation procedure by an AD software tool. Otherwise, the error-free derivation of a derivative generation scheme for a function defined by possibly several thousand lines of source code would be nearly impossible. Also for every new function there would be the need for a lot of additional manual work. This would be not desirable, if at all feasible, if an AD approach should be used as part of a multiple purpose simulation or optimization code. Fortunately, all the presented schemes can be derived efficiently from the function definition in an automatic way. A good overview on available AD tools, as well as on publications related to AD, can be found on the "AutoDiff" homepage [Bis]. AD software can be separated mainly into two large groups, depending on the approach it is based on. In any case, for most of the tools some modifications of the original function evaluation code are needed, e.g., to specify independent and dependent variables.
The first group of AD software is based on source code transformation. Here from the given source for the evaluation of the function that is to be differentiated, the AD tool generates as a kind of precompiler a source code for the computation of the derivative of the function by using one of the presented AD schemes. This source code for the derivative can then be compiled and linked to the program or library where it should be used. A big advantage of this approach is that, since the generation of the derivative code is not done at run-time and usually only once for a function, a deep analysis of the function evaluation procedure and a very sophisticated approach for the

generation of the derivative scheme can usually be applied. Furthermore, the instructions of the resulting derivative code are known and can be analyzed and optimized by the compiler, leading to a very efficient derivative evaluation. Note that this compiler based optimization would be most efficient when also the order of the derivatives to be generated, the number of directions, etc., would already be known at compile time, which unfortunately is usually not the case. Also a new compilation process is needed whenever the definition of the underlying function changes or a new function is defined. This makes a fully automatic handling inside of a multiple purpose code more difficult, as, e.g., user specified functions are normally not known at the time of the compilation of the core code. There sometimes also exist technical problems, e.g., when the source code for the function evaluation is scattered over more than one source file. Nowadays, there exist many AD tools based on source code transformation for computer languages like Fortran (e.g., ADIFOR [BCKM96], OpenAD [HNN02], TAPENADE [HP04]), C/C++ (ADIC [BRM97], RAPSODIA [CU09]) or MATLAB (ADiMat [BBL+02], tomsym [Inc]). While nearly all of them support at least first order forward/reverse schemes, Jacobian and maybe even Hessian generation, only very few of them support forward TC propagation for a predefined order and until now none of them supports forward/reverse TC propagation schemes of arbitrary order. Considering the supported languages, it should be noted that the target language does not have to be the same as the source language. There exist several specialized tools, that generate for a special function class from its problem specific description language expressions for function and derivative evaluation in another language like Fortran, C, or C++. Examples are the code of Rücker [Rüc99] that generates from a description of a chemical reaction system Fortran or C source code for the model equations and their first (forward) derivatives, or the ADOPT package by Schmidt [Sch08]. ADOPT uses the SBML [HFH+07] description of a process in system biology to generate C++ function and derivative code that supports forward/reverse TC propagation and that is optimized for the use as model description in the `SolvIND` integrator suite.

The second group of AD tools is based on operator overloading. This means the definitions of the elemental functions in a computer language that allows operator overloading, like C++, are redefined, usually for a new data type specific to the AD tool. For example, they could be changed to the corresponding operations in Taylor arithmetic to perform a forward TC sweep. If used in this sense, operator overloading is in the end quite similar to a source code transformation, only that the transformation is done in this case directly by the compiler itself. However, often the elemental functions are adapted to record the operations performed during a function evaluation onto an operation tape. This can later be used as basis for the subsequent execution of forward and reverse derivative generation schemes at different evaluation points and for different derivative directions and derivative orders. Other than in the case of source code transformation, this means that effectively the derivative generation scheme is constructed at run-time of the calling program. This makes the tools based on operator overloading in general a lot more flexible and more suited for the use in multiple-purpose codes, where the number of derivative directions, as well as type and order of derivatives, might not be known at compile-time. In exchange, the derivative evaluation they provide is usually slower than for code obtained from source code transformation, as it cannot be optimized by a compiler for the actual system architecture. Furthermore, additional memory is needed to store the tape of operations. Like for the approach of source code transformation there

exists a variety of AD tools based on operator overloading for different languages like MATLAB, newer Fortran versions or C++. Examples are here ADMAT [CV98], COSY INFINITY [Ber02], TaylUR [vH06], FADBAD++/FADBAD-TADIFF [BS96], CppAD [Bel] and ADOL-C [GJU96]. All of the tools support one kind of first order derivative generation, some also higher-order forward derivative schemes, but so far ADOL-C and CppAD seem to remain the only ones to support forward/reverse TC propagation for higher-order derivative generation.
Note that not all AD tools can be clearly put into one of the two categories, as for example they might generate derivative code that contains overloaded operations for specific AD data types. Alternatively, they might use operator overloading to record the operations occurring during a function evaluation to build up a representation of the function evaluation and generate afterwards, based on this representation, source code for the evaluation of the derivatives. The partition of the tools presented here is mainly based on whether the construction of the derivative generation scheme is done at compile-time or at run-time.

2.5 Generation of sparse derivatives

While most strategies and algorithms presented in this thesis strive to avoid the forming of complete Jacobians or full higher-order derivative tensors and aim to minimize the number of directional derivatives, sometimes the building of a full Jacobian matrix cannot be avoided. This is for example the case in the presented schemes for numerical integration of ODEs and DAEs. However, if the computation of a full Jacobian or derivative tensor cannot be avoided, its specific structure should be exploited if possible. This is especially true for high-dimensional functions, where otherwise the memory consumption and also the operation count for operations involving these tensors becomes prohibitively large. Hence we give in this section a short overview on strategies for the generation of sparse Jacobian and Hessian matrices, i.e., matrices where the number elements that might be nonzero is significantly smaller than the total number of matrix elements. The first group of strategies is based on AD ideas and relies on the availability of an elemental representation of the function or the computational graph of the function, respectively. In this case, sparse storage versions of the presented first order forward and reverse mode can be obtained quite easily [BKBC96]. However, their implementation may suffer a lack of efficiency when the internal dependencies of the function evaluations are not known at compile time. Similarly, also second-order schemes using sparse storage can be derived. More interesting AD-based approaches are working on the computational graph of the function. They also use the chain rule, but make use of the freedom that it does not have to be applied in a strictly sequential manner to accumulate derivative information, like it is done in the forward and in the reverse mode. Instead, it can also be used to eliminate vertices or edges from the computational graph in arbitrary order, until a bipartite graph only containing dependent and independent variables remains. From this graph the Jacobian and also its sparsity pattern can be immediately obtained. Furthermore, it should be noted that in this approach the sparsity structure is automatically exploited. The problem of finding the accumulation scheme for a Jacobian with the minimum number of floating point operations is in general NP-complete [Nau08]. However, there exist a number of heuristic accumulation strategies, see, e.g., [GR91, Nau99, Nau02, Gri00] and references therein. These

are usually much more efficient than a Jacobian computation by a (sparse) forward or a (sparse) reverse mode. Heuristics have also been proposed for the accumulation of Hessians [GR05].
Another group of strategies for sparse Jacobian computation is based on compression. They are not relying on the availability of an elemental representation of the function, but only on the possibility to evaluate the function in a black-box manner and on the knowledge of the sparsity pattern of the Jacobian. Compression based approaches use the sparsity pattern of the Jacobian $\mathbf{J} \in \mathbb{R}^{n_{\text{dep}} \times n_{\text{indep}}}$ for the construction of so-called seed matrices $\mathbf{S} \in \mathbb{R}^{n_{\text{indep}} \times p}$ and/or $\mathbf{T} \in \mathbb{R}^{n_{\text{dep}} \times q}$ that allow the reconstruction of all Jacobian entries from the computation of first order directional derivatives. The use of directional forward derivatives of the form

$$\mathbf{B} = \mathbf{J} \cdot \mathbf{S} \tag{2.75}$$

is called row compression, whereas the use of adjoint directional derivatives

$$\mathbf{C}^T = \mathbf{T}^T \cdot \mathbf{J}, \tag{2.76}$$

is called column compression. In both cases the strategies aim to minimize the number of needed directional derivatives, i.e., the number of columns p and q, respectively, of the seed matrices. The matrices $\mathbf{B}$ and $\mathbf{C}$ are called compressed Jacobians. The evaluation of the directional derivatives can be done with any suitable strategy presented above, depending on the accuracy requirements. Let $\mathcal{X}_i$ be the set of column numbers containing nonzero elements in the i-th row of $\mathbf{J}$ and let $\mathbf{b_i} \in \mathbb{R}^p$ the i-th row of $\mathbf{B}$ and $\mathbf{j_i} \in \mathbb{R}^{p_i}$, $p_i := |\mathcal{X}_i|$, the vector of the nonzero elements in the i-th row, ordered by their size. If we furthermore define $\mathbf{S_i} := ((\mathbf{e_j^{n_{\text{indep}}}})^T \mathbf{S})_{j \in \mathcal{X}_i} \in \mathbb{R}^{p_i \times p}$ as the "projection" of $\mathbf{S}$ onto the nonzero columns in the i-th row of $\mathbf{J}$ we see that for a given choice of $\mathbf{S}$ the row of the compressed Jacobian $\mathbf{b_i}$ is determined by

$$\mathbf{S_i}^T \mathbf{j_i} = \mathbf{b_i}. \tag{2.77}$$

This means on the other hand that the row $\mathbf{j_i}$ of the uncompressed Jacobian can only be determined uniquely from $\mathbf{b_i}$, if $p \geq p_i$. An immediate consequence for the complexity of the row compression is that for the number of directional derivatives needed it holds $p \geq \max_{1 \leq i \leq n_{\text{dep}}} p_i$. Analogously, the bound $q \geq \max_{1 \leq i \leq n_{\text{indep}}} \mathcal{Y}_i$ for the column compression can be derived, where $\mathcal{Y}_i$ is the number of nonzero elements in the i-th column of $\mathbf{J}$.
For the construction of $\mathbf{S}$ there exist different approaches, where the mostly used ones were proposed by Curtis, Powell and Reid (CPR) [CPR74] and by Newsam and Ramsdell (NR) [NR83]. CPR seeding constructs seed matrices $\mathbf{S} \in \{0,1\}^{n_{\text{indep}} \times p}$ for which the $\mathbf{S_i}$ are permutations of $[\mathbb{I}_{p_i}\ \mathbf{0}] \in \mathbb{R}^{p_i \times p}$. Because finding such a seed matrix with minimal p is equivalent to the graph coloring problem and hence NP-hard [CM84], CPR uses heuristic strategies, which usually work fine, but may also lead to a p much larger than needed theoretically. An advantage of CPR seeding is that once a suitable $\mathbf{S}$ is found, the entries of the uncompressed Jacobian can be obtained directly from the compressed one without further algorithmic operations. NR seeding on the other hand uses Vandermonde matrices for the $\mathbf{S}$, where p is chosen at its lower bound. The drawback is that linear systems have to be solved to reconstruct the uncompressed Jacobian from the compressed one. The choice of the Vandermonde matrices was inspired by complexity considerations, as linear

systems involving these matrices can be solved quite efficiently [GvL96], especially compared to the effort for the derivative computation. On the other hand these systems might be ill-conditioned such that modifications for stability improvement have been proposed, e.g., the use of orthogonal polynomials or coloring based approaches [GUG96]. Note that both the CPR and the NR seeding can also be adapted for the use in column compression. Furthermore also a combination of row and column compression has been proposed [CV96], leading to a significantly reduced number of needed derivatives for special matrix types. Concerning Hessian computation there also exist compression techniques based on two-sided compression of the Hessian tensor that also exploit its symmetry, as, e.g., presented in [CC86].
Summarizing, it can be said that also for the exploitation of sparsity in Jacobian and Hessian computation there exists no optimal approach for all cases. Instead, it depends much on the specific problem which approach performs best. The choice of the most suitable approach is also strongly influenced by the availability of an elemental representation of the function to be differentiated and by the question whether the computational graph or the sparsity patterns is known already at compile-time or only at run-time.

3 Nonlinear Programming

In this chapter the problem class of (smooth) Nonlinear Programs (NLPs) and their basic properties are presented. We give a short overview over the local optimality conditions of these type of problems as well as their numerical solution using Newton-type methods. Finally, the topic of globalization is addressed shortly. In this chapter we consider only the general NLP formulation, e.g., we neglect any internal structure that the problems might have. The topics discussed in this chapter are the foundation for the lifted Newton-type methods presented in Chapter 4 and Chapter 7.

3.1 Notation and definitions

In this section we give the basic problem formulation and the definitions needed in the discussions that follow. The definitions and the (omitted) proofs of the presented theorems can be found in many textbooks on optimization, e.g., in the book of Nocedal and Wright [NW99] or the book of Fiacco and McCormick [FM90].

We start by defining the fundamental problem class.

Definition 3.1 (Nonlinear Program (NLP))
Let $c \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R})$, $\mathbf{g} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{eq}}})$ and $\mathbf{h} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{ineq}}})$ be nonlinear functions. The general NLP is then given by

$$\min_{\mathbf{u} \in \mathbb{R}^{n_u}} c(\mathbf{u}) \tag{3.1a}$$

subject to

$$\mathbf{g}(\mathbf{u}) = \mathbf{0} \tag{3.1b}$$
$$\mathbf{h}(\mathbf{u}) \geq \mathbf{0}. \tag{3.1c}$$

Here c represents a cost function, and $\mathbf{g}$ and $\mathbf{h}$ represent equality and inequality constraints, respectively. The inequality in (3.1c) is to be understood component-wise. We denote the scalar-valued component functions of $\mathbf{g}$ and $\mathbf{h}$ with $g_i(\mathbf{u})$, $1 \leq i \leq n_{\text{eq}}$ and $h_i(\mathbf{u})$, $1 \leq i \leq n_{\text{ineq}}$.

We also state here two important subclasses of NLPs. The nonlinear least-squares program occurs often in practice, e.g., in the treatment of parameter estimation problems or tracking problems.

Definition 3.2 (Nonlinear least-squares program)
Let $\mathbf{r} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{res}}})$, $\mathbf{g} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{eq}}})$ *and* $\mathbf{h} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{ineq}}})$ *be nonlinear functions. The nonlinear least-squares problem is then given by*

$$\min_{\mathbf{u}\in\mathbb{R}^{n_u}} \frac{1}{2}\|\mathbf{r}(\mathbf{u})\|_2^2 \tag{3.2a}$$

subject to

$$\mathbf{g}(\mathbf{u}) = \mathbf{0} \tag{3.2b}$$
$$\mathbf{h}(\mathbf{u}) \geq \mathbf{0}. \tag{3.2c}$$

Here we call $\mathbf{r}$ *the residual function and* $\mathbf{g}$ *and* $\mathbf{h}$ *represent, as before, equality and inequality constraints.*

The Quadratic Program (QP) also often occurs in practice, and it is the common subproblem class to be solved in each iteration of the Newton-type methods presented in this and later chapters.

Definition 3.3 (Quadratic Program (QP))
A Quadratic Program (QP) is an optimization problem with a quadratic cost function and linear constraints, e.g., of the form

$$\min_{\mathbf{u}\in\mathbb{R}^{n_u}} \frac{1}{2}\mathbf{u}^T\mathbf{A}\mathbf{u} + \mathbf{a}^T\mathbf{u} \tag{3.3a}$$

subject to

$$\mathbf{g} + \mathbf{G}\mathbf{u} = \mathbf{0} \tag{3.3b}$$
$$\mathbf{h} + \mathbf{H}\mathbf{u} \geq \mathbf{0}. \tag{3.3c}$$

where $\mathbf{a} \in \mathbb{R}^{n_u}$, $\mathbf{g} \in \mathbb{R}^{n_{\text{eq}}}$, $\mathbf{h} \in \mathbb{R}^{n_{\text{ineq}}}$ *are given vectors and* $\mathbf{A} \in \mathbb{R}^{n_u \times n_u}$, $\mathbf{G} \in \mathbb{R}^{n_{\text{eq}} \times n_u}$, $\mathbf{H} \in \mathbb{R}^{n_{\text{ineq}} \times n_u}$ *given matrices.*

After presenting the fundamental problem classes we define the notation of points fulfilling the constraints of the NLP.

Definition 3.4 (Feasibility)
A point $\tilde{\mathbf{u}} \in \mathbb{R}^{n_u}$ *is feasible with respect to the (equality) constraint* $g_i(\mathbf{u}) = 0$, *iff* $g_i(\tilde{\mathbf{u}}) = 0$. *Then we say that the constraint* g_i *is satisfied at* $\tilde{\mathbf{u}}$. *If this is not the case, we call* $\tilde{\mathbf{u}}$ *infeasible and say that the constraint* g_i *is violated at* $\tilde{\mathbf{u}}$.
A point $\tilde{\mathbf{u}} \in \mathbb{R}^{n_u}$ *is feasible with respect to the (inequality) constraint* $h_i(\mathbf{u}) \geq 0$, *iff* $h_i(\tilde{\mathbf{u}}) \geq 0$. *Then we say that the constraint* h_i *is satisfied at* $\tilde{\mathbf{u}}$. *If this is not the case, we call* $\tilde{\mathbf{u}}$ *infeasible and say that the constraint* h_i *is violated at* $\tilde{\mathbf{u}}$.
We call $\tilde{\mathbf{u}}$ *a feasible point (of the NLP (3.1)), iff it is feasible with respect to all constraints. We denote the set of all feasible points with* $\mathcal{F}$.

The following definition characterizes a solution (or minimum) of the NLP.

Definition 3.5 (Local and global minimum)
Let $\mathbf{u}^* \in \mathbb{R}^{n_u}$ *be a feasible point.*

$\mathbf{u}^*$ *is a local minimum of the NLP* (3.1)*, iff there exists an* $\varepsilon > 0$*, such that*

$$c(\mathbf{u}^*) \leq c(\mathbf{u}) \qquad \forall \mathbf{u} \in \mathcal{F} \cap \mathcal{U}_\varepsilon(\mathbf{u}^*). \tag{3.4}$$

We call $\mathbf{u}^*$ *a strict local minimum, iff there exists an* $\varepsilon > 0$*, such that*

$$c(\mathbf{u}^*) < c(\mathbf{u}) \qquad \forall \mathbf{u} \in \mathcal{F} \cap \mathcal{U}_\varepsilon(\mathbf{u}^*), \mathbf{u} \neq \mathbf{u}^*. \tag{3.5}$$

$\mathbf{u}^*$ *is a global minimum, iff*

$$c(\mathbf{u}^*) \leq c(\mathbf{u}) \qquad \forall \mathbf{u} \in \mathcal{F}. \tag{3.6}$$

A strict global minimum is defined analogously. A local (or global) minimum, that is not strict, is also called a weak local (or global) minimum.

Based on this definition, the next important question is how to test, whether a given feasible point $\mathbf{u}$ is a local (or global) minimum. For the nonlinear problems occurring in practice, it is usually impossible to obtain a closed representation of the set of feasible points in the neighborhood of $\mathbf{u}$ and to test its optimality using the criteria of Definition 3.5.
Note that in general finding a global minimum of the NLP is a significantly different and usually also more difficult task compared to finding a local minimum. Only for some special cases, e.g., convex problems, it can be shown that a local minimum is also a global minimum. In the following we restrict ourselves to algorithms that find local minima of the NLP only. For an overview on possible approaches to find global minima, refer ,e.g., to the review article [FAC+05].
As a result, we need tools to systematically describe and characterize the set of feasible points in the direct neighborhood of a feasible point $\mathbf{u}$.

Definition 3.6 (Feasible path and feasible direction)
Let $\mathbf{u} \in \mathbb{R}^{n_u}$ *be a feasible point and* $\varepsilon > 0$*.*
A feasible path for the point $\mathbf{u}$ *is a mapping* $\boldsymbol{\psi} \in \mathcal{C}^1([0, \varepsilon), \mathcal{F})$ *with*

$$\boldsymbol{\psi}(0) = \mathbf{u} \text{ and } \boldsymbol{\psi}(s) \neq \mathbf{u} \; \forall s \in (0, \varepsilon).$$

The tangent to the feasible path in the point $\mathbf{u}$ *is called a feasible direction.*

Loosely speaking, a feasible path is a way starting in $\mathbf{u}$ and remaining in the feasible set. Generally, a feasible path has to be nonlinear to achieve this. A feasible direction describes a direction in which we could go from $\mathbf{u}$ at least an infinitesimal small step while remaining feasible.

We now divide the constraints into two separate classes to study how each of them influences the set of feasible directions.

Definition 3.7 (Active set and active constraints)
Let $\tilde{\mathbf{u}} \in \mathbb{R}^{n_u}$ be a feasible point.

An inequality constraint h_i is called active in $\tilde{\mathbf{u}}$, iff $h_i(\tilde{\mathbf{u}}) = 0$. If $h_i(\tilde{\mathbf{u}}) > 0$ it is called inactive in $\tilde{\mathbf{u}}$. The active set of the inequality constraints $\mathbf{h}$ in a point $(\tilde{\mathbf{u}})$ is defined as

$$\mathcal{I}(\tilde{\mathbf{u}}) := \{i \in \{1, \ldots, n_{\text{ineq}}\} | h_i(\tilde{\mathbf{u}}) = 0\}, \tag{3.7}$$

i.e., as the set of indices of the active inequality constraints h_i in that point.

We define the combination of equality constraints and active inequality constraints as active constraints

$$\tilde{\mathbf{g}}(\mathbf{u}) := \begin{pmatrix} (\mathbf{g}(\mathbf{u}))_{1 \leq i \leq n_{\text{eq}}} \\ (\mathbf{h}(\mathbf{u}))_{i \in \mathcal{I}(\mathbf{u})} \end{pmatrix} \in \mathbb{R}^{n_{\text{eq}} + |\mathcal{I}(\mathbf{u})|} \tag{3.8}$$

Based on these definitions we observe the following: An active inequality constraint allows (infinitesimal) movements into directions where the constraint remains active, and also into directions where it becomes inactive. An inactive inequality constraint on the other side does allow infinitesimal movements in all directions, hence it does not restrict the set of possible feasible directions. An equality constraint finally allows only movements into directions where one remains on the manifold defined by the constraint. We can formalize this by the following Lemma.

Lemma 3.8
Let $\mathbf{u} \in \mathbb{R}^{n_u}$ be a feasible point for the NLP (3.1) and $\boldsymbol{\psi} \in \mathcal{C}^1([0, \varepsilon), \mathcal{F})$ a feasible path for $\mathbf{u}$. Then the corresponding feasible direction $\mathbf{d} = \frac{\partial \boldsymbol{\psi}(s)}{\partial s}\Big|_{s=0}$ fulfills

$$\nabla_{\mathbf{u}} g_i(\mathbf{u})^T \mathbf{d} = 0, \qquad 1 \leq i \leq n_{\text{eq}}, \tag{3.9a}$$
$$\nabla_{\mathbf{u}} h_i(\mathbf{u})^T \mathbf{d} \geq 0, \qquad i \in \mathcal{I}(\mathbf{u}). \tag{3.9b}$$

Proof:

This follows directly from the feasibility conditions for the path, i.e., $\mathbf{g}(\boldsymbol{\psi}(s)) = \mathbf{0} \quad \forall s \in [0, \varepsilon)$ and $\mathbf{h}(\boldsymbol{\psi}(s)) \geq 0 \quad \forall s \in [0, \varepsilon)$.

□

Hence the equations (3.9) are necessary conditions for every feasible direction. But in general they are not sufficient, i.e., not every vector fulfilling (3.9) is a feasible direction. Only in case of linear constraint functions $\mathbf{g}$ and $\mathbf{h}$ the conditions (3.9) completely characterize the set of feasible directions. For nonlinear constraints, one usually imposes additional conditions, so-called constraint qualifications, to avoid such problems. There exist several concepts of constraint qualifications, but we will only state here a very commonly used one that we need also for the derivations that follow.

Definition 3.9 (First order constraint qualification)
Let $\mathbf{u} \in \mathbb{R}^{n_u}$ be a feasible point. We say that the first order constraint qualification holds in $\mathbf{u}$, iff every vector $\mathbf{d} \in \mathbb{R}^{n_u}$, $\mathbf{d} \neq \mathbf{0}$, fulfilling the conditions (3.9) is a feasible direction.

As we want to establish later also second order optimality conditions, the following constraint qualification is needed.

Definition 3.10 (Second order constraint qualification)
Let $\mathbf{u} \in \mathbb{R}^{n_u}$ be a feasible point. We say that the second order constraint qualification holds in $\mathbf{u}$, iff every vector $\mathbf{d} \in \mathbb{R}^{n_u}$, $\mathbf{d} \neq \mathbf{0}$, that fulfills

$$\nabla_{\mathbf{u}} g_i(\mathbf{u})^T \mathbf{d} = 0, \qquad 1 \leq i \leq n_{\text{eq}} \tag{3.10a}$$
$$\nabla_{\mathbf{u}} h_i(\mathbf{u})^T \mathbf{d} = 0, \qquad i \in \mathcal{I}(\mathbf{u}), \tag{3.10b}$$

is a feasible direction of a twice continuously differentiable feasible path $\boldsymbol{\psi}(s)$, for which additionally $h_i(\boldsymbol{\psi}(s)) \equiv 0$ holds for all $i \in \mathcal{I}(\mathbf{u})$.

A commonly used criterion to ensure that these constraint qualifications hold is the following condition.

Definition 3.11 (Linear Independence Constraint Qualification (LICQ))
We say that the Linear Independence Constraint Qualification (LICQ) holds in a feasible point $\mathbf{u}$, if the gradients of the active constraints $\nabla_{\mathbf{u}} \tilde{g}_i(\mathbf{u})$, $1 \leq i \leq n_{\text{eq}} + |\mathcal{I}(\mathbf{u})|$, are linearly independent.

Definition 3.12 (Regular point)
A regular point is a feasible point that fulfills the LICQ.

For a regular point, finally the following theorem can be proven, see, e.g., [FM90].

Theorem 3.13
Let $\mathbf{u} \in \mathbb{R}^{n_u}$ be a regular point. Then the first and the second order constraint qualifications hold in $\mathbf{u}$.

Note that also in non-regular points the constraint qualification might hold, i.e., the preceding theorem is only a sufficient and not a necessary condition. However, in practical problems the solution $\mathbf{u}^*$ will often be a regular point.
Based on the preceding characterization of the set of feasible directions, we present in the next section first and second order conditions for local minima of the NLP (3.1).

3.2 Local optimality conditions

The tool that we will use in the following for the determination and characterization of possible solutions of the NLP, e.g., in the optimality conditions, is the so-called Lagrange function.

Definition 3.14 (Lagrange function and Lagrange multipliers)
The Lagrange function of the NLP (3.1) *is defined by*

$$\mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := c(\mathbf{u}) - \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{u}) - \boldsymbol{\mu}^T \mathbf{h}(\mathbf{u}). \tag{3.11}$$

Here $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are called Lagrange multiplier.

Using the Lagrange function, we can formulate the well-known Karush-Kuhn-Tucker (KKT) conditions, which are first order necessary conditions for a local minimum of the NLP. They were first derived by Karush [Kar39] and later independently by Kuhn and Tucker [KT51]. A proof can, e.g., be found in [NW99].

Theorem 3.15 (Karush-Kuhn-Tucker (KKT) conditions)
Let the first order constraint qualification hold in $\mathbf{u}^* \in \mathbb{R}^{n_u}$ and let furthermore $\mathbf{u}^*$ be a local minimum of the NLP (3.1).

Then there exist Lagrange multiplier $\boldsymbol{\lambda}^* \in \mathbb{R}^{n_{\text{eq}}}$ and $\boldsymbol{\mu}^* \in \mathbb{R}^{n_{\text{ineq}}}$, such that $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfy:

$$\nabla_{\mathbf{u}} \mathcal{L}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0} \tag{3.12a}$$
$$\mathbf{g}(\mathbf{u}^*) = \mathbf{0} \tag{3.12b}$$
$$\mathbf{h}(\mathbf{u}^*) \geq \mathbf{0} \tag{3.12c}$$
$$\boldsymbol{\mu}^* \geq \mathbf{0} \tag{3.12d}$$
$$\mu_i^* h_i^*(\mathbf{u}^*) = 0, \qquad 1 \leq i \leq n_{\text{ineq}}. \tag{3.12e}$$

Here the condition (3.12e) *is called complementarity condition.*

A triple $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is called a KKT point and is a candidate for a local minimum of the NLP.

If the local minimum is also a regular point, the following theorem can be proven.

Theorem 3.16 (Multiplier uniqueness)
Let $\mathbf{u}^* \in \mathbb{R}^{n_u}$ be a regular point and also a local minimum of the NLP (3.1). *Then the Lagrange multiplier $\boldsymbol{\lambda}^* \in \mathbb{R}^{n_{\text{eq}}}$ and $\boldsymbol{\mu}^* \in \mathbb{R}^{n_{\text{ineq}}}$ in Theorem 3.15 are unique.*

The complementary conditions of Theorem 3.15 imply that the multiplier corresponding to inactive constraints are equal to zero. Depending on whether the multiplier for the active inequality constraints are zero or not, we classify the corresponding constraints.

Definition 3.17 (Weakly and strongly active constraints, strict complementarity)
Let $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be a KKT point. Then we say that an active inequality constraint $h_i(\mathbf{u}^*) = 0$ is weakly active, if $\mu_i = 0$. Otherwise we say the constraint is strongly active.
We denote the set of the indices of the weakly active constraints with

$$\mathcal{I}_0(\mathbf{u}^*) := \{i \in \{1, \ldots, n_{\text{ineq}}\} | h_i(\mathbf{u}^*) = 0 \wedge \mu_i^* = 0\},$$

and the set of the indices of the strongly active constraints with

$$\mathcal{I}_+(\mathbf{u}^*) := \{i \in \{1, \ldots, n_{\text{ineq}}\} | h_i(\mathbf{u}^*) = 0 \wedge \mu_i^* > 0\},$$

such that $\mathcal{I}(\mathbf{u}^*) = \mathcal{I}_0(\mathbf{u}^*) \dot\cup \mathcal{I}_+(\mathbf{u}^*)$.
If all active constraints are strongly active, i.e., $\mathcal{I}(\mathbf{u}^*) = \mathcal{I}_+(\mathbf{u}^*)$*, we say that the KKT point fulfills the strong complementary condition.*

Remark 3.18
The geometrical interpretation of the first condition (3.12a) is that the gradient of the objective function can be expressed as linear combination of the gradients of the constraint functions. Alternatively, we can say that $\mathbf{u}^*$ is a stationary point for the Lagrange function $\mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ (with respect to $\mathbf{u}$ and provided that $\boldsymbol{\lambda} = \boldsymbol{\lambda}^*$ and $\boldsymbol{\mu} = \boldsymbol{\mu}^*$).

Remark 3.19 (Shadow prices)
The Lagrange multiplier of the active constraints can be interpreted as the costs of fulfilling the corresponding constraints and are hence often called shadow prices. More strictly speaking, if we interpret the cost function (via the changing KKT point) as a function of the right-hand side of the constraints, the Lagrange multiplier are the values of the directional derivative of the cost function in the positive direction. This means they describe how the cost function varies in the optimal solution, if the right-hand side of a constraint is changed from zero to an infinitesimal small positive value. This interpretation also gives an intuitive explanation for the condition (3.12d) that demands nonnegativity of the inequality multipliers in an optimal solution.

Example 3.20 (KKT conditions for QPs)
Consider the QP described in Definition 3.3. Then the KKT conditions of the problem are given for a point $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ by

$$\begin{aligned}
\mathbf{A}\mathbf{u}^* + \mathbf{a} - \mathbf{G}^T\boldsymbol{\lambda}^* - \mathbf{H}^T\boldsymbol{\mu}^* &= \mathbf{0} && \text{(3.13a)}\\
\mathbf{g} + \mathbf{G}\mathbf{u}^* &= \mathbf{0} && \text{(3.13b)}\\
\mathbf{h} + \mathbf{H}\mathbf{u}^* &\geq \mathbf{0} && \text{(3.13c)}\\
\boldsymbol{\mu}^* &\geq \mathbf{0} && \text{(3.13d)}\\
\mu_j^*(h_j + \mathbf{H}_{j\cdot}\mathbf{u}^*) &= 0, \qquad 1 \leq i \leq n_{\text{ineq}}, && \text{(3.13e)}
\end{aligned}$$

where $\mathbf{H}_{j\cdot}$ denotes here the j-th row of the matrix $\mathbf{H}$.

Definition 3.21 (KKT matrix)
If we consider a QP without inequalities, the KKT conditions in Example 3.20 can be written in a more compact form as

$$\begin{pmatrix} \mathbf{A} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u}^* \\ -\boldsymbol{\lambda}^* \end{pmatrix} = - \begin{pmatrix} \mathbf{a} \\ \mathbf{g} \end{pmatrix}, \tag{3.14}$$

where the matrix on the left side in called KKT matrix. It can be shown that the KKT matrix is regular, provided that $\mathbf{G}$ *has full rank (e.g., when* $\mathbf{u}^*$ *is a regular point) and that* $\mathbf{A}$ *is positive*

definite on the null space of $\mathbf{G}$. *For a proof, see, e.g., in [NW99]. In this case the equality constrained QP has exactly one KKT point. For inequality constrained QPs the existence and uniqueness of a KKT point can be proven, e.g., under the additional conditions that there exist feasible points and that the combined constraint Jacobian* $(\mathbf{G}^T, \mathbf{H}^T)^T$ *has full rank.*

While the presented first order conditions offer a possibility to determine candidates for local minima, it is not possible to give first order sufficient conditions without imposing strong conditions on the cost and constraint functions. Hence in the following we present second order necessary and sufficient conditions, that allow additional tests for (non-)optimality at a given KKT point. Here we assume that all functions occurring in the NLP are at least twice continuously differentiable.

Theorem 3.22 (Second order necessary conditions)
Let the first and second order constraint qualifications hold in $\mathbf{u}^* \in \mathbb{R}^{n_u}$ *and let furthermore* $\mathbf{u}^*$ *be a local minimum of the NLP* (3.1).

Then there exist Lagrange multiplier $\boldsymbol{\lambda}^* \in \mathbb{R}^{n_{\text{eq}}}$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^{n_{\text{ineq}}}$, *such that* $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *is a KKT point and that every vector* $\mathbf{d} \in \mathbb{R}^{n_u}$, $\mathbf{d} \neq \mathbf{0}$ *with*

$$\begin{aligned} \nabla_{\mathbf{u}} g_i(\mathbf{u}^*)^T \mathbf{d} &= 0, \qquad 1 \leq i \leq n_{\text{eq}} && \text{(3.15a)} \\ \nabla_{\mathbf{u}} h_i(\mathbf{u}^*)^T \mathbf{d} &= 0, \qquad i \in \mathcal{I}(\mathbf{u}^*), && \text{(3.15b)} \end{aligned}$$

also fulfills

$$\mathbf{d}^T \nabla^2_{\mathbf{uu}} \mathcal{L}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{d} \geq 0. \qquad \text{(3.15c)}$$

This means that the Hessian of the Lagrange function with respect to (w.r.t.) $\mathbf{u}$ *needs to be positive semidefinite on the subspace defined by* (3.15a) *and* (3.15b).

This second order necessary condition can be used to eliminate possible solution candidates that represent, e.g., local maxima or saddle-points which would fulfill the first order necessary conditions.
Next, we formulate a sufficient condition which allows us to actually verify whether or not a point is a strict local minimum.

Theorem 3.23 (Second order sufficient conditions)
Let $\mathbf{u}^* \in \mathbb{R}^{n_u}$ *be feasible.*

If there exist Lagrange multiplier $\boldsymbol{\lambda}^* \in \mathbb{R}^{n_{\text{eq}}}$ *and* $\boldsymbol{\mu}^* \in \mathbb{R}^{n_{\text{ineq}}}$, *such that* $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *is a KKT point and if every vector* $\mathbf{d} \in \mathbb{R}^{n_u}$, $\mathbf{d} \neq \mathbf{0}$ *with*

$$\begin{aligned} \nabla_{\mathbf{u}} g_i(\mathbf{u}^*)^T \mathbf{d} &= 0, \qquad 1 \leq i \leq n_{\text{eq}} && \text{(3.16a)} \\ \nabla_{\mathbf{u}} h_i(\mathbf{u}^*)^T \mathbf{d} &= 0, \qquad i \in \mathcal{I}_+(\mathbf{u}^*), && \text{(3.16b)} \\ \nabla_{\mathbf{u}} h_i(\mathbf{u}^*)^T \mathbf{d} &\geq 0, \qquad i \in \mathcal{I}_0(\mathbf{u}^*), && \text{(3.16c)} \end{aligned}$$

also fulfills

$$\mathbf{d}^T \nabla^2_{\mathbf{uu}} \mathcal{L}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{d} > 0, \qquad \text{(3.16d)}$$

then $\mathbf{u}^*$ *is a strict local minimum of the NLP* (3.1).
This means that the Hessian of the Lagrange function w.r.t. $\mathbf{u}$ *needs to be positive definite on the subspace defined by* (3.16a) *to* (3.16c). *A similar sufficient condition can be given for weak local minima, see, e.g., [FM90].*

Example 3.24 (Minimum of a convex QP)
Consider a convex QP, i.e., a QP of the type (3.3) with positive definite matrix $\mathbf{A}$. Furthermore, let $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be a KKT point of the problem. Then $\mathbf{u}^*$ is the only local minimum of the QP and also strict. Additionally, it can be shown that it is also the global minimum of the problem.

Note that there might exist points which fulfill the necessary conditions of Theorem 3.22 but cannot be verified using the sufficient condition in Theorem 3.23. It is possible to formulate more complex conditions that are both necessary and sufficient at once. But, as they often cannot be verified in practice, we will not state them here.
We finally formulate conditions that assure not only the existence of a local solution, but also guarantee the nonsingularity of the Jacobian of the KKT conditions in the solution. This allows in principle the successful use of Newton's method for the solution of the NLPs, provided the iterations are started with initial guesses for variables and multipliers that are close enough to the solution.

Theorem 3.25 (Sufficient condition for nonsingularity of KKT Jacobian)
Let $\mathbf{u}^* \in \mathbb{R}^{n_u}$ *be regular and* $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *a KKT point in which the strict complementarity condition holds.*
Furthermore, assume that every vector $\mathbf{d} \in \mathbb{R}^{n_u}$, $\mathbf{d} \neq \mathbf{0}$ *with*

$$\nabla_{\mathbf{u}} g_i(\mathbf{u}^*)^T \mathbf{d} = 0, \qquad 1 \leq i \leq n_{\text{eq}} \tag{3.17a}$$
$$\nabla_{\mathbf{u}} h_i(\mathbf{u}^*)^T \mathbf{d} = 0, \qquad i \in \mathcal{I}_+(\mathbf{u}^*), \tag{3.17b}$$

also fulfills

$$\mathbf{d}^T \nabla^2_{\mathbf{uu}} \mathcal{L}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{d} > 0. \tag{3.17c}$$

Then $\mathbf{u}^*$ *is a strict local minimum of the NLP* (3.1) *and the Jacobian* $\mathbf{J}^{\text{KKT}}$ *of the KKT conditions* (3.12a), (3.12b) *and* (3.12e) *in the KKT point, which is given by*

$$\mathbf{J}^{\text{KKT}}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \begin{pmatrix} \nabla^2_{\mathbf{uu}} \mathcal{L}(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) & -\nabla \mathbf{g}(\mathbf{u}^*) & -\nabla \mathbf{h}(\mathbf{u}^*) \\ \nabla \mathbf{g}(\mathbf{u}^*)^T & 0 & 0 \\ \operatorname{diag}(\boldsymbol{\mu}) \mathbf{h}(\mathbf{u}^*)^T & 0 & \operatorname{diag}(\mathbf{h}(\mathbf{u}^*)) \end{pmatrix} \tag{3.18}$$

is nonsingular.

3.3 Newton-type methods for NLP solution

Solution methods for the constrained NLP (3.1) are often based on the principle of attacking a simpler problem, that is in some sense related to the original NLP and that tries to captures its essential properties. In practice, this idea leads to iterative methods for the solution of the NLP

that solve a series of simpler subproblems where each, at least locally, approximates the original NLP in the current iterate.
In the case of an unconstrained problem it is for example very common to use a local quadratic approximation of the cost function. For the constrained NLP (3.1) we will discuss here methods that are based on a quadratic approximation of the Langrage function of the NLP and a linearization of the constraint functions in the current iterate. In each iteration, these methods solve a QP of the type (3.3) and are hence called Sequential Quadratic Programming (SQP) methods. The motivation to choose this specific type of subproblem is given by the sufficient conditions for optimality presented in the last section. There, it has be shown that the optimality of a point can be decided based on the curvature of the Lagrange function on the null-space of the Jacobians of the active constraints.
In the following, we present the general class of SQP methods as well as two important members of the class in more detail. We also show how the members of the SQP-family can be understood as Newton-type methods for the solution of the KKT conditions and address their local convergence properties. The approximation of NLPs by QPs is in general only locally of acceptable accuracy and this has also to be taken into account in practice during the computation of the next iterate. Therefore, we will shortly present globalization strategies, that ensure that in each iteration an actual progress towards the NLP solution is made by adapting the step length or restricting the step to an area where the approximation can be trusted.

3.3.1 The SQP framework

An SQP method tries to find a KKT point of the NLP (3.1) iteratively, starting at a initial guess $(\mathbf{u}^{(0)}, \boldsymbol{\lambda}^{(0)}, \boldsymbol{\mu}^{(0)})$ for optimization variables and Lagrange multiplier. Afterwards, it iterates

$$\begin{pmatrix} \mathbf{u}^{(k+1)} \\ \boldsymbol{\lambda}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^{(k)} \\ \boldsymbol{\lambda}^{(k)} \\ \boldsymbol{\mu}^{(k)} \end{pmatrix} + \alpha^{(k)} \begin{pmatrix} \boldsymbol{\Delta}\mathbf{u}^{(k)} \\ \boldsymbol{\lambda}^{\mathrm{QP}(k)} - \boldsymbol{\lambda}^{(k)} \\ \boldsymbol{\mu}^{\mathrm{QP}(k)} - \boldsymbol{\mu}^{(k)} \end{pmatrix}, \qquad k = 0, 1, 2, \ldots, \tag{3.19}$$

until a KKT point has been found, or a prescribed termination criterion is fulfilled, respectively. The $\alpha^{(k)} \in (0, 1]$ is called the stepsize of the SQP step. If $\alpha^{(k)}$ is set to one for all steps we speak of a full step method.
The increment $\boldsymbol{\Delta}\mathbf{u}^{(k)}$ in step k and the multiplier $\boldsymbol{\lambda}^{\mathrm{QP}(k)}$ and $\boldsymbol{\mu}^{\mathrm{QP}(k)}$ are computed from the solution of the following QP.

$$\min_{\boldsymbol{\Delta}\mathbf{u} \in \boldsymbol{\Omega}^{(k)} \subseteq \mathbb{R}^{n_u}} \quad \frac{1}{2} \boldsymbol{\Delta}\mathbf{u}^T \mathbf{B}^{(k)} \boldsymbol{\Delta}\mathbf{u}^T + \nabla_{\mathbf{u}} c(\mathbf{u}^{(k)})^T \boldsymbol{\Delta}\mathbf{u} \tag{3.20a}$$

subject to

$$\mathbf{g}(\mathbf{u}^{(k)}) + \mathbf{G}^{(k)} \boldsymbol{\Delta}\mathbf{u} = \mathbf{0} \tag{3.20b}$$

$$\mathbf{h}(\mathbf{u}^{(k)}) + \mathbf{H}^{(k)} \boldsymbol{\Delta}\mathbf{u} \geq \mathbf{0}. \tag{3.20c}$$

Here $\mathbf{B}^{(k)}$ is usually chosen as $\mathbf{B}^{(k)} \approx \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\mu}^{(k)})$, i.e., as an approximation of the Hessian of the Lagrange function with respect to $\mathbf{u}$ at the current values of optimization variables and multiplier. $\mathbf{G}^{(k)} := \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{u}^{(k)})$ and $\mathbf{H}^{(k)} := \frac{\partial \mathbf{h}}{\partial \mathbf{u}}(\mathbf{u}^{(k)})$ are the Jacobians of the constraint functions with respect to $\mathbf{u}$.
In practice, SQP methods differ mainly in the choice of the Hessian approximation $\mathbf{B}^{(k)}$ and in the choice, or the heuristic, respectively, to determine the stepsizes $\alpha^{(k)}$ and/or the region of allowed steps $\boldsymbol{\Omega}^{(k)} \subseteq \mathbb{R}^{n_u}$ in the QP subproblem. By suitable choices of $\alpha^{(k)}$ or $\boldsymbol{\Omega}^{(k)}$ a convergence of the algorithm to a local minimum (if existing) can be achieved regardless of the initial guess. This is addressed in Section 3.3.5.
Note that for the solution of the QP subproblems there exist several sophisticated algorithms with efficient implementations, which contribute to a large part to the overall efficiency and also the success of SQP methods in practice. Other favorable properties of SQP methods are that they are self-terminating and, especially, that they are able to identify the correct active set and the multiplier values of the NLP solution from the QP subproblem without the need for a prior knowledge of them. This is formulated in the following Lemma.

Lemma 3.26 (Self-termination and identification of the active set)
Let $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be a KKT point of the NLP, i.e., fulfilling the KKT conditions (3.12). Then the QP subproblem formulated in $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ has the solution $(\mathbf{0}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$. If the Hessian approximation $\mathbf{B}$ in the QP subproblem is positive definite on the null space of the constraint Jacobian we can see from the remarks in Definition 3.21 that this is also the unique KKT point of the QP subproblem. Hence, the QP subproblem will in this case never generate a step away from the KKT point of the NLP. The other way round it is true that if the QP subproblem formulated at a point $(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\mu}^{(k)})$ has the KKT point $(\mathbf{0}, \boldsymbol{\lambda}^{(k)}_{\mathrm{QP}}, \boldsymbol{\mu}^{(k)}_{\mathrm{QP}})$, then $(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}_{\mathrm{QP}}, \boldsymbol{\mu}^{(k)}_{\mathrm{QP}})$ is a KKT point of the NLP. If we assume that in $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ additionally the second order sufficient conditions of Theorem 3.23 as well as the strict complementarity condition are fulfilled it can be shown that the SQP algorithm determines the correct active set also in a neighborhood of the solution $(\mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$.

Proof:

The first claims follow directly from the comparison of the KKT conditions for the NLP (3.12) with the KKT conditions for the corresponding QP. A more detailed discussion and a proof of the last claim can be found in [Rob74].

□

Remark 3.27 (Warm start of the QP subproblems)
While in the solution of the QP subproblems the active set and the multiplier values for the current QP subproblem can be found independently of their initial guesses, SQP methods benefit in practice very much of the possibility to warm start the QP subproblem solution in connection with active set QP solvers. The solution process of the QP is often much faster if the active set and the multiplier of the current QP are initialized with the values from the last QP subproblem. This is especially the case if the algorithm comes closer to the solution, as here the steps become

smaller and one can then argue that the subproblems (and hence also their active sets) are more and more related to each other.

3.3.2 Full step exact-Hessian SQP

The full step exact-Hessian SQP method has first been presented by [Wil63]. It is characterized by the choice of $\alpha^{(k)} = 1$ and $\mathbf{\Omega}^{(k)} = \mathbb{R}^{n_u}$ for all k as well as $\mathbf{B}^{(k)} = \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\mu}^{(k)})$. This choice of the Hessian approximation leads to a very good local convergence behavior. To see this, we consider first an equality constrained NLP. The KKT conditions for the QP subproblem solution $(\boldsymbol{\Delta}\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}_{\mathrm{QP}})$ in the point $(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)})$ are then given by

$$\begin{aligned} \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)})\boldsymbol{\Delta}\mathbf{u}^{(k)} + \nabla_{\mathbf{u}}c(\mathbf{u}^{(k)}) - \mathbf{G}(\mathbf{u}^{(k)})^T\boldsymbol{\lambda}^{(k)}_{\mathrm{QP}} &= \mathbf{0} \quad (3.21a) \\ \mathbf{g}(\mathbf{u}^{(k)}) + \mathbf{G}(\mathbf{u}^{(k)})\boldsymbol{\Delta}\mathbf{u}^{(k)} &= \mathbf{0}. \quad (3.21b) \end{aligned}$$

This can be written in matrix form as

$$\begin{pmatrix} \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}) & -\mathbf{G}(\mathbf{u}^{(k)})^T \\ \mathbf{G}(\mathbf{u}^{(k)}) & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta}\mathbf{u}^{(k)} \\ \boldsymbol{\lambda}^{(k)}_{\mathrm{QP}} \end{pmatrix} = - \begin{pmatrix} \nabla_{\mathbf{u}}c(\mathbf{u}^{(k)}) \\ \mathbf{g}(\mathbf{u}^{(k)}) \end{pmatrix}. \quad (3.22)$$

By rewriting $\boldsymbol{\lambda}^{(k)}_{\mathrm{QP}} = \boldsymbol{\lambda}^{(k)} + \boldsymbol{\Delta}\boldsymbol{\lambda}^{(k)}$ and regarding that $\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}) = \nabla_{\mathbf{u}}c(\mathbf{u}^{(k)}) - \mathbf{G}(\mathbf{u}^{(k)})^T\boldsymbol{\lambda}^{(k)}$ we obtain

$$\begin{pmatrix} \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}) & -\mathbf{G}(\mathbf{u}^{(k)})^T \\ \mathbf{G}(\mathbf{u}^{(k)}) & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta}\mathbf{u}^{(k)} \\ \boldsymbol{\Delta}\boldsymbol{\lambda}^{(k)} \end{pmatrix} = - \begin{pmatrix} \nabla_{\mathbf{u}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}) \\ \mathbf{g}(\mathbf{u}^{(k)}) \end{pmatrix}. \quad (3.23)$$

This rule for the determination of the step in variables and multipliers of the NLP in the exact-Hessian SQP is now identical to the formula for the increment in Newton's method, if it is applied to the KKT conditions of the NLP given by

$$\begin{pmatrix} \nabla_{\mathbf{u}}\mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)}) \\ \mathbf{g}(\mathbf{u}^{(k)}) \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \quad (3.24)$$

Hence the convergence of the full step exact-Hessian SQP is locally quadratic, like the convergence of Newton's method. Note that it is sufficient to consider an equality constrained problem also to analyze the local convergence behavior of the method in the case of an NLP with inequality constraints, provided the KKT point fulfills the sufficient optimality conditions of Theorem 3.23 and the strict complementarity conditions. In this case, like addressed shortly in Lemma 3.26, the active set remains constant in the vicinity of the KKT point, the QP subproblem can be interpreted as a small perturbation of the original problem and the SQP method will eventually find the correct active set. This is proven and discussed in more detail in [Rob74].
Based on the considerations above, both the initial guess for the variables as well as for the multiplier would have to be close enough to the solution values to obtain the quadratic convergence. However, by exploiting the special structure of the KKT system and its linearization, this restriction on the initial multiplier guess can be softened. This has been proven by Fletcher [Fle87] in the following theorem.

Theorem 3.28 (Local convergence of full step exact-Hessian SQP)
Let $(\mathbf{u}^*, \boldsymbol{\lambda}^*)$ satisfy the sufficient optimality conditions of Theorem 3.23 for an equality constrained NLP (3.1). If the initial value $\mathbf{u}^{(0)}$ is close enough to $\mathbf{u}^*$ and the initial multiplier value $\boldsymbol{\lambda}^{(0)}$ is chosen such that the KKT matrix

$$\begin{pmatrix} \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}^{(0)}, \boldsymbol{\lambda}^{(0)}) & \mathbf{G}(\mathbf{u}^{(0)}, \boldsymbol{\lambda}^{(0)})^T \\ \mathbf{G}(\mathbf{u}^{(0)}, \boldsymbol{\lambda}^{(0)}) & \mathbf{0} \end{pmatrix} \tag{3.25}$$

is nonsingular, then the sequence of iterates $(\mathbf{u}^{(k)}, \boldsymbol{\lambda}^{(k)})$ obtained by applying the full step exact-Hessian SQP method converges quadratically to $(\mathbf{u}^*, \boldsymbol{\lambda}^*)$, i.e.,

$$\left\| \begin{pmatrix} \mathbf{u}^{(k+1)} - \mathbf{u}^* \\ \boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^* \end{pmatrix} \right\| \leq \kappa \left\| \begin{pmatrix} \mathbf{u}^{(k)} - \mathbf{u}^* \\ \boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^* \end{pmatrix} \right\|^2, \tag{3.26}$$

with a constant $\kappa \geq 0$.

Remark 3.29
The equivalence of the solution of the QP subproblem in the exact-Hessian SQP and a Newton step for the solution of the KKT conditions of the NLP motivates the interpretation of SQP methods as Newton-type methods for the NLP solution. The use of an approximation of the Hessian of the Lagrangian in an SQP method can be understood as the use of an approximation of the Jacobian in the corresponding Newton method.

3.3.3 Full step constrained Gauss-Newton

The constrained Gauss-Newton method performs very well on the special class of Nonlinear Least-Squares (NLSQ) problems defined in (3.2). It is characterized by the choices $\alpha^{(k)} = 1$ and $\boldsymbol{\Omega}^{(k)} = \mathbb{R}^{n_u}$ for all k and the Hessian approximation

$$\mathbf{B}^{(k)} = \mathbf{J}^T\mathbf{J}, \qquad \text{with } \mathbf{J} := \frac{\partial \mathbf{r}}{\partial \mathbf{u}}(\mathbf{u}^{(k)}). \tag{3.27}$$

If we compare this approximation with the exact Hessian of the Lagrangian of the problem, which is given by

$$\begin{aligned} \nabla^2_{\mathbf{uu}}\mathcal{L}(\mathbf{u}, \boldsymbol{\lambda}) &= \frac{\partial}{\partial \mathbf{u}}\left[\frac{\partial \mathbf{r}}{\partial \mathbf{u}}(\mathbf{u})^T\mathbf{r}(\mathbf{u}) + \mathbf{G}(\mathbf{u})^T\boldsymbol{\lambda} + \mathbf{H}(\mathbf{u})^T\boldsymbol{\mu}\right] \\ &= \underbrace{\frac{\partial \mathbf{r}}{\partial \mathbf{u}}(\mathbf{u})^T\frac{\partial \mathbf{r}}{\partial \mathbf{u}}(\mathbf{u})}_{=\mathbf{J}^T\mathbf{J}} + \sum_{i=1}^{n_{\text{res}}} r_i(\mathbf{u})\frac{\partial^2 r_i}{\partial \mathbf{u}^2}(\mathbf{u}) + \sum_{i=1}^{n_{\text{eq}}} \lambda_i \frac{\partial^2 g_i}{\partial \mathbf{u}^2}(\mathbf{u}) + \sum_{i=1}^{n_{\text{ineq}}} \mu_i \frac{\partial^2 h_i}{\partial \mathbf{u}^2}(\mathbf{u}), \end{aligned} \tag{3.28}$$

we can expect that the approximation will be good in the vicinity of a solution, e.g., if the problem functions are only mildly nonlinear, or if the residual function $\mathbf{r}$ becomes small and additionally the multiplier can be bounded in terms of the residual function.

Note that the Gauss-Newton approximation of the Hessian only needs first order derivatives of the residual function, which makes the computation somewhat simpler than that of the exact Hessian. Furthermore, it is independent of the Lagrange multiplier. It should also be noted that for the resulting subproblem class specialized solutions approaches exist, which do not need the explicit computation of the matrix $\mathbf{J}^T\mathbf{J}$, but rather work based on $\mathbf{J}$, see, e.g., [Boc87].
The local convergence behavior of the full-step constrained Gauss-Newton method is determined by the accuracy of the Hessian approximation. In general, only linear convergence can be expected. A criterion to determine whether the method converges at all and to estimate the convergence rate is given by the local contraction theorem for Newton-type methods which can be found in [Boc87], and also later in this work as Proposition 5.62 on page 131.

3.3.4 Other SQP variants

In the last two sections we presented two common SQP-type methods that are also of importance later in this thesis. Besides them, there exist a variety of other SQP-type methods. Most prominent among them and widely used are SQP methods that use a Quasi-Newton method for the approximation of the Hessian of the Lagrangian. This means that the Hessian approximation $\mathbf{B}^{(k+1)}$ is obtained based on $\mathbf{B}^{(k)}$ and an update formula such as the Davidon-Fletcher-Powell (DFP) or Broyden-Fletcher-Goldfarb-Shanno (BFGS) update. This idea was first proposed by Garcia Palomares and Mangasarian in [PM76]. It was further developed by Han [Han76] and Powell [Pow78c, Pow78a, Pow78b], who also presented a very successful implementation of his algorithm. These algorithms achieve usually a locally superlinear convergence to a local minimum fulfilling the usual conditions. An overview on these and more SQP variants working with approximations of the Hessian can be found in [Lei99].
Another class of SQP methods is given by methods that do not only approximate the Hessian of the Lagrangian, but also the constraint Jacobians $\mathbf{G}$ and/or $\mathbf{H}$. This kind of methods is called inexact SQP methods, where the term inexact refers here to the Jacobian approximation, not to an inexact solution of the QP subproblem, e.g., using iterative methods, as it can also be found in literature. The local convergence of these methods can again be proven, depending on the quality of the approximation, by interpreting them as Newton-type methods and applying the local contraction theorem. They can be very efficient in practice, because for each iteration they need, besides the computation of the Hessian and Jacobian approximations, mainly the gradient of the Lagrangian which can be computed efficiently using the adjoint mode of Automatic Differentiation (AD). For more information on inexact methods for equality constrained problems we refer to [JS97, HV01] and the works of Griewank and Walther [GW02, Wal08] as well as the references therein. For the extension to problems with inequality constraints consult the works of Bock, Diehl and coworkers [BDK04, BDKS07, DWBK09] and the references therein. An implementation of an inexact SQP method in the context of Nonlinear Model Predictive Control (NMPC) is described in [Wir06], an application to optimal control in [WAK+08].
Although we will not go deeper into the details of these class of methods in this work, the lifting approach presented in this thesis can in principle also be used in connection with update strategies and inexact methods to obtain efficient algorithms.

3.3.5 Globalization strategies

Until now, we only considered the local convergence properties of SQP methods. In practice, however, it is important that the methods also converge to a local minimum if they are not started in the direct vicinity of the solution. In general, this cannot be guaranteed for full step methods, hence a so-called globalization strategy is needed. The globalization strategy ensures that during the iterations a "sufficient" progress towards the solution is made.
This progress can be measured in the unconstrained case directly by the achieved reduction in the cost functional. In the constrained case this is more difficult, as it is a priori not possible to say whether it is better to improve in a step the cost function while increasing the infeasibility of the constraints or vice versa. A common tool to measure the progress to the solution in the constrained case are so-called merit functions, that usually combine the cost function with weighted penalty terms for constraint violations. Ideally, a merit function is exact, i.e., that a local minimum of the merit function is also a local minimum of the corresponding NLP. A popular choice is the l_1 penalty function

$$\mathcal{P}(\mathbf{u}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = c(\mathbf{u}) + \sum_{i=1}^{n_{\text{eq}}} \bar{\lambda}_i |g_i(\mathbf{u})| + \sum_{i=1}^{n_{\text{ineq}}} \bar{\mu}_i |\min(0, h_i(\mathbf{u}))|. \tag{3.29}$$

This merit function is exact, provided that the penalty factors are chosen in a way that

$$\bar{\boldsymbol{\lambda}} \geq \boldsymbol{\lambda}^* \quad \text{and} \quad \bar{\boldsymbol{\mu}} \geq \boldsymbol{\mu}^*. \tag{3.30}$$

Then. a KKT point fulfilling the conditions of Theorem 3.25 is a strict local minimizer not only of the NLP, but also of the l_1 penalty function (see, e.g., [Fle87, Han77]). Another important requirement is the compatibility of the step direction $\boldsymbol{\Delta u}$ generated by the QP subproblem with the merit function, i.e., that $\boldsymbol{\Delta u} \neq \mathbf{0}$ should always be a descent direction for the merit function. The merit function can then be used to test the progress to the NLP solution after the computation of the step. If the progress is not sufficient, then the step is rejected and will be modified by the globalization strategy until it is accepted.
An alternative to the use of merit functions, is the so-called class of filter methods [FL02, WB02] employing ideas from multi-objective optimization. These methods take into account the (independent) objectives of feasibility and minimization of the cost functional and accept a new iterate if it improves at least one of these objectives compared to the iterates already present in the filter. Afterwards, this new iterate might be added to the filter, based on some heuristic criterion. Usually, filter methods are not scaling invariant with respect to the variables and the problem functions, contrary to the approaches based on the l_1 merit function which we address in this thesis.
Ideally, a globalization strategy should be designed in a way that the usually good local convergence properties of the SQP method are not destroyed by rejecting full steps close to the solution, thus avoiding, e.g., the Maratos effect [Mar78]. In practice there exist two major classes of globalization strategies in connection with merit functions: line-search strategies and trust-region strategies. We will shortly explain both of them in the following. A more detailed overview on globalization strategies including measures to avoid the Maratos effect, like second-order-correction [MP82] or a watchdog strategy, [CLPP82] is given, e.g., in [Lei99].

Line-search methods

Line search strategies for globalization work by modifying the step size $\alpha^{(k)}$ of the SQP step. This means first the step direction $\mathbf{\Delta u}^{(k)}$ is computed by solving the QP subproblem with $\mathbf{\Omega}^{(k)} = \mathbb{R}^{n_u}$, and afterwards $\alpha^{(k)} \in (0, 1]$ is determined such that enough progress to the solution is made. In an exact line search strategy the minimum of the merit function on the line $\{\tilde{\mathbf{u}} | \tilde{\mathbf{u}} = \mathbf{u}^{(k)} + \alpha \mathbf{\Delta u}^{(k)}, 0 < \alpha \leq 1\}$ is determined and the corresponding α is taken as steplength for the current step. Although theoretically very appealing, the exact line search is normally not used in practice due to its effort. Hence, heuristics are employed that usually start by testing if a full-step will give "sufficient" progress and reduce the stepsize if necessary by certain rules until the test is fulfilled. Note that line search strategies require in general a positive definite Hessian approximation in the SQP method (or at least the positive definiteness of the projected Hessian, i.e., the Hessian approximation on the null space of the Jacobians of the active constraints) to ensure that the computed direction is indeed a descend direction for the merit function.

Trust-region methods

Trust-region strategies are based on the idea of determining a region in which the current quadratic approximation of the NLP is reasonable, the so-called trust region. The trust region is imposed as additional restriction on the QP subproblem in form of the set $\mathbf{\Omega}^{(k)}$, while $\alpha^{(k)}$ is set to 1. Usually the trust region is chosen in form of $\|\mathbf{\Delta u}\| \leq \rho$, where as norm mostly the 1-norm or the euclidean norm in $\mathbb{R}^{n_u}$ are used. ρ is here called the trust radius. The radius of the trust region is usually adapted heuristically from iteration to iteration, based on the performance of the the computed step, or the new iterate based on this step, respectively. If the expected improvement in the merit function has been achieved, the trust region is normally increased. If not, it is decreased until the improvement is satisfying. A trust region strategy is especially well-suited for exact-Hessian SQP methods, as here indefinite Hessians might occur regularly during the iteration process. The additional trust-region constraints help to cope with the problem of an indefinite projected Hessian in the QP subproblem and with the resulting possibility of an unbounded QP solution. One drawback of the trust-region strategy is that it might leads to infeasible QP subproblems. A reason for this is, e.g., that at some point, if the trust region becomes too small, the trust regions constraints will probably contradict the linearized NLP constraints in the QP. In this case a relaxation of the constraints might be a possible remedy. We will shortly address later in Section 7.2.2 a suitable choice of a merit function and the implementation of a corresponding trust-region strategy in the framework of our lifted exact-Hessian SQP algorithm.

4 Lifted methods for Nonlinear Programming

In this chapter we describe a new "lifting" approach for the solution of nonlinear optimization problems (NLPs) that have objective and constraint functions with intermediate variables. The approach has first been presented in [AD10]. It has been observed, in particular in the context of the solution of boundary value problems by shooting methods, that transferring a nonlinear root finding problem into a higher-dimensional space might offer advantages in terms of convergence rates and region of attraction [Boc78a, Osb69, SB92].

This classical "multiple shooting" method works, like described in Section 1.2.3 for the Optimal Control Problem (OCP) case, by introducing intermediate variables as additional degrees of freedom and corresponding constraints to ensure equivalence of the solutions with the original problem. It then solves this equivalent augmented system - which we call the "lifted" system - instead of the original system by a Newton-type method. At first sight, the increased size of the lifted system seems to render each Newton-type iteration more expensive. This can be overcome, however, to a large extent by structure-exploiting linear algebra in each Newton-type step (see [Sch88, Lei99, Sch05]).

Besides the classical domain of multiple shooting, parameter estimation and optimal control for ODEs and DAEs, where the natural choice of intermediate values are the system states at different timepoints, there exist other problem classes that can benefit from "lifting". A direct transfer of the idea can be made to the case of discrete time models or optimization in the context of kinematic chains arising in robotics.

However, "multiple shooting" and related "lifted" approaches are often not used due to the increased programming burden. Usually, it would be necessary to split up the original algorithm according to the choice of intermediate values and the structure of the problem into a sequence of subfunctions. From these, one has to compute and to assemble the quantities and derivatives of the augmented system. Afterwards, the augmented problem has to be "condensed" again to obtain small reduced subproblems needed for an efficient step computation. All of these steps are technically non-trivial to implement.

In this chapter we propose algorithms for the solution of nonlinear optimization problems that solve the augmented system by a structure-exploiting Newton-type method, yet do not require any additional user knowledge about the structure of the problem functions or the meaning of the intermediate variables. We show that the cost of each iteration of these "lifted" methods is nearly identical to the cost of one iteration for the solution of the original problems. Furthermore, we make a first step towards proving the superior local convergence speed of lifted Newton methods.

We first explain the idea of lifting in Section 4.1 at the example of Newton's method for a root

finding problem and derive a "lifted" Newton algorithm. In Section 4.2 we discuss the application of the lifting approach to optimization and derive a lifted Gauss-Newton method as well as a lifted SQP method for equality and inequality constrained optimization problems that is based on adjoint gradient computations. Equivalence of the last method with the iterations obtained by a full-space SQP method is proven. In Section 4.3 we discuss under which circumstances "lifted" approaches converge faster than non-lifted ones, and give a proof in a simplified setting. In Section 4.4 we present a first tutorial application example for the lifted Newton algorithm. More numerical examples are given later on in Chapter 8.

4.1 The Lifted Newton Method

We first consider the problem of solving a nonlinear system of equations, represented by

$$\mathbf{f}(\mathbf{u}) = \mathbf{0}, \tag{4.1}$$

where the evaluation of the function $\mathbf{f} \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_u})$ is given in form of a possibly complex algorithm with several intermediate variables. Denoting these intermediate variables by $\mathbf{x_i} \in \mathbb{R}^{n_i}$, for $i = 1, 2, \ldots, m$, and disregarding further internal structure, we summarize the algorithm in the generic form

$$\mathbf{x_i} := \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{i-1}}), \quad \text{for} \quad i = 1, 2, \ldots, m, \tag{4.2}$$

where the final output $\mathbf{f}(\mathbf{u})$ is given by

$$\boldsymbol{\phi_f}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}). \tag{4.3}$$

It is straightforward to see that the original system (4.1) is equivalent to the "lifted" system of nonlinear equations

$$\mathbf{g}(\mathbf{u}, \mathbf{x}) = \mathbf{0}, \tag{4.4}$$

with $n_x = \sum_{i=1}^{m} n_i$, $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_m})$ and where $\mathbf{g} \in \mathcal{C}^1(\mathbb{R}^{n_u} \times \mathbb{R}^{n_x}, \mathbb{R}^{n_u} \times \mathbb{R}^{n_x})$ is given by

$$\mathbf{g}(\mathbf{u}, \mathbf{x}) = \begin{pmatrix} \boldsymbol{\phi_1}(\mathbf{u}) & - & \mathbf{x_1} \\ \boldsymbol{\phi_2}(\mathbf{u}, \mathbf{x_1}) & - & \mathbf{x_2} \\ \vdots & & \\ \boldsymbol{\phi_m}(\mathbf{u}, \mathbf{x_1}, \ldots, \mathbf{x_{m-1}}) & - & \mathbf{x_m} \\ \boldsymbol{\phi_f}(\mathbf{u}, \mathbf{x_1}, \ldots, \mathbf{x_m}) & & \end{pmatrix}. \tag{4.5}$$

To solve the augmented system (4.4) the lifted Newton method iterates, starting at a guess $(\mathbf{x}^{(0)}, \mathbf{u}^{(0)})$,

$$\begin{pmatrix} \mathbf{x}^{(k+1)} \\ \mathbf{u}^{(k+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{x}^{(k)} \\ \mathbf{u}^{(k)} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\Delta}\mathbf{x}^{(k)} \\ \boldsymbol{\Delta}\mathbf{u}^{(k)} \end{pmatrix} \tag{4.6a}$$

with

$$\begin{pmatrix} \boldsymbol{\Delta}\mathbf{x}^{(k)} \\ \boldsymbol{\Delta}\mathbf{u}^{(k)} \end{pmatrix} = - \left[\frac{\partial \mathbf{g}}{\partial(\mathbf{u}, \mathbf{x})}(\mathbf{x}^{(k)}, \mathbf{u}^{(k)}) \right]^{-1} \mathbf{g}(\mathbf{x}^{(k)}, \mathbf{u}^{(k)}). \tag{4.6b}$$

To initialize the value $\mathbf{x}^{(0)}$, we can simply call the algorithm defining the original function $\mathbf{f}(\mathbf{u}^{(0)})$ and add a few lines to output all intermediate variables, i.e., we call Algorithm 4.1. Please note that we are also free to choose the intermediate values otherwise, and that it is often advantageous to do so.

Algorithm 4.1: Function with output of intermediate variables

Input : $\mathbf{u} \in \mathbb{R}^{n_u}$
Output: $\mathbf{x_1} \in \mathbb{R}^{n_1}, \ldots, \mathbf{x_m} \in \mathbb{R}^{n_m}, \mathbf{f} \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\mathbf{x_i} = \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{i-1}})$;
  end for
  $\mathbf{f} := \boldsymbol{\phi_f}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m})$;
end

By modifying the user given Algorithm 4.1 slightly, we can easily define the residual function $\mathbf{g}(\mathbf{u}, \mathbf{y})$, as follows by Algorithm 4.2.

Algorithm 4.2: Residual function $\mathbf{g}(\mathbf{u}, \mathbf{y})$

Input : $\mathbf{u}, \mathbf{y_1}, \ldots, \mathbf{y_m}$
Output: $\mathbf{g_1}, \ldots, \mathbf{g_m}, \mathbf{f}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\mathbf{x_i} = \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{i-1}})$;
    $\mathbf{g_i} = \mathbf{x_i} - \mathbf{y_i}$;
    $\mathbf{x_i} = \mathbf{y_i}$;
  end for
  $\mathbf{f} = \boldsymbol{\phi_f}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m})$;
end

Thus, it is easy to transform a given user function into a function that outputs the residuals. Note that it is not necessary that all $\mathbf{x_i}$ are distinct variables with separately allocated memory within the program code. The only code modification is to add after each computation of an intermediate variable two lines (or even only one line calling a more convenient function defined in Algorithm 4.7 on page 85) that store the residual and set the variable to the given input value $\mathbf{y_i}$.

For notational convenience, we define

$$\boldsymbol{\phi}(\mathbf{u},\mathbf{x}) := \begin{pmatrix} \boldsymbol{\phi}_{\mathbf{1}}(\mathbf{u}) \\ \boldsymbol{\phi}_{\mathbf{2}}(\mathbf{u},\mathbf{x}_{\mathbf{1}}) \\ \vdots \\ \boldsymbol{\phi}_{\mathbf{m}}(\mathbf{u},\mathbf{x}_{\mathbf{1}},\ldots,\mathbf{x}_{\mathbf{m-1}}) \end{pmatrix}, \tag{4.7}$$

such that

$$\mathbf{g}(\mathbf{u},\mathbf{x}) = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{u},\mathbf{x}) - \mathbf{x} \\ \boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u},\mathbf{x}) \end{pmatrix}. \tag{4.8}$$

It is straightforward to see that Algorithm 4.1 delivers, for given $\mathbf{u}$, the unique solution $\tilde{\mathbf{x}}$ of $\boldsymbol{\phi}(\mathbf{u},\tilde{\mathbf{x}}) - \tilde{\mathbf{x}} = \mathbf{0}$.

To perform a Newton method on the augmented system we have to calculate the increments in (4.6b). Dropping the index k for notational convenience, we have to solve in every iteration the linear system

$$\boldsymbol{\phi}(\mathbf{u},\mathbf{x}) - \mathbf{x} + \left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x}) - \mathbb{I}_{n_x}\right)\boldsymbol{\Delta}\mathbf{x} + \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}}(\mathbf{u},\mathbf{x})\boldsymbol{\Delta}\mathbf{u} = \mathbf{0}, \tag{4.9a}$$

$$\boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u},\mathbf{x}) + \frac{\partial \boldsymbol{\phi}_{\mathbf{f}}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x})\boldsymbol{\Delta}\mathbf{x} + \frac{\partial \boldsymbol{\phi}_{\mathbf{f}}}{\partial \mathbf{u}}(\mathbf{u},\mathbf{x})\boldsymbol{\Delta}\mathbf{u} = \mathbf{0}, \tag{4.9b}$$

where $\mathbb{I}_{n_x}$ represents the identity operator in $\mathbb{R}^{n_x}$. Due to the fact that the square matrix $\left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x}) - \mathbb{I}_{n_x}\right)$ is lower triangular with nonzero diagonal, and thus invertible, (4.9a) is equivalent to

$$\begin{aligned} \boldsymbol{\Delta}\mathbf{x} &= \underbrace{-\left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x}) - \mathbb{I}_{n_x}\right)^{-1}(\boldsymbol{\phi}(\mathbf{u},\mathbf{x}) - \mathbf{x}) +}_{=:\mathbf{a}} \\ &\quad \underbrace{-\left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x}) - \mathbb{I}_{n_x}\right)^{-1}\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}}(\mathbf{u},\mathbf{x})}_{=:\mathbf{A}}\boldsymbol{\Delta}\mathbf{u} \\ &= \mathbf{a} + \mathbf{A}\boldsymbol{\Delta}\mathbf{u}. \end{aligned} \tag{4.10}$$

Based on this, we can "condense" the second equation to

$$\begin{array}{rcccccc} 0 &=& \boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u},\mathbf{x}) + \frac{\partial \boldsymbol{\phi}_{\mathbf{f}}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x})\mathbf{a} &+& \left(\frac{\partial \boldsymbol{\phi}_{\mathbf{f}}}{\partial \mathbf{u}}(\mathbf{u},\mathbf{x}) + \frac{\partial \boldsymbol{\phi}_{\mathbf{f}}}{\partial \mathbf{x}}(\mathbf{u},\mathbf{x})\mathbf{A}\right) & \boldsymbol{\Delta}\mathbf{u} \\ &=:& \mathbf{b} &+& \mathbf{B} & \boldsymbol{\Delta}\mathbf{u} \end{array} \tag{4.11}$$

If the "reduced quantities" $\mathbf{a}$, $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{B}$ were known, we could easily compute the step by

$$\boldsymbol{\Delta}\mathbf{u} = -\mathbf{B}^{-1}\,\mathbf{b} \tag{4.12a}$$

$$\boldsymbol{\Delta}\mathbf{x} = \mathbf{a} + \mathbf{A}\,\boldsymbol{\Delta}\mathbf{u}. \tag{4.12b}$$

4.1.1 An algorithmic trick for efficient computation of reduced quantities

In the following we will present an algorithmic trick to compute the vectors $\mathbf{a}$, $\mathbf{b}$ and the matrices $\mathbf{A}$, $\mathbf{B}$ efficiently, without the need to form or invert the derivatives of $\boldsymbol{\phi}$ explicitly. This trick is a generalization of "Schlöder's trick" [Sch88] which is since long known in the context of multiple shooting for parameter estimation, and was extended to optimal control by Schäfer [Sch05]. In all these existing approaches, however, the algorithms are specially tailored to specific sparsity structures. On the other hand, the new generalized trick does not require any structure or user input, apart from a minimal number of extra lines of code into the function to be "lifted", as illustrated in Algorithm 4.6.
To derive the trick, we introduce a function $\mathbf{z}(\mathbf{u}, \mathbf{d})$ as follows. For given vectors $\mathbf{u}$ and $\mathbf{d}$, the unique solution $\tilde{\mathbf{z}}$ of the system $\boldsymbol{\phi}(\mathbf{u}, \tilde{\mathbf{z}}) - \tilde{\mathbf{z}} - \mathbf{d} = \mathbf{0}$, that we will denote in the following by $\mathbf{z}(\mathbf{u}, \mathbf{d})$, can be computed easily by Algorithm 4.3. The algorithm simultaneously computes the value $\boldsymbol{\phi}_\mathbf{f}(\mathbf{u}, \mathbf{z}(\mathbf{u}, \mathbf{d}))$.

Algorithm 4.3: Modified function $\mathbf{z}(\mathbf{u}, \mathbf{d})$

Input : $\mathbf{u}, \mathbf{d_1}, \ldots, \mathbf{d_m}$
Output: $\mathbf{z_1}, \ldots, \mathbf{z_m}, \mathbf{f}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\mathbf{x_i} = \boldsymbol{\phi}_\mathbf{i}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{i-1}})$;
    $\mathbf{z_i} = \mathbf{x_i} - \mathbf{d_i}$;
    $\mathbf{x_i} = \mathbf{z_i}$;
  end for
  $\mathbf{f} = \boldsymbol{\phi}_\mathbf{f}(\mathbf{u}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m})$;
end

The derivatives of the function $\mathbf{z}(\mathbf{u}, \mathbf{d})$ with respect to $\mathbf{u}$ and $\mathbf{d}$ help us in computing the vector $\mathbf{a}$ and the matrix $\mathbf{A}$ as well as $\mathbf{b}$ and $\mathbf{B}$. To see this, we first observe that

$$\mathbf{z}(\mathbf{u}, \boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x}) = \mathbf{x}.$$

Thus, by setting $\mathbf{d} = \boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x}$, we can call Algorithm 4.3 to obtain $\mathbf{x} = \mathbf{z}(\mathbf{u}, \mathbf{d})$ and $\boldsymbol{\phi}_\mathbf{f}(\mathbf{u}, \mathbf{x})$. On the other hand, from the defining equation of $\mathbf{z}(\mathbf{u}, \mathbf{d})$, namely

$$\boldsymbol{\phi}(\mathbf{u}, \mathbf{z}(\mathbf{u}, \mathbf{d})) - \mathbf{z}(\mathbf{u}, \mathbf{d}) - \mathbf{d} = \mathbf{0},$$

we obtain the following two equations by total differentiation with respect to u and d:

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{x}) + \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u}, \mathbf{x}) \frac{\partial \mathbf{z}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{d}) - \frac{\partial \mathbf{z}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{d}) = \mathbf{0}$$

and

$$\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u}, \mathbf{x}) \frac{\partial \mathbf{z}}{\partial \mathbf{d}}(\mathbf{u}, \mathbf{d}) - \frac{\partial \mathbf{z}}{\partial \mathbf{d}}(\mathbf{u}, \mathbf{d}) - \mathbb{I}_{n_x} = \mathbf{0}.$$

Note that we have assumed $\mathbf{d} = \boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x}$ so that $\mathbf{z}(\mathbf{u}, \mathbf{d}) = \mathbf{x}$. The two relations above are equivalent to

$$\frac{\partial \mathbf{z}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{d}) = -\left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u}, \mathbf{x}) - \mathbb{I}_{n_x}\right)^{-1} \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{x})$$

and

$$\frac{\partial \mathbf{z}}{\partial \mathbf{d}}(\mathbf{u}, \mathbf{d}) = \left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{x}}(\mathbf{u}, \mathbf{x}) - \mathbb{I}_{n_x}\right)^{-1}.$$

Therefore, the derivatives $\mathbf{a}$ and $\mathbf{A}$ can efficiently be computed as directional derivatives of $\mathbf{z}$,

$$\mathbf{a} = -\frac{\partial \mathbf{z}}{\partial \mathbf{d}}(\mathbf{u}, \mathbf{d})\ \mathbf{d} \quad \text{and} \quad \mathbf{A} = \frac{\partial \mathbf{z}}{\partial \mathbf{u}}(\mathbf{u}, \mathbf{d}),$$

by differentiation of Algorithm 4.3. As a by-product, the vector $\mathbf{b}$ and the matrix $\mathbf{B}$ are obtained as the derivatives of the last output $\mathbf{f}$ of Algorithm 4.3.
Summarizing, we propose Algorithm 4.4 on the facing page to perform the computations within the lifted Newton method.

4.1.2 Simple practical implementation

The described Algorithms 4.1 to 4.3 used in the lifted Newton method can be obtained with minimal modification of the original function evaluation by adding calls to a "node" function. For this, we assume that the original function evaluation is given in the abstract form of Algorithm 4.5 on page 84, where the $\mathbf{w_i}, i = 1, \ldots, m$ now denote the intermediate values that should be used for lifting. It is then sufficient, as illustrated in Algorithm 4.6 on page 84, to add after each computation of an intermediate value $\mathbf{w_i}$ a call to the node function defined in Algorithm 4.7 on page 85. This modified function then evaluates the different Algorithms 4.1 to 4.3, depending on the value of the global flag `mode`, which has to be set appropriately by the calling function. The global variables $\mathbf{x}$, $\mathbf{z}$ and $\mathbf{d}$ serve as input/output values, depending on the chosen algorithm.

Algorithm 4.4: The Lifted Newton Method

Input : $\mathbf{u}^{(0)}$, `tol`, `manNodeInit`, $\mathbf{x}_{\text{user}}^{(0)}$
Output: $\mathbf{u}^*, \mathbf{x}^*, \mathbf{d}^*, \mathbf{f}^*$
begin
  Set $k = 0$;
  if `manNodeInit` == false **then**
    `// Initialize node values` $\mathbf{x}^{(0)}$ `by function evaluation`
    Set $\mathbf{d}^{(0)} = \mathbf{0}$;
    Call Algorithm 4.3 with inputs $\mathbf{u}^{(0)}, \mathbf{d}^{(0)}$ and set
    $\mathbf{x}^{(0)} = \mathbf{z}(\mathbf{u}^{(0)}, \mathbf{d}^{(0)})$;
    $\mathbf{f}^{(0)} = \boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}^{(0)}, \mathbf{z}(\mathbf{u}^{(0)}, \mathbf{d}^{(0)}))$;
  else
    `// Initialize node values` $\mathbf{x}^{(0)}$ `manually`
    Set $\mathbf{x}^{(0)} = \mathbf{x}_{\text{user}}^{(0)}$;
    Call Algorithm 4.2 with inputs $\mathbf{u}^{(0)}, \mathbf{x}^{(0)}$ to obtain $\mathbf{d}^{(0)}, \mathbf{f}^{(0)}$;
  end
  while $\|\mathbf{f}^{(k)}\| + \|\mathbf{d}^{(k)}\| \geq$ `tol` **do**
    Differentiate Algorithm 4.3 directionally at $(\mathbf{u}^{(k)}, \mathbf{d}^{(k)})$ to obtain
    $\mathbf{a}^{(k)} = -\frac{\partial \mathbf{z}}{\partial \mathbf{d}}(\mathbf{u}^{(k)}, \mathbf{d}^{(k)})\mathbf{d}$;
    $\mathbf{A}^{(k)} = \frac{\partial \mathbf{z}}{\partial \mathbf{u}}(\mathbf{u}^{(k)}, \mathbf{d}^{(k)})$;
    $\mathbf{b}^{(k)} = \mathbf{f}^{(k)} - \frac{\mathrm{d}\boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}, \mathbf{z}(\mathbf{u}, \mathbf{d}))}{\mathrm{d}\mathbf{d}}\mathbf{d}$;
    $\mathbf{B}^{(k)} = \frac{\mathrm{d}\boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}, \mathbf{z}(\mathbf{u}, \mathbf{d}))}{\mathrm{d}\mathbf{u}}$;
    Solve the condensed Newton system to obtain
    $\boldsymbol{\Delta}\mathbf{u}^{(k)} = -(\mathbf{B}^{(k)})^{-1}\mathbf{b}^{(k)}$;
    Perform the Newton step
    $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{a}^{(k)} + \mathbf{A}^{(k)}\boldsymbol{\Delta}\mathbf{u}^{(k)}$;
    $\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \boldsymbol{\Delta}\mathbf{u}^{(k)}$;
    Call Algorithm 4.2 with inputs $(\mathbf{u}^{(k+1)}, \mathbf{x}^{(k+1)})$ to obtain
    $\mathbf{d}^{(k+1)} = \boldsymbol{\phi}(\mathbf{u}^{(k+1)}, \mathbf{x}^{(k+1)}) - \mathbf{x}^{(k+1)}$;
    $\mathbf{f}^{(k+1)} = \boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}^{(k+1)}, \mathbf{x}^{(k+1)})$;
    Set $k = k + 1$;
  end
  Set
  $\mathbf{u}^* = \mathbf{u}^{(k)}$;
  $\mathbf{x}^* = \mathbf{x}^{(k)}$;
  $\mathbf{d}^* = \mathbf{d}^{(k)}$;
  $\mathbf{f}^* = \mathbf{f}^{(k)}$;
end

Algorithm 4.5: Original function evaluation

Input : $\mathbf{u} \in \mathbb{R}^{n_u}$
Output: $\mathbf{f} \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\mathbf{w_i} = \boldsymbol{\phi}_\mathbf{i}(\mathbf{u}, \mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_{i-1}})$;
  end for
  $\mathbf{f} := \boldsymbol{\phi}_\mathbf{f}(\mathbf{u}, \mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_m})$;
end

Algorithm 4.6: Function evaluation modified for use in lifted algorithms

Input : $\mathbf{u} \in \mathbb{R}^{n_u}$
Output: $\mathbf{f} \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\mathbf{w_i} = \boldsymbol{\phi}_\mathbf{i}(\mathbf{u}, \mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_{i-1}})$;
    Call node($\mathbf{w_i}$);
  end for
  $\mathbf{f} := \boldsymbol{\phi}_\mathbf{f}(\mathbf{u}, \mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_m})$;
end

Algorithm 4.7: The node function: node ($\mathbf{v}$)

```
Global Variables: x_1, ..., x_m, z_1, ..., z_m, d_1, ..., d_m, i, mode
Input / Output : v
begin
    switch mode do
        case mode == "original"
        end
        case mode == "init"
            // (see Algorithm 4.1);
            x_i := v;
            i := i + 1;
        end
        case mode == "residual"
            // (see Algorithm 4.2);
            d_i := v − x_i;
            v := x_i;
            i := i + 1;
        end
        case mode == "reduced"
            // (see Algorithm 4.3);
            z_i := v − d_i;
            v := z_i;
            i := i + 1;
        end
    end
end
```

4.2 Application to Optimization

The idea of lifting can also be used in the context of optimization. Let us see how we can use the new trick to efficiently generate the quantities needed for "lifted" Gauss-Newton and Sequential Quadratic Programming (SQP) methods. The new methods shall need to solve only subproblems in the original degrees of freedom to determine a step, while the iterations will be operating in the whole variable space. We also need to show that the iterations made by the proposed lifting approach and by full-space methods are identical.

4.2.1 A lifted Gauss-Newton method

Using the idea of lifting from above, we can develop a lifted Gauss-Newton method to solve least-square type problems with (in)equality constraints. We consider the following nonlinear least-squares optimization problem

$$\min_{\mathbf{u}} \quad \tfrac{1}{2}\|\mathbf{r}(\mathbf{u})\|_2^2 \tag{4.13a}$$

$$\text{s.t.} \quad \mathbf{h}(\mathbf{u}) \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} \mathbf{0}, \tag{4.13b}$$

where $\mathbf{r} : \mathbb{R}^{n_u} \to \mathbb{R}^{n_{\text{res}}}$ is a nonlinear vector valued function describing the residual vector, and the function $\mathbf{h} : \mathbb{R}^{n_u} \to \mathbb{R}^{n_c}$ represents nonlinear equality and inequality constraints.
The Gauss-Newton approach then uses, as mentioned in Section 3.3.3, the Jacobian $\mathbf{J_r}(\mathbf{u}) := \frac{\partial \mathbf{r}}{\partial \mathbf{u}}(\mathbf{u})$ to compute an approximation $\mathbf{J_r}(\mathbf{u})^T\mathbf{J_r}(\mathbf{u})$ of the Hessian of the Lagrangian of the system. The increments $\mathbf{\Delta u}^{(k)}$ are in the constrained Gauss-Newton method computed via the solution of the subproblem

$$\min_{\mathbf{\Delta u}} \quad \tfrac{1}{2}\|\mathbf{r}(\mathbf{u}^{(k)}) + \mathbf{J_r}(\mathbf{u}^{(k)})\mathbf{\Delta u}\|_2^2 \tag{4.14a}$$

$$\text{s.t.} \quad \mathbf{h}(\mathbf{u}^{(k)}) + \mathbf{J_h}(\mathbf{u}^{(k)})\mathbf{\Delta u} \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} \mathbf{0}. \tag{4.14b}$$

The Gauss-Newton approach can also be lifted, which offers impressive advantages in convergence speed, as has been demonstrated by Bock [Boc87], Schlöder [Sch88] and Kallrath, et al. [KBS93]. The augmented problem after the introduction of intermediate values $\mathbf{x_i}, i = 1, \ldots, m$ reads then in the notation from Section 4.1

$$\min_{\mathbf{u},\mathbf{x}} \quad \tfrac{1}{2}\|\boldsymbol{\phi}_\mathbf{r}(\mathbf{u}, \mathbf{x})\|_2^2 \tag{4.15a}$$

$$\text{s.t.} \quad \boldsymbol{\phi}_\mathbf{h}(\mathbf{u}, \mathbf{x}) \left\{\begin{matrix}=\\ \geq\end{matrix}\right\} \mathbf{0} \tag{4.15b}$$

$$\boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x} = \mathbf{0}. \tag{4.15c}$$

This results in the following augmented Quadratic Program (QP) for the step determination:

$$\min_{\mathbf{\Delta u},\mathbf{\Delta x}} \quad \frac{1}{2}\left\|\boldsymbol{\phi}_{\mathbf{r}}(\mathbf{u}^{(k)},\mathbf{x}^{(k)}) + \frac{\partial \boldsymbol{\phi}_{\mathbf{r}}}{\partial(\mathbf{u},\mathbf{x})}(\mathbf{u}^{(k)},\mathbf{x}^{(k)})\begin{pmatrix}\mathbf{\Delta u}\\ \mathbf{\Delta x}\end{pmatrix}\right\|_2^2 \tag{4.16a}$$

s.t.

$$\boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}^{(k)}) + \frac{\partial \boldsymbol{\phi}_{\mathbf{h}}}{\partial(\mathbf{u},\mathbf{x})}(\mathbf{u}^{(k)},\mathbf{x}^{(k)})\begin{pmatrix}\mathbf{\Delta u}\\ \mathbf{\Delta x}\end{pmatrix} \quad \begin{Bmatrix}=\\ \geq\end{Bmatrix} \quad \mathbf{0} \tag{4.16b}$$

$$\boldsymbol{\phi}(\mathbf{u}^{(k)},\mathbf{x}^{(k)}) - \mathbf{x}^{(k)} + \frac{\partial \boldsymbol{\phi}}{\partial(\mathbf{u},\mathbf{x})}(\mathbf{u}^{(k)},\mathbf{x}^{(k)})\begin{pmatrix}\mathbf{\Delta u}\\ \mathbf{\Delta x}\end{pmatrix} - \mathbf{\Delta x} \quad = \quad \mathbf{0}. \tag{4.16c}$$

While at first sight this transformation seems to be disadvantageous due to the increased size of the QP and the need to compute the derivatives of the functions also with respect to the intermediate values $\mathbf{x}$, the problem can be set up and solved at roughly the cost of one Gauss-Newton iteration of the original problem, which is formulated only with variables $\mathbf{u}$ as degrees of freedom. This was discovered by Schlöder [Sch88] in the context of multiple shooting for parameter estimation and extended to direct multiple shooting for optimal control by Schäfer [Sch05].
In order to compute the iterates efficiently we propose here to simply lift the evaluation of $\mathbf{f}(\mathbf{u}) := (\mathbf{r}(\mathbf{u})^T, \mathbf{h}(\mathbf{u})^T)^T$ and use Algorithm 4.4 to compute directly the condensed quantities $\mathbf{a}$, $\mathbf{A}$, $\mathbf{b} = (\mathbf{b_1}^T, \mathbf{b_2}^T)^T$ and $\mathbf{B} = (\mathbf{B_1}^T, \mathbf{B_2}^T)^T$ needed for the condensed QP and the following step expansion. The condensed QP is of the form

$$\min_{\mathbf{\Delta u}} \quad \tfrac{1}{2}\|\mathbf{b_1} + \mathbf{B_1}\mathbf{\Delta u}\|_2^2 \tag{4.17a}$$

s.t.

$$\mathbf{b_2} + \mathbf{B_2}\mathbf{\Delta u} \begin{Bmatrix}=\\ \geq\end{Bmatrix} \mathbf{0}. \tag{4.17b}$$

It is then solved using the same least-squares QP solver as in the non-lifted Gauss-Newton method to generate a solution $\mathbf{\Delta u}_k$. This is then expanded using the relation $\mathbf{\Delta x}^{(k)} = \mathbf{a} + \mathbf{A}\mathbf{\Delta u}^{(k)}$. This procedure obviously delivers the same result as solving (4.15). Numerical results for our new way to implement this well-known approach are given in Section 8.1 and Section 8.3.

4.2.2 Nonlinear optimization via the lifted Newton method

We can also derive a partially-reduced Sequential Quadratic Programming (SQP) method for nonlinear optimization that is based on the lifting idea. By "partially-reduced" we understand in this context that the constraints resulting from the introduction of intermediate variables and the intermediate variables themselves are eliminated from the subproblems, while the constraints of the non-lifted nonlinear problem remain. Fully reduced methods also eliminate these constraints, which leads to some inconveniences in the treatment of inequality constraints. For a discussion of reduced SQP method see, e.g., [Sch96]. In the following we first show equivalence of the lifted SQP iterations with a classical full-space SQP approach in the simpler unconstrained case, and afterwards we treat the general constrained case.

Full-space exact-Hessian SQP iterations

Let us consider the solution of an unconstrained Nonlinear Program (NLP). Assume that we want to minimize a scalar function $c(\mathbf{u}), c : \mathbb{R}^{n_u} \to \mathbb{R}$ and we have lifted the evaluation of c by introducing additional variables $\mathbf{w_i}, i = 1 \ldots, m$. This results in the augmented optimization problem (with the notation from above):

$$\min_{\mathbf{u},\mathbf{w}} \quad \phi_c(\mathbf{u}, \mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_m}) \tag{4.18a}$$

$$\text{s.t.} \quad \mathbf{g}(\mathbf{u}, \mathbf{w}) = \begin{pmatrix} \boldsymbol{\phi_1}(\mathbf{u}) & - & \mathbf{w_1} \\ \boldsymbol{\phi_2}(\mathbf{u}, \mathbf{w_1}) & - & \mathbf{w_2} \\ \vdots & & \\ \boldsymbol{\phi_m}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{m-1}}) & - & \mathbf{w_m} \end{pmatrix} = \mathbf{0}, \tag{4.18b}$$

with the corresponding KKT system for the first order necessary optimality conditions

$$\nabla_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}) = \nabla_{\mathbf{u}} \phi_c(\mathbf{u}, \mathbf{w}) + \nabla_{\mathbf{u}} \mathbf{g}(\mathbf{u}, \mathbf{w}) \boldsymbol{\lambda} = \mathbf{0} \tag{4.19a}$$

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}) = \nabla_{\mathbf{w}} \phi_c(\mathbf{u}, \mathbf{w}) + \nabla_{\mathbf{w}} \mathbf{g}(\mathbf{u}, \mathbf{w}) \boldsymbol{\lambda} = \mathbf{0} \tag{4.19b}$$

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}) = \mathbf{g}(\mathbf{u}, \mathbf{w}) = \mathbf{0}, \tag{4.19c}$$

using the notation $\nabla_{\mathbf{u}} f(\mathbf{u}) = \frac{\partial f}{\partial \mathbf{u}}(\mathbf{u})^T$. The variables $\boldsymbol{\lambda}$ are the Lagrange multipliers for the equality constraints concerning the intermediate values $\mathbf{w}$ and $\mathcal{L}(\mathbf{u}, \mathbf{w})$ is the Lagrange function of the augmented optimization system. The standard full-space exact-Hessian approach then employs a standard Newton method to solve this root finding problem, iterating in the full variable space of $(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda})$.

How to compute the full-space iterations efficiently

To use the lifting approach efficiently we start by evaluating the gradient ∇c of the original objective using the principles of the adjoint mode of automatic differentiation that we described earlier in Section 2.4.2 on page 32. This leads to the following evaluation sequence of the function

$\mathbf{f} : \mathbf{u} \mapsto \bar{\mathbf{u}} \equiv \nabla_{\mathbf{u}} c$ with intermediate values $\mathbf{w}$ and $\bar{\mathbf{w}}$:

$$\begin{aligned}
\mathbf{w_1} &= \boldsymbol{\phi}_{\mathbf{1}}(\mathbf{u}) && (4.20a)\\
\mathbf{w_2} &= \boldsymbol{\phi}_{\mathbf{2}}(\mathbf{u}, \mathbf{w_1}) && (4.20b)\\
&\vdots \\
\mathbf{w_m} &= \boldsymbol{\phi}_{\mathbf{m}}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{m-1}}) && (4.20c)\\
y &\equiv \phi_c(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m}) && (4.20d)\\
\bar{\mathbf{w}}_{\mathbf{m}} &= \nabla_{\mathbf{w_m}} \phi_c && (4.20e)\\
\bar{\mathbf{w}}_{\mathbf{m-1}} &= \nabla_{\mathbf{w_{m-1}}} \phi_c + \nabla_{\mathbf{w_{m-1}}} \boldsymbol{\phi}_{\mathbf{m}} \bar{\mathbf{w}}_{\mathbf{m}} && (4.20f)\\
&\vdots \\
\bar{\mathbf{w}}_{\mathbf{1}} &= \nabla_{\mathbf{w_1}} \phi_c + \sum_{i=2}^{m} \nabla_{\mathbf{w_1}} \boldsymbol{\phi}_{\mathbf{i}} \bar{\mathbf{w}}_{\mathbf{i}} && (4.20g)\\
\bar{\mathbf{u}} &= \nabla_{\mathbf{u}} \phi_c + \sum_{i=1}^{m} \nabla_{\mathbf{u}} \boldsymbol{\phi}_{\mathbf{i}} \bar{\mathbf{w}}_{\mathbf{i}}. && (4.20h)
\end{aligned}$$

We now lift all intermediate variables $\mathbf{x} := (\mathbf{w_1}, \ldots, \mathbf{w_m}, \bar{\mathbf{w}}_{\mathbf{m}}, \ldots, \bar{\mathbf{w}}_{\mathbf{1}})$ in the gradient evaluation procedure $\mathbf{f}(\mathbf{u})$, i.e., we interpret (4.20a)-(4.20g) to be $\boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x} = \mathbf{0}$, as before, and (4.20h) as $\bar{\mathbf{u}} = \boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}, \mathbf{x})$. Doing this, we can show that the lifted Newton iterations towards the solution of

$$\begin{aligned}
\boldsymbol{\phi}_{\mathbf{f}}(\mathbf{u}, \mathbf{x}) &= \mathbf{0} && (4.21a)\\
\boldsymbol{\phi}(\mathbf{u}, \mathbf{x}) - \mathbf{x} &= \mathbf{0} && (4.21b)
\end{aligned}$$

and the iterations of a full-space exact-Hessian SQP method to solve system (4.19) in variables $\mathbf{u}$, $\mathbf{w}$ and $\boldsymbol{\lambda}$ are identical. To see this first observe that (4.20a) - (4.20c) is equivalent to (4.19c), (4.20e) - (4.20g) to (4.19b) if we set $\boldsymbol{\lambda} \equiv \bar{\mathbf{w}}$ and that (4.20h) with $\bar{\mathbf{u}} = \mathbf{0}$ is equivalent to (4.19a). As a result, we obtain the following theorem.

Theorem 4.1
The lifted Newton iterations in variables $(\mathbf{u}, \mathbf{x})$ *applied to the lifted equivalent of the function* $\mathbf{f}(\mathbf{u}) := \nabla_{\mathbf{u}} c(\mathbf{u})$ *are identical to the exact-Hessian full-space SQP iterates in variables* $(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda})$.

4.2.3 A lifted SQP method

Let us now consider the constrained NLP

$$\begin{aligned}
\min_{\mathbf{u}} \quad & c(\mathbf{u}) && (4.22a)\\
\text{s.t.} \quad & \\
& \mathbf{h}(\mathbf{u}) \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \mathbf{0}, && (4.22b)
\end{aligned}$$

where $c : \mathbb{R}^{n_u} \to \mathbb{R}$ is a nonlinear cost function and $\mathbf{h} : \mathbb{R}^{n_u} \to \mathbb{R}^{n_{\text{constr}}}$ represents again nonlinear equality and inequality constraints. The unlifted full step exact-Hessian SQP of Section 3.3.2 then computes the increments by solving the quadratic problem

$$\begin{aligned} \min_{\mathbf{\Delta u}} \quad & \tfrac{1}{2}\mathbf{\Delta u}^T \mathbf{B}^{(k)} \mathbf{\Delta u} + \nabla_{\mathbf{u}} c(\mathbf{u}^{(k)})^T \mathbf{\Delta u} && \text{(4.23a)} \\ \text{s.t.} \quad & \\ & \mathbf{h}(\mathbf{u}^{(k)}) + \nabla_{\mathbf{u}} \mathbf{h}(\mathbf{u}^{(k)})^T \mathbf{\Delta u} \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \mathbf{0}, && \text{(4.23b)} \end{aligned}$$

where $\mathbf{B}^{(k)} = \nabla^2_{\mathbf{uu}} \mathcal{L}(\mathbf{u}^{(k)}, \boldsymbol{\mu}^{(k)})$. The iteration uses the QP solution $\mathbf{\Delta u}_k$ as step in the primal variables, and the corresponding QP multipliers as new multiplier guess $\boldsymbol{\mu}^{(k+1)}$. If we now introduce intermediate variables $\mathbf{w_i}, i = 1, \ldots, m$ we obtain the augmented optimization problem of the form

$$\begin{aligned} \min_{\mathbf{u}} \quad & \phi_c(\mathbf{u}, \mathbf{w}) && \text{(4.24a)} \\ \text{s.t.} \quad & \\ & \boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}, \mathbf{w}) \quad \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \quad \mathbf{0} && \text{(4.24b)} \\ & \mathbf{g}(\mathbf{u}, \mathbf{w}) \quad = \quad \mathbf{0}, && \text{(4.24c)} \end{aligned}$$

with the Lagrangian $\mathcal{L}(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \phi_c(\mathbf{u}, \mathbf{w}) + \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{u}, \mathbf{w}) + \boldsymbol{\mu}^T \boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}, \mathbf{w})$. The full-space QP subproblem in case of an exact-Hessian SQP method then reads

$$\begin{aligned} \min_{\mathbf{\Delta u}, \mathbf{\Delta w}} \quad & \frac{1}{2} \begin{pmatrix} \mathbf{\Delta u} \\ \mathbf{\Delta w} \end{pmatrix}^T \nabla^2 \mathcal{L}(\cdot) \begin{pmatrix} \mathbf{\Delta u} \\ \mathbf{\Delta w} \end{pmatrix} + \nabla \phi_c(\mathbf{u}^{(k)}, \mathbf{w}^{(k)})^T \begin{pmatrix} \mathbf{\Delta u} \\ \mathbf{\Delta w} \end{pmatrix} && \text{(4.25a)} \\ \text{s.t.} \quad & \\ & \boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}^{(k)}, \mathbf{w}^{(k)}) + \nabla \boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}^{(k)}, \mathbf{w}^{(k)})^T \begin{pmatrix} \mathbf{\Delta u} \\ \mathbf{\Delta w} \end{pmatrix} \quad \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \quad \mathbf{0} && \text{(4.25b)} \\ & \mathbf{g}(\mathbf{u}^{(k)}, \mathbf{w}^{(k)}) + \nabla \mathbf{g}(\mathbf{u}^{(k)}, \mathbf{w}^{(k)})^T \begin{pmatrix} \mathbf{\Delta u} \\ \mathbf{\Delta w} \end{pmatrix} \quad = \quad \mathbf{0}. && \text{(4.25c)} \end{aligned}$$

Again, we can use the lifting algorithm to compute the iterates more efficiently than by solving the full-space QP. By a straightforward application of the lifting idea to the combined function evaluation $\mathbf{f}(\mathbf{u}, \boldsymbol{\mu}) := \nabla \mathcal{L}(\mathbf{u}, \boldsymbol{\mu}) \equiv \begin{pmatrix} \nabla_{\mathbf{u}} c(\mathbf{u}) + \nabla_{\mathbf{u}} \mathbf{h}(\mathbf{u}) \boldsymbol{\mu} \\ \mathbf{h}(\mathbf{u}) \end{pmatrix}$ we obtain the required condensed quantities $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{B_1}$ and $\mathbf{B_2}$ for the condensed QP

$$\begin{aligned} \min_{\mathbf{\Delta u}} \quad & \tfrac{1}{2}\mathbf{\Delta u}^T \mathbf{B_1} \mathbf{\Delta u} + \mathbf{b_1}^T \mathbf{\Delta u} && \text{(4.26a)} \\ \text{s.t.} \quad & \\ & \mathbf{b_2} + \mathbf{B_2} \mathbf{\Delta u} \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \mathbf{0}, && \text{(4.26b)} \end{aligned}$$

which can afterwards be expanded using the multiplier of the QP subproblem solution to a step $\boldsymbol{\Delta\mu}$ and using $\mathbf{a}$ and $\mathbf{A}$ finally to steps $\boldsymbol{\Delta w}$ and $\boldsymbol{\Delta\lambda}$. Note that it is sufficient in this case to build the condensed quantities only for the degrees of freedom u to obtain valid $\mathbf{b_1}, \mathbf{b_2}$ and $\mathbf{B_1}, \mathbf{B_2}$ for the condensed QP. After the QP is solved and $\boldsymbol{\Delta\mu}$ is computed, one has to compute one additional directional derivative to expand the step to $\boldsymbol{\Delta w}$ and $\boldsymbol{\Delta\lambda}$. Again, equivalence of iterations generated by the lifted algorithm and the full-space SQP iterations can be shown, provided that the lifting is done as described in the unconstrained case. The derivation of this equivalence and the details of the step expansion procedure for the constrained case are described in the following section.
Note that all directional derivatives of both the lifted constraints and the lifted Lagrage gradient needed in this context can be computed efficiently by the means of Automatic Differentiation (AD) and the forward/adjoint Taylor Coefficient (TC) propagation presented in Chapter 2. For that, the implementation of the Algorithms 4.7 and 4.4 can be modified for the use in connection with an AD tool like ADOL-C such that only the combined evaluation of the Lagrange function and constraints, including calls to the node function, has to be supplied.

4.2.4 Equivalence of lifted SQP and full-space iterations in the constrained case

We consider here the equality constrained nonlinear optimization problem

$$\min_{\mathbf{u}} c(\mathbf{u}) \tag{4.27a}$$

$$\text{subject to}$$

$$\mathbf{h}(\mathbf{u}) = \mathbf{0}, \tag{4.27b}$$

with $c \in \mathcal{C}^2(\mathbb{R}^{n_u}, \mathbb{R})$, $\mathbf{h} \in \mathcal{C}^2(\mathbb{R}^{n_u}, \mathbb{R}^{n_{\text{eq}}})$, to show the equivalence and to explain the computation of the reduces quantities. The results follow directly for the case of inequality constraints from the equivalence of Newton's method for the KKT conditions and the exact-Hessian SQP presented in Section 3.3.2.

Augmented problem

Introducing additionally intermediate values $\mathbf{w}$ occurring during evaluation of c and $\mathbf{h}$ as variables, we obtain the equivalent augmented full-space problem

$$\min_{\mathbf{u},\mathbf{w}} \phi_c(\mathbf{u}, \mathbf{w}) \tag{4.28a}$$

$$\text{subject to}$$

$$\mathbf{g}(\mathbf{u}, \mathbf{w}) = \mathbf{0}, \tag{4.28b}$$

$$\boldsymbol{\phi}_{\mathbf{h}}(\mathbf{u}, \mathbf{w}) = \mathbf{0}, \tag{4.28c}$$

with $\phi_c \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R})$, $\mathbf{g} \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R}^{n_w})$, $\boldsymbol{\phi}_{\mathbf{h}} \in \mathcal{C}^2(\mathbb{R}^{n_u} \times \mathbb{R}^{n_w}, \mathbb{R}^{n_{\text{eq}}})$.

KKT-Conditions for the full-space problem

We define the Lagrangian of the augmented problem

$$\mathcal{L}(\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := \phi_c(\mathbf{u}, \mathbf{w}) + \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{u}, \mathbf{w}) + \boldsymbol{\mu}^T \boldsymbol{\phi}_\mathbf{h}(\mathbf{u}, \mathbf{w}).$$

The necessary first order optimality conditions are then given by

$$\nabla \mathcal{L}(\mathbf{u}^*, \mathbf{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) := \begin{pmatrix} \nabla_\mathbf{u}\phi_c(\mathbf{v}^*) + \nabla_\mathbf{u}\mathbf{g}(\mathbf{v}^*)\boldsymbol{\lambda}^* + \nabla_\mathbf{u}\boldsymbol{\phi}_\mathbf{h}(\mathbf{v}^*)\boldsymbol{\mu}^* \\ \nabla_\mathbf{w}\phi_c(\mathbf{v}^*) + \nabla_\mathbf{w}\mathbf{g}(\mathbf{v}^*)\boldsymbol{\lambda}^* + \nabla_\mathbf{w}\boldsymbol{\phi}_\mathbf{h}(\mathbf{v}^*)\boldsymbol{\mu}^* \\ \mathbf{g}(\mathbf{v}^*) \\ \boldsymbol{\phi}_\mathbf{h}(\mathbf{v}^*) \end{pmatrix} \stackrel{!}{=} \mathbf{0}. \tag{4.29}$$

Here and in the following $\nabla_\mathbf{y}\psi \equiv \frac{\partial \psi}{\partial \mathbf{y}}^T$ denotes the transpose of the Jacobian of a function ψ with respect to $\mathbf{y}$, $\nabla^2_{\mathbf{yz}}\psi \equiv \frac{\partial^2 \psi}{\partial \mathbf{y} \partial \mathbf{z}}$ the (here mixed) Hessian and $\mathbf{v} = (\mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$. For improved readability we sometimes omit the arguments of the functions, if no ambiguity is caused.

Full-space iterations

We use Newton's Method to solve the system for the KKT-conditions, which is then an exact-Hessian SQP method. It iterates

$$\begin{pmatrix} \nabla^2_{\mathbf{uu}}\mathcal{L} & \nabla^2_{\mathbf{uw}}\mathcal{L} & \nabla_\mathbf{u}\mathbf{g} & \nabla_\mathbf{u}\boldsymbol{\phi}_\mathbf{h} \\ \nabla^2_{\mathbf{wu}}\mathcal{L} & \nabla^2_{\mathbf{ww}}\mathcal{L} & \nabla_\mathbf{w}\mathbf{g} & \nabla_\mathbf{w}\boldsymbol{\phi}_\mathbf{h} \\ \nabla_\mathbf{u}\mathbf{g}^T & \nabla_\mathbf{w}\mathbf{g}^T & \mathbf{0} & \mathbf{0} \\ \nabla_\mathbf{u}\boldsymbol{\phi}_\mathbf{h}{}^T & \nabla_\mathbf{w}\boldsymbol{\phi}_\mathbf{h}{}^T & \mathbf{0} & \mathbf{0} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{\Delta}\mathbf{u} \\ \boldsymbol{\Delta}\mathbf{w} \\ \boldsymbol{\Delta}\boldsymbol{\lambda} \\ \boldsymbol{\Delta}\boldsymbol{\mu} \end{pmatrix} = - \begin{pmatrix} \nabla_\mathbf{u}\mathcal{L} \\ \nabla_\mathbf{w}\mathcal{L} \\ \mathbf{g} \\ \boldsymbol{\phi}_\mathbf{h} \end{pmatrix}. \tag{4.30}$$

From the assumed structure of $\mathbf{g}$

$$\mathbf{g}(\mathbf{u}, \mathbf{w}) = \begin{pmatrix} \boldsymbol{\phi}_\mathbf{1}(\mathbf{u}) & -\mathbf{w}_\mathbf{1} \\ \boldsymbol{\phi}_\mathbf{2}(\mathbf{u}, \mathbf{w}_\mathbf{1}) & -\mathbf{w}_\mathbf{2} \\ \vdots & \vdots \\ \boldsymbol{\phi}_\mathbf{m}(\mathbf{u}, \mathbf{w}_\mathbf{1}, \ldots, \mathbf{w}_\mathbf{m-1}) & -\mathbf{w}_\mathbf{m} \end{pmatrix} \tag{4.31}$$

we have

$$\nabla_\mathbf{w}\mathbf{g}^T(\mathbf{u}, \mathbf{w}) = \begin{pmatrix} -\mathbb{I}_{n_{w_1}} & \mathbf{0} & \ldots & \mathbf{0} \\ \frac{\partial \boldsymbol{\phi}_\mathbf{2}}{\partial \mathbf{w}_\mathbf{1}} & -\mathbb{I}_{n_{w_2}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \frac{\partial \boldsymbol{\phi}_\mathbf{m}}{\partial \mathbf{w}_\mathbf{1}} & \ldots & \frac{\partial \boldsymbol{\phi}_\mathbf{m}}{\partial \mathbf{w}_\mathbf{m-1}} & -\mathbb{I}_{n_{w_m}} \end{pmatrix} \tag{4.32}$$

and thus $\nabla_\mathbf{w}\mathbf{g}$ and $\nabla_\mathbf{w}\mathbf{g}^T$ are invertible. Therefore we can solve the third equation of (4.30) for $\boldsymbol{\Delta}\mathbf{w}$ and the second equation for $\boldsymbol{\Delta}\boldsymbol{\lambda}$ to obtain

$$\boldsymbol{\Delta}\mathbf{w} = \mathbf{a}_\mathbf{x} + \mathbf{A}^\mathbf{u}_\mathbf{x}\boldsymbol{\Delta}\mathbf{u}, \qquad \text{with } \mathbf{a}_\mathbf{x} := -\nabla_\mathbf{w}\mathbf{g}^{-T}\mathbf{g},\ \mathbf{A}^\mathbf{u}_\mathbf{x} := -\nabla_\mathbf{w}\mathbf{g}^{-T}\nabla_\mathbf{u}\mathbf{g}^T \tag{4.33}$$

and

$$\boldsymbol{\Delta\lambda} = \mathbf{a}_{\boldsymbol{\lambda}} + \mathbf{A}^{\mathbf{u}}_{\boldsymbol{\lambda}}\boldsymbol{\Delta}\mathbf{u} + \mathbf{A}^{\boldsymbol{\mu}}_{\boldsymbol{\lambda}}\boldsymbol{\Delta\mu} \tag{4.34}$$

with

$$\mathbf{a}_{\boldsymbol{\lambda}} := -\nabla_{\mathbf{w}}\mathbf{g}^{-1}\left(\nabla_{\mathbf{w}}\mathcal{L} + \nabla^2_{\mathbf{ww}}\mathcal{L}(-\nabla_{\mathbf{x}}\mathbf{g}^{-T}\mathbf{g})\right) \tag{4.34a}$$

$$\mathbf{A}^{\mathbf{u}}_{\boldsymbol{\lambda}} := -\nabla_{\mathbf{w}}\mathbf{g}^{-1}\left(\nabla^2_{\mathbf{ww}}\mathcal{L}(-\nabla_{\mathbf{w}}\mathbf{g}^{-T}\nabla_{\mathbf{u}}\mathbf{g}^{T}) + \nabla^2_{\mathbf{wu}}\mathcal{L}\right) \tag{4.34b}$$

$$\mathbf{A}^{\boldsymbol{\mu}}_{\boldsymbol{\lambda}} := -\nabla_{\mathbf{w}}\mathbf{g}^{-1}\nabla_{\mathbf{w}}\boldsymbol{\phi}_{\mathbf{h}}. \tag{4.34c}$$

Now we can condense the problem to

$$\begin{aligned}\begin{pmatrix}\nabla^2_{\mathbf{uu}}\mathcal{L} + \nabla^2_{\mathbf{uw}}\mathcal{L}\mathbf{A}^{\mathbf{u}}_{\mathbf{x}} + \nabla_{\mathbf{u}}\mathbf{g}\mathbf{A}^{\mathbf{u}}_{\boldsymbol{\lambda}} & \nabla_{\mathbf{u}}\boldsymbol{\phi}_{\mathbf{h}} + \nabla_{\mathbf{u}}\mathbf{g}\mathbf{A}^{\boldsymbol{\mu}}_{\boldsymbol{\lambda}} \\ \nabla_{\mathbf{u}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T} + \nabla_{\mathbf{w}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T}\mathbf{A}^{\mathbf{u}}_{\mathbf{x}} & \mathbf{0}\end{pmatrix} \cdot \begin{pmatrix}\boldsymbol{\Delta}\mathbf{u} \\ \boldsymbol{\Delta\mu}\end{pmatrix} \\ = -\begin{pmatrix}\nabla_{\mathbf{u}}\mathcal{L} + \nabla^2_{\mathbf{uw}}\mathcal{L}\,\mathbf{a}_{\mathbf{x}} + \nabla_{\mathbf{u}}\mathbf{g}\,\mathbf{a}_{\boldsymbol{\lambda}} \\ \boldsymbol{\phi}_{\mathbf{h}} + \nabla_{\mathbf{w}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T}\mathbf{a}_{\mathbf{x}}\end{pmatrix},\end{aligned} \tag{4.35}$$

solve for $\boldsymbol{\Delta}\mathbf{u}, \boldsymbol{\Delta\mu}$ and afterwards expand to $\boldsymbol{\Delta}\mathbf{w}, \boldsymbol{\Delta\lambda}$ using (4.33) and (4.34) to obtain the complete full-space SQP step. Note that this system is symmetric of the form

$$\begin{pmatrix}\mathbf{B_1} & \mathbf{B_2}{}^{T} \\ \mathbf{B_2} & \mathbf{0}\end{pmatrix} \cdot \begin{pmatrix}\boldsymbol{\Delta}\mathbf{u} \\ \boldsymbol{\Delta\mu}\end{pmatrix} = -\begin{pmatrix}\nabla_{\mathbf{u}}\mathcal{L} + \nabla^2_{\mathbf{uw}}\mathcal{L}\,\mathbf{a}_{\mathbf{x}} + \nabla_{\mathbf{u}}\mathbf{g}\,\mathbf{a}_{\boldsymbol{\lambda}} \\ \boldsymbol{\phi}_{\mathbf{h}} + \nabla_{\mathbf{w}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T}\mathbf{a}_{\mathbf{x}}\end{pmatrix}, \tag{4.36}$$

with symmetric

$$\begin{aligned}\mathbf{B_1} = \nabla^2_{\mathbf{uu}}\mathcal{L} - \nabla^2_{\mathbf{uw}}\mathcal{L}\nabla_{\mathbf{w}}\mathbf{g}^{-T}\nabla_{\mathbf{u}}\mathbf{g}^{T} - \nabla_{\mathbf{u}}\mathbf{g}\nabla_{\mathbf{w}}\mathbf{g}^{-1}\nabla^2_{\mathbf{wu}}\mathcal{L} \\ +\nabla_{\mathbf{u}}\mathbf{g}\nabla_{\mathbf{w}}\mathbf{g}^{-1}\nabla^2_{\mathbf{ww}}\mathcal{L}\nabla_{\mathbf{w}}\mathbf{g}^{-T}\nabla_{\mathbf{u}}\mathbf{g}^{T}\end{aligned} \tag{4.37}$$

and

$$\mathbf{B_2} = \nabla_{\mathbf{u}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T} - \nabla_{\mathbf{w}}\boldsymbol{\phi}_{\mathbf{h}}{}^{T}\nabla_{\mathbf{w}}\mathbf{g}^{-T}\nabla_{\mathbf{u}}\mathbf{g}^{T}. \tag{4.38}$$

Lifted Newton formulation

We assume that the combined evaluation of

$$\begin{pmatrix}\nabla_{\mathbf{u}}\mathcal{L}^{\text{orig}} \\ \mathbf{h}\end{pmatrix} = \begin{pmatrix}\nabla_{\mathbf{u}}c + \nabla_{\mathbf{u}}\mathbf{h}\boldsymbol{\mu} \\ \mathbf{h}\end{pmatrix}, \tag{4.39}$$

is lifted in such a way that the computation of $\mathcal{L}$ and $\mathbf{h}$ use common intermediate values $\mathbf{w}$ which are introduced as nodes. Furthermore, the evaluation of $\nabla_{\mathbf{u}}\mathcal{L}$ is done in the way of the adjoint mode of automatic differentiation, and the corresponding adjoint values $\bar{\mathbf{w}}$ are also introduced as nodes.

The nodes $(\mathbf{w}, \bar{\mathbf{w}})$ are then defined by the equations

$$\boldsymbol{\gamma}(\mathbf{u}, \mathbf{w}, \bar{\mathbf{w}}, \boldsymbol{\mu}) = \begin{pmatrix} \boldsymbol{\gamma_1}(\mathbf{u}, \mathbf{w}) \\ \hline \boldsymbol{\gamma_2}(\mathbf{u}, \mathbf{w}, \bar{\mathbf{w}}, \boldsymbol{\mu}) \end{pmatrix} = \begin{pmatrix} \mathbf{g}(\mathbf{u}, \mathbf{w}) \\ \hline \boldsymbol{\gamma_2}(\mathbf{u}, \mathbf{w}, \bar{\mathbf{w}}, \boldsymbol{\mu}) \end{pmatrix} =$$

$$\left(\begin{array}{cc} \boldsymbol{\phi_1}(\mathbf{u}) & -\mathbf{w_1} \\ \boldsymbol{\phi_2}(\mathbf{u}, \mathbf{w_1}) & -\mathbf{w_2} \\ \vdots & \\ \boldsymbol{\phi_m}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{m-1}}) & -\mathbf{w_m} \\ \hline \frac{\partial \phi_c(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_m}} + \frac{\partial \boldsymbol{\phi_h}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_m}}^T \boldsymbol{\mu} & -\bar{\mathbf{w}}_\mathbf{m} \\ \vdots & \\ \frac{\partial \phi_c(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_2}} + \frac{\partial \boldsymbol{\phi_h}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_2}}^T \boldsymbol{\mu} + \sum_{i=3}^{m} \frac{\partial \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{i-1}})}{\partial \mathbf{w_2}}^T \bar{\mathbf{w}}_\mathbf{i} & -\bar{\mathbf{w}}_\mathbf{2} \\ \frac{\partial \phi_c(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_1}} + \frac{\partial \boldsymbol{\phi_h}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{w_1}}^T \boldsymbol{\mu} + \sum_{i=2}^{m} \frac{\partial \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{i-1}})}{\partial \mathbf{w_1}}^T \bar{\mathbf{w}}_\mathbf{i} & -\bar{\mathbf{w}}_\mathbf{1} \end{array}\right) = 0 \tag{4.40}$$

and for the gradient $\nabla_\mathbf{u}\mathcal{L}$ of the Lagrangian we have

$$\nabla_\mathbf{u}\mathcal{L}(\mathbf{u}, w, \bar{\mathbf{w}}, \boldsymbol{\mu}) = \left(\frac{\partial \phi_c(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{u}} + \frac{\partial \boldsymbol{\phi_h}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_m})}{\partial \mathbf{u}}^T \boldsymbol{\mu} + \sum_{i=1}^{m} \frac{\partial \boldsymbol{\phi_i}(\mathbf{u}, \mathbf{w_1}, \ldots, \mathbf{w_{i-1}})}{\partial \mathbf{u}}^T \bar{\mathbf{w}}_\mathbf{i} \right). \tag{4.41}$$

Following the lifting approach we then apply Newton to solve the root finding problem

$$\begin{pmatrix} \nabla_\mathbf{u}\mathcal{L}(\mathbf{u}, \mathbf{w}, \bar{\mathbf{w}}, \boldsymbol{\mu}) \\ \boldsymbol{\phi_h}(\mathbf{u}, \mathbf{w}) \\ \mathbf{g}(\mathbf{u}, \mathbf{w}) \\ \boldsymbol{\gamma_2}(\mathbf{u}, \mathbf{w}, \bar{\mathbf{w}}, \boldsymbol{\mu}) \end{pmatrix} = 0. \tag{4.42}$$

We compare now this system with system (4.30) from the full-space approach. We observe that, if identifying $\bar{\mathbf{w}}_\mathbf{i}$ with $\boldsymbol{\lambda}_\mathbf{i}$ and exchanging equations 2 and 4 in (4.42), the systems are identical. As a result, the Newton method we apply to these systems in both cases will lead to the same iterations, which shows the equivalence of the lifted SQP approach and full-space exact-Hessian iterations. Note that by construction the lifted approach leads to the same condensed system as the full-space approach.
Furthermore, we observe that for the computation of $\mathbf{B_1}$ and $\mathbf{B_2}$ of the symmetric system (4.36) by using the modified function of the lifting approach only directional derivatives in $\mathbf{u}$ are needed. After the computation of $\boldsymbol{\Delta u}$ and $\boldsymbol{\Delta \mu}$ from the condensed system we first can expand the step to $\boldsymbol{\Delta w}$ via (4.33), as $\mathbf{a_x}$ and $\mathbf{A_x^u}$ have been computed together with $\mathbf{B_1}, \mathbf{B_2}$. Afterwards, we need one additional directional derivative of the modified function to expand the step to $\boldsymbol{\Delta \lambda}$ via (4.34).

4.3 Local Convergence Analysis of Lifted Newton Methods

While a main contribution of this thesis lies in the derivation of easy-to-implement algorithms for lifted Newton methods that each have the same computational complexity per iteration as

the corresponding non-lifted methods, the idea of "lifting" in itself is an old idea. However, its convergence properties are not well understood. For solution of boundary value problems with underlying nonlinear ODE models, the multiple shooting method (which can be regarded a lifted algorithm) is since long known to outperform the single shooting method [Osb69]. Three reasons are often cited for the superiority of the "lifted" compared to the non-lifted Newton approaches:

- more freedom for initialization,
- better conditioned and block-sparse linear systems, and
- faster local convergence.

While the first two reasons are well-understood, no detailed local convergence analysis exists so far that explains this superior local convergence of "lifted" Newton methods. In order to make a first step to approach this question, we regard a model root finding problem $\mathbf{f}(\mathbf{u}) = \mathbf{0}$ where we have a chain of nonlinear functions that each only depend on the output of the immediate predecessor function, and that all have the same input and output dimensions:

$$\mathbf{x_1} = \boldsymbol{\phi_1}(\mathbf{u}),\ \mathbf{x_2} = \boldsymbol{\phi_2}(\mathbf{x_1}), \ldots, \mathbf{x_m} = \boldsymbol{\phi_m}(\mathbf{x_{m-1}}) \text{ and } \boldsymbol{\phi_f}(\mathbf{x_m}) \equiv \mathbf{x_{m+1}}(\mathbf{x_m}).$$

In order to further simplify the following discussion, we will now restrict ourselves to the simplest case, were u and all other variables are scalar. We regard the local convergence rate in the neighborhood of the solution. At this solution $(u^*, \mathbf{x}^*)$, all Jacobians must be invertible. Therefore, by suitable affine variable transformations for $x_0 := u$ and for $x_1, \ldots, x_m$ we can both assume that the solution is zero for all variables, and that all functions ϕ_i are given by

$$\phi_i(x) = x + b_i(x)^2 + O(|x|^3). \tag{4.43}$$

Here the affine transformations to the new variables and functions can be expressed as follows

$$x_i^{\text{new}} = (x_i - x_i^*)/a_i \qquad a_0 := 1, \quad a_i := \prod_{j=1}^{i} \phi_j'(x_{j-1}^*), 1 \le i \le m+1, \quad \text{and}$$

$$\phi_i^{\text{new}}(x_{i-1}^{\text{new}}) = \frac{1}{a_i}\left(\phi_i(x_{i-1}) - x_i^*\right), \quad b_i := \frac{\phi_i''(x_{i-1}^*)}{2\phi_i'(x_{i-1}^*)} \prod_{j=1}^{i-1} \phi_j'(x_{j-1}^*)$$

4.3.1 Local Convergence of the Non-Lifted Newton Method

In the simplified setting outlined above, the non-lifted function $f(u)$ is given by

$$f(u) = \phi_{m+1}(\phi_m(\ldots\phi_1(u)\ldots)) = u + \Big(\sum_{i=1}^{m+1} b_i\Big)u^2 + O(|u|^3)$$

and its derivative is given by

$$f'(u) := \frac{\partial f}{\partial u}(u) = 1 + 2\Big(\sum_{i=1}^{m+1} b_i\Big)u + O(|u|^2).$$

To regard the local convergence behavior near zero, we first note that for any Newton method holds that

$$u^{(k+1)} = u^{(k)} - f'(u^{(k)})^{-1} f(u^{(k)}) = f'(u^{(k)})^{-1}\Big(f'(u^{(k)})u^{(k)} - f(u^{(k)})\Big)$$

and due to the fact that in our case

$$f'(u^{(k)})u^{(k)} = \left(u^{(k)} + 2\Big(\sum_{i=1}^{m+1} b_i\Big)(u^{(k)})^2 + O(|u^{(k)}|^3)\right)$$

this leads to the iteration formula

$$u^{(k+1)} = \Big(\sum_{i=1}^{m+1} b_i\Big)(u^{(k)})^2 + O(|u^{(k)}|^3).$$

Thus, the local contraction constant for quadratic convergence is given by $\Big(\sum_{i=1}^{m+1} b_i\Big)$.

4.3.2 Local Convergence of the Lifted Newton Method

In the simplified setting outlined above, the lifted function $\mathbf{g}(u, \mathbf{x})$ is given by

$$\mathbf{g}(u,\mathbf{x}) = \begin{pmatrix} u + b_1 u^2 & - & x_1 \\ x_1 + b_2 x_1^2 & - & x_2 \\ \vdots & & \\ x_{m-1} + b_m x_{m-1}^2 & - & x_m \\ x_m + b_{m+1} x_m^2 & & \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix} \right\|^3\right) \tag{4.44}$$

and its derivative $\frac{\partial \mathbf{g}(u,\mathbf{x})}{\partial (u,\mathbf{x})}$ is given by

$$\begin{pmatrix} 1+2b_1u & -1 & & & \\ & 1+2b_2x_1 & -1 & & \\ & & \ddots & \ddots & \\ & & 1+2b_mx_{m-1} & -1 & \\ & & & 1+2b_{m+1}x_m & \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix} \right\|^2\right).$$

From this particular form follows first, that

$$\frac{\partial \mathbf{g}(u,\mathbf{x})}{\partial (u,\mathbf{x})} \cdot \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} u + 2b_1 u^2 & - & x_1 \\ x_1 + 2b_2 x_1^2 & - & x_2 \\ \vdots & & \\ x_{m_1} + 2b_m x_{m_1}^2 & - & x_m \\ x_m + 2b_{m+1} x_m^2 & & \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix} \right\|^3\right)$$

and second, that

$$\frac{\partial \mathbf{g}(u,\mathbf{x})}{\partial(u,\mathbf{x})}^{-1} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ & 1 & 1 & \dots & 1 \\ & & \ddots & \ddots & \vdots \\ & & & 1 & 1 \\ & & & & 1 \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix} \right\|\right).$$

Using again the formula for the Newton iteration

$$\begin{pmatrix} u \\ \mathbf{x} \end{pmatrix}^{(k+1)} = \frac{\partial \mathbf{g}(u^{(k)},\mathbf{x}^{(k)})}{\partial(u,\mathbf{x})}^{-1} \left(\frac{\partial \mathbf{g}(u^{(k)},\mathbf{x}^{(k)})}{\partial(u,\mathbf{x})} \cdot \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix}^{(k)} - \mathbf{g}(u^{(k)},\mathbf{x}^{(k)}) \right)$$

we obtain the iteration

$$\begin{pmatrix} u \\ x_1 \\ \vdots \\ x_{m-1} \\ x_m \end{pmatrix}^{(k+1)} = \begin{pmatrix} b_1 \left(u^{(k)}\right)^2 + \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{(k)}\right)^2 \\ \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{(k)}\right)^2 \\ \vdots \\ b_m \left(x_{m-1}^{(k)}\right)^2 + b_{m+1} \left(x_m^{(k)}\right)^2 \\ b_{m+1} \left(x_m^{(k)}\right)^2 \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ \mathbf{x} \end{pmatrix}^{(k)} \right\|^3 \right).$$

It can be seen that, neglecting third order terms, the last component, x_m, converges independently from all others with quadratic contraction constant b_{m+1}. All other components x_i converge based on their own quadratic contraction constant, b_{i+1}, and those of the higher indexed components, and the same holds for u. Thus, x_m is leading the convergence, with x_{m-1} as follower, etc, until u.

4.3.3 Comparison of Lifted and Non-Lifted Newton Method

We have to compare the non-lifted quadratic convergence constant

$$\Big(\sum_{i=1}^{m+1} b_i \Big)$$

with the interdependent chain of quadratically converging sequences x_i in the lifted case, each with its dominant quadratic convergence constant b_{i+1}. Let us assume we start the non-lifted variant close to the solution with $u^{(0)} = \epsilon$, and the lifted variant with the corresponding values resulting from a forward function evaluation, which in our special setting turn out to be $x_i^{(0)} = \epsilon + O(|\epsilon|^2)$. As expected, the first step in u is identical in both methods, and results in

$$u^{(1)} = \Big(\sum_{i=1}^{m+1} b_i \Big) \epsilon^2 + O(|\epsilon|^3).$$

However, in the lifted variant, the values x_i have been contracted according to their own contraction constants, to values

$$x_j^{(1)} = \Big(\sum_{i=j+1}^{m+1} b_i \Big) \epsilon^2 + O(|\epsilon|^3).$$

In the second iteration, the differing values for $\mathbf{x}$ will already lead to different iterates $u^{(2)}$. Which of the two methods converges faster depends on the signs of the b_i.

Same direction of curvature If all b_i have the same sign, i.e., all subfunctions ϕ_i are curved in the same direction, then the contraction constants for all x_i are better than the non-lifted variant, with the last component x_m converging fastest. The improved convergence speed of $\mathbf{x}$ spills over to the convergence of u and therefore makes the lifted Newton method converge faster.
To see the effect at an example, let us regard the simplest setting with only one intermediate function evaluation, i.e., $m = 1$, with constants $b_1 = b_2 = 1$. After four iterations, the value of u in the non-lifted variant is $u^{(4)} = 2^{15}\ \epsilon^{16}$ while in the lifted variant it is $u^{(4)} = 677\ \epsilon^{16}$ which is more than two decimal digits more accurate.

Opposite directions of curvature In the other extreme, let us regard a setting where all b_i add to zero, but are each independently different from zero. Note that this can only occur if the subfunctions ϕ_i have different directions of curvature. In this case, the non-lifted variant converges even faster than quadratically, while the lifted variant has the usual quadratic rate.
To see this at at an even more extreme example, regard the simple chain of two functions $\phi_1(u) = \frac{1}{2}(1+u)^2 - \frac{1}{2}$ and $\phi_2(x_1) = \sqrt{1+2x_1} - 1$. These functions satisfy our assumptions with $b_1 = 1$ and $b_2 = -1$. Moreover, they are constructed such that $f(u) = u$. As f is a linear function, the non-lifted Newton method converges in the first iteration, $u^{(1)} = 0$, while the lifted Newton method performs the same favorable first step in u, but as x_1 is not yet converged, it will continue iterating and changing u until both variables have been converged to sufficient accuracy.

Practical Advice In a practical application, even if we would have a chain of subsequent functions each depending only on the output of its predecessor, we do not know which local curvature constants the typically multi-input-multi-output functions $\boldsymbol{\phi}_\mathbf{i}$ would have relatively to each other, after the affine variable transformation based on linearization at the solution, to make them comparable. However, we might make an educated guess in the following case that occurs, e.g., in the simulation of continuous time dynamic systems: if we have repeated calls of the same function, i.e., $\boldsymbol{\phi}_{\mathbf{i+1}} = \boldsymbol{\phi}_\mathbf{i}$, and the variables $\mathbf{x}_{\mathbf{i+1}}$ and $\mathbf{x}_\mathbf{i}$ differ only slightly then we can expect lifting to have a favorable effect on the required number of Newton iterations, even if both methods are initialized identically. A second case where a lifted approach surely is beneficial is the case where the freedom for initializing the $\mathbf{x}_\mathbf{i}$ based on extra knowledge can be used, e.g., when state measurement data are present in parameter estimation problems [Boc87]. On the other hand, lifting should not be applied to simple linear subfunctions $\boldsymbol{\phi}_\mathbf{i}$, i.e., scalar multiplications and additions/subtractions, as no accelerated convergence can be gained, but memory requirement and operation counts are increased. In all other cases, we do not dare to make predictions, but suggest to experiment with

the lifted and non-lifted Newton methods in specific application examples. It is one aim of this thesis to propose an algorithmic trick and a software package that makes the switching between the two methods as simple as possible. In the past, the implementation of structure-exploiting lifted algorithms was often a tedious task, deterring many users that might have benefitted from the lifted approach. The insight gained in this section can be expressed by the following theorem that characterizes the local convergence speed of lifted and non-lifted Newton methods.

Theorem 4.2 (Local convergence speed of lifted and non-lifted Newton methods) *Let $f_i : \mathbb{R} \to \mathbb{R}$, $1 \le i \le m+1$ be a chain of twice continuously differentiable scalar functions, such that $f(u) = \phi_{m+1}(\phi_m(\ldots(\phi_1(u))\ldots))$, $x_1 = \phi_1(u)$ and $x_i = \phi_i(x_{i-1})$, $2 \le i \le m+1$. Assume that the solution of the problem $f(u) = 0$ is given by u^* and that in the solution the Jacobians of $\phi_i, 1 \le i \le m+1$ are invertible. Define*

$$a_i := \prod_{j=1}^{i} \phi_j'(x_{j-1}^*), \qquad 1 \le i \le m+1 \tag{4.45a}$$

$$b_1 := \frac{\phi_1''(u^*)}{2\phi_1'(u^*)}, \tag{4.45b}$$

$$b_i := \frac{\phi_i''(x_{i-1}^*)}{2\phi_i'(x_{i-1}^*)} \phi_{i-1}'(x_{i-2}^*) \ldots \phi_1'(u^*), \qquad 2 \le i \le m+1, \tag{4.45c}$$

with $x_i^* = \phi_i(\phi_{i-1}(\ldots(\phi_1(u^*))\ldots))$. Then the local convergence speed of an non-lifted Newton method for the solution of $f(u) = 0$ is given by

$$|u^{(k+1)} - u^*| \le \left(\sum_{i=1}^{m+1} b_i \right) |u^{(k)} - u^*|^2 + O(|u^{(k)} - u^*|^3),$$

and the local convergence of the lifted Newton iterates is componentwise staggered, following the estimation

$$\left| \frac{x_i^{(k+1)} - x_i^*}{a_i} \right| \le \sum_{j=i+1}^{m+1} b_j \left| \frac{x_{j-1}^{(k)} - x_{j-1}^*}{a_j} \right|^2 + O\left(\left\| \begin{pmatrix} u^{(k)} - u^* \\ \mathbf{x}^{(k)} - \mathbf{x}^* \end{pmatrix} \right\|^3 \right), \quad 1 \le i \le m$$

$$|u^{[k+1]} - u^*| \le b_1 |u^{(k)} - u^*|^2 + \sum_{j=i+1}^{m+1} b_j \left| \frac{x_{j-1}^{(k)} - x_{j-1}^*}{a_j} \right|^2 + O\left(\left\| \begin{pmatrix} u^{(k)} - u^* \\ \mathbf{x}^{(k)} - \mathbf{x}^* \end{pmatrix} \right\|^3 \right).$$

Cost comparison

To compare the overall numerical effort for the solution of the example we analyze the cost of a single iteration in the lifted, non-lifted and also full-space approach in more detail. We estimate computational effort in terms of floating point operations (flops). We follow the usual convention that an addition, subtraction and multiplication, as well as a combined multiply-add cost 1 flop, while one division takes 4 flops. The cost in flops for the evaluation of the subfunctions $\boldsymbol{\phi}_\mathbf{i}$ are

denoted with $c_{\boldsymbol{\phi}_\mathbf{i}}$, where we set for notational convenience $\boldsymbol{\phi}_{\mathbf{m+1}} := \boldsymbol{\phi}_\mathbf{f}$. The cost to compute their directional derivative $c_{d\boldsymbol{\phi}_\mathbf{i}}$. To the cost to solve a n-dimensional linear equation system we refer with $c_{LS}(n)$, which can in general be estimated with $c_{LS}(n) = \frac{2n^3+3n^2-5n}{6} + 2n(n+1)$ flops. For simplicity we assume one-dimensional node values and omit the cost of memory access. We split the cost for one iteration into the cost for (cf. Algorithm 4.4)

- evaluation of the residuals and the value of $\mathbf{f}$,
- computation of the quantities of the Newton system,
- the solution of the Newton system to compute the step,
- application of the step, which in the lifted case includes the cost for the expansion of the step in the controls $\mathbf{u}$ to the the step in the nodes.

Table 4.1 on the next page shows a comparison of these costs for the three approaches, as well as the resulting overall effort. Note that the cost estimation in the full-space approach is based on the assumption that no further internal structure of the problem is known and exploited than the decomposition of $\mathbf{f}(\mathbf{u})$ into the sequence of mappings ϕ_i. Further exploitation of the internal structure might lead to a higher efficiency of a full-space approach, but already this relative simple exploitation of structure will lead to significantly higher implementation effort compared to the non-lifted and lifted approach. Additionally, to be efficient, it has to be adjusted manually for each new problem (and also each new decomposition of $\mathbf{f}$).

4.4 A tutorial root finding example

To end this chapter, we illustrate the numerical behavior of the lifted Newton method, along with a convergence and cost comparison discussion, using a tutorial root finding problem that is given by

$$f(u) := u^{16} - 2 = 0.$$

We introduce $m = 4$ intermediate values $x_1, \ldots, x_4$ and lift the evaluation of f in the following way:

$$\begin{aligned} x_1 &:= u^2, & x_2 &:= x_1^2, \\ x_3 &:= x_2^2, & x_4 &:= x_3^2, \\ f &:= x_4 - 2. \end{aligned}$$

To solve the problem we employ Algorithm 4.4, as well as a standard full-step Newton iteration applied to the non-lifted problem. The termination criterion is based on the Euclidean norm of the function value, plus, in the lifted case, the Euclidean norm of the actual node residual. We require this sum to be smaller than 10^{-6}.
For the initial value $u^{(0)} = 0.8$ we obtain convergence after 7 iterations in the lifted case and 27 iterations in the non-lifted. The progress of the iterates towards the solution during the iterations is

	non-lifted	lifted
residual evaluation	$\sum_{i=1}^{m+1} c_{\phi_i}$	$\sum_{i=1}^{m+1} c_{\phi_i} + m$
Newton system	$n_u \sum_{i=1}^{m+1} c_{d\phi_i}$	$(n_u+1)(\sum_{i=1}^{m+1} c_{d\phi_i} + 2m) + n_u$
step computation	$c_{LS}(n_u)$	$c_{LS}(n_u)$
step application	n_u	$n_u + m + n_u m$

	full-space
residual evaluation	$\sum_{i=1}^{m+1} c_{\phi_i} + m$
Newton system	$\sum_{i=1}^{m+1} (n_u + i - 1) c_{d\phi_i}$
step computation	$c_{LS}(n_u + m)$
step application	$n_u + m$

	overall effort per iteration
non-lifted	$c_{LS}(n_u) + \sum_{i=1}^{m+1} [c_{\phi_i} + n_u c_{d\phi_i}] + n_u$
lifted	$c_{LS}(n_u) + \sum_{i=1}^{m+1} [c_{\phi_i} + (n_u+1) c_{d\phi_i}] + 3n_u m + 3m + 2n_u$
full-space	$c_{LS}(n_u + m) + \sum_{i=1}^{m+1} [c_{\phi_i} + (n_u + i - 1) c_{d\phi_i}] + 2m + n_u$

Table 4.1: Cost analysis (in flops) for one iteration of an non-lifted, lifted and a full-space Newton method. Described are the costs for the different phases of an iteration (top, middle) and the overall cost for one iteration (bottom). n_u is the number of controls, m the number of nodes and c_{ϕ_i}, $c_{d\phi_i}$ describe the cost of an evaluation and a directional derivative of the subfunction ϕ_i, respectively. The quantity $c_{LS}(n)$ stands for the effort to solve a linear equation system with n unknowns.

depicted in Figure 4.1 on the following page. The first iteration is identical, as it is always the case if the node values $\mathbf{x}$ in the lifted algorithm are initialized by a function evaluation (cf. Algorithm 4.1). In subsequent iterations we observe that the lifted version benefits from the additional degrees of freedom which results in a much faster progress towards the solution $u^* \approx 1.044$.

Convergence analysis

We now analyze the local convergence of the test example. f was decomposed into $\phi_i(x) = x^2$, $1 \le i \le 4$ and $\phi_5(x) = x - 2$, with the solution $(u^*, x_1^*, x_2^*, x_3^*, x_4^*) = (\sqrt[16]{2}, \sqrt[8]{2}, \sqrt[4]{2}, \sqrt{2}, 2)$. We then compute using Theorem 4.2

$$
\begin{aligned}
b_1 &= \frac{1}{2\sqrt[16]{2}} \approx 0.478801, \\
b_2 &= \frac{1}{2\sqrt[8]{2}} 2\sqrt[16]{2} = \frac{1}{\sqrt[16]{2}} \approx 0.957603, \\
b_3 &= \frac{1}{2\sqrt[4]{2}} 4\sqrt[16]{2}\sqrt[8]{2} = \frac{2}{\sqrt[16]{2}} \approx 1.91521, \\
b_4 &= \frac{1}{2\sqrt[2]{2}} 8\sqrt[16]{2}\sqrt[8]{2}\sqrt[4]{2} = \frac{4}{\sqrt[16]{2}} \approx 3.83041, \\
b_5 &= 0.
\end{aligned}
$$

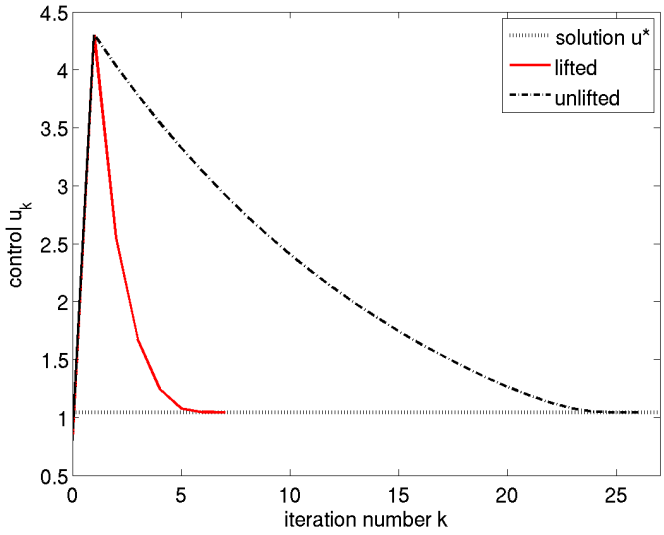

Figure 4.1: Iterates $u^{(k)}$ of lifted and non-lifted approach for the solution of $f(u) := u^{16} - 2 = 0$, and solution value $u^* \approx 1.044$. After an identical first iteration, which is due to the automatic initialization of the intermediate values using a function evaluation, the lifted method makes much faster progress towards the solution.

This leads to a non-lifted local quadratic contraction constant of $b \approx 7.18202$.
Figure 4.2 on the next page shows the error of the iterates and the convergence rates for each iteration during solution with the non-lifted and the lifted Newton method. The convergence rates are determined numerically for each component using the formula

$$\beta_y^{(k)} \approx \frac{\|y^{(k)} - y^*\|}{\|y^{(k+1)} - y^*\|^2}.$$

We observe, besides the faster convergence of the lifted iterations already described, that, as we approach the solution, the predicted convergence rates are reached in both cases. Additionally, it can be seen, as predicted by Theorem 4.2, that in the lifted case the components converge in a staggered way, starting with x_4, which converges due to linearity of ϕ_5 in one step, followed by x_3, x_2, ... until finally u is converged.

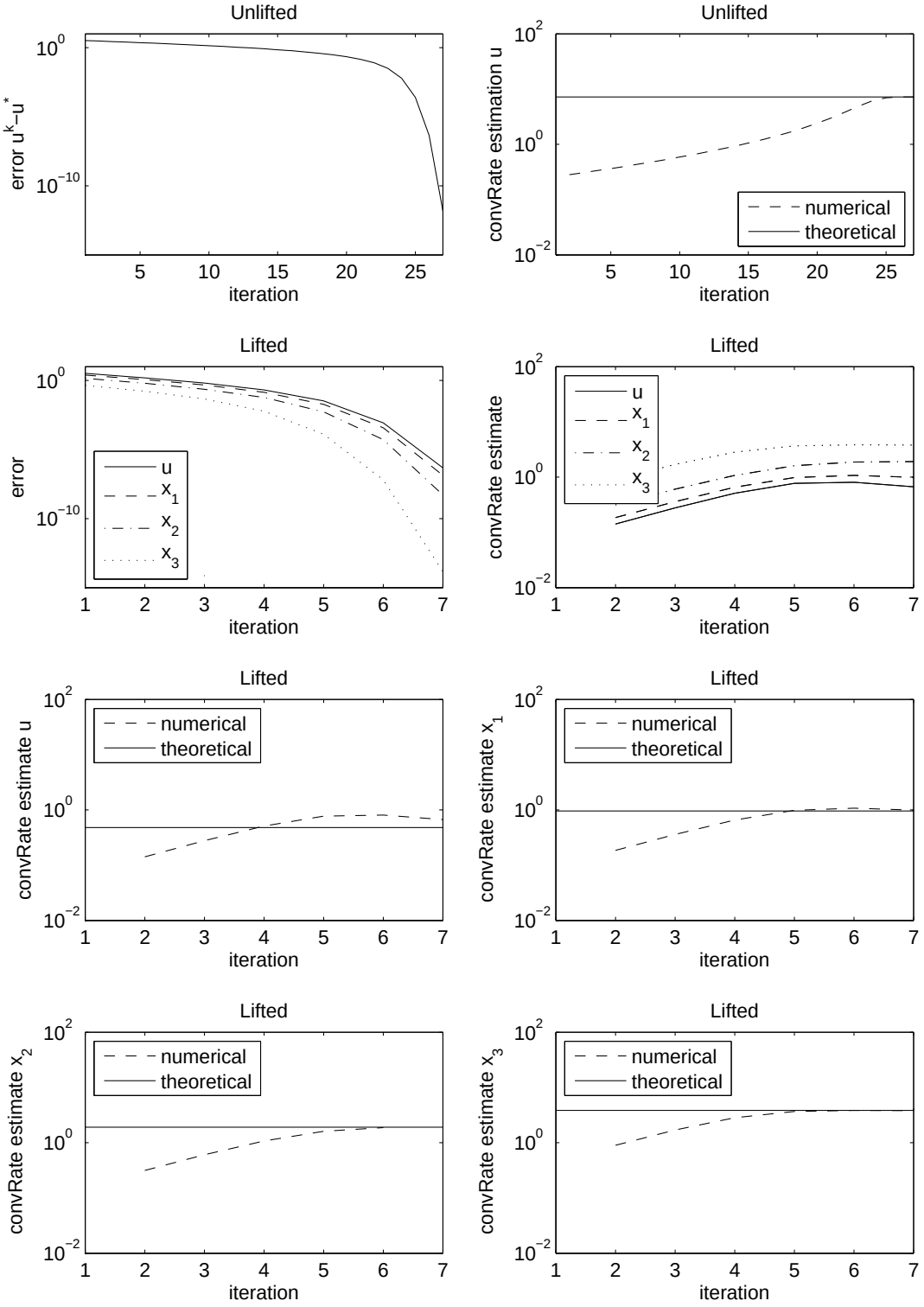

Figure 4.2: Shown is a comparison of the behavior of the iterates of the non-lifted and lifted Newton method on the root finding example $f(u) = u^{16} - 2 = 0$ with start value $u_0 = 0.8$ and tolerance $\texttt{tol} = 10^{-6}$. Depicted is the Euclidean norm of the errors of the iterates (upper left pictures) and the local convergence estimates (upper right pictures), where in the lifted case the quantities are depicted componentwise with exception of x_4, as it converges after the first iteration due to linearity of ϕ_5. Additionally, the lower 4 pictures show for the lifted method for each component (except x_4) the convergence rate estimate as well as its theoretical prediction. We observe that in both the non-lifted and the lifted case the convergence rates finally reach the predicted values. Furthermore the lifted case shows the staggered convergence of the components that is predicted by theorem 4.2.

5 Solution of initial value problems for ODEs and index 1 DAEs

In this chapter we address the task of computing solutions of Initial Value Problems (IVPs) for Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs) reliably and efficiently. We present the needed underlying theory as well as efficient methods for the fast numerical solution of these types of IVPs. The topic of efficient derivative computation for these solutions is addressed in the next chapter.
This chapter is organized as follows. In Section 5.1 we state some well-known facts from DAE theory. In Section 5.2 the theoretical foundation and the basic properties of the numerical methods we use for the solution of the ODE- and DAE-IVPs are presented. Finally, in Section 5.3 we address the specific strategies implemented in our solver `DAESOL-II` to efficiently solve the ODE/DAE-IVPs based on variable order variable stepsize backward differentiation formulas (BDF).

5.1 Basic DAE theory

In this section we shortly present the definitions and theoretical properties of DAE systems we use later during the description of the strategies for the numerical solution. Furthermore, we shortly address existence and uniqueness of the solutions.

5.1.1 Notation and definitions

The most general type of a DAE is the fully implicit DAE, which cannot explicitly be split up into a differential and an algebraic part.

Definition 5.1 (Fully implicit DAE)
A fully implicit *DAE is a system of equations*

$$\mathbf{b}(t, \mathbf{y}(t), \dot{\mathbf{y}}(t)) = \mathbf{0}, \tag{5.1}$$

where $\mathbf{b} : \mathbb{R} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_y}$ *and* $\mathbf{y} : \mathbb{R} \to \mathbb{R}^{n_y}$ *are vector-valued functions and* t *is the independent variable. Here and in the following* $\dot{\mathbf{y}}$ *means the derivative of* $\mathbf{y}$ *with respect to* t*.*

It is often the case in our applications that the DAE problem can be described in a more structured way, which leads to the type of the linearly implicit DAE.

Definition 5.2 (Linearly implicit DAE)
A DAE is called linearly implicit, *if it can be divided into a differential and an algebraic part with corresponding* differential *variables* $\mathbf{x}$ *and* algebraic *variables* $\mathbf{z}$, *and the time derivatives of* $\mathbf{x}$ *enter the problem linearly, which results in the following form.*

$$\begin{aligned} \mathbf{A}(t, \mathbf{x}(t), \mathbf{z}(t))\, \dot{\mathbf{x}}(t) &= \mathbf{f}(t, \mathbf{x}(t), \mathbf{z}(t)) \in \mathbb{R}^{n_x} \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}(t), \mathbf{z}(t)) \in \mathbb{R}^{n_z}, \end{aligned} \tag{5.2}$$

where $\mathbf{A}(t, \mathbf{x}(t), \mathbf{z}(t)) \in \mathbb{R}^{n_x \times n_x}$, $\mathbf{x} : \mathbb{R} \to \mathbb{R}^{n_x}$ *and* $\mathbf{z} : \mathbb{R} \to \mathbb{R}^{n_z}$. *If* $\mathbf{A}$ *is equal to the identity matrix, we call the DAE* semi-explicit.

Definition 5.3 (Solution of a DAE)
A classical solution of a DAE on an interval $\mathcal{I} \subset \mathbb{R}$ *is a continuously differentiable function*

$$\mathbf{y}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{z}(t) \end{pmatrix} : \mathbb{R} \to \mathbb{R}^{n_y} = \mathbb{R}^{n_x} \times \mathbb{R}^{n_z},$$

which fulfills the given equations for all $t \in \mathcal{I}$.

From these definitions we see that a DAE can be understood as an ODE with some algebraic constraints. These constraints define a manifold on which the solution of the DAE remains. Furthermore, every sufficiently smooth DAE can be transformed into an ODE by differentiation of the algebraic part of the system with respect to t. For the characterization of the algebraic part of a DAE system and the relationship between ODEs and DAEs the *differential index*, invented by Gear [Gea88], plays an important role.

Definition 5.4 (Differential index of a DAE)
The implicit DAE

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

is of differential index $k \in \mathbb{N}$ *(short:* index*), if* k *is the smallest number, such that* $\dot{\mathbf{y}}(t)$ *is fully determined by the* $(k+1)$ *equations:*

$$\begin{aligned} \mathbf{b}(t, \mathbf{y}(t), \dot{\mathbf{y}}(t)) &= \mathbf{0} \\ \frac{d}{dt}\mathbf{b}(t, \mathbf{y}(t), \dot{\mathbf{y}}(t)) &= \mathbf{0} \\ &\vdots \\ \frac{d^k}{dt^k}\mathbf{b}(t, \mathbf{y}(t), \dot{\mathbf{y}}(t)) &= \mathbf{0}. \end{aligned}$$

Example 5.5 (Example of an index 1 DAE)
Consider a semi-explicit DAE and the (total) derivative of the algebraic equations $\mathbf{g}(t, \mathbf{x}(t), \mathbf{z}(t)) = \mathbf{0}$ with respect to t. We obtain (omitting the arguments)

$$\mathbf{g_t} + \mathbf{g_x}\dot{\mathbf{x}} + \mathbf{g_z}\dot{\mathbf{z}} = \mathbf{0},$$

where subscripts denote the partial derivative with respect to the corresponding variables. Let $\mathbf{g_z}$ be regular, then we can transform the equation to

$$\dot{\mathbf{z}} = -\mathbf{g_z}^{-1}(\mathbf{g_t} + \mathbf{g_x}\dot{\mathbf{x}})$$

and we end up with the coupled ODE system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(t, \mathbf{x}(t), \mathbf{z}(t)) \\ \dot{\mathbf{z}}(t) &= -\mathbf{g_z}^{-1}(t, \mathbf{x}(t), \mathbf{z}(t)) \left[\mathbf{g_t}(t, \mathbf{x}(t), \mathbf{z}(t)) + \mathbf{g_x}(t, \mathbf{x}(t), \mathbf{z}(t))\dot{\mathbf{x}}(t)\right]. \end{aligned}$$

Therefore, by Definition 5.4 a semi-explicit DAE is of differential index 1 if $\mathbf{g_z}$ is regular.

Remark 5.6 (DAEs of index larger than 1)
A DAE with index $k > 1$ can be transformed into an index 1 DAE by differentiating the algebraic equations $(k-1)$ times with respect to t. The analytic solution of an index-reduced system fullfills the algebraic equations of the original problem and their first $(k-1)$ derivatives. Hence it is identical to the analytic solution of the original problem. The system consisting of the algebraic equations and their first $(k-1)$ derivatives is called the *invariants* of the index reduced system.

Remark 5.7 (Numerical problems)
The numerical solution of DAEs with index larger than 1 is more complicated than it would seem after the previous explications. During the numerical computation it is inevitable that the numerical solution leaves the manifold defined by the invariants due to discretization and round-off errors. Therefore, it has to be ensured that the invariants remain fulfilled during the computation. The numerical treatment of such problems is described in more details in the works of Eich [Eic91, Eic93], von Schwerin [vS97], Petzold et al. [PRG+97] or Pantelides et al. [PSV94] and will not further be discussed in this thesis.

The next theoretical concept that is important for the numerical treatment of DAEs is the perturbation index. It describes the sensitivity of the system with respect to small disturbances in the right hand side or the initial values of the system. It was introduced by Hairer et al. [HLR89].

Definition 5.8 (Perturbation index of a DAE)
For a DAE (5.1) *the* perturbation index *along a solution* $\mathbf{y}(t)$, $t \in [t_0, t_f]$ *is defined as the smallest natural number* k, *such that for all functions* $\hat{\mathbf{y}}(t)$ *with the defect*

$$\mathbf{b}(t, \hat{\mathbf{y}}(t), \dot{\hat{\mathbf{y}}}(t)) = \boldsymbol{\delta}(t)$$

the estimate

$$\begin{aligned} \| \hat{\mathbf{y}}(t) - \mathbf{y}(t) \| \leq\ & C(\mathbf{b}, |t_f - t_0|)\Bigg(\| \hat{\mathbf{y}}(0) - \mathbf{y}(0) \| \\ &+ \max_{0 \leq \tilde{t} \leq t} \| \int_0^{\tilde{t}} \boldsymbol{\delta}(\tau) d\tau \| \\ &+ \max_{0 \leq \tilde{t} \leq t} \| \boldsymbol{\delta}(\tilde{t}) \| + \ldots + \max_{0 \leq \tilde{t} \leq t} \| \boldsymbol{\delta}^{(k-1)}(\tilde{t}) \| \Bigg) \end{aligned} \tag{5.3}$$

holds, provided that $\boldsymbol{\delta}(t)$ is small enough. Here $C(\mathbf{b}, |t_f - t_0|)$ stands for a constant that depends only on the function $\mathbf{b}$ and the length of the time horizon.

Remark 5.9
The importance of the perturbation index for the numerical treatment lies in the characterization of the influence of round-off errors in $\mathbf{b}$ on the numerical solution. If the DAE is of perturbation index k, we have as part of (5.3) the $(k-1)$-th derivative of $\boldsymbol{\delta}$. While the perturbation $\boldsymbol{\delta}$ itself can be very small (e.g., in the order of the machine precision), its derivative can be very large. In the end this may lead to problems in the numerical solution if k is larger than 1. It can be shown that the accuracy of the numerical solution is influenced by round-off and discretization errors with order $\mathcal{O}(h^{(1-k)})$, where h is the maximum stepsize during the numerical solution.

Gear showed the following connection between differential index and perturbation index of a DAE, for a proof see [HW96].

Theorem 5.10 (Gear, 1990)
For the DAE (5.1) it holds that

$$pi \leq di + 1$$

if the differential index di and the perturbation index pi exist.

Remark 5.11
The perturbation index is 0, if the estimate

$$\| \hat{\mathbf{y}}(t) - \mathbf{y}(t) \| \leq C(\mathbf{b}, |t_f - t_0|) \left(\| \hat{\mathbf{y}}(0) - \mathbf{y}(0) \| + \max_{0 \leq \tilde{t} \leq t} \| \int_0^{\tilde{t}} \boldsymbol{\delta}(\tau) d\tau \| \right)$$

holds. This is always the case for ODEs with Lipschitz-continuous right hand side. In case of semi-explicit DAEs it can be shown that the differential index is equal to the perturbation index.

5.1.2 Existence and uniqueness of solutions

In the theory of ODE systems there exist quite simple theorems concerning existence and uniqueness of initial values problems, e.g., the well-known theorems of Peano and Picard-Lindelöf (see, e.g., [Wal93]). Some results from ODE theory can be transferred to the DAE context by understanding DAEs as ODEs with the restriction that the solution has to lie on a specific manifold, as described above. This approach was used for example by Rheinboldt [Rhe84].
For the general solvability of DAEs we have the following theorem.

Definition 5.12 (Solvability of DAEs)
Let $\mathcal{I} \subset \mathbb{R}$ be open, Ω an open and connected subset of $\mathbb{R} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_y}$ and $\mathbf{b} : \Omega \to \mathbb{R}^{n_y}$ differentiable. The DAE (5.1) is then solvable in Ω on the interval $\mathcal{I}$, if there is an r-dimensional family of solutions $\mathbf{Y}(t, \mathbf{c})$, that are defined on a connected set $\mathcal{I} \times \tilde{\Omega}$, $\tilde{\Omega} \subset \mathbb{R}^r$, such that

1. *$\mathbf{Y}(t, \mathbf{c})$ is defined for all $t \in \mathcal{I}$ and for all $\mathbf{c} \in \tilde{\Omega}$,*

2. $(t, \mathbf{Y}(t,\mathbf{c}), \dot{\mathbf{Y}}(t,\mathbf{c})) \in \Omega$ *for* $(t,\mathbf{c}) \in \mathcal{I} \times \tilde{\Omega}$,

3. *if* $\mathbf{w}(t)$ *is another solution with* $(t, \mathbf{w}(t), \dot{\mathbf{w}}(t)) \in \Omega$, *then it holds* $\mathbf{w}(t) = \mathbf{Y}(t,\mathbf{c})$ *for a certain* $\mathbf{c} \in \tilde{\Omega}$,

4. *the graph of* $\mathbf{Y}$ *as function of* $(t,\mathbf{c})$ *is a* $(r+1)$*-dimensional manifold.*

This concept of solvability implies that no bifurcations exist. For the case of a linearly implicit DAE (5.2) of index 1 one obtains the following results.

Proposition 5.13 (Existence and uniqueness for index 1 DAEs)
Let $\mathbf{A} : \mathbb{R} \times \mathcal{S} \longrightarrow \mathbb{R}^{n_x} \times \mathbb{R}^{n_z}$, $\mathbf{f} : \mathbb{R} \times \mathcal{S} \longrightarrow \mathbb{R}^{n_x}$ *and* $\mathbf{g} : \mathbb{R} \times \mathcal{S} \longrightarrow \mathbb{R}^{n_z}$ *be* $\mathcal{C}^r$*-functions,* $r \geq 2$, $\mathcal{S} \subset \mathbb{R}^{n_x+n_z}$ *open and* $y = (x^T, z^T)^T \in \mathbb{R}^{n_y}$.
Then

$$\mathcal{S}_0 = \left\{ (t,\mathbf{y}) \in \mathbb{R} \times \mathcal{S} : \mathit{rank} \begin{pmatrix} \mathbf{A}(t,\mathbf{y}) & \mathbf{0} & -\mathbf{f}(t,\mathbf{y}) \\ \mathbf{g_x}(t,\mathbf{y}) & \mathbf{g_z}(t,\mathbf{y}) & \mathbf{g_t}(t,\mathbf{y}) \end{pmatrix} = n_y \right\}$$

is an open subset of $\mathbb{R}^{1+n_y}$. *For the manifold*

$$\mathcal{M}(\mathbf{g},\mathcal{S}) = \{(t,\mathbf{y}) \in \mathbb{R} \times \mathcal{S} : \mathbf{g}(t,\mathbf{y}) = \mathbf{0}\},$$

it holds in the case of $\mathcal{M}_0 = \mathcal{M}(\mathbf{g},\mathcal{S}) \cap \mathcal{S}_0 \neq \emptyset$ *that* $\mathcal{M}_0$ *is a submanifold of* $\mathcal{M}(\mathbf{g},\mathcal{S})$ *and there exists for all* $(t_0, \mathbf{y_0})$ *a unique* $\mathcal{C}^{r-1}$*-solution of* (5.2) *which goes through* $(t_0, \mathbf{y_0})$.

Remark 5.14 (Numerical solution of index 1 DAEs)
Be

$$\mathcal{S}_1 = \left\{ (t,\mathbf{y}) \in \mathbb{R} \times \mathcal{S} : \operatorname{rank} \begin{pmatrix} \mathbf{A}(t,\mathbf{y}) & \mathbf{0} \\ \mathbf{g_x}(t,\mathbf{y}) & \mathbf{g_z}(t,\mathbf{y}) \end{pmatrix} = n_y \right\}$$

and

$$\mathcal{M}_1 = \mathcal{M}(\mathbf{g},\mathcal{S}) \cap \mathcal{S}_1 \neq \emptyset.$$

For the numerical solution of DAE (5.2) with standard methods additionally $(t,\mathbf{y}) \in \mathcal{M}_1$ should hold for all points of the solution.
Then the matrices $\mathbf{A}$ and $\mathbf{g_z}$ are regular for all $(t,\mathbf{y}) \in \mathcal{M}_1$, their inverse matrices are bounded and for consistent initial values $(t_0, \mathbf{y}_0) \in \mathcal{M}_1$ the initial value problem for (5.2) has a unique solution $\mathbf{y}(t)$. This solution depends continuously and $(r-1)$-times differentiable on the initial values $\mathbf{x}_0$. In this case $\mathbf{z_0}$ is uniquely determined by $\mathbf{x_0}$ via the consistency conditions $\mathbf{g}(t_0, \mathbf{x_0}, \mathbf{z_0}) = 0$.

5.2 Numerical solution of initial value problems for DAEs of index 1

In the following we introduce the class of Linear Multistep Methods (LMMs) for the solution of IVPs for ODEs and DAEs. We will lay the focus on the Backward Differentiation Formulas (BDFs), a particular class of LMMs which has proven itself to be very efficient for the numerical

solution of stiff ODEs and DAEs. We will present and analyze the methods first for ODEs on equidistant grids. Afterwards we switch to variable grids and finally to the case of linearly implicit DAEs. For further details on these topics we refer, e.g., to the text book of Strehmel and Weiner [SW95] or the text books of Hairer et al. [HW96, HNW93]. In these books also the proofs of that theorems that are just stated here, and for which no other references are given, can be found.

5.2.1 Linear multistep methods on equidistant grids

We first consider the numerical solution of initial value problems for ODEs on equidistant grids. It is assumed that the initial value problems are of the following type.

Definition 5.15 (IVP for ODEs)
Let be $\mathcal{I} = [t_0, t_f] \subset \mathbb{R}$, $\mathbf{f} : \mathbb{R} \times \mathbb{R}^{n_x} \longrightarrow \mathbb{R}^{n_x}$. *The* initial value problem *is defined as the problem of finding a function* $\mathbf{x} : \mathbb{R} \longrightarrow \mathbb{R}^{n_x}$ *which fulfills the system*

$$\dot{\mathbf{x}}(t) \quad = \quad \mathbf{f}(t, \mathbf{x}(t))$$

and the initial values

$$\mathbf{x}(t_0) = \mathbf{x_0}.$$

t_0 *is called the start or initial time and* t_f *is called the end or final time.*

Remark 5.16 (Parameter dependency of the IVP)
Note that especially in the framework of an optimization problem the right hand side function $\mathbf{f}$, and possibly also the initial values $\mathbf{x_0}$, could also depend on some system parameter, control functions or similar. As these are usually either constant or depend at most on t for a specific IVP solution process, we can skip them for notational simplicity in the following discussion. We will consider the more general parameter dependent case in Chapter 6 on sensitivity generation.

Remark 5.17 (Stiffness)
Particularly dynamic models of chemical and biological processes, occasionally also of mechanical and electrical processes, often possess a certain property called stiffness. This was discovered and first described by Curtiss and Hirschfelder [CH52]. Summarized their description is that stiff equations are equations on which certain implicit methods, namely BDF methods, work better, usually tremendously better, than explicit ones. Hairer and Wanner [HW96] characterize this even more concise: Stiff equations are problems for which explicit methods do not work. Until today there is no standardized definition of stiffness. Usual characterizations for stiff systems are

- The IVP has slowly changing solutions and other solutions in their neighborhood approach them fast,
- There exist eigenvalues λ_i of the matrix $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ for which

  $$\Re(\lambda_i) \ll 0$$

  holds, where $\mathbf{f}$ is the right hand side of an ODE with

  $$\left\| \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(t, \mathbf{x}(t)) \right\| |t_f - t_0| \gg 1.$$

A well known example for a stiff system is a system of chemical reactions where some reactions take place at a much faster rate than others. It should be noted that stiffness is in general not a property of the whole system but, more precisely, a property of the initial value problem which is stiff for certain initial values.

Definition 5.18 (General linear multistep methods)
The general form of a k-step linear multistep method *for the computation of a grid function* $\mathbf{x_h}(t)$ *on an equidistant grid*

$$I_h = \{t \in [t_0, t_f] : t = t_m, m = 0, 1, \ldots, N, t_m = t_0 + mh\}$$

as approximation of the solution $\mathbf{x}(t)$ *of an IVP is defined by*

1. *k start values* $\mathbf{x_m} = \mathbf{x_h}(t_m), m = 0, 1, \ldots, k-1$ *and*
2. *a difference equation for the computation of the next approximated values* $\mathbf{x_{m+k}}$

$$\sum_{l=0}^{k} \alpha_l \mathbf{x_{m+l}} = h \sum_{l=0}^{k} \beta_l \mathbf{f}(t_{m+l}, \mathbf{x_{m+l}}), \quad m = 0, 1, \ldots, N-k, \tag{5.4}$$

with $\alpha_l, \beta_l \in \mathbb{R}$ *and* $\alpha_k \neq 0, |\alpha_0| + |\beta_0| \neq 0$.

Remark 5.19

- The method is called linear because the method function

$$\phi(t_m, \ldots, t_{m+k}, \mathbf{x_m}, \ldots, \mathbf{x_{m+k}}; h) = h \sum_{l=0}^{k} \beta_l \mathbf{f}(t_{m+l}, \mathbf{x_{m+l}})$$

  depends linearly on $\mathbf{f}(t_{m+l}, \mathbf{x_{m+l}})$.

- The condition $a_k \neq 0$ ensures that the implicit equation (5.4) has a (locally) unique solution $\mathbf{x_{m+k}}$ (at least for sufficiently small h).

- The condition $|\alpha_0| + |\beta_0| \neq 0$ ensures that the number k of steps is uniquely determined.

- For $\beta_k = 0$ the method is explicit and the solution can be computed directly. Otherwise the method is implicit and in every step we have to solve an equation system of the type

$$\mathbf{x_{m+k}} = h \frac{\beta_k}{\alpha_k} \mathbf{f}(t_{m+k}, \mathbf{x_{m+k}}) + \mathbf{v}, \tag{5.5}$$

  where

$$\mathbf{v} = \frac{1}{\alpha_k} \sum_{l=0}^{k-1} [h\beta_l \mathbf{f}(t_{m+l}, \mathbf{x_{m+l}}) - \alpha_l \mathbf{x_{m+l}}]$$

  is independent of $(t_{m+k}, \mathbf{x_{m+k}})$. This system is in general nonlinear and it is usually solved, depending on the problem class, by functional iteration or Newton's (or a Newton-type) method.

Remark 5.20
By Definition 5.18 we observe that a LMM consists of 2 phases:

- The start-up phase where the approximations $\mathbf{x_1}, \ldots, \mathbf{x_{k-1}}$ are computed using, e.g., a one-step method or a LMM with fewer steps, and
- the run phase described by (5.4) where in every step a system of equations has to be solved that in general is nonlinear.

Order of a LMM

In this part we introduce concepts that allow us to quantify how exact the approximations generated by a LMM are.

Definition 5.21 (Generating polynomials)
For a LMM 5.18 we define the generating polynomials *as*

$$\rho(\xi) = \alpha_k \xi^k + \alpha_{k-1}\xi^{k-1} + \cdots + \alpha_0 \tag{5.6}$$
$$\chi(\xi) = \beta_k \xi^k + \beta_{k-1}\xi^{k-1} + \cdots + \beta_0. \tag{5.7}$$

Definition 5.22 (Local discretization error, consistency error)
The local discretization error σ *and the* consistency error τ *of a LMM are defined as*

$$h\tau(\mathbf{x}(t), h) := \sigma[\mathbf{x}(t), h] := L[\mathbf{x}(t), h] := \sum_{l=0}^{k} \left[\alpha_l \mathbf{x}(t + lh) - h\beta_l \dot{\mathbf{x}}(t + lh)\right],$$

where L is called the linear difference operator of the LMM.

Definition 5.23 (Consistency order of a LMM)
A LMM is of consistency order *p, if for all $\mathbf{x} \in \mathcal{C}^{p+1}([t_0, t_f], \mathbb{R}^{n_x})$*

$$L[\mathbf{x}(t), h] = \mathcal{O}(h^{p+1})$$

holds for $h \to 0$. The consistency order describes how fast the local error tends to zero for $h \to 0$.

Lemma 5.24
A LMM is of *consistency order p*, if the following conditions are fulfilled

$$\sum_{l=0}^{k} \alpha_l = 0, \tag{5.8}$$
$$\sum_{l=0}^{k} (l\alpha_l - \beta_l) = 0, \tag{5.9}$$
$$\sum_{l=0}^{k} \left[\frac{1}{\nu!} l^\nu \alpha_l - \frac{1}{(\nu-1)!} l^{\nu-1} \beta_l\right] = 0, \quad \nu = 2, 3, \ldots, p. \tag{5.10}$$

Remark 5.25 (Consistency conditions)
The necessary conditions for a LMM to be of consistency order 1 are called consistency conditions. They can be written using the generating polynomials as

$$\rho(1) = 0, \qquad \rho'(1) = \chi(1).$$

If the consistency order of a method is at least one, then $\xi_1 = 1$ is a root of $\rho(\xi)$ and the method is called *consistent*.

Zero stability of a LMM

One observes that a consistent LMM is not necessarily convergent, even if it is of high consistency order and the local error is therefore small. This is due to the error propagation through inexact earlier approximations $\mathbf{x_m}$ and the evaluations of the right hand side $\mathbf{f}$ at these values.
If we neglect for the moment the error propagation through the $\mathbf{f_m}$-terms and analyze the case $h \to 0$, we obtain the concept of zero stability. Zero stability of a method assures that the error propagation through the historical values remains bounded.

Definition 5.26 (Zero stability)
A LMM is called zero stable*, if the generating polynomial $\rho(\xi)$ satisfies:*

a) The roots of $\rho(\xi)$ lie on or inside the unit circle and

b) the roots on the unit circle are simple roots.

Remark 5.27
If all roots besides $\xi_1 = 1$ lie inside the unit circle, the method is called *strongly stable*.

Convergence of a LMM

Dahlquist showed in 1956 [Dah56] that consistency and zero stability are necessary and sufficient conditions for the convergence of a LMM.

Definition 5.28 (Convergence of a LMM)
A LMM is convergent*, if for all IVPs with Lipschitz-continuous right hand side $\mathbf{f}$ on $\mathcal{S} := \{(t, \mathbf{x}) : t_0 \le t \le t_f, \mathbf{x} \in \mathbb{R}^{n_x}\}$ and for all start values $\mathbf{x_h}(t_0 + mh)$, $m = 0, 1, \ldots, k-1$ with*

$$||\mathbf{x}(t_0 + mh) - \mathbf{x_h}(t_0 + mh)|| \to 0 \ \text{ for } h \to 0$$

holds that

$$\epsilon(\mathbf{x}(t), h) := ||\mathbf{x}(t) - \mathbf{x_h}(t)|| \to 0, \ h \to 0, \ t \in [t_0, t_f],$$

where ϵ is called the global error *of the LMM.*

Definition 5.29 (Order of convergence)
A LMM is convergent of order p *if for all IVPs with* $\mathbf{f}$ *smooth enough on* $\mathcal{S}$ *there exists an* $h_0 > 0$ *such that for all start values* $\mathbf{x_h}(t_0 + mh)$, $m = 0, 1, \ldots, k-1$ *with*

$$||\mathbf{x}(t_0 + mh) - \mathbf{x_h}(t_0 + mh)|| \leq C_0 h^p \quad \text{for } h \in (0, h_0]$$

$$||\mathbf{x}(t) - \mathbf{x_h}(t)|| \leq C h^p, \quad \text{for } h \in (0, h_0]$$

holds.

Definition 5.30 (Stability of LMMs)
A LMM is stable *if for all sufficiently smooth functions* $\mathbf{y}(t)$ *there exist constants* $C_1, C_2 \in \mathbb{R}$ *that are independent of* h *such that for all grids* I_h *it holds:*

$$\max_{t \in I_h} ||\epsilon(\mathbf{x}(t), t)|| \leq C_1 \max_{t \in I_h} ||\tau(\mathbf{x}(t), t)|| + C_2 \, \epsilon_{\text{start}},$$

where ϵ_{start} *is the maximum of the norms of the errors in the start values.*

Corollary 5.31
If a LMM is consistent and stable and the errors in the start values are zero, then it is convergent and the order of convergence is equal to the consistency order.

Theorem 5.32 (Dahlquist)
Let $\mathbf{y}(t) \in \mathcal{C}^2(I, \mathbb{R}^{n_y})$ *and* $\mathbf{f}$ *be Lipschitz-continuous, then a zero stable and consistent LMM is stable.*

Theorem 5.33 (Criterion for convergence)
A LMM is convergent iff it is zero stable and consistent.
A LMM is convergent of order p *iff it is zero stable and of consistency order* p.

The order of a k-step LMM cannot be arbitrarily high, as Dahlquist showed in [Dah56] that the consistency order of a zero stable k-step LMM is bounded. This fact is called the first Dahlquist barrier.

Theorem 5.34 (First Dahlquist barrier)
The consistency order p *of a zero stable* k*-step LMMs satisfies*

$$\begin{aligned} p &\leq k+2 \quad \text{if } k \text{ even}, \\ p &\leq k+1 \quad \text{if } k \text{ odd}, \\ p &\leq k \quad \text{if } \frac{\beta_k}{\alpha_k} \leq 0 \text{ (i.e., particularly for explicit LMMs).} \end{aligned}$$

Absolute stability of a LMM

The concept of zero stability neglects the error propagation through the right hand side terms $\mathbf{f}(t_m, \mathbf{x_m})$, as it analyzes the behavior for $h \to 0$. This may cause the fact that even if a LMM is zero stable and consistent, and thus convergent, reasonable results can only be achieved for very small stepsizes h. Analyzing the error propagation through the right hand side leads to the concept of *absolute stability*.
This analysis is usually done by investigating the behavior of the LMM on a characteristic scalar test equation, the so called Dahlquist test equation

$$\dot{\mathbf{x}}(t) = \lambda \mathbf{x}(t),\ \lambda \in \mathbb{C}, \mathbf{x}(t) \in \mathbb{C}. \tag{5.11}$$

If we apply a k-step LMM to solve this equation we obtain the linear difference equation

$$(\alpha_k - h\lambda\beta_k)\mathbf{x_{m+k}} + \ldots + (\alpha_0 - h\lambda\beta_0)\mathbf{x_m} = \mathbf{0}. \tag{5.12}$$

This equation has stable solutions $\mathbf{x_l}$, iff all roots of the *characteristic equation*

$$\rho(\xi) - h\lambda\chi(\xi) = 0$$

lie on or inside the unit circle, and multiple roots lie inside the unit circle.
This inspires the following definition of the region of absolute stability.

Definition 5.35 (Region of absolute stability)
The region of absolute stability *(or short stability region) of a LMM is defined as*

$$\mathcal{D} := \Big\{ h\lambda \in \mathbb{C} : \textit{All roots of } (5.12) \textit{ satisfy } |\xi_i(h\lambda)| \leq 1,$$
$$\textit{and multiple roots additionally satisfy } |\xi_i(h\lambda)| < 1 \Big\}.$$

Remark 5.36
For $h = 0$ this is the definition of zero stability. Therefore zero stability is equivalent to $0 \in \mathcal{D}$.

Example 5.37 (Stability regions of the Euler methods)
For the characteristic equation of the explicit Euler method $\mathbf{x_{m+1}} = \mathbf{x_m} + h\mathbf{f}(t_m, \mathbf{x_m})$ we have

$$-1 + \xi - h\lambda = 0,$$

and hence $|1 + h\lambda| \leq 1$ for $h\lambda \in \mathcal{D}$.
For the implicit Euler method $\mathbf{x_{m+1}} = \mathbf{x_m} + h\mathbf{f}(t_{m+1}, \mathbf{x_{m+1}})$ we obtain

$$-1 + \xi - h\lambda\xi = 0.$$

Therefore $|1 - h\lambda| \geq 1$ for $h\lambda \in \mathcal{D}$ (cf. Figure 5.1).

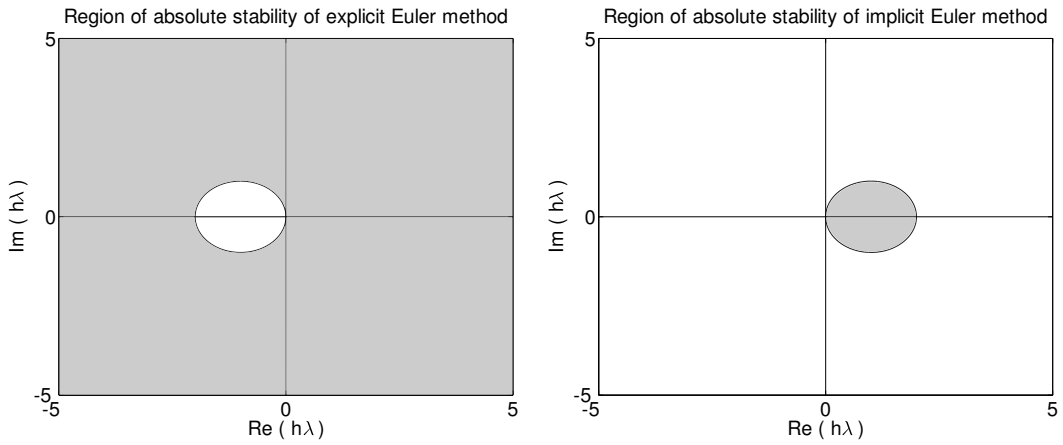

Figure 5.1: Stability regions for the explicit (left) and implicit (right) Euler methods. The stability regions are displayed as white areas.

Remark 5.38
The explicit Euler method is a representative example for the inefficiency of explicit methods in the treatment of stiff problems. Transferred to the test equation stiffness corresponds usually to $\Re(\lambda) \ll 0$. Because of the bounded stability region (which is a characteristic of all explicit methods) we may be forced to choose very small stepsizes, regardless of the local discretization error. For instance $\Re(\lambda) = -1000$ implies necessarily $h < 10^{-3}$.

Therefore the property that $\mathbb{C}^- = \{z \in \mathbb{C} : \Re(z) \leq 0\}$ is a subset of the stability region $\mathcal{D}$ is highly desired in the treatment of stiff problems. These property has been defined by Dahlquist as A-stability.

Definition 5.39 (A-stability of a LMM)
A LMM is A-stable *if* $\mathbb{C}^- \subset \mathcal{D}$.

Dahlquist showed in [Dah63] that there exists an order limit for an A-stable LMM.

Theorem 5.40 (Second Dahlquist barrier)
An A-stable LMM has a consistency order of $p \leq 2$.

A consistency order of at most 2 is not sufficient in pratice to solve problems efficiently. This led to a slightly weaker definition, introduced by Widlund, to describe nearly A-stable methods.

Definition 5.41 (A(α)-stability of a LMM)
A convergent LMM is A(α)-stable *with* $0 < \alpha < \pi/2$ *if*

$$\mathcal{D}_\alpha := \{h\lambda : |\arg(-h\lambda)| < \alpha,\ h\lambda \neq 0\} \subset \mathcal{D},$$

where arg stands for the complex argument. In other words, α describes the angle between a straight line through the origin and the negative real axis in $\mathbb{C}^-$, such that the area between the straight line and the negative real axis belongs to the stability region.

5.2.2 LMMs on variable grids

After having introduced basic notation and definitions as well as having presented the most important properties of LMMs on equidistant grids, we now proceed with the case of variable grids. We will see that most of the definitions and results can be transferred with only minor modifications.

Definition 5.42 (General LMM on variable grids)
A general k-step LMM on a variable grid

$$I_h = \{t \in [t_0, t_f] : t = t_m, m = 0, 1, \ldots, N,\ t_m = t_{m-1} + h_{m-1} \text{ for } m \neq 0\}$$

is defined by

$$\sum_{l=0}^{k} \alpha_{lm} \mathbf{x_{m+l}} = h_{m+k-1} \sum_{l=0}^{k} \beta_{lm} \mathbf{f}(t_{m+l}, \mathbf{x_{m+l}}),\ m = 0, \ldots, N-k, \tag{5.13}$$

$$\mathbf{x_h}(t_m) = \mathbf{x_m}, \qquad m = 0, \ldots, k-1,$$

where $\alpha_{lm}, \beta_{lm} \in \mathbb{R}$, $|\alpha_{0m}| + |\beta_{0m}| \neq 0$ *and* α_{lm}, β_{lm} *depend on the stepsize changes* $\omega_i := h_i/h_{i-1}$, $i = m+1, \ldots, m+k-1$.

Consistency order of LMM on variable grids

Similar to 5.23 we define the consistency of a LMM on a variable grid.

Definition 5.43 (Consistency order on variable grids)
A LMM (5.13) *has* consistency order p, *if for all polynomials* $q(t)$ *with* $\deg(q) \leq p$ *and all grids* I_h *it holds that*

$$\sum_{l=0}^{k} \alpha_{lm} q(t_{m+l}) = h_{m+k-1} \sum_{l=0}^{k} \beta_{lm} \dot{q}(t_{m+l}).$$

Theorem 5.44
Be the LMM (5.13) *of consistency order* p *and* $f \in \mathcal{C}^p(S, \mathbb{R}^{n_x})$. *Additionally it shall hold:*

1. *The stepsize changes* $\omega_i = h_i/h_{i-1}$, $i = m+1, \ldots, m+k-1$, *are bounded for all* m *and*
2. *the coefficients* α_{lm}, β_{lm} *are bounded.*

Then the local discretization error, which is defined analogously to the equidistant case, is of order $\mathcal{O}(h_m^{p+1})$.

Stability

Definition 5.45
Assume that the coefficients α_{lm} of the LMM are normalized, such that $\alpha_{km} = 1$. We then define for a LMM (5.13) with coefficients α_{lm} the matrices

$$\mathcal{A}_{\mathbf{m}} := \begin{pmatrix} -\alpha_{k-1,m} & -\alpha_{k-2,m} & \dots & -\alpha_{1,m} & -\alpha_{0,m} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

Definition 5.46 (Stability on variable grids)
A LMM on a variable grid is stable, *if there exists $M \in \mathbb{R}$ such that*

$$||\mathcal{A}_{\mathbf{m+j}}\mathcal{A}_{\mathbf{m+j-1}} \dots \mathcal{A}_{\mathbf{m+1}}\mathcal{A}_{\mathbf{m}}|| \leq M$$

for all m and $j \geq 0$.

Crouzeix and Lisbona [CL84] have proven the following connection between stability on variable grids and strong stability on equidistant grids.

Theorem 5.47 (Stability on variable grids)
Assume for the LMM (5.13) it holds that

1. $\sum_{l=0}^{k} \alpha_{lm} = 0$,
2. *the coefficients $\alpha_{lm} = \alpha_{lm}(\omega_{m+1}, \dots, \omega_{m+k-1})$ are continuous in a neighborhood of $(1, \dots, 1)$ and*
3. *the LMM is strongly stable on all equidistant grids.*

Then there exist $\omega, \Omega \in \mathbb{R}$ with $\omega < 1 < \Omega$, such that the LMM is stable provided

$$\omega \leq \omega_m \leq \Omega$$

holds for all m.

Convergence

Finally we can formulate a convergence criterion for variable grids.

Theorem 5.48 (Convergence of a LMM on variable grid)
Let the LMM (5.13) be stable, of consistency order p and let the coefficients α_{lm}, β_{lm} be bounded. For the start values $\mathbf{x_h}(t_m),\ m = 0, 1, \dots, k-1$ it holds that

$$||\mathbf{x}(t_m) - \mathbf{x_h}(t_m)|| = \mathcal{O}(h_0^p)$$

and the stepsize changes ω_m are bounded for all $m \geq 1$.
Then the LMM is convergent of order p, i.e., there exists a $C \in \mathbb{R}$, such that

$$||\mathbf{x}(t_m) - \mathbf{x_h}(t_m)|| \leq Ch^p, \quad t_m \in [t_0, t_f], \quad h = \max_m h_m.$$

5.2.3 BDF methods

After the analysis of general LMMs on equidistant and variable grids we now present the class of Backward Differentiation Formulas (BDF). This specific class of LMMs is the basis of `DAESOL-II`, our numerical integration code. We employ BDF methods in this thesis as they are very efficient for the solution of stiff problems. BDF methods were invented by Curtiss and Hirschfelder [CH52] for the solution of stiff ODEs and became more famous through the analysis of Gear [Gea71] who used them for the first time also in the DAE context. We start with BDF methods and their properties for ODEs and explain afterwards how the results can be transferred to the DAE case.

BDF methods for ODEs

The underlying idea of BDF methods is to interpolate the last $(k+1)$ values $\mathbf{x_m}, \ldots, \mathbf{x_{m+k}}$ using a polynomial and to require that the interpolation polynomial satisfies the ODE at point t_{m+k}.

Definition 5.49 (BDF method)
The k-step BDF method *is defined by specifying the k start values and the difference equation*

$$\sum_{l=0}^{k} \alpha_{lm} \mathbf{x_{m+l}} = h_{m+k-1} \mathbf{f}(t_{m+k}, \mathbf{x_{m+k}}), \quad m = 0, \ldots, N-k,$$

with $\alpha_{lm} \in \mathbb{R}$, $\alpha_0, \alpha_k \neq 0$. The α_{lm} are obtained as the coefficients of the derivative of the interpolation polynomial multiplied by h_{m+k-1}. BDF methods are implicit methods as $\beta_k = 1 \neq 0$.

Theorem 5.50 (Order of BDF methods)
A k-step BDF method is by construction of consistency order k.

Unlike other LMMs the BDF methods are not zero stable by construction. Cryer [Cry72] showed for the zero stability of BDF methods the following theorem.

Theorem 5.51 (Zero stability of BDF methods)
BDF methods (on equidistant grids) are zero stable for $k \leq 6$ and unstable for $k \geq 7$.

As direct consequence of this and Theorem 5.33, BDF methods with $k \leq 6$ are also convergent of order k. Hence in practice only methods up to $k = 6$ are relevant, and the method with $k = 7$ is only of use for local error estimation.

Theorem 5.52 (Stability regions of BDF methods)
BDF methods are A-stable for $k \leq 2$ and $A(\alpha)$-stable for $k \leq 6$ with the following values for α:

k	1	2	3	4	5	6
α	90°	90°	86.03°	73.35°	51.84°	17.84°

The stability regions of the BDF methods with an order up to 7 are displayed in Figure 5.2 on the facing page.

Griegorieff [Gri83] analyzed the behavior of BDF methods on variable grids and proved stability under specific restrictions on the stepsize changes:

Theorem 5.53 (Stability of BDF methods on variable grids)
BDF methods are stable on variable grids, if the following bounds on stepsize changes are satisfied:

k	2	3	4	5	6
ω	0	0.836	0.979	0.997	$1-\delta_1$
Ω	2.414	1.127	1.019	1.003	$1+\delta_2$

with $0 < \delta_1, \delta_2 < 0.001$. *The 1-step BDF method is a one-step method which is stable on every grid.*

Remark 5.54 (Stepsize changes)
The above bounds for stepsize changes in methods of order $k \geq 3$ are very restrictive since they take all possible series of stepsize changes into account. For an efficient stepsize strategy in practical applications these bounds are of no use. We will address this issue in more detail in Section 5.3.4. It should be noted, however, that the specified bound for order $k = 2$ is also necessary, i.e., the method is unstable for larger stepsize changes.

5.2.4 BDF methods for index 1 DAEs

We consider now the IVP for the linearly implicit DAE (5.2)

$$\begin{aligned}
\mathbf{A}(t,\mathbf{x}(t),\mathbf{z}(t))\,\dot{\mathbf{x}}(t) &= \mathbf{f}(t,\mathbf{x}(t),\mathbf{z}(t)), \quad \mathbf{x}(t_0)=\mathbf{x_0}\in\mathbb{R}^{n_x}, \\
\mathbf{0} &= \mathbf{g}(t,\mathbf{x}(t),\mathbf{z}(t)), \quad \mathbf{z}(t_0)=\mathbf{z_0}\in\mathbb{R}^{n_z}, \\
t &\in [t_0,t_f]\subset\mathbb{R}, \\
\mathbf{x}(t) &\in \mathbb{R}^{n_x}, \\
\mathbf{z}(t) &\in \mathbb{R}^{n_z}, \\
\mathbf{A}(t,\mathbf{x}(t),\mathbf{z}(t)) &\in \mathbb{R}^{n_x\times n_x} \text{ regular} \\
\mathbf{g_z}(t,\mathbf{x}(t),\mathbf{z}(t)) &\in \mathbb{R}^{n_z\times(n_x+n_z)} \text{ regular.}
\end{aligned} \tag{5.14}$$

We choose the so-called indirect approach to transfer the BDF discretization scheme from ODEs to index 1 DAEs. As we assume that $\mathbf{g_z}$ and $\mathbf{A}$ are regular in this case, we can use the implicit function theorem to obtain a unique local representation of the algebraic variables through the differential ones. More specifically, the implicit function theorem assures that there exist in a neighborhood of a solution for all $t \in [t_0, t_f]$ a smooth function $\tilde{\mathbf{g}} : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_z}$ and a locally

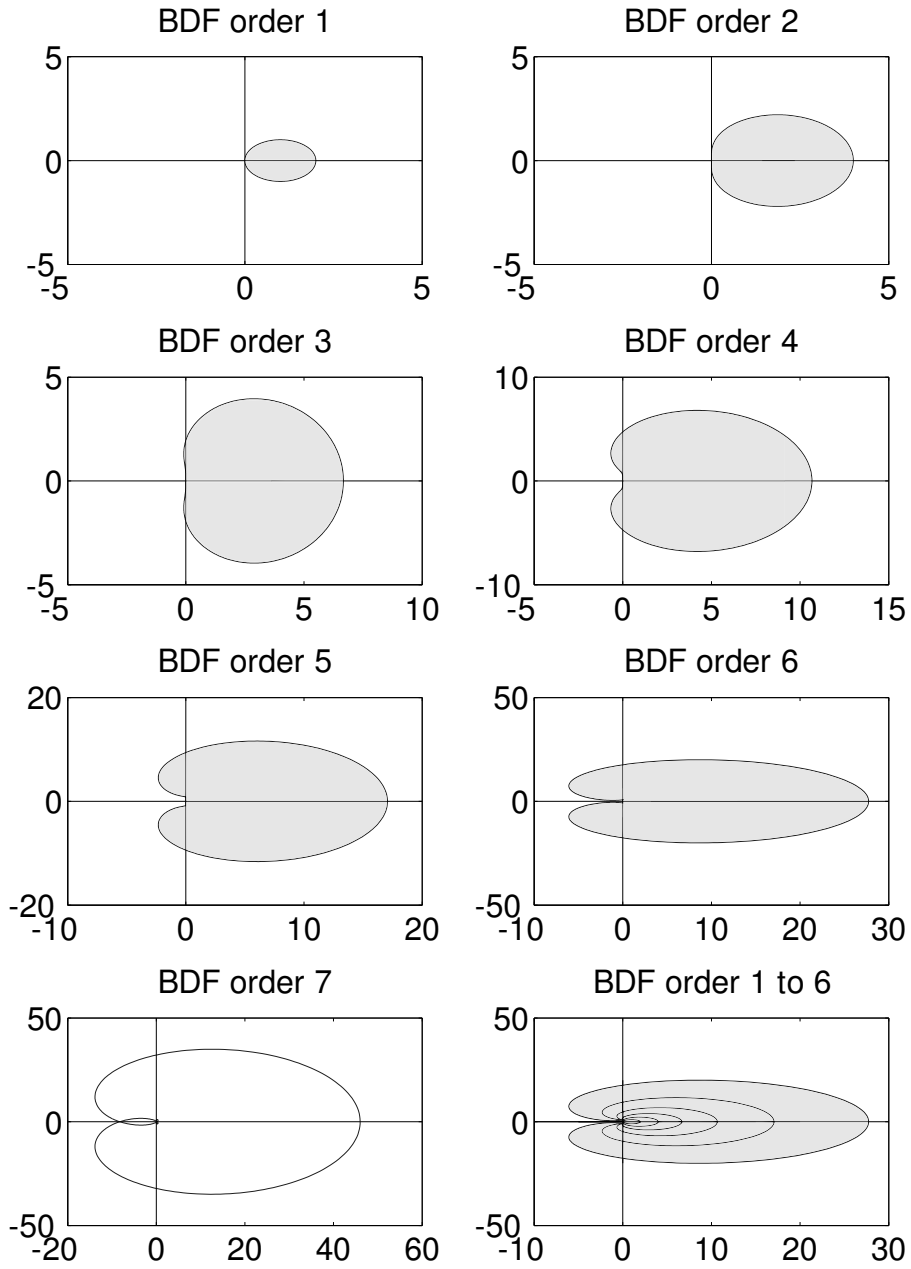

Figure 5.2: The upper three rows show the domains of absolute stability for the BDF methods of order 1 to 6, where the stable area is drawn in white. We see that for order 1 and 2 the methods are A-stable and A(α)-stable with decreasing α up to order 6. The lower row shows on the left the outline the stability region for the method of order 7, which is not zero-stable any more, i.e., zero is not in the stability domain. On the lower right plot a comparison of the stability regions for the orders 1 to 6 is given. In every plot the x-axis corresponds to the real part of $h\lambda$ and the y-axis to the imaginary part of $h\lambda$.

unique solution $\mathbf{z}(t) = \tilde{\mathbf{g}}(\mathbf{x}(t))$ of the algebraic equations.
Inserting this solution into the original problem leads to the IVP

$$\mathbf{A}(t, \mathbf{x}(t), \tilde{\mathbf{g}}(\mathbf{x}(t)))\, \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \tilde{\mathbf{g}}(\mathbf{x}(t))), \quad \mathbf{x}(t_0) = \mathbf{x_0} \in \mathbb{R}^{n_x}.$$

We apply now a BDF method to this IVP and obtain on a variable grid the discretization scheme

$$\begin{aligned} \mathbf{A}(t_{m+k}, \mathbf{x_{m+k}}, \mathbf{z_{m+k}}) \sum_{l=0}^{k} \alpha_{lm} \mathbf{x_{m+l}} &= h_{m+k-1} \mathbf{f}(t_{m+k}, \mathbf{x_{m+k}}, \mathbf{z_{m+k}}) \\ \mathbf{0} &= \mathbf{g}(t_{m+k}, \mathbf{x_{m+k}}, \mathbf{z_{m+k}}), \end{aligned} \tag{5.15}$$

with $m = 0, \ldots, N-k$.
The convergence of BDF methods applied to the IVP (5.14) follows from the following theorem.

Theorem 5.55
Let the LMM (5.13) *be of consistency order p and let 0 be in the stability region. Furthermore, let the start values of the DAE be consistent, i.e., $\mathbf{g}(t_0, \mathbf{x_0}, \mathbf{z_0}) = \mathbf{0}$ and assume that the errors in the start values of the BDF method are of order $\mathcal{O}(h^p)$. Then the LMM is convergent of order p.*

5.3 Strategies used in DAESOL-II

In this section we describe the strategies implemented in our integrator DAESOL-II which is part of our SolvIND integrator suite. We address here only the aspects directly relevant for the IVP solution. The strategies related to sensitivity generation are presented at the end of the corresponding Chapter 6. Note that some of the more technical details concerning the IVP solution strategies can only be roughly sketched in the frame of this thesis. For a more detailed description we refer to [Alb05], as well as to the works of Bleser [Ble86], Eich [Eic87], and Bauer [Bau99] which describe in more detail the ideas already implemented in the preceding Fortran code DAESOL.

5.3.1 Representation of the interpolation polynomials

The core of DAESOL-II is a BDF method for the solution of linearly implicit DAE-IVPs as described in Section 5.2.4. To represent the interpolation polynomials needed for the BDF method we use Newton's representation. This allows an efficient storage as well as an efficient update between integration steps.

Definition 5.56 (Newton's representation)
In Newton's representation *the interpolation polynomial $\mathcal{P}$ through the $(k+1)$ points $\mathbf{v_i} = \mathbf{v}(t_i) \in \mathbb{R}^{n_v}, i = 0, \ldots, k$ at point t is given by*

$$\mathcal{P}(t; \mathbf{v_0}, \ldots, \mathbf{v_k}) = \sum_{i=0}^{k} \mathcal{N}_i(t) \mathbf{v}[t_k, \ldots, t_{k-i}]. \tag{5.16}$$

Here the divided differences $\mathbf{v}[\ldots]$ *are defined recursively by*

$$\begin{aligned} \mathbf{v}[t_i] &:= \mathbf{v}(t_i) \\ \mathbf{v}[t_{i+j}, \ldots, t_i] &:= \frac{\mathbf{v}[t_{i+j}, \ldots, t_{i+1}] - \mathbf{v}[t_{i+j-1}, \ldots, t_i]}{t_{i+j} - t_i} \end{aligned} \tag{5.17}$$

and the Newton polynomials $\mathcal{N}_i$ *as*

$$\mathcal{N}_i(t) = \begin{cases} \prod_{l=0}^{i-1}(t - t_{k-l}) & \text{for } i = 1, \ldots, k \\ 1 & \text{for } i = 0. \end{cases} \tag{5.18}$$

For the implementation of the BDF method in the n-th integration step the method's corresponding interpolation polynomial $\mathcal{P}^{\mathbf{C}}_{\mathbf{n+1}}$ is needed for the computation of $\mathbf{y_{n+1}}$. In a k-step BDF method this polynomial is of degree k and interpolates the $(k+1)$ values $\mathbf{x_{n+1-i}}, i = 0, \ldots, k$ of the differential variables and $\mathcal{P}^{\mathbf{C}}$, respectively its time derivative, satisfies the ODE/DAE at point t_{n+1}. By these conditions $\mathcal{P}^{\mathbf{C}}_{\mathbf{n+1}}$ and its value $\mathbf{x}^{\mathbf{C}}_{\mathbf{n+1}} = \mathcal{P}^{\mathbf{C}}_{\mathbf{n+1}}(t_{n+1})$ are uniquely defined. We call this interpolation polynomial *corrector polynomial* and its value $\mathbf{x}^{\mathbf{C}}_{\mathbf{n+1}}$ at t_{n+1} the *corrector*.
As the equation system in the discretization scheme (5.15) is nonlinear, we solve it iteratively. Therefore a start value for the differential and algebraic variables is needed, which we call *predictor*. We obtain the predictor by use of a second interpolation polynomial $\mathcal{P}^{\mathbf{P}}_{\mathbf{n+1}}$ of degree k through the last $k+1$ values $\mathbf{y_{n+1-i}}, i = 1, \ldots, k+1$ that is called *predictor polynomial*.
We define the following notation to efficiently describe how the interpolation polynomials are calculated and stored as well as how the transition from one integration step to the next is made.

Definition 5.57
We define the factors of Newton's polynomials (5.18) *(at point* $t = t_{n+1}$*)*

$$\psi_i(n+1) := t_{n+1} - t_{n+1-i} = h_n + \ldots + h_{n+1-i} = \psi_{i-1}(n) + h_n, \tag{5.19}$$

the quotients of the new and old $\mathcal{N}_i$*'s*

$$\chi_i(n+1) := \begin{cases} 1 & \text{for } i = 1 \\ \frac{\psi_1(n+1) \cdot \ldots \cdot \psi_{i-1}(n+1)}{\psi_1(n) \cdot \ldots \cdot \psi_{i-1}(n)} & \text{for } i > 1, \end{cases} \tag{5.20}$$

the modified divided differences

$$\mathbf{\Phi_i}(n) := \begin{cases} \mathbf{y_n} & \text{for } i = 1 \\ \psi_1(n) \cdot \ldots \cdot \psi_{i-1}(n)\ \mathbf{y}[t_n, \ldots, t_{n-i+1}] & \text{for } i > 1, \end{cases} \tag{5.21}$$

and furthermore

$$\mathbf{\Phi}^*_{\mathbf{i}}(n) := \chi_i(n+1)\ \mathbf{\Phi_i}(n) \tag{5.22}$$

$$\gamma_i(n+1) := \sum_{j=1}^{i-1} \frac{1}{\psi_j(n+1)} \tag{5.23}$$

$$\boldsymbol{\delta}_{\mathbf{i}}(n+1) := \sum_{j=1}^{i} \mathbf{\Phi}^*_{\mathbf{j}}(n). \tag{5.24}$$

Here empty products should have the value 1, empty sums the value 0.

Now we show how predictor and corrector can be represented using these quantities.

Lemma 5.58 (Representation of the predictor)
For the predictor polynomial in step $n+1$ it holds that

$$\mathcal{P}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1-i}) = \mathbf{y_{n+1-i}},\ i = 1, \ldots, k+1. \tag{5.25}$$

Using Newton's representation (5.16) and the definitions above we obtain the representation

$$\begin{aligned}
\mathbf{y^P_{n+1}} &= \mathcal{P}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1}) \\
&= \sum_{j=0}^{k} \prod_{i=0}^{j-1} (t_{n+1} - t_{n-i})\ \mathbf{y}[t_n, \ldots, t_{n-j}] \\
&= \sum_{j=1}^{k+1} \psi_1(n+1) \ldots \psi_{j-1}(n+1)\ \mathbf{y}[t_n, \ldots, t_{n-j+1}] \\
&= \sum_{j=1}^{k+1} \mathbf{\Phi^*_j}(n) \\
&= \boldsymbol{\delta}_{\mathbf{k+1}}(n+1).
\end{aligned} \tag{5.26}$$

Hence $\boldsymbol{\delta}_{\mathbf{i}}(n+1)$ are the partial sums of the corrector in step $n+1$. Note that in case of DAEs we denote the differential part of the predictor with $\mathbf{x^P_{n+1}}$ and the algebraic part with $\mathbf{z^P_{n+1}}$.

The time derivative $\dot{\mathcal{P}}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1})$ at point t_{n+1} is then obtained as

$$\begin{aligned}
\dot{\mathcal{P}}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1}) &= \sum_{j=0}^{k} \frac{d}{dt} \prod_{i=0}^{j-1} (t - t_{n-i})\ \mathbf{y}[t_n, \ldots, t_{n-j}] \Bigg|_{t=t_{n+1}} \\
&= \sum_{j=0}^{k} \sum_{l=0}^{j-1} \prod_{i=0, i\neq l}^{j-1} (t - t_{n-i})\ \mathbf{y}[t_n, \ldots, t_{n-j}] \Bigg|_{t=t_{n+1}} \\
&= \sum_{j=0}^{k} \sum_{l=0}^{j-1} \frac{1}{\psi_{l+1}(n+1)} \prod_{i=0}^{j-1} \psi_{i+1}(n+1)\ \mathbf{y}[t_n, \ldots, t_{n-j}] \\
&= \sum_{j=0}^{k} \gamma_{j+1}(n+1) \mathbf{\Phi^*_{j+1}}(n) \\
&= \sum_{j=1}^{k+1} \gamma_j(n+1) \mathbf{\Phi^*_j}(n).
\end{aligned} \tag{5.27}$$

Lemma 5.59 (Representation of the corrector derivative)
Both predictor polynomial and corrector polynomial interpolate the k values $\mathbf{x_n}, \ldots, \mathbf{x_{n-k+1}}$ exactly. If we take only the differential part of the predictor into account, we can write the difference of predictor and corrector polynomial as

$$\mathcal{P}^{\mathbf{C}}_{\mathbf{n+1}}(t) - \mathcal{P}^{\mathbf{P}}_{\mathbf{n+1}}(t) = \Delta(t)(\mathbf{x^C_{n+1}} - \mathbf{x^P_{n+1}}). \tag{5.28}$$

As $\Delta(t)$ is the difference of two polynomials of degree k, it is itself a polynomial of degree $\leq k$, that is uniquely defined by the $k+1$ conditions

$$\Delta(t_{n+1-i}) = \begin{cases} 1 & \text{for } i = 0 \\ 0 & \text{for } i = 1, \ldots, k. \end{cases} \tag{5.29}$$

Therefore, we obtain

$$\Delta(t) = \prod_{j=1}^{k} \frac{t - t_{n+1-j}}{t_{n+1} - t_{n+1-j}} \tag{5.30}$$

and by differentiation and evaluation at point t_{n+1}

$$\begin{aligned} \dot{\Delta}(t) &= \frac{d}{dt} \prod_{j=1}^{k} \frac{t - t_{n+1-j}}{t_{n+1} - t_{n+1-j}} \Bigg|_{t=t_{n+1}} \\ &= \sum_{j=1}^{k} \frac{1}{t_{n+1} - t_{n+1-j}} \prod_{i=1, i\neq j}^{k} \frac{t - t_{n+1-i}}{t_{n+1} - t_{n+1-i}} \Bigg|_{t=t_{n+1}} \\ &= \sum_{j=1}^{k} \frac{1}{t_{n+1} - t_{n+1-j}} \\ &= \gamma_{k+1}(n+1). \end{aligned} \tag{5.31}$$

We differentiate (5.28) and obtain finally at t_{n+1} using (5.27) and (5.26)

$$\begin{aligned} \dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} &= \dot{\mathcal{P}}^{\mathbf{C}}_{\mathbf{n+1}}(t_{n+1}) \\ &= \dot{\mathcal{P}}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1}) + \dot{\Delta}(t_{n+1})(\mathbf{x^C_{n+1}} - \mathbf{x^P_{n+1}}) \\ &= \sum_{j=1}^{k+1} \gamma_j(n+1)\mathbf{\Phi^*_j}(n) + \gamma_{k+1}(n+1)(\mathbf{x^C_{n+1}} - \sum_{j=1}^{k+1} \mathbf{\Phi^*_j}(n)) \\ &= \sum_{j=1}^{k+1} (\gamma_j(n+1) - \gamma_{k+1}(n+1))\mathbf{\Phi^*_j}(n) + \gamma_{k+1}(n+1)\mathbf{x^C_{n+1}} \\ &= -\sum_{j=1}^{k} \frac{1}{\psi_j(n+1)} \sum_{i=1}^{j} \mathbf{\Phi^*_i}(n) + \gamma_{k+1}(n+1)\mathbf{x^C_{n+1}} \\ &= -\sum_{j=1}^{k} \frac{1}{\psi_j(n+1)} \boldsymbol{\delta}_{\mathbf{j}}(n+1) + \gamma_{k+1}(n+1)\mathbf{x^C_{n+1}}. \end{aligned} \tag{5.32}$$

Here we again took only the differential part of $\dot{\mathcal{P}}^{\mathbf{P}}_{\mathbf{n+1}}$ and the $\mathbf{\Phi}^*_{\mathbf{i}}(n)$ and $\boldsymbol{\delta}_{\mathbf{i}}(n)$ into account. The term

$$\mathbf{x}^{\mathbf{CC}}_{\mathbf{n+1}} := -\sum_{j=1}^{k} \frac{1}{\psi_j(n+1)} \gamma_j(n+1) \tag{5.33}$$

describing the part of the representation of the corrector derivative that does not depend on the corrector value is called *corrector constant*.

Lemma 5.60 (Update of the modified divided differences)
For given $\mathbf{y_n}$ and $\boldsymbol{\delta}_{\mathbf{i}}(n)$ we obtain $\mathbf{\Phi_{i+1}}(n)$ as

$$\begin{aligned}
\mathbf{\Phi_{i+1}}(n) &= \psi_1(n) \dots \psi_i(n) \mathbf{y}[t_n, \dots, t_{n-i}] \\
&= \psi_1(n) \dots \psi_{i-1}(n) \frac{\psi_i(n)}{t_n - t_{n-i}} (\mathbf{y}[t_n, \dots, t_{n-i+1}] - \mathbf{y}[t_{n-1}, \dots, t_{n-i}]) \\
&= \psi_1(n) \dots \psi_{i-1}(n) (\mathbf{y}[t_n, \dots, t_{n-i+1}] - \mathbf{y}[t_{n-1}, \dots, t_{n-i}]) \\
&= \mathbf{\Phi_i}(n) - \mathbf{\Phi^*_i}(n-1) \\
&= \mathbf{y_n} - \boldsymbol{\delta}_{\mathbf{i}}(n).
\end{aligned} \tag{5.34}$$

Hence we obtain for $\mathbf{\Phi}^*_{\mathbf{i}}(n)$

$$\mathbf{\Phi}^*_{\mathbf{i}}(n) = \chi_i(n+1)(\mathbf{y_n} - \boldsymbol{\delta}_{\mathbf{i-1}}(n)), \tag{5.35}$$

and we can compute the term $\mathbf{\Phi}^*_{\mathbf{i}}(n)$ needed in step n using the $\boldsymbol{\delta}_{\mathbf{i-1}}(n)$ from the last step $(n-1)$ and the last value $\mathbf{y_n}$.

Implementation in the code DAESOL-II

The practical computation of predictor and corrector in step $(n+1)$, based on the step n, is done as follows.

Let $\psi_i(n)$, $\boldsymbol{\delta}_{\mathbf{i}}(n)$ and $\mathbf{y_n}$ be given from step n. Then we compute in the following order

a) the $\psi_i(n+1)$ with (5.19)

$$\psi_i(n+1) = \psi_{i-1}(n) + h_n, \; i = k, \dots, 2, \qquad \psi_1(n+1) = h_n,$$

b) the $\chi_i(n+1)$ using (5.20)

$$\chi_1(n+1) = 1, \qquad \chi_i(n+1) = \chi_{i-1}(n+1) \frac{\psi_{i-1}(n+1)}{\psi_{i-1}(n)}, \; i = 2, \dots, k,$$

c) $\gamma_{k+1}(n+1)$ using (5.23)

$$\gamma_{k+1}(n+1) = \sum_{i=1}^{k} \frac{1}{\psi_i(n+1)},$$

d) the new predictor $\mathbf{y^P_{n+1}}$ with (5.26)

$$\begin{aligned}\mathbf{y^P_{n+1}} &= \sum_{j=1}^{k+1} \mathbf{\Phi^*_j}(n) \\ &= \mathbf{y_n} + \sum_{j=1}^{k} \mathbf{\Phi^*_{j+1}}(n) \\ &= \mathbf{y_n} + \sum_{j=1}^{k} \chi_{j+1}(n+1)[\mathbf{y_n} - \boldsymbol{\delta_j}(n)] \text{ using (5.35)},\end{aligned}$$

e) and the new corrector constant

$$\mathbf{x^{CC}_{n+1}} = -\sum_{j=1}^{k} \frac{1}{\psi_j(n+1)} \boldsymbol{\delta_j}(n+1).$$

The corrector derivative can then be represented as

$$\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} = \gamma_{k+1}(n+1)\mathbf{x^C_{n+1}} + \mathbf{x^{CC}_{n+1}}$$

and for the coefficients α_k of the BDF method it holds that

$$\alpha_k = h_n \gamma_{k+1}(n+1). \tag{5.36}$$

Again, here only the differential part of the $\boldsymbol{\delta_j}(n+1)$ is used.

Remark 5.61
During integration only the last values of $\psi_i(n)$, $\boldsymbol{\delta_i}(n)$ and $\mathbf{y_n}$ need to be stored. Note that the $\boldsymbol{\delta_i}(n+1)$ are stored during the computations under d), i.e., they do not have to be computed separately.

5.3.2 Error estimation

In an efficient adaptive numerical method we want to adapt the stepsize and the order of the method to the actual problem in such a way that the effort for performing the integration becomes minimal, while the error remains below a user given tolerance. To do this, obviously an estimate for the error in each integration step is needed. Based on this estimate we can then decide whether to accept the step, or to reject it and repeat it with an adapted stepsize.
Ideally, we would compute the global error of the method. Unfortunately, this is not easy in the actual practical setup: The global error is the accumulation of the local errors made in the individual steps and the error propagation. Especially the error propagation is not easily

accessible for a quantitative analysis during the normal integration procedure. An approximation of the global error $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}^{\boldsymbol{x}T}, \boldsymbol{\epsilon}^{\boldsymbol{z}T})^T$ in step $(n+1)$ is given in [Bau99] by the formula

$$\begin{pmatrix} \alpha_k \mathbf{A} + \mathbf{A_x}\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} + h\mathbf{f_x} & \mathbf{A_z}\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} + h\mathbf{f_z} \\ \mathbf{g_x} & \mathbf{g_z} \end{pmatrix} \begin{pmatrix} \boldsymbol{\epsilon}^{\mathbf{x}}_{\mathbf{n+1}} \\ \boldsymbol{\epsilon}^{\mathbf{z}}_{\mathbf{n+1}} \end{pmatrix} = \begin{pmatrix} h\boldsymbol{\nu_1} - \mathbf{A}(h\boldsymbol{\sigma}^{\mathbf{x}}_{\mathbf{n+1}} - h\boldsymbol{\epsilon}^{\mathbf{CC}}_{\mathbf{n+1}}) \\ \boldsymbol{\nu_2} \end{pmatrix}. \tag{5.37}$$

Here we summarize in $\boldsymbol{\nu_1}$ and $\boldsymbol{\nu_2}$ the error from premature termination of the iterative method used to solve the nonlinear equation system and the higher orders of the errors $\boldsymbol{\epsilon}^{\mathbf{x}}, \boldsymbol{\epsilon}^{\mathbf{z}}$ and $\boldsymbol{\sigma}$, respectively. These quantities, as well as the corrector constant $\boldsymbol{\epsilon}^{\mathbf{CC}}$, are not directly computable. We will see later at the end of Chapter 6 how an approximation of the global error can be obtained a posteriori, i.e., after the integration, by the use of adjoint sensitivity information. For now we restrict ourselves to an error control based on an estimation of the local discretization error, in the way it is found in [Bau99] or [Eic91] and similar also in [Gea71] and [LP86, PL86].

Estimation of the local error

From Definition 5.22 and the representation of the derivative of the method's interpolation polynomial we obtain in step $(n+1)$ the local discretization error for a k-step BDF method as

$$\begin{aligned} \boldsymbol{\sigma}_{\mathbf{n+1}} := \boldsymbol{\sigma}[\mathbf{y}^{(\mathrm{ex})}(t_{n+1}), h_n] &= \dot{\mathbf{y}}^{(\mathrm{ex})}(t_{n+1}) - \dot{\mathbf{y}}^{\mathbf{C}(\mathrm{ex})}_{\mathbf{n+1}} \\ &= \dot{\mathbf{y}}^{(\mathrm{ex})}(t_{n+1}) - \dot{\mathcal{P}}^{\mathbf{P}(\mathrm{ex})}_{\mathbf{n+1}}(t_{n+1}) \\ &\quad -\gamma_{k+1}(n+1)\left(\mathbf{y}^{(\mathrm{ex})}(t_{n+1}) - \mathcal{P}^{\mathbf{P}(\mathrm{ex})}_{\mathbf{n+1}}(t_{n+1})\right). \end{aligned}$$

Here and in the following the superscript $(\cdot)^{(\mathrm{ex})}$ denotes that the corresponding quantities are computed using the exact solution $\mathbf{y}^{(\mathrm{ex})}(t)$ of the DAE. The definition of the divided differences and the Newton representation of $\mathcal{P}^{\mathbf{P}(\mathrm{ex})}_{\mathbf{n+1}}(t)$ now imply

$$\mathbf{y}^{(\mathrm{ex})}(t) - \mathcal{P}^{\mathbf{P}(\mathrm{ex})}_{\mathbf{n+1}}(t) = \prod_{i=0}^{k}(t - t_{n-i})\ \mathbf{y}^{(\mathrm{ex})}[t, t_n, \ldots, t_{n-k}]. \tag{5.38}$$

By differentiation we obtain

$$\begin{aligned} \dot{\mathbf{y}}^{(\mathrm{ex})}(t) - \dot{\mathcal{P}}^{\mathbf{P}(\mathrm{ex})}_{\mathbf{n+1}}(t) &= \frac{d}{dt}\left(\prod_{i=0}^{k}(t - t_{n-i})\right)\ \mathbf{y}^{(\mathrm{ex})}[t, t_n, \ldots, t_{n-k}] \\ &\quad + \prod_{i=0}^{k}(t - t_{n-i})\ \frac{d}{dt}\mathbf{y}^{(\mathrm{ex})}[t, t_n, \ldots, t_{n-k}]. \end{aligned}$$

Furthermore, we have

$$\frac{d}{dt}\mathbf{y}^{(\mathrm{ex})}[t, t_n, \ldots, t_{n-k}] = \mathbf{y}^{(\mathrm{ex})}[t, t, t_n, \ldots, t_{n-k}],$$

where we define $\mathbf{y}^{(\mathrm{ex})}[t,t] := \dot{\mathbf{y}}^{(\mathrm{ex})}(t)$. Additionally, it holds that

$$\begin{aligned}
\prod_{i=0}^{k}(t_{n+1}-t_{n-i}) &= \prod_{i=1}^{k+1}\psi_i(n+1) \text{ and} \\
\frac{d}{dt}\prod_{i=0}^{k}(t_{n+1}-t_{n-i}) &= \sum_{j=1}^{k+1}\frac{1}{\psi_j(n+1)}\prod_{i=0}^{k}(t_{n+1}-t_{n-i}) \\
&= \gamma_{k+2}(n+1)\prod_{i=1}^{k+1}\psi_i(n+1).
\end{aligned}$$

So we obtain for $\boldsymbol{\sigma_{n+1}}$

$$\begin{aligned}
\boldsymbol{\sigma_{n+1}} &= \gamma_{k+2}(n+1)\prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},\ldots,t_{n-k}] \\
&\quad + \prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},t_{n+1},\ldots,t_{n-k}] \\
&\quad - \gamma_{k+1}(n+1)\prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},\ldots,t_{n-k}] \\
&= (\gamma_{k+2}(n+1)-\gamma_{k+1}(n+1))\prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},\ldots,t_{n-k}] \\
&\quad + \prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},t_{n+1},\ldots,t_{n-k}] \qquad (5.39) \\
&= \left(\sum_{i=1}^{k+1}\frac{1}{\psi_i(n+1)} - \sum_{i=1}^{k}\frac{1}{\psi_i(n+1)}\right)\boldsymbol{\Phi}^{(\mathrm{ex})}_{\mathbf{k+2}}(n+1) \\
&\quad + \prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},t_{n+1},\ldots,t_{n-k}] \\
&= \frac{1}{\psi_{k+1}(n+1)}\boldsymbol{\Phi}^{(\mathrm{ex})}_{\mathbf{k+2}}(n+1) + \prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+1},t_{n+1},\ldots,t_{n-k}].
\end{aligned}$$

The terms that have been computed here using the exact solution are replaced in the practical implementation with the corresponding terms of the numerically approximated solution, as the exact solution is in general not available. Gear showed that this estimation is asymptotically correct for equidistant grids [Gea74].
Similar to the approximation of the global error (5.37), the following approximation of the local error made in one integration step, denoted with $\boldsymbol{\mu}$, can be derived based on the local discretization

error $\boldsymbol{\sigma}$ [Bau99].

$$\begin{pmatrix} \alpha_k \mathbf{A} + \mathbf{A_x}\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} + h\mathbf{f_x} & \mathbf{A_z}\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} + h\mathbf{f_z} \\ \mathbf{g_x} & \mathbf{g_z} \end{pmatrix} \begin{pmatrix} \boldsymbol{\mu}^{\mathbf{x}}_{\mathbf{n+1}} \\ \boldsymbol{\mu}^{\mathbf{z}}_{\mathbf{n+1}} \end{pmatrix} = \begin{pmatrix} -h\mathbf{A}\boldsymbol{\sigma}^{\mathbf{x}}_{\mathbf{n+1}} \\ \mathbf{0} \end{pmatrix}. \tag{5.40}$$

Practical implementation of the error estimation in DAESOL-II

The calculation of the local error (and also of a new stepsize in the framework of stepsize control) using the Formula (5.40) would be computationally very expensive, as in every step the equation system had to be solved. Hence in DAESOL-II the following simplified formula for local error estimation is used:

$$\begin{aligned} E_k(n+1,h_n) &= h_n||\boldsymbol{\sigma}_{\mathbf{n+1}}|| \\ &= h_n\Big|\Big|\frac{1}{\psi_{k+1}(n+1)}\boldsymbol{\Phi}_{\mathbf{k+2}}(n+1) \\ &\quad + \prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}[t_{n+1},t_{n+1},\ldots,t_{n-k}]\Big|\Big| \\ &\doteq h_n\frac{1}{\psi_{k+1}(n+1)}\Big|\Big|\prod_{i=1}^{k+1}\psi_i(n+1)\ \mathbf{y}[t_{n+1},\ldots,t_{n-k}]\Big|\Big| \\ &= h_n\prod_{i=1}^{k}\psi_i(n+1)\ \Big|\Big|\mathbf{y}[t_{n+1},\ldots,t_{n-k}]\Big|\Big|. \end{aligned} \tag{5.41}$$

After every integration step it is checked whether this estimation for $E_k(n+1,h_n)$ is smaller than the user given tolerance. If this is the case, the step is accepted. Otherwise, the step is rejected and repeated with a reduced stepsize. More details on the reduction of the stepsize are given in Section 5.3.4.

5.3.3 Solution of the nonlinear corrector equation

As BDF methods are implicit methods, in every integration step a nonlinear system of equations has to be solved. Because the problems considered in this thesis are usually stiff, we use in DAESOL-II for the solution of these systems a Newton-like method together with a "monitor strategy" that we discuss in the following. As the equation solver is modularized in DAESOL-II, also other equation solving approaches (possibly specifically designed for certain problem classes) may be implemented without too much programming effort.
We write the discretization scheme (5.15) of the BDF method for a linearly implicit DAE at time t_{n+1} as

$$\mathbf{f}^{\mathbf{BDF}}_{\mathbf{n}}(\mathbf{y_{n+1}}) = \mathbf{0}, \quad \text{where again } \mathbf{y_{n+1}} = (\mathbf{x_{n+1}}^T, \mathbf{z_{n+1}}^T)^T. \tag{5.42}$$

Newton's method for the iterative solution of the equation system in step $(n+1)$ is then given by a start value $\mathbf{y}^{(\mathbf{0})}_{\mathbf{n+1}}$ and the iteration

$$\mathbf{y}^{(\mathbf{i+1})}_{\mathbf{n+1}} = \mathbf{y}^{(\mathbf{i})}_{\mathbf{n+1}} + \boldsymbol{\Delta}\mathbf{y}^{(\mathbf{i})}_{\mathbf{n+1}},$$

where $\mathbf{\Delta y^{(i)}_{n+1}}$ is obtained as solution of the linear equation system

$$\mathbf{J_n}(\mathbf{y^{(i)}_{n+1}})\mathbf{\Delta y^{(i)}_{n+1}} = -\mathbf{f_n^{BDF}}(\mathbf{y^{(i)}_{n+1}}). \tag{5.43}$$

Here $\mathbf{J_n}(\mathbf{y^{(i)}_{n+1}}) := \frac{\partial \mathbf{f_n^{BDF}}(\mathbf{y^{(i)}_{n+1}})}{\partial \mathbf{y}}$ denotes the Jacobian of $\mathbf{f_n^{BDF}}$. The Jacobian has the form

$$\mathbf{J_n}(\mathbf{y^{(i)}_{n+1}}) = \begin{pmatrix} \alpha\mathbf{A} + \mathbf{A_x}(\alpha\mathbf{x^{(i)}_{n+1}} + h_n\,\mathbf{x^{CC}_{n+1}}) + h_n\,\mathbf{f_x} & \mathbf{A_z}(\alpha\mathbf{x^{(i)}_{n+1}} + h_n\,\mathbf{x^{CC}_{n+1}}) + h_n\,\mathbf{f_z} \\ \mathbf{g_x} & \mathbf{g_z} \end{pmatrix}, \tag{5.44}$$

where α is the coefficient of the BDF-method that corresponds to $\mathbf{x_{n+1}}$. The start value is obtained as the value $\mathbf{y^{(0)}_{n+1}} := \mathbf{y^P_{n+1}} = \mathcal{P}^{\mathbf{P}}_{\mathbf{n+1}}(t_{n+1})$ of the predictor polynomial at time t_{n+1}.
To solve the equation systems using the original Newton's method one has to construct and decompose the Jacobian in every iteration. Usually this is the most expensive part in the integration. Especially in the case of large ODE/DAE systems and if the evaluation of the derivatives of the model functions $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{A}$ is very costly.
However, often the Jacobian does not change very much from iteration to iteration, or even during several integration steps of the BDF method. This motivates a strategy to reduce the computational effort, where the Jacobian is kept constant as long as possible. This Newton-like method has slightly inferior convergence properties compared to the pure Newton method. As a result, some additional iterations are needed, but this additional effort is in general much smaller than the benefits gained by fewer evaluations and decompositions of the Jacobian.
We present now a monitor strategy for the Jacobians that assures the convergence of the Newton-like method in a few iteration steps while reusing the Jacobian and the model function derivatives as long as possible. This monitor strategy was first presented by Bock and Eich and can be found in [Eic87].
As foundation of the strategy we first analyze the convergence behavior of Newton-like methods, for which Bock [Boc87] proved the following theorem:

Proposition 5.62 (Local contraction theorem)
Let $D \subseteq \mathbb{R}^{n_y}$ be open and $\mathbf{v} : D \longrightarrow \mathbb{R}^{n_y}$ a C^1-function. We denote with $\mathbf{J}(\mathbf{y}) = \frac{\partial \mathbf{v}}{\partial \mathbf{y}}(\mathbf{y})$ the Jacobian of $\mathbf{v}$ and with $\mathbf{M}$ an approximation of the inverse of $\mathbf{J}$.
Assume that for all $\tau \in [0,1]$, for all $\mathbf{y}, \mathbf{y} + \mathbf{\Delta y} \in D$ with $\mathbf{\Delta y} = -\mathbf{M}\,\mathbf{v}(\mathbf{y})$ and for all m there exist ω and κ, such that:

1. *A generalized Lipschitz condition holds:*

$$\frac{\|\mathbf{M}[\mathbf{J}(\mathbf{y^{(m)}} + \tau\mathbf{\Delta y^{(m)}}) - \mathbf{J}(\mathbf{y^{(m)}})]\mathbf{\Delta y^{(m)}}\|}{\tau\|\mathbf{\Delta y^{(m)}}\|^2} \leq \omega^{(m)}, \;\; \omega^{(m)} \leq \omega \leq \infty, \tag{5.45}$$

2. *the quality of the approximated inverse $\mathbf{M}$ in direction of the Newton-increments is sufficient:*

$$\frac{\|\mathbf{M}[\mathbf{v}(\mathbf{y^{(m)}}) - \mathbf{J}(\mathbf{y^{(m)}})\mathbf{\Delta y^{(m)}}]\|}{\|\mathbf{\Delta y^{(m)}}\|} \leq \kappa^{(m)}, \;\; \kappa^{(m)} \leq \kappa < 1, \tag{5.46}$$

3. *the start value* $\mathbf{y^{(0)}}$ *of the iteration satisfies the condition*

$$\delta_0 := \frac{\omega^{(0)}}{2}\|\mathbf{\Delta y^{(0)}}\| + \kappa^{(0)} < 1, \tag{5.47}$$

4. *and the ball* D_0 *with center* $\mathbf{y^{(0)}}$ *and radius* $r = \frac{\|\mathbf{\Delta y^{(0)}}\|}{1-\delta_0}$ *lies in* D.

Then it follows:

1. *The iteration* $\mathbf{y^{(m+1)}} = \mathbf{y^{(m)}} + \mathbf{\Delta y^{(m)}}$, $\mathbf{\Delta y^{(m)}} = -\mathbf{M}\,\mathbf{v}(\mathbf{y^{(m)}})$ *is well-defined and remains in* D_0,

2. *there exists a root* $\mathbf{y}^* \in D_0$, *against which the series* $\mathbf{y^{(m)}}$ *converges,*

3. *the series* $\mathbf{y^{(m)}}$ *converges at least linearly with*

$$\|\mathbf{\Delta y^{(m)}}\| \leq \left(\frac{\omega^{(m-1)}}{2}\|\mathbf{\Delta y^{(m-1)}}\| + \kappa^{(m-1)}\right)\|\mathbf{\Delta y^{(m-1)}}\| \leq \|\mathbf{\Delta y^{(m-1)}}\|$$

4. *and the a priory estimate*

$$\|\mathbf{y^{(m)}} - \mathbf{y}^*\| \leq \frac{\delta_0^m}{1-\delta_0}\|\mathbf{\Delta y^{(0)}}\|$$

holds.

The convergence of the Newton-like method inside the BDF method can then be controlled using an estimate of the convergence rate δ_0. This estimate can be obtained after 2 Newton iterations by

$$\tilde{\delta}_0 := \frac{\|\mathbf{\Delta y^{(1)}}\|}{\|\mathbf{\Delta y^{(0)}}\|} \leq \delta_0.$$

The termination criterion for the Newton-like method is based on the increment norm $\|\mathbf{\Delta y^{(i)}}\|$. If $\|\mathbf{\Delta y^{(i)}}\| \leq c_{\text{newton}} \cdot \texttt{tol}$ then the method is considered as converged. Here $c_{\text{newton}} < 1$ and `tol` is the user given relative tolerance.
Because usually the start value $\mathbf{y^{(0)}} = \mathbf{y^P}$ lies near the solution $\mathbf{y}^*$, the start value should be inside the local convergence region of the Newton-like method. Therefore, with the exception of very nonlinear problems, the aim is not to find a solution at all, but to compute it with the smallest possible effort per integration step. The effort consists of the calculation and decomposition of the Jacobian and of the solution of the linear equation system using the decomposed matrix. To limit the effort connected to the solution of the equation systems we demand a convergence rate that assures convergence after at most three iterations: Two iterations are needed in any case to estimate the convergence rate, and we admit an additional third iteration, if the estimated convergence rate predicts convergence for it.
A survey on the choice of the limit for $\tilde{\delta}_0$ depending on the number of desired iterations m and the desired error improvement factor c_{red} in the estimation $\|\mathbf{y^{(m)}} - \mathbf{y}^*\| \leq c_{\text{red}}\|\mathbf{y^{(0)}} - \mathbf{y}^*\|$ after

m iterations is given by von Schwerin [vS97]. For DAESOL-II we choose $m = 2$ iterations and $c_{\text{red}} = \frac{1}{12}$ and obtain the condition $\delta_0 \leq 0.25$.
This means in practice that we have to test besides the termination criterion also if the estimated convergence rate remains below 0.25 after the second iteration. If this is not the case, i.e., if the convergence rate of the Newton-like method is too bad, this can have several reasons:

a) The quality of the iteration matrix is not good enough, i.e., the κ in the local contraction theorem is too large:
   - The coefficient α of the BDF method or the stepsize h have changed considerably, e.g., through stepsize or order changes.
   - The derivatives of the model functions $\mathbf{f}$, $\mathbf{g}$ or $\mathbf{A}$ have changed considerably.

b) The start value $\mathbf{y}^{(\mathbf{0})}$ is too far away from the solution, e.g., if the problem is "too nonlinear". This means the ω in the local contraction theorem is too large and the start values do not lie in the local convergence region.

Based on these considerations we present the following so-called monitor strategy for the Newton-like iterations, which aims to reduce the effort for the solution of the corrector equation as much as possible.

Implementation of the monitor strategy in DAESOL-II

In DAESOL-II the monitor strategy to control the Newton-like iterations is implemented in the following way:

a) After every iteration the termination criterion is tested. If it is fulfilled the solution of the system is considered successful and is terminated.

b) After two iterations the convergence rate is estimated. If the estimation is smaller than 0.25, a third iteration is admitted, otherwise the iteration is solution as failed and aborted.

c) If after 3 iterations the termination criterion is not fulfilled, the solution is also aborted.

The reuse or the recalculation of the derivatives and the decomposition of the iteration matrix, respectively, is done according to the following hierarchical scheme:

1. As long as the estimate of the convergence rate $\tilde{\delta}_0$ remains below 0.25, the Jacobian approximation is kept constant and its decomposition is reused.

2. If no convergence is achieved, a new Jacobian approximation is built using the actual values of α and h and the "old" derivative values $\mathbf{f_y}$, $\mathbf{g_y}$, and $\mathbf{A_y}\dot{\mathbf{x}}^{\mathbf{C}}$. It is decomposed and the solution is repeated with this new iteration matrix.

3. If again no convergence is achieved, also the derivatives $\mathbf{f_y}$, $\mathbf{g_y}$, and $\mathbf{A_y}\dot{\mathbf{x}}^{\mathbf{C}}$ are computed anew, are stored and the Jacobian approximation is rebuild and decomposed. The solution is repeated.

4. If still no convergence is achieved, the complete BDF-step is repeated using a smaller stepsize. The details of the stepsize reduction strategy are discussed in Section 5.3.4.

Here especially the second step is not used widely in other integrator codes. However if the derivatives of the model functions change only slowly and are expensive to evaluate, this approach promises a large benefit [Eic87].

Stability regions of BDF predictor-corrector scheme

In the previous sections we have presented details on the practical implementation of the predictor-corrector method used in DAESOL-II to solve the nonlinear equations that arise in BDF methods. The focus of the presented monitor strategy is the efficient reduction of the error in the corrector equation, but not necessarily a solution of the equation up to machine precision, which would be unnecessarily expensive. This means in practice that we do not employ a truly implicit method. Although this approach has been applied very successfully during the last decades and similar approaches are used in other codes based on BDF methods, this gives rise to the question how the stability regions of the method are influenced by this. For Adams-PECE methods there have been some works ([Cha62],[CK65],[Kro66],[Ste68]) that show that the stability regions are seriously reduced compared to the implicit schemes and that propose adapted schemes to improve stability. For BDF methods the only existing analysis is given by Krogh and Steward [KS84], analyzing the special case of asymptotic absolute stability of BDF predictor-corrector methods for the case $h \to \infty$. Hence we present in the following a heuristic analysis of the regions of absolute stability that is more specifically tailored to our setup of BDF based predictor-corrector schemes.
We perform the analysis as it is usual on an equidistant grid with stepsize h for the test equation of Dahlquist (5.11). Other than in the analysis of the implicit method, we have to imitate now the predictor-corrector scheme and substitute the formula for the predictor value into the corrector equation and perform a number of Newton-like iterations. In the framework of this analysis we assume a fixed number of s Newton-like iterations per integration step, and the quality of our Jacobian approximation is described by the value of κ in the local contraction theorem (5.46).
We demonstrate this now at the example of the method with order 2. The predictor value for $\mathbf{x}^{\mathbf{P}}_{\mathbf{n+1}}$ is then obtained as

$$\mathbf{x}^{\mathbf{P}}_{\mathbf{n+1}} = 3\mathbf{x_n} - 3\mathbf{x_{n-1}} + \mathbf{x_{n-2}}. \tag{5.48}$$

The predictor value is then inserted into the corresponding corrector equation

$$\frac{3}{2}\mathbf{x}^{\mathbf{C}}_{\mathbf{n+1}} - 2\mathbf{x_n} + \frac{1}{2}\mathbf{x_{n-1}} - (h\lambda)\mathbf{x}^{\mathbf{C}}_{\mathbf{n+1}} \stackrel{!}{=} \mathbf{0} \tag{5.49}$$

and we apply a Newton-like iteration with the Jacobian approximation

$$\tilde{\mathbf{J}}(\kappa,\mu) = \frac{3}{2} - \mu + \frac{1}{2}\frac{\kappa\,(-3+2\,\mu)}{-1+\kappa}, \tag{5.50}$$

where we have set $\mu := h\lambda$. This leads then to the first iterate

$$\frac{\mathbf{x_{n-1}} - 4\,\mathbf{x_{n-2}} + \kappa(8\,\mathbf{x_{n-1}} - 5\,\mathbf{x_{n-2}} - 3\,\mathbf{x_n}) + \kappa\mu(2\,\mathbf{x_n} - 6\,\mathbf{x_{n-1}} + 6\,\mathbf{x_{n-2}})}{-3+2\,\mu} \tag{5.51}$$

from which we obtain the characteristic equation for the method with order 2 and one Newton-like iteration

$$\frac{-3\,z^3 + 2\,\mu\,z^3 - z + 4\,z^2 - 8\,z\kappa + 5\,z^2\kappa + 3\,\kappa - 2\,\kappa\,\mu + 6\,\mu\,z\kappa - 6\,z^2\mu\,\kappa}{z^2\,(-3 + 2\,\mu)} = 0. \tag{5.52}$$

Using Definition 5.35 of the stability region we can determine the root locus curve, i.e., the potential bound of the stability region, for a given κ by solving this equation for each $z = e^{i\theta}$ with $0 \leq \theta < 2\pi$. The same analysis can be repeated using a higher number of Newton-like iterations to obtain a series of stability regions for a given Jacobian quality κ. The Figures 5.3, 5.4 and 5.5 show the stability regions for the predictor-corrector methods of order 1,2 respectively 5 using between 1 and 4 Newton-like iterations for the different values $\kappa_1 = \frac{1}{4}$, $\kappa_2 = -\frac{1}{4}$, $\kappa_3 = \frac{i}{4}$ and $\kappa_4 = \frac{1}{\sqrt{32}} + \frac{i}{\sqrt{32}}$ of the left-hand term of (5.46), such that always $|\kappa_i| = \frac{1}{4}$ holds.
This is motivated by the constraint in our monitor strategy to allow only Jacobian approximations with $|\kappa| < \frac{1}{4}$ (as $\omega = 0$ in this case), i.e., the chosen values represent in this sense examples for the worst case of the admitted Jacobians in our strategy. In general it can be observed that the stability regions become with increasing number of Newton-like iterations more and more similar to the regions of the implicit methods. Not surprisingly this occurs the faster, the smaller the absolute value of κ is. Interesting for our setup is that even for examples of the worst case of κ the method of order 1 remains A-stable and the method of order 2 preserves A-stability if at least 2 iterations are made. The higher-order methods might require, especially in the case of a complex κ, several Newton-like iterations to obtain reasonable stability regions similar to their implicit counterparts. But considering that in practice most of the time Jacobian approximation with far better values for κ are used, we can reasonably expect that our predictor-corrector scheme will in general not be affected by a serious loss of stability. Otherwise, more Newton-like iterations should be enforced in the monitor strategy to assure better stability properties.

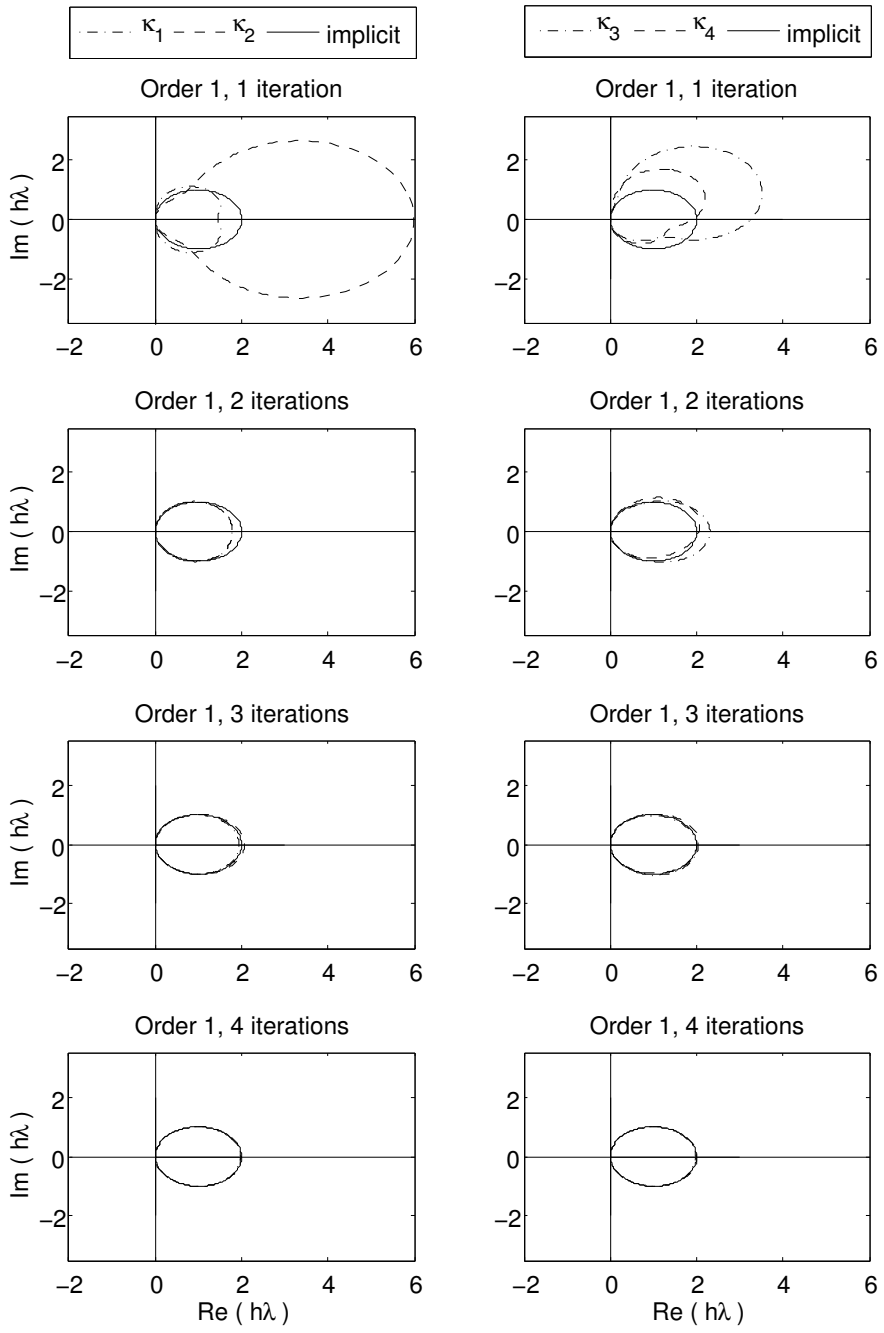

Figure 5.3: Depicted are the borders of the domains of absolute stability for the predictor-corrector BDF schemes (dashed and dashed-dotted lines) and for the truly implicit BDF scheme (solid lines) of order 1 for different Jacobian approximations and different numbers of Newton-like iterations between 1 and 4. The values of κ for the Jacobian approximations are $\kappa_1 = \frac{1}{4}$, $\kappa_2 = -\frac{1}{4}$, $\kappa_3 = \frac{i}{4}$ and $\kappa_4 = \frac{1}{\sqrt{32}} + \frac{i}{\sqrt{32}}$. We see that for order 1 the predictor-corrector scheme remains A-stable like the implicit method, and if using 2 or more Newton-like iterations the stability regions of the predictor-corrector scheme and the implicit method very much coincide.

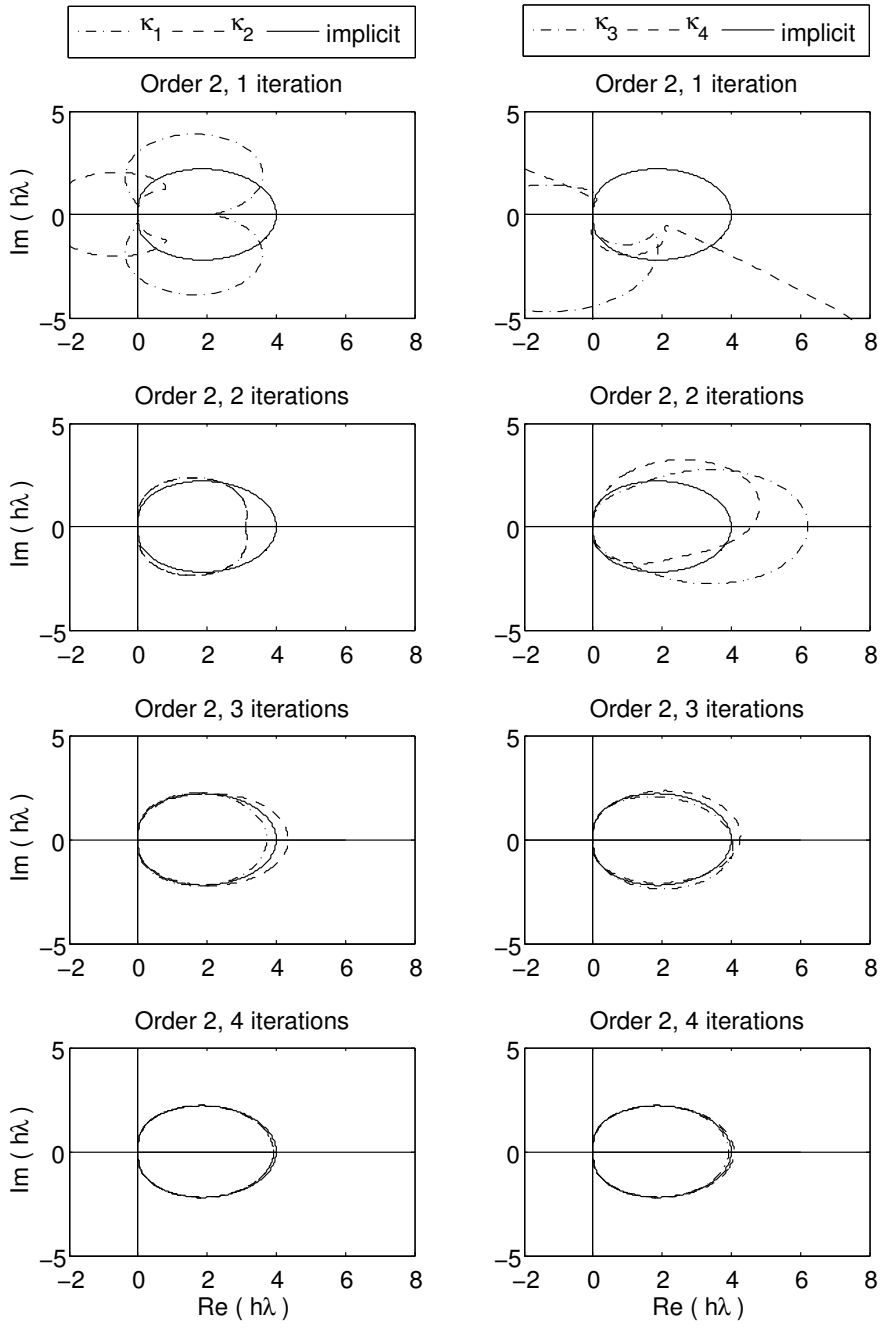

Figure 5.4: Depicted are the borders of the domains of absolute stability for the predictor-corrector BDF schemes (dashed and dashed-dotted lines) and for the truly implicit BDF scheme (solid lines) of order 2 for different Jacobian approximations and different numbers of Newton-like iterations between 1 and 4. The values of κ for the Jacobian approximations are $\kappa_1 = \frac{1}{4}$, $\kappa_2 = -\frac{1}{4}$, $\kappa_3 = \frac{i}{4}$ and $\kappa_4 = \frac{1}{\sqrt{32}} + \frac{i}{\sqrt{32}}$. We see that for order 2 the predictor-corrector scheme remains A-stable if 2 or more Newton-like iterations are applied. For more than 2 iterations the stability regions of the predictor-corrector scheme and the implicit method are nearly the same.

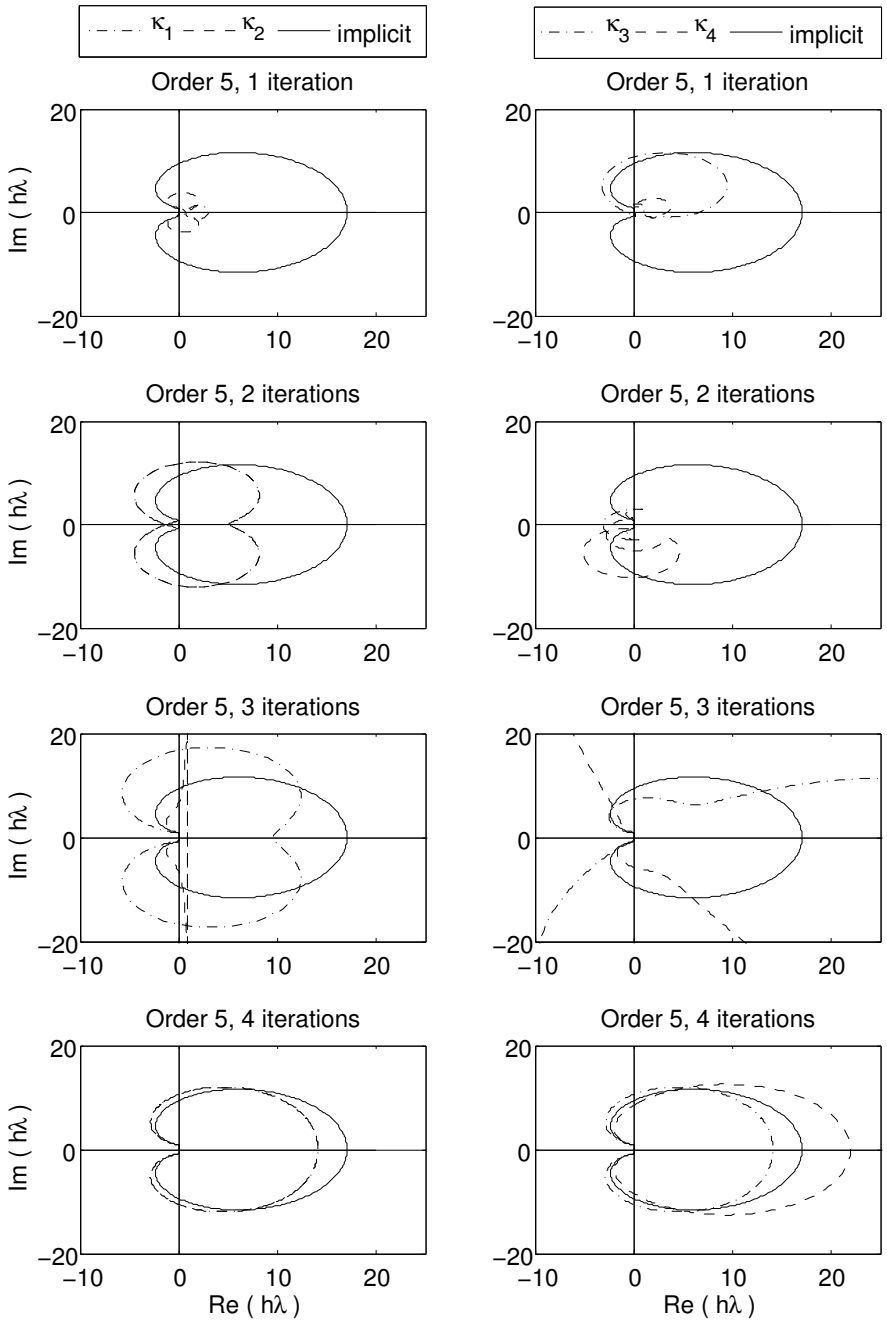

Figure 5.5: Depicted are the borders of the domains of absolute stability for the predictor-corrector BDF schemes (dashed and dashed-dotted lines) and for the truly implicit BDF scheme (solid lines) of order 5 for different Jacobian approximations and different numbers of Newton-like iterations between 1 and 4. The values of κ for the Jacobian approximations are $\kappa_1 = \frac{1}{4}$, $\kappa_2 = -\frac{1}{4}$, $\kappa_3 = \frac{i}{4}$ and $\kappa_4 = \frac{1}{\sqrt{32}} + \frac{i}{\sqrt{32}}$. We see that for order 5 the predictor-corrector is unstable if less than 3 Newton-like iterations are applied. For complex values of κ this is also the case if 3 iterations are made. For 4 iterations the stability regions of the predictor-corrector scheme and the implicit method are similar.

5.3.4 Stepsize and order control

In this section we describe the strategy for the stepsize and order control used in `DAESOL-II`. Its aim is to plan the next integration step with a stepsize as large as possible while keeping the occurring local error below a prescribed tolerance `tol` such that the step will be accepted.

Estimation of h_{n+1}

Assume for the following that step $(n+1)$ has been accepted. To determine a new stepsize h_{n+1} for the next step, we need an estimate for the local error in step $(n+2)$, assuming that the new step is performed with an order of k and the stepsize h_{n+1}. Based on formula (5.39) for the local discretization error $\boldsymbol{\sigma_{n+1}}$ we obtain for $\boldsymbol{\sigma_{n+2}}$

$$\begin{aligned}\boldsymbol{\sigma_{n+2}} &= \prod_{i=1}^{k} \psi_i(n+2)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+2},\ldots,t_{n-k+1}] \\ &\quad + \prod_{i=1}^{k+1} \psi_i(n+2)\ \mathbf{y}^{(\mathrm{ex})}[t_{n+2},t_{n+2},\ldots,t_{n-k+1}],\end{aligned}$$

and hence the estimation for the local error

$$\begin{aligned}E_k(n+2,h_{n+1}) &= h_{n+1}\|\boldsymbol{\sigma_{n+1}}\| \\ &\doteq h_{n+1}\prod_{i=1}^{k}\psi_i(n+2)\Big\|\mathbf{y}[t_{n+2},\ldots,t_{n-k+1}]\Big\|. \qquad (5.53)\end{aligned}$$

Here, the still unknown divided difference can be estimated conservatively by [BSS94]

$$\begin{aligned}\Big\|\mathbf{y}[t_{n+2},\ldots,t_{n-k+1}]\Big\| &\le \Big\|\mathbf{y}[t_{n+1},\ldots,t_{n-k}]\Big\| \\ &\quad +\psi_{k+2}(n+2)\Big\|\mathbf{y}[t_{n+1},\ldots,t_{n-k-1}]\Big\|. \qquad (5.54)\end{aligned}$$

We end up with the condition for h_{n+1}

$$\begin{aligned}E_k(n+2,h_{n+1}) &\doteq h_{n+1}\prod_{i=1}^{k}\psi_i(n+2)\cdot\Big(\big\|\mathbf{y}[t_{n+1},\ldots,t_{n-k}]\big\| \\ &\qquad +\psi_{k+2}(n+2)\big\|\mathbf{y}[t_{n+1},\ldots,t_{n-k-1}]\big\|\Big) \\ &\le c\cdot\mathtt{tol}, \qquad (5.55)\end{aligned}$$

where $c \le 1$ is a safety factor, e.g., $c = 0.5$. Because this inequality depends polynomially on h_{n+1}, it is difficult to obtain a suitable h_{n+1} directly. Therefore we first compute the largest admittable stepsize for the given tolerance on an equidistant grid as an approximation and reduce it, if needed, according to the inequality above.

On an equidistant grid we obtain for a step with order k the estimation

$$\hat{E}_k(n+2,h) = k!\, h^{k+1} \|\mathbf{y}[t_{n+1},\ldots,t_{n-k}]\|.$$

Demanding $\hat{E}_k(n+2,h) \leq c \cdot \texttt{tol}$ this leads to the so-called *maximal uniform stepsize* [Ble86]:

$$\hat{h}_{\hat{k}} = \sqrt[\hat{k}+1]{\frac{c \cdot \texttt{tol}}{\hat{k}!\ \|\mathbf{y}[t_{n+1},\ldots,t_{n-\hat{k}}]\|}}. \tag{5.56}$$

$\hat{h}_{\hat{k}}$ is computed for the actual integration order $\hat{k} = k$, as well as for $\hat{k} = k-1$ and $\hat{k} = k+1$. The values are compared and the order is changed for the next step, if for the new order the stepsize would be significantly larger. Finally $\hat{h}_{k^*}$ is checked against the error estimation formula

$$E_{k^*}(n+2,\hat{h}_{k^*}) \leq c \cdot \texttt{tol}$$

on variable grid. If this check is passed, the stepsize is accepted, and the next step is planned using $h_{n+1} = \hat{h}_{\hat{k}}$. Otherwise the step stepsize is adjusted using (5.55). For that, we use the formula

$$h_{k^*}^2 = \frac{c \cdot \texttt{tol}}{q(\hat{h}_{k^*})\ (\|\mathbf{y}[t_{n+1},\ldots,t_{n-k^*}]\| + (\hat{h}_{k^*} + \psi_{k+1}(n+1))\|\mathbf{y}[t_{n+1},\ldots,t_{n-k-1}]\|)}, \tag{5.57}$$

where

$$q(h) := \prod_{i=1}^{k^*-1} (h + \psi_i(n+1))$$

is monotonically increasing in h.
As $h_{k^*} < \hat{h}_{k^*}$ holds, it follows $h_{k^*}^2 q(h_{k^*}) < h_{k^*}^2 q(\hat{h}_{k^*})$ and h_{k^*} fulfills

$$E_{k^*}(n+2,h_{k^*}) \leq c \cdot \texttt{tol}$$

and is used as stepsize for the next step.

This strategy performs order and stepsize control based on the actual variable grid and allows a flexible adaption of stepsizes and orders. As Bleser [Ble86] and later Eich [Eic91] have demonstrated, this strategy is very efficient. It reduces the number of step rejections, allows to switch the order of the steps faster and leads to a more stable integration procedure compared to ordinary stepsize strategies based on uniform grids.

Remark 5.63 (Stepsize changes and stability)
As presented in Section 5.2.2, stability of BDF methods on variable grids can without further assumptions only be proven for very limited stepsize changes. These strict bounds prohibit especially for higher orders an efficient stepsize control. As a consequence, investigations were made to establish less strict bounds on the allowed stepsize changes under some additional assumptions. Calvo et al. [CLM87] computed for BDF methods on pseudo-equidistant grids (Nordsieck-methods, [Nor62]) bounds for stepsize changes that guarantee stability. Here the stepsize is kept constant

for some time after a stepsize change, which allows more relaxed stepsize bounds. The bounds become the more relaxed the longer stepsize is kept constant. Gear and Tu [GT74] proved under the assumption of continuous stepsize changes less strict bounds on the allowed stepsize changes than Griegorieff in [Gri83].
The stepsize control in `DAESOL-II` aims for continuous stepsize changes and avoids all-too large jumps in the stepsizes. Therefore the narrow bounds from theorem 5.53 do not need to be fulfilled strictly in practice. We also can allow in `DAESOL-II` stepsize changes after every integration step.

Remark 5.64 (Order changes and stability)
Like stepsize changes, also changes of the order of the BDF method can lead to stability problems during integration. Strictly speaking, an order change is in the end a change of the used LMM. We take care of this point in `DAESOL-II` by freezing the order for a certain amount of steps after an order change, like proposed by Gear and Wanatabe [GW74]. Depending on the last order and if the order was previously increased or decreased, the order is kept constant for between one and four integration steps.

Stepsize reduction after step rejections

The analysis given above considered the estimation of a new stepsize for the case that the actual step has been accepted. The question remains how to choose the stepsize for the repetition of the actual step $(n+1)$, if it was rejected. Here we distinguish between the two possible reasons for a step rejection in `DAESOL-II`:

a) The estimated error $E_k(n+1, h_n)$ is too large.

b) The Newton-like method does not converge in three steps despite rebuild and decomposition of the Jacobian.

In case a) we compute a new stepsize for the step repetition from the error formula (5.55) on variable grid by

$$h_n^{(\text{new})} = h_n^{(\text{old})} \sqrt{\frac{c \cdot \texttt{tol}}{E_k(n+1, h_n^{(\text{old})})}}. \tag{5.58}$$

In case b) the stepsize is reduced implicitly by imposing a stricter tolerance. Here the new stepsize shall be chosen in a way, that for the expected new convergence rate $\delta_0^{\text{new}} \leq 0.25$ holds. This corresponds to a predicted convergence of the Newton-like method in two steps [Enk84].
Because the step rejection is the last action in the monitor strategy, the actual Jacobian including up-to-date model derivatives has already been computed and decomposed. Therefore in the quality criterion for the iteration matrix in the local contraction theorem (5.46) $\kappa = 0$ holds.
An estimation of $\omega^{(0)}$ based on the quantities computed in the rejected step is obtained using (5.47) as:

$$\omega^{(0)} \approx 2 \frac{\delta_0^{(\text{rejected})}}{\|\boldsymbol{\Delta}\mathbf{y}_{\mathbf{n+1}(\text{rejected})}^{(\mathbf{0})}\|}. \tag{5.59}$$

Then the condition

$$\delta_0^{(\text{new})} = \frac{\omega^{(0)}}{2} \|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}(\text{new})}\| \leq \frac{1}{4}$$

can be transformed into

$$\|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}(\text{new})}\| \leq \frac{\|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}(\text{rejected})}\|}{4\ \delta_0^{(\text{rejected})}}. \tag{5.60}$$

Considering the estimation of the local error, then with

$$\|\mathbf{y^C_{n+1}} - \mathbf{y^P_{n+1}}\| = \prod_{i=1}^{k+1} \psi_i(n+1) \|\mathbf{y}[t_{n+1}, \ldots, t_{n-k}]\|$$

and the approximation

$$\|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}}\| \approx \|\mathbf{y^C_{n+1}} - \mathbf{y^P_{n+1}}\|$$

we obtain using (5.41) the approximation

$$E_k(n+1, h) = \frac{h}{\psi_{k+1}(n+1)} \|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}}\|. \tag{5.61}$$

If we now tighten `tol`, such that

$$\overline{\texttt{tol}} = \frac{h}{\psi_{k+1}(n+1)} \frac{\|\mathbf{\Delta y}^{(\mathbf{0})}_{\mathbf{n+1}(\text{rejected})}\|}{4\delta_0^{(\text{rejected})}} \tag{5.62}$$

and therefore demand in the stepsize determination that

$$E_k(n+1, h_n^{(\text{new})}) \leq c \cdot \overline{\texttt{tol}}$$

we finally accomplish

$$\delta_0^{(\text{new})} \leq \frac{1}{4}.$$

5.3.5 Start-up of the BDF method

A k-step BDF method needs at least k "historical" values. At the begin of the integration of an IVP only the initial value $\mathbf{y_0} = (\mathbf{x_0}^T, \mathbf{z_0}^T)^T$ is available. There exist several possibilities to start the integration.

In a *self starting method* we start with a one-step BDF method and increase the order subsequently according to the stepsize and order control. A disadvantage of this approach might be, that in the beginning due to the low order often very small steps have to be made. Possibly the solution might be less accurate and errors that are made at the begin of the integration might dominate the overall error of the method for some time (cf. [Bau99]). Additionally, several rebuilds and decompositions of the iteration matrix are to be expected during the process of increasing the stepsize and order.

Another possibility would be to make a small integration step backwards using a one-step method, and to start with a two-step method afterwards.
Finally, a one-step method of higher order could be used to produce several start values to start afterwards with a LMM of higher order. Gear [Gea80] used for the first time a Runge-Kutta-method to start a LMM. Von Schwerin and Bock [vSB95] used an explicit Runge-Kutta-method (RK) as a starter for an Adams-method. Other than in the approach of Gear, in this method some of the internal stages are of higher order, such that with one step of the Runge-Kutta-method all needed start values can be produced.
For BDF methods Bauer [Bau99] developed an implicit RK-method that follows the approach of Bock and von Schwerin: Several internal stages are of higher order. Hence with one step of the RK-method, the start values for a fourth order BDF method can be produced. Furthermore this method is designed in such a way that the iteration matrix of the RK-method can be reused in the BDF method.

In `DAESOL-II` there are currently two of these approaches implemented. The RK-starter approach of Bauer and the self starting approach. As in the first integration step of a self starting method there are also not enough historical values available to build the predictor polynomial, here an explicit Euler step is used to obtain the predictor value. The starter strategy of the integration is also modularized, such that a later extension with additional approaches is possible. An efficient start-up strategy is especially important in the case of discontinuities of the model functions, or their lower order derivatives, because in all these points the integration has to be stopped and restarted. Also in the treatment of problems where the dynamic model switches depending on the current state as well as during the solution of delay differential equations often integration restarts are needed.

5.3.6 Computation of consistent initial values

To solve an initial value problem for DAEs, consistent initial values are needed. In an actual application these might not be known exactly, e.g., because they are not measurable directly. In the fully-implicit setup consistency means that $\mathbf{b}(t_0, \mathbf{y_0}, \dot{\mathbf{y}}_\mathbf{0}) = \mathbf{0}$ holds. In the linearly implicit index 1 case the differential variables $\mathbf{x_0}$ are free, and the algebraic variables $\mathbf{z_0}$ are implicitly defined due to the regularity of $\mathbf{g_z}$ and the consistency condition $\mathbf{g}(t_0, \mathbf{x_0}, \mathbf{z_0}) = \mathbf{0}$.
If the integration would be started with non-consistent initial values, after one step a consistent point would be reached, provided the iterative Newton-like method for the corrector equation has converged. But in this case problems with the error estimation and the quality of the computed "solution" would arise, because in the first step $h_0 \to 0$ does not imply any more that the error tends to 0, as a non-vanishing contribution of the inconsistency remains.
For a linearly implicit index 1 DAE-IVP the generation of initial values is equivalent to finding the root of the algebraic equations $\mathbf{g}(t_0, \mathbf{x_0}, \mathbf{z})$ with only $\mathbf{z}$ as variables.
A simple approach to solve this problem implemented in `DAESOL-II` is a full-step Newton method. The drawback here is that convergence is only achieved, if the initial estimates for the algebraic variables lie in the local convergence region (see also 5.62) of the Newton method.
For highly nonlinear problems or very bad initial estimates it is therefore possible that no consistent

algebraic start value can be found. In this case globally convergent root finding methods would be needed, like, e.g., damped Newton or homotopy methods, which are described in [BSS94] or [Bau99]. As the routines for the consistent initialization are implemented as modules, some of these more advanced approaches can be added later to DAESOL-II if needed.

5.3.7 Relaxed formulation of the algebraic equations

For the iterative solution of an optimization problem involving a DAE model it is often more convenient (and also more efficient) to allow the solution of DAE-IVPs with originally inconsistent initial values. For example, if the algebraic variables depend on free variables that are to be optimized, the consistency conditions will usually be violated after every optimization iteration. In this case a relaxed formulation of the algebraic variables is of advantage [BES88], where the optimization algorithm has to ensure consistency in the solution of the optimization problem. This can be achieved for example by adding the algebraic equations in the start points of the IVPs as equality constraints to the optimization problem, as, e.g., done in [Lei99]. The relaxed formulation leads to an DAE-IVP of the type

$$\begin{aligned} \mathbf{A}(t,\mathbf{x},\mathbf{z})\dot{\mathbf{x}} &= \mathbf{f}(t,\mathbf{x},\mathbf{z}) \\ \mathbf{0} &= \mathbf{g}(t,\mathbf{x},\mathbf{z}) - \theta(t)\mathbf{g}(t_0,\mathbf{x_0},\mathbf{z_0}), \\ \mathbf{x}(t_0) &= \mathbf{x_0}, \\ \mathbf{z}(t_0) &= \mathbf{z_0}. \end{aligned} \tag{5.63}$$

Here we assume that the damping function $\theta : \mathbb{R} \to \mathbb{R}$ is sufficiently smooth, that $\theta(t_0) = 1$, $\theta(t) \geq 0$ and that $\theta(t)$ is monotonically decreasing. This formulation means that we force the initial values to be consistent by a slight change of the constraint manifold of the DAE and hence of the actual problem we solve. This allows us now in the optimization context to start the integration with arbitrary initial values. And if we finally use at some point of the optimization "truly" consistent initial values, the relaxing term vanished and we solve again our original problem.

The standard choice in DAESOL-II is the damping function

$$\theta(t) = e^{-p_{\text{relax}}\left(\frac{t-t_0}{t_{\text{end}}-t_0}\right)},$$

where p_{relax} may be chosen by the user and is per default set to $p_{\text{relax}} = 5$.

Note that if the damping is sufficiently strong and the integration horizon is performed over a long horizon, the use of $\mathbf{f} \equiv \mathbf{0}$ and the "normal" $\mathbf{g}$ provides another way to compute consistent initial values.

5.3.8 Scaling

When solving problems numerically, it is necessary to take into account the possibly different order of magnitudes of different components of the variables. Hence in the strategies for error

estimation and stepsize control a weighted norm is used instead of the plain Euclidean l_2-norm $\|y\|_2$. It is calculated by the formula

$$\left\|\mathbf{y_{n+1}}\right\| = \frac{1}{n_y}\sqrt{\sum_{i=1}^{n_y}\left(\frac{y_{n+1,i}}{v_{\mathrm{scal}_{n+1},i}}\right)^2}, \tag{5.64}$$

where $\mathbf{v}_{\mathrm{scal}} \in \mathbb{R}^{n_y}$ is the scaling vector containing the different scaling factors and the i denotes the i-th component of the corresponding vector.
Up to now, in `DAESOL-II` are four different scaling methods implemented which differ in the choice of the update strategy of the scaling vector:

1. The DASSL scaling [BCP96] which is also used in many integrator code besides DASSL:
$$v_{\mathrm{scal}_{n+1},i} = |y_{n+1,i}| + \mathtt{atol}_i/\mathtt{tol},\ i = 1,\ldots,n_y.$$
2. Gear scaling:
$$v_{\mathrm{scal}_{n+1},i} = \max\{|y_{n+1,i}|, v_{\mathrm{scal}_n,i}\},\ i = 1,\ldots,n_y.$$
3. Old Deuflhard scaling:
$$v_{\mathrm{scal}_{n+1},i} = \max\{|y_{n+1,i}|, v_{\mathrm{scal}_n,i}, c_{\min}\},\ i = 1,\ldots,n_y,$$
with $c_{\min} = \max_{i=1}^{n_y} y_{n+1,i} \cdot \epsilon_{\mathrm{mach}} \cdot \frac{100}{\mathtt{tol}}$.
4. New Deuflhard scaling which is used in the LIMEX code:
$$v_{\mathrm{scal}_{n+1},i} = \max\{|y_{n+1,i}|, v_{\mathrm{scal}_n,i}, \mathtt{atol}_i\},\ i = 1,\ldots,n_y.$$

We use here $\mathbf{v}_{\mathrm{scal}_0} = \mathbf{0}$, ϵ_{mach} is the machine precision, `tol` the user given relative tolerance and `atol` a user given absolute tolerance.
With the exception of the DASSL scaling the scaling vector depends in every step on its last value in order to take the maximal absolute values of the components into account. If one component becomes very small during integration but is still significant for the computation, then the DASSL scaling should be preferred.
Different orders of magnitude in the different components can be taken into account by a corresponding choice of the components of the `atol` vector. In the DASSL scaling `atol` has the meaning of an absolute error tolerance, in the new Deuflhard scaling it corresponds to a scaling factor.

5.3.9 Continuous representation of the solutions

The polynomial interpolation of the latest computed trajectory values, which is the foundation of the BDF methods, provides a simple and efficient possibility to generate a continuous representation of the solution. This allows us to decouple the output grid of the solution from the actual integration grid and to evaluate the trajectories error-controlled at arbitrary points that are not contained in the integration grid. This representation is also the foundation for extensions of the integrator code for the treatment of problems with implicitly defined switching points, where the model

equations change or non-differentiabilities occur (cf. [Kir06],[Ehr05],[BP04], [Eic91],[GO84]). Also for the treatment of delay differential equations this representation plays a crucial role. In the general case, where the delays are state-dependent, historical trajectory values are needed at points that usually do not belong to the discretization grid and that are not known in advance.

We obtain the continuous representation of the solution in the interval $[t_n, t_{n+1}]$ from the interpolation polynomial $\mathcal{P}^{\mathbf{I}}_{\mathbf{n+1}}(t)$ that interpolates the values $\mathbf{y_{n+1}}, \ldots, \mathbf{y_{n-k}}$ exactly. This means that it is identical with the corrector polynomial of the actually completed step. If no order change occurs, it is also identical with the predictor polynomial of the next step. Hence we get

$$\begin{aligned} \mathcal{P}^{\mathbf{I}}_{\mathbf{n+1}}(t) &= \sum_{j=0}^{k} \prod_{i=0}^{j-1} (t - t_{n+1-i})\ \mathbf{y}[t_{n+1}, \ldots, t_{n+1-j}] \\ &= \mathcal{P}^{\mathbf{C}}_{\mathbf{n+1}}(t) \\ &= \mathcal{P}^{\mathbf{P}}_{\mathbf{n+2}}(t) \text{ (if order remains the same).} \end{aligned} \tag{5.65}$$

It can be shown that this continuous interpolation of the solution fulfills the conditions of "natural interpolation" [BS81] at least on equidistant grids. This means that, asymptotically, the interpolation error from the evaluation of $\mathcal{P}^{\mathbf{I}}$ is smaller than the discretization error of the BDF method at the gridpoints.

The output of solutions in DAESOL-II is based on this representation and uses a plug-in system implemented in the SolvIND interface. The user can pass a self-written plug-in to the integrator, defining different actions for different kinds of events, as well as an arbitrary output grid, for which interpolated values are generated by the integrator and passed back to the plug-in.

6 Sensitivity generation

In this chapter we present efficient strategies and algorithms for the numerical computation of sensitivities, i.e., the derivatives of solutions of Initial Value Problems (IVPs) for stiff Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs) of index 1 with respect to initial values and/or parameter. The efficient and accurate computation of sensitivities is of interest, e.g., in the analysis of dynamic systems or in model reduction strategies. Furthermore, it is an essential part of all derivative based algorithms for the optimization of dynamic systems such as Gauss-Newton and SQP methods. In these algorithms the sensitivity generation is usually the most time consuming task, especially for large scale systems.

The type of required sensitivities depends on the application context. While normally for ordinary Gauss-Newton-type methods, e.g., for parameter estimation, first order forward sensitivities are sufficient, Gauss-Newton approaches for robust optimization and exact-Hessian SQP methods as presented in this thesis need at least second order sensitivity information. For optimal experimental design even (directional) sensitivities of third order might be of interest. Inexact SQP approaches, on the other hand, need adjoint sensitivity information to be efficient. In the following we present numerical schemes for the computation of all these types of sensitivities. For the derivation of these schemes we use the idea of Internal Numerical Differentiation (IND), invented by Bock [Boc81, Boc83], which leads to sensitivity generation schemes strongly intertwined with the numerical schemes used for the solution of the corresponding nominal IVPs. While the presented first order forward schemes are well-known, we present new adjoint based IND schemes for LMMs and the first schemes for arbitrary order sensitivity generation at all. Furthermore, we give the first approach for the propagation of higher-order directional sensitivities across switching events.

This chapter is organized as follows. First, we recall in Section 6.1 the definition of the sensitivity generation problem. Then, we describe in Section 6.2 how sensitivities could be obtained analytically as solution of variational DAEs. Afterwards, in Section 6.3, we introduce the idea of IND, before in Section 6.4 IND-based schemes for first order sensitivity generation are given. In Section 6.5 we present numerical schemes for the computation of sensitivities of arbitrary order. We compare numerical effort and memory usage of the presented strategies in Section 6.6, before we finally address in Section 6.7 the strategies implemented in `DAESOL-II` that are related to sensitivity generation.

6.1 Problem formulation

In this chapter we consider the computation of sensitivities, i.e., the computation of the derivatives of the solutions of relaxed parameter dependent initial value problems for linearly implicit index 1 DAEs

$$\mathbf{A}(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p})\, \dot{\mathbf{x}}(\tau) - \mathbf{f}(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p}) = \mathbf{0}, \qquad \mathbf{x}(\tau_0) = \mathbf{x_0}, \tag{6.1a}$$

$$\mathbf{g}(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p}) - \theta(\tau)\, \mathbf{g}(\tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) = \mathbf{0}, \qquad \mathbf{z}(\tau_0) = \mathbf{z_0}, \tag{6.1b}$$

with respect to the initial values $\mathbf{x_0}$, $\mathbf{z_0}$ and the parameter $\mathbf{p}$. In this formulation we assume to be in the direct multiple shooting context such that $\tau \in [\tau_0, \tau_e] \subset [0, 1]$ is already the normalized time and $\mathbf{p} = (\mathbf{p_q}^T, \mathbf{p_p}^T, \mathbf{p_h}^T)^T \in \mathbb{R}^{n_q+n_p+n_h}$ a set of parameter combining the parameter of a possible control parametrization, the system parameter as well as the stage lengths in case of a multistage formulation. We also assume here that the initial values do not depend on the parameter vector $\mathbf{p}$. If this should be the case, the corresponding sensitivity information with respect to $\mathbf{p}$ can be easily obtained by application of the chain rule.

Wronski matrices

We denote by

$$\mathcal{W}(\tau; \tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) := \frac{\mathrm{d}\mathbf{y}(\tau; \tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p})}{\mathrm{d}(\mathbf{x_0}, \mathbf{z_0}, \mathbf{p})} \in \mathbb{R}^{(n_x+n_z)\times(n_x+n_z+n_p)} \tag{6.2}$$

the (first order) sensitivities of the DAE-IVP solution $\mathbf{y}(\tau) = (\mathbf{x}(\tau)^T, \mathbf{z}(\tau)^T)^T$ at time τ for the initial time τ_0, initial values $\mathbf{x_0}$, $\mathbf{z_0}$ and parameter $\mathbf{p}$. These sensitivity matrices are called Wronski matrices. To describe the submatrices of the Wronski matrix we define (omitting the arguments)

$$\mathcal{W} := \begin{pmatrix} \mathcal{W}^{\mathbf{x}} \\ \mathcal{W}^{\mathbf{z}} \end{pmatrix} := \begin{pmatrix} \mathcal{W}^{\mathbf{x}}_{\mathbf{y_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{p}} \\ \mathcal{W}^{\mathbf{z}}_{\mathbf{y_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{p}} \end{pmatrix} := \begin{pmatrix} \mathcal{W}^{\mathbf{x}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{p}} \\ \mathcal{W}^{\mathbf{z}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{p}} \end{pmatrix}. \tag{6.3}$$

In the following we present strategies to compute approximations for the Wronski matrices $\mathcal{W}$ as well as for matrix-vector and vector-matrix products between direction vectors and $\mathcal{W}$. This leads in one case to so-called forward sensitivities $\mathcal{W}^{\mathbf{D}}$ with a matrix of forward directions $\mathbf{D} \in \mathbb{R}^{(n_x+n_z+n_p)\times n_{\mathrm{fwdDir}}}$

$$\mathcal{W}^{\mathbf{D}} = \mathcal{W} \cdot \mathbf{D} \in \mathbb{R}^{(n_x+n_z)\times n_{\mathrm{fwdDir}}} \tag{6.4}$$

and in the other case to adjoint sensitivities $\bar{\mathcal{W}}^{\bar{\mathbf{Y}}}$ with a matrix of adjoint directions $\bar{\mathbf{Y}} \in \mathbb{R}^{(n_x+n_z)\times n_{\mathrm{adjDir}}}$

$$\bar{\mathcal{W}}^{\bar{\mathbf{Y}}} = \mathcal{W}^T \cdot \bar{\mathbf{Y}} \in \mathbb{R}^{(n_x+n_z+n_p)\times n_{\mathrm{adjDir}}}. \tag{6.5}$$

For a more convenient notation we omit here and in the following the sensitivity of the DAE solution with respect to the initial time τ_0, as it is usually not needed for the algorithms presented in this thesis. However, the derivations and strategies presented in this chapter can be transferred with at most small modifications to this case, leading to schemes for the generation of sensitivities with respect to the initial time.

6.2 Variational DAEs

We now discuss how the sensitivities can be described in an analytical way. Provided that the model functions $\mathbf{A}$, $\mathbf{f}$ and $\mathbf{g}$ are smooth enough, the solution of the so-called *nominal* IVP (6.1) is differentiable with respect to the initial time τ_0, the initial values $\mathbf{x_0}$, $\mathbf{z_0}$ and the parameter $\mathbf{p}$. We can thus derive for the nominal DAE-IVP the corresponding variational DAE-IVPs to compute sensitivities.

Forward variational DAE

The simple formal differentiation of the solution of the nominal IVP leads to an IVP for the sensitivities, also called forward variational DAE, which has the form (omitting the arguments)

$$\mathbf{A}\dot{\mathcal{W}}^{\mathbf{x}} = \begin{pmatrix} \mathbf{f_x} - \mathbf{A_x} & \mathbf{f_z} - \mathbf{A_z}\dot{\mathbf{x}} & \mathbf{f_p} - \mathbf{A_p}\dot{\mathbf{x}} \end{pmatrix} \begin{pmatrix} \mathcal{W}^{\mathbf{x}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{p}} \\ \mathcal{W}^{\mathbf{z}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{p}} \\ \mathbf{0} & \mathbf{0} & \mathbb{I}_{n_p} \end{pmatrix} \tag{6.6a}$$

$$\mathbf{0} = \begin{pmatrix} \mathbf{g_x} & \mathbf{g_z} & \mathbf{g_p} \end{pmatrix} \begin{pmatrix} \mathcal{W}^{\mathbf{x}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{x}}_{\mathbf{p}} \\ \mathcal{W}^{\mathbf{z}}_{\mathbf{x_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{z_0}} & \mathcal{W}^{\mathbf{z}}_{\mathbf{p}} \\ \mathbf{0} & \mathbf{0} & \mathbb{I}_{n_p} \end{pmatrix} - \theta \cdot \begin{pmatrix} \mathbf{g}_{\mathbf{x},0} & \mathbf{g}_{\mathbf{z},0} & \mathbf{g}_{\mathbf{p},0} \end{pmatrix} \tag{6.6b}$$

with initial values

$$\mathcal{W}(\tau_0; \tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) = \begin{pmatrix} \mathbb{I}_{n_x} & 0 & 0 \\ 0 & \mathbb{I}_{n_z} & 0 \end{pmatrix}. \tag{6.6c}$$

Here the index zero at the $\mathbf{g}$ terms indicates the evaluation at the initial time and for the initial values. Alternatively, the variational DAE can be formulated directly for a matrix of directions

$$\mathbf{D} = \begin{pmatrix} \mathbf{D_x} \\ \mathbf{D_z} \\ \mathbf{D_p} \end{pmatrix} \in \mathbb{R}^{(n_x+n_z+n_p)\times n_{\text{fwdDir}}} \tag{6.7}$$

to compute the forward sensitivities $\mathcal{W}^{\mathbf{D}}$ via the IVP

$$\mathbf{A}\dot{\mathcal{W}}^{\mathbf{D},\mathbf{x}} = \begin{pmatrix} \mathbf{f_x} - \mathbf{A_x}\dot{\mathbf{x}} & \mathbf{f_z} - \mathbf{A_z}\dot{\mathbf{x}} & \mathbf{f_p} - \mathbf{A_p}\dot{\mathbf{x}} \end{pmatrix} \begin{pmatrix} \mathcal{W}^{\mathbf{D},\mathbf{x}} \\ \mathcal{W}^{\mathbf{D},\mathbf{z}} \\ \mathbf{D_p} \end{pmatrix} \tag{6.8a}$$

$$\mathbf{0} = \begin{pmatrix} \mathbf{g_x} & \mathbf{g_z} & \mathbf{g_p} \end{pmatrix} \begin{pmatrix} \mathcal{W}^{\mathbf{D},\mathbf{x}} \\ \mathcal{W}^{\mathbf{D},\mathbf{z}} \\ \mathbf{D_p} \end{pmatrix} - \theta \cdot \begin{pmatrix} \mathbf{g}_{\mathbf{x},0} & \mathbf{g}_{\mathbf{z},0} & \mathbf{g}_{\mathbf{p},0} \end{pmatrix} \cdot \mathbf{D} \tag{6.8b}$$

with initial values

$$\mathcal{W}^{\mathbf{D}}(\tau_0; \tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) = \begin{pmatrix} \mathbf{D_x} \\ \mathbf{D_z} \end{pmatrix}. \tag{6.8c}$$

Note that both versions for the forward sensitivity DAEs are linear and obviously have the same stiffness properties as the nominal IVP. Furthermore, for the solution of the variational IVP we

need the solution of the nominal IVP, such that the solution of both systems should be performed simultaneously in practice. In principle every numerical integration code suitable for the solution of the nominal IVP could be used in a black-box manner to solve the combined systems as well. However, because of the close relation between nominal and variational IVP, there is a lot of structure that should be exploited in the solution process. The algorithmic schemes based on IND presented later in this chapter exploit this structure automatically by construction, while this is not the case for commonly used integration routines, where special modifications have to be made.

Adjoint variational DAE

An alternative possibility to compute the sensitivities is based on the solution of the adjoint variational DAE which we derive in the following for the relaxed index 1 DAE-IVP case. We do this similar to the derivation that Cao et al. [CLPS03] give for the case of a non relaxed fully-implicit DAE. For this derivation we assume that $\varphi(\mathbf{x}(\tau_\mathrm{e}), \mathbf{z}(\tau_\mathrm{e}))$ is a smooth scalar function that depends on the values of the solution of the DAE-IVP (6.1) at the final time. Examples for φ could be a component of the solution or a linear combination of them. Based on the derivative of φ with respect to $\mathbf{x_0}$, $\mathbf{z_0}$ and $\mathbf{p}$ we can obtain adjoint sensitivities of type (6.5) via suitable choices of φ. Consider now the function

$$\begin{aligned}\chi\big(\mathbf{x}(\tau_\mathrm{e};\mathbf{x_0},\mathbf{z_0},\mathbf{p}),\mathbf{z}(\tau_\mathrm{e};\mathbf{x_0},\mathbf{z_0},\mathbf{p})\big) &= \varphi\big(\mathbf{x}(\tau_\mathrm{e};\mathbf{x_0},\mathbf{z_0},\mathbf{p}),\mathbf{z}(\tau_\mathrm{e};\mathbf{x_0},\mathbf{z_0},\mathbf{p})\big)\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T \underbrace{\left[\mathbf{A}(\tau,\mathbf{x}(\tau),\mathbf{z}(\tau),\mathbf{p})\dot{\mathbf{x}}(\tau) - \mathbf{f}(\tau,\mathbf{x}(\tau),\mathbf{z}(\tau),\mathbf{p})\right]}_{\equiv\mathbf{0}} d\tau\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T \underbrace{\left[\mathbf{g}(\tau,\mathbf{x}(\tau),\mathbf{z}(\tau),\mathbf{p}) - \theta(\tau)\mathbf{g}(\tau_0,\mathbf{x_0},\mathbf{z_0},\mathbf{p})\right]}_{\equiv\mathbf{0}} d\tau.\end{aligned} \tag{6.9}$$

We then obtain for the derivative of φ with respect to $\mathbf{y_0} = \begin{pmatrix}\mathbf{x_0}\\ \mathbf{z_0}\end{pmatrix}$ (only displaying the time arguments)

$$\begin{aligned}\frac{\mathrm{d}\varphi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{y_0}} &= \frac{\mathrm{d}\chi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{y_0}}\\ &= \varphi_{\mathbf{x}}(\tau_\mathrm{e})\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}} + \varphi_{\mathbf{z}}(\tau_\mathrm{e})\frac{\mathrm{d}\mathbf{z}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}}\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T \left[\mathbf{A}(\tau)\frac{\mathrm{d}\dot{\mathbf{x}}(\tau)}{\mathrm{d}\mathbf{y_0}} + (\mathbf{A}_{\mathbf{y}}(\tau)\dot{\mathbf{x}}(\tau))\frac{\mathrm{d}\mathbf{y}(\tau)}{\mathrm{d}\mathbf{y_0}} - \mathbf{f}_{\mathbf{y}}(\tau)\frac{\mathrm{d}\mathbf{y}(\tau)}{\mathrm{d}\mathbf{y_0}}\right] d\tau\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T \left[\mathbf{g}_{\mathbf{y}}(\tau)\frac{\mathrm{d}\mathbf{y}(\tau)}{\mathrm{d}\mathbf{y_0}} - \theta(\tau)\mathbf{g}_{\mathbf{y}}(\tau_0)\right] d\tau.\end{aligned} \tag{6.10}$$

Integration by parts leads to

$$\int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T\mathbf{A}(\tau)\frac{\mathrm{d}\dot{\mathbf{x}}(\tau)}{\mathrm{d}\mathbf{y_0}}d\tau = \left[\boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T\mathbf{A}(\tau)\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\mathbf{y_0}}\right]\Bigg|_{\tau_0}^{\tau_\mathrm{e}} - \int_{\tau_0}^{\tau_\mathrm{e}} \frac{\mathrm{d}(\boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T\mathbf{A}(\tau))}{\mathrm{d}\tau}\frac{\mathrm{d}\mathbf{x}(\tau)}{\mathrm{d}\mathbf{y_0}}d\tau. \tag{6.11}$$

Together with (6.10) this results in

$$\begin{aligned}\frac{\mathrm{d}\varphi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{y_0}} &= \left[\varphi_{\mathbf{x}}(\tau_\mathrm{e}) - \boldsymbol{\lambda}^{\mathbf{x}}(\tau_\mathrm{e})^T\mathbf{A}(\tau_\mathrm{e})\right]\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}} + \varphi_{\mathbf{z}}(\tau_\mathrm{e})\frac{\mathrm{d}\mathbf{z}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}} + \boldsymbol{\lambda}^{\mathbf{x}}(\tau_0)^T\mathbf{A}(\tau_0)\begin{pmatrix}\mathbb{I}_{n_x} & \mathbf{0}\end{pmatrix}\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \underbrace{\left[\begin{pmatrix}-\frac{\mathrm{d}(\boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T\mathbf{A}(\tau))}{\mathrm{d}\tau} & \mathbf{0}\end{pmatrix} + \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T(\mathbf{A}_{\mathbf{y}}(\tau)\dot{\mathbf{x}}(\tau)) - \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T\mathbf{f}_{\mathbf{y}}(\tau)\right]}_{=:a_1}\frac{\mathrm{d}\mathbf{y}(\tau)}{\mathrm{d}\mathbf{y_0}}d\tau\\ &\quad - \int_{\tau_0}^{\tau_\mathrm{e}} \underbrace{\left[\boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T\mathbf{g}_{\mathbf{y}}(\tau)\right]}_{=:a_2}\frac{\mathrm{d}\mathbf{y}(\tau)}{\mathrm{d}\mathbf{y_0}}d\tau + \int_{\tau_0}^{\tau_\mathrm{e}}\theta(\tau)\boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T\mathbf{g}_{\mathbf{y}}(\tau_0)d\tau. \end{aligned} \tag{6.12}$$

We require now $a_1 + a_2 \equiv \mathbf{0}$ which leads to the adjoint DAE system

$$\mathbf{A}(\tau)^T\dot{\boldsymbol{\lambda}}^{\mathbf{x}}(\tau) = (\mathbf{A}_{\mathbf{x}}(\tau)\dot{\mathbf{x}}(\tau) - \mathbf{f}_{\mathbf{x}}(\tau))^T\boldsymbol{\lambda}^{\mathbf{x}}(\tau) + \mathbf{g}_{\mathbf{x}}(\tau)^T\boldsymbol{\lambda}^{\mathbf{z}}(\tau) - \mathbf{A}_t(\tau)^T\boldsymbol{\lambda}^{\mathbf{x}}(\tau) \tag{6.13a}$$

$$\mathbf{0} = (\mathbf{A}_{\mathbf{z}}(\tau)\dot{\mathbf{x}}(\tau) - \mathbf{f}_{\mathbf{z}}(\tau))^T\boldsymbol{\lambda}^{\mathbf{x}}(\tau) + \mathbf{g}_{\mathbf{z}}(\tau)^T\boldsymbol{\lambda}^{\mathbf{z}}(\tau). \tag{6.13b}$$

Furthermore, we can use the implicit function theorem on the relaxed algebraic equations at the final time to express $\frac{\mathrm{d}\mathbf{z}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}}$ in terms of $\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}}$ and the relaxation as

$$\frac{\mathrm{d}\mathbf{z}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}} = -\mathbf{g}_{\mathbf{z}}^{-1}(\tau_\mathrm{e})\left(\mathbf{g}_{\mathbf{x}}(\tau_\mathrm{e})\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}} - \theta(\tau_\mathrm{e})\mathbf{g}_{\mathbf{y}}(\tau_0)\right). \tag{6.14}$$

Then, Equation (6.12) for the derivative becomes

$$\begin{aligned}\frac{\mathrm{d}\varphi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{y_0}} &= \boldsymbol{\lambda}^{\mathbf{x}}(\tau_0)^T\mathbf{A}(\tau_0)\begin{pmatrix}\mathbb{I}_{n_x} & \mathbf{0}\end{pmatrix}\\ &\quad + \underbrace{\left[\varphi_{\mathbf{x}}(\tau_\mathrm{e}) - \boldsymbol{\lambda}^{\mathbf{x}}(\tau_\mathrm{e})^T\mathbf{A}(\tau_\mathrm{e}) - \varphi_{\mathbf{z}}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{z}}^{-1}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{x}}(\tau_\mathrm{e})\right]}_{=:b}\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{y_0}}\\ &\quad + \theta(\tau_\mathrm{e})\varphi_{\mathbf{z}}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{z}}^{-1}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{y}}(\tau_0) + \int_{\tau_0}^{\tau_\mathrm{e}}\theta(\tau)\boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T\mathbf{g}_{\mathbf{y}}(\tau_0)d\tau. \end{aligned} \tag{6.15}$$

Demanding $b \equiv 0$ defines the initial values for the adjoint DAE system (6.13)

$$\mathbf{A}(\tau_\mathrm{e})^T\boldsymbol{\lambda}^{\mathbf{x}}(\tau_\mathrm{e}) = \varphi_{\mathbf{x}}(\tau_\mathrm{e})^T - \mathbf{g}_{\mathbf{x}}(\tau_\mathrm{e})^T\mathbf{g}_{\mathbf{z}}(\tau_\mathrm{e})^{-T}\varphi_{\mathbf{z}}(\tau_\mathrm{e})^T. \tag{6.16}$$

Solving the adjoint DAE system (6.13) with these initial values backwards in time from τ_e to τ_0 allows us to express the derivative as

$$\begin{aligned}\frac{\mathrm{d}\varphi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{y_0}} &= \boldsymbol{\lambda}^{\mathbf{x}}(\tau_0)^T\mathbf{A}(\tau_0)\begin{pmatrix}\mathbb{I}_{n_x} & \mathbf{0}\end{pmatrix}\\ &\quad + \theta(\tau_\mathrm{e})\varphi_{\mathbf{z}}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{z}}^{-1}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{y}}(\tau_0) + \int_{\tau_0}^{\tau_\mathrm{e}}\theta(\tau)\boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T\mathbf{g}_{\mathbf{y}}(\tau_0)d\tau, \end{aligned} \tag{6.17}$$

where $\boldsymbol{\lambda}^{\mathbf{x}}(\tau_0)$ is the solution of the adjoint DAE. The integral term can be integrated along with the solution of the adjoint DAE. Analogously, the derivative of φ with respect to the parameter $\mathbf{p}$ can be derived, leading to the expression

$$\begin{aligned}\frac{\mathrm{d}\varphi(\mathbf{x},\mathbf{z})}{\mathrm{d}\mathbf{p}} &= -\int_{\tau_0}^{\tau_\mathrm{e}} \boldsymbol{\lambda}^{\mathbf{x}}(\tau)^T((\mathbf{A}_{\mathbf{p}}(\tau)\dot{\mathbf{x}}(\tau) - \mathbf{f}_{\mathbf{p}}(\tau)) + \boldsymbol{\lambda}^{\mathbf{z}}(\tau)^T(\mathbf{g}_{\mathbf{p}}(\tau) - \theta(\tau)\mathbf{g}_{\mathbf{p}}(\tau_0))d\tau \\ &\quad -\varphi_{\mathbf{z}}(\tau_\mathrm{e})\mathbf{g}_{\mathbf{z}}^{-1}(\tau_\mathrm{e})(\mathbf{g}_{\mathbf{p}}(\tau_\mathrm{e}) - \theta(\tau_\mathrm{e})\mathbf{g}_{\mathbf{p}}(\tau_0)). \qquad (6.18)\end{aligned}$$

Note that also in this case the integral part can be computed along with the solution of the adjoint DAE-IVP, such that all derivatives of a function φ can be computed by a single solution of the adjoint variational DAE. The adjoint variational DAE is, like the forward variational DAE, linear. If we solve now the adjoint variational DAEs for a sequence of φ corresponding to the single components of the DAE solution, we can compute the full Wronskian matrix (6.3) with $(n_x + n_z)$ solutions of the adjoint variational DAE. These are fewer solutions than needed in the case of using the forward variational DAE (6.6), which requires $(n_x + n_z + n_p)$ solutions. Note also that only n_{adjDir} solutions of the adjoint variational DAE are needed to compute an adjoint sensitivity of type (6.5).
A drawback of the adjoint approach is that the nominal IVP and the adjoint variational IVP cannot be solved simultaneously, but in general the solution values of the nominal IVP are nevertheless needed for the solution of the adjoint IVP. Hence a nominal integration has to be performed first to obtain a representation of the nominal solution and afterwards the adjoint IVP can be solved. In codes not based on IND, such as CVODES and IDAS of the SUNDIALS [HBG+05] suite (and all other codes capable of adjoint sensitivity computation in the DASSL family tree), the needed nominal solution values are usually obtained by interpolation based on the solution values on the discretization grid of the nominal IVP solution. This is necessary in these codes, since the discretization grids for the nominal IVP and the adjoint IVP in general do not coincide. This increases the computational effort and possibly introduces further numerical errors. As we will see later, the IND-based adjoint schemes developed in this thesis can reuse here more information that already has been computed during solution of the nominal IVP.

Higher-order sensitivities

For the computation of higher-order sensitivities theoretically the differentiation procedure leading to the forward variational DAE could be iterated, resulting in forward variational DAEs for the computation of higher-order sensitivities. For second order sensitivities for DAEs this is described, e.g., in [Bau99, Kör02]. For higher orders it becomes even more tedious and error-prone to perform this process. A combined forward/adjoint approach inspired by AD techniques to obtain a variational ODE for reduced second order sensitivities is given in [OB05]. The numerical solution of this variational DAE, however, needs also a specially tailored ODE solver.

After defining forward and adjoint sensitivities and describing the variational DAE-IVPs that are commonly used for their approximation, we now describe the principle of IND and how it can be used to obtain efficient schemes for sensitivity generation.

6.3 The principle of internal numerical differentiation

In this section we describe how, based on a numerical scheme for the solution of the nominal problem, a scheme for the numerical approximation of the sensitivities can be derived. The simplest approach, using a given integration code to obtain sensitivity information, treats the integrator as black box. It calculates finite differences after solving the nominal IVP for the original initial values and parameter and once again for slightly perturbed initial values and/or perturbed parameter. This approach is called external numerical differentiation (END). Although very easy to implement, the END suffers from the fact that the output of an adaptive integrator usually does not depend continuously on the input: Jumps in the range of the integration tolerance can always occur for different sets of initial values and parameter, e.g., due to a change in the stepsize and order strategy. Therefore, the number of significant digits in the solution of the IVP has to be approximately twice as high as the needed accuracy of the derivatives. This leads to a very high and often unacceptable numerical effort.
The idea of Internal Numerical Differentiation (IND) [Boc81, Boc83] is to freeze the adaptive components of the integrator and to differentiate not the whole adaptive integrator code, but the adaptively generated discretization scheme (fixing the used stepsizes, orders, iteration matrices and number of Newton-like iterations). This scheme can be interpreted as a sequence of differentiable mappings, each leading from the solution at one timepoint of the discretization grid via intermediate values to the next. Hence it can be differentiated, for example, using finite differences, the complex step method or the techniques of automatic differentiation. This leads to numerical schemes for the computation of the sensitivities that are strongly intertwined with the computation of the nominal solution.

In fact, by using the principle of IND we are not only able to reuse a lot of information already computed during the solution of the nominal problem, but we also obtain, together with the approximation of the analytical sensitivities, the exact derivative of the numerical solution of the nominal problem. This is a very desirable property of IND-based schemes that most of the other available codes do not share. They often use only the same numerical method for the nominal and the variational IVP, but not the same discretization scheme, even when they exploit some parts of the problem structure. This may lead to problems in adaptive optimization algorithms relying on the sensitivity information, e.g., when for lower integration tolerances the behavior of the numerical nominal solution predicted by the sensitivities does not correspond to its actual behavior.

Following the idea of IND, we now decompose the integration scheme generated adaptively during the numerical solution of the nominal problem into a sequence of differentiable "elementary" mappings. Using the derivatives of these elementary mappings and applying the chain rule, we can derive numerical schemes for sensitivity computation using the AD techniques presented in Chapter 2. Depending on the used approach we will obtain IND schemes for the approximation of first order forward or adjoint sensitivities as well as of higher-order sensitivities. Although the presentation in the following is given mainly in the context of BDF methods, the presented strategies, including the new adjoint and the arbitrary order schemes, can be immediately transferred to other implicit LMMs and even easier to explicit LMMs and Runge-Kutta methods.

For the derivation of the IND schemes we first recall from Section 5.3.3 the computations made during an (accepted) step for the computation of $\mathbf{y_{n+1}}$. We denote the stepsize of this step with h_n, the BDF order with k_n, the iteration matrix with $\mathbf{M_n}$ and the number of performed Newton-like iterations with s_n.
In a single BDF step, first the predictor value $\mathbf{y^P_{n+1}}$ is computed by the interpolation polynomial through the last $k_n + 1$ values, extrapolated at time t_{n+1}, resulting in

$$\mathbf{y^P_{n+1}} = \sum_{i=0}^{k_n} \alpha^P_{i,n}\, \mathbf{y_{n-i}}, \tag{6.19}$$

with predictor coefficients $\alpha^P_{i,n}$ depending only on the stepsize series $h_n, \ldots, h_{n-k_n}$. The predictor value is then used as start value for the iterative solution of the implicit corrector equation. The time derivative $\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}}$ of the corrector polynomial through the values $\mathbf{x_{n+1}}, \ldots, \mathbf{x_{n+1-k_n}}$ can be written as

$$\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}} = \sum_{i=0}^{k_n} \alpha^C_{i,n}\, \mathbf{x_{n+1-i}} = \mathbf{x^{CC}_{n+1}} + \alpha^C_{0,n}\mathbf{x_{n+1}}, \tag{6.20}$$

with corrector constant

$$\mathbf{x^{CC}_{n+1}} = \sum_{i=1}^{k_n} \alpha^C_{i,n}\, \mathbf{x_{n+1-i}}. \tag{6.21}$$

It summarizes the contribution of the last differential states to $\dot{\mathbf{x}}^{\mathbf{C}}_{\mathbf{n+1}}$, with the exception of the value $\mathbf{x_{n+1}}$ which is to be determined by the integration step. Note that for an easier notation in this context we have relabeled the coefficients from Chapter 5, such that $\alpha^C_{i,n}$ now corresponds to the $\alpha_{(k_n-i)(n+1-k_n)}$ of Definition 5.18.

For the solution of the implicit system (5.15) we apply a Newton-like method using the approximation $\mathbf{M_n}$ instead of the actual Jacobian $\mathbf{J_n} := \frac{\partial \mathbf{f^{BDF}_n}}{\partial \mathbf{y}}$ of the method function

$$\mathbf{f^{BDF}_n}(\mathbf{x_{n+1}}, \mathbf{z_{n+1}}; \mathbf{x^{CC}_{n+1}}, \mathbf{p}, \mathbf{x_0}, \mathbf{z_0}) := \begin{pmatrix} \mathbf{A}(t_{n+1}, \mathbf{x_{n+1}}, \mathbf{z_{n+1}}, \mathbf{p})(\mathbf{x^{CC}_{n+1}} + \alpha^C_{0,n}\mathbf{x_{n+1}}) - h_n \mathbf{f}(t_{n+1}, \mathbf{x_{n+1}}, \mathbf{z_{n+1}}, \mathbf{p}) \\ \mathbf{g}(t_{n+1}, \mathbf{x_{n+1}}, \mathbf{z_{n+1}}, \mathbf{p}) - \theta(t_{n+1})\, \mathbf{g}(t_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) \end{pmatrix}. \tag{6.22}$$

The iterations to find a root of $\mathbf{f^{BDF}_n}$ are started with the predictor value $\mathbf{y^{(0)}_{n+1}} = \mathbf{y^P_{n+1}}$. We perform s_n Newton iterations

$$\mathbf{y^{(i+1)}_{n+1}} = \mathbf{y^{(i)}_{n+1}} + \mathbf{\Delta y^{(i)}_{n+1}}, \qquad 0 \le i \le s_n - 1, \tag{6.23}$$

where the increment is given by

$$\mathbf{\Delta y^{(i)}_{n+1}} = -\mathbf{M_n^{-1}} \mathbf{f^{BDF}_n}(\mathbf{y^{(i)}_{n+1}}). \tag{6.24}$$

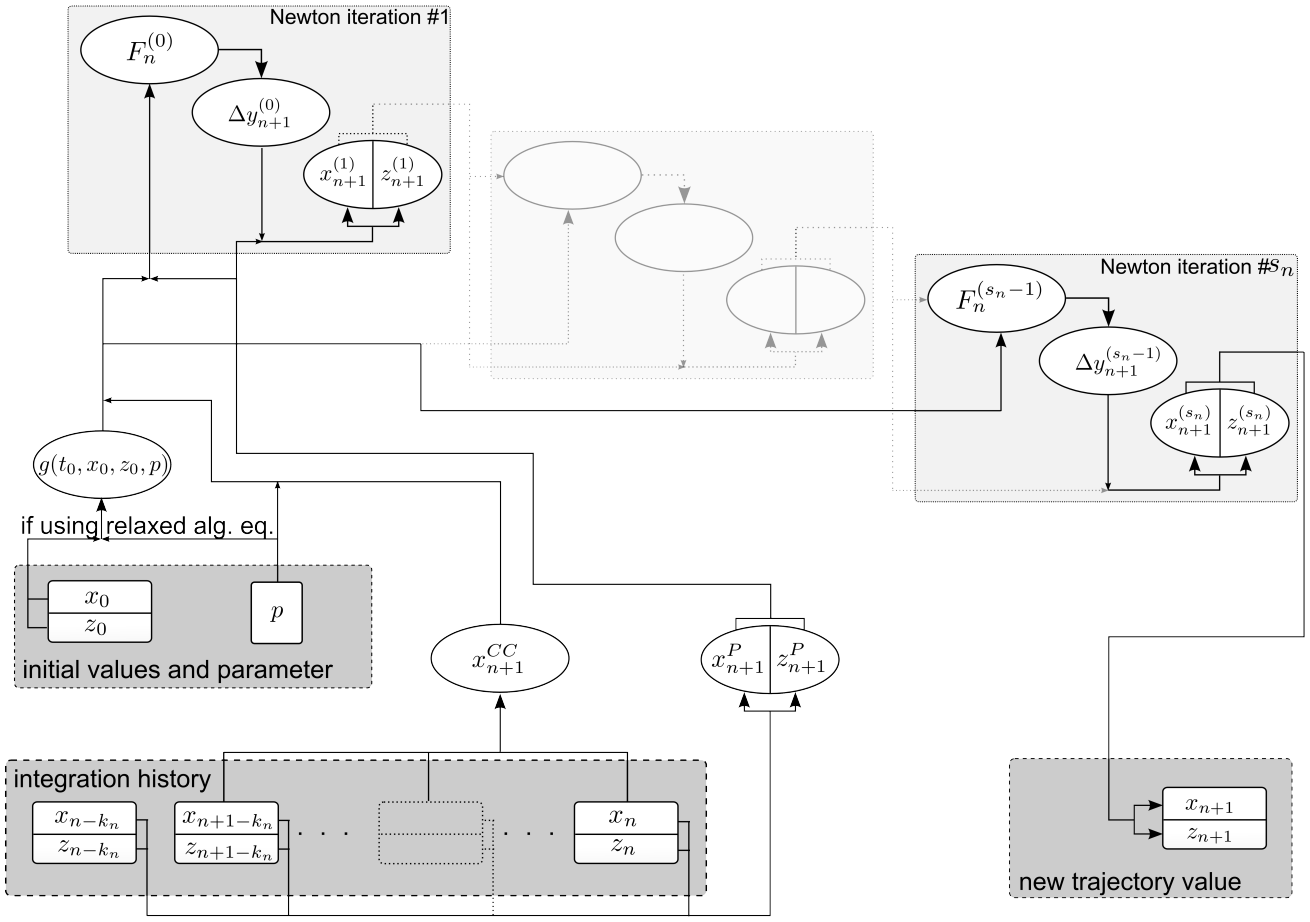

Figure 6.1: The dependencies in the calculations made during the n-th step of a BDF method with order k_n and iteration matrix M_n.

Now, we interpret the computations made in the equations (6.19)-(6.24) as elementary operations for which the partial derivatives are easily determined, cf. Section 6.4.1. Figure 6.1 shows the dependencies of the quantities and the elementary operations used in one BDF step. In the following section we use the AD techniques described in Chapter 2 on this elemental representation of a BDF step to propagate sensitivities through the integration process.

6.4 First order sensitivity generation for DAEs

In this section we derive IND-based schemes for the computation of first order sensitivities. The first IND-based schemes for the computation of first order forward sensitivities for ODEs were derived by Bock [Boc81, Boc83] for the code METAN1 of Deuflhard and Bader [DB83], using the semi-implicit midpoint rule, and the code DIFSYS of Bulirsch and Stoer [BS66], using the explicit midpoint rule. Plitt [Pli81] and later von Schwerin [vS92] used first order forward IND schemes derived from Runge-Kutta-Fehlberg [Feh69, Feh70] codes. Von Schwerin and Winckler [SW96]

presented first order forward IND schemes for ODE and DAE systems with discontinuities based on Dormand-Prince Runge-Kutta methods [DP80, PD81] and on an Adams PECE [BA83, Mil26, Mou26] method implemented in the MBSSIM [SW94] integrator library. The first forward IND schemes for the computation of first order DAE sensitivities using BDF methods were presented in the code `DAESOL` by Eich [Eic87]. Later, Støren and Hertzberg [SH99] augmented the DAE code DASSL [Pet82, BCP96] by a first order forward IND scheme and called this method DASSP. First order adjoint IND schemes for DAEs have first been presented for BDF methods by Albersmeyer [AB08] in the code `DAESOL-II`, for ODEs using explicit Runge-Kutta methods by Bock [Boc87] and later using explicit Runge-Kutta-Fehlberg methods by Wirsching [Wir06]. As a starting point we present now the derivation of first order IND schemes using the nomenclature and presentation style of AD.

6.4.1 Forward sensitivity generation

For the derivation of first order forward IND schemes we have the choice between two slightly different variants. The strict application of the IND principle to the predictor-corrector scheme of the nominal solution leads to the so-called iterative IND scheme. The (not entirely appropriate) assumption that the implicit equation of the nominal problem has been solved exactly leads, together with the application of the implicit function theorem, to the so-called direct IND scheme.

Iterative forward IND

To differentiate the predictor-corrector scheme we define the elementary functions corresponding to the equations (6.19)-(6.24)

$$\boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{P}}(\mathbf{y_n}, \ldots, \mathbf{y_{n-k_n}}) := \sum_{i=0}^{k_n} \alpha_{i,n}^{P}\, \mathbf{y_{n-i}}, \tag{6.25a}$$

$$\boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{CC}}(\mathbf{x_n}, \ldots, \mathbf{x_{n+1-k_n}}) := \sum_{i=1}^{k_n} \alpha_{i,n}^{C}\, \mathbf{x_{n+1-i}}, \tag{6.25b}$$

$$\boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{it}}(\mathbf{y_{n+1}}, \boldsymbol{\Delta}\mathbf{y_{n+1}}) := \mathbf{y_{n+1}} + \boldsymbol{\Delta}\mathbf{y_{n+1}}, \tag{6.25c}$$

$$\boldsymbol{\phi}_{\mathbf{n}}^{\boldsymbol{\Delta}}(\mathbf{y_{n+1}}, \mathbf{x_{n+1}^{CC}}, \mathbf{p}, \mathbf{y_0}, \mathbf{z_0}) := -\mathbf{M_n^{-1}}\mathbf{f_n^{BDF}}(\mathbf{y_{n+1}}; \mathbf{x_{n+1}^{CC}}, \mathbf{p}, \mathbf{y_0}, \mathbf{z_0}). \tag{6.25d}$$

The partial derivatives of these elementary functions are then easily calculated as (omitting the arguments)

$$\frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{P}}}{\partial \mathbf{y}_{\mathbf{n-i}}} = \alpha_{i,n}^{P}\, \mathbb{I}_{n_y}, \qquad 0 \le i \le k_n \tag{6.26a}$$

$$\frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{CC}}}{\partial \mathbf{x}_{\mathbf{n+1-i}}} = \alpha_{i,n}^{C}\, \mathbb{I}_{n_x}, \qquad 1 \le i \le k_n \tag{6.26b}$$

$$\frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{it}}}{\partial \mathbf{y}_{\mathbf{n+1}}} = \mathbb{I}_{n_y}, \qquad \frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{it}}}{\partial \boldsymbol{\Delta}\mathbf{y}_{\mathbf{n+1}}} = \mathbb{I}_{n_y}, \tag{6.26c}$$

$$\frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\boldsymbol{\Delta}}}{\partial \mathbf{v}} = -\mathbf{M}_{\mathbf{n}}^{-1} \frac{\partial \mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}}}{\partial \mathbf{v}} \quad \text{for } \mathbf{v} \in \{\mathbf{x}_{\mathbf{n+1}}^{\mathbf{CC}}, \mathbf{p}, \mathbf{x_0}, \mathbf{z_0}\}. \tag{6.26d}$$

In this way, the whole nominal integration performed using the predictor-corrector scheme can be understood as a sequence of elementary functions $\boldsymbol{\phi}_{\mathbf{m}}(\boldsymbol{\phi}_{\mathbf{m-1}}(\ldots(\boldsymbol{\phi}_{\mathbf{0}}(\mathbf{y_0}, \mathbf{p})\ldots)))$ of the above types. As in a first order AD forward sweep (see Section 2.4.2), we propagate the sensitivity information starting with a sensitivity direction $\mathbf{d} = (\mathbf{d}_{\mathbf{x}}^T, \mathbf{d}_{\mathbf{z}}^T, \mathbf{d}_{\mathbf{p}}^T)^T \in \mathbb{R}^{n_x+n_z+n_p}$ along with the computation of the nominal values. We use the common AD notation and denote the intermediate quantities in the forward sweep by a dot over the corresponding nominal value. For the elementary functions in (6.25) and their results we then obtain (again omitting the arguments)

$$\dot{\mathbf{y}}_{\mathbf{n+1}}^{\mathbf{P}} = \sum_{i=0}^{k_n} \frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{P}}}{\partial \mathbf{y}_{\mathbf{n-i}}} \dot{\mathbf{y}}_{\mathbf{n-i}} = \sum_{i=0}^{k_n} \alpha_{i,n}^{P}\, \dot{\mathbf{y}}_{\mathbf{n-i}}, \tag{6.27a}$$

$$\dot{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}} = \sum_{i=1}^{k_n} \frac{\partial \boldsymbol{\phi}_{\mathbf{n}}^{\mathbf{CC}}}{\partial \mathbf{x}_{\mathbf{n+1-i}}} \dot{\mathbf{x}}_{\mathbf{n+1-i}} = \sum_{i=1}^{k_n} \alpha_{i,n}^{C}\, \dot{\mathbf{x}}_{\mathbf{n+1-i}}, \tag{6.27b}$$

$$\dot{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l+1})} = \dot{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})} + \boldsymbol{\Delta}\dot{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})}, \tag{6.27c}$$

$$\begin{aligned} \boldsymbol{\Delta}\dot{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l+1})} = -\mathbf{M}_{\mathbf{n}}^{-1} \Bigg[& \mathbf{J}_{\mathbf{n}} \dot{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})} + \mathbf{A}_{\mathbf{n+1}} \dot{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}} \\ & + \begin{pmatrix} \mathbf{A}_{\mathbf{p,n+1}}(\mathbf{x}_{\mathbf{n+1}}^{\mathbf{CC}} + \alpha_{0,n}^{C}\mathbf{x}_{\mathbf{n+1}}) - h_n \mathbf{f}_{\mathbf{p,n+1}} \\ \mathbf{g}_{\mathbf{p,n+1}} - \theta_{n+1}\mathbf{g}_{\mathbf{p,0}} \end{pmatrix} \dot{\mathbf{p}} \\ & + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -\theta_{n+1}\mathbf{g}_{\mathbf{x,0}} & -\theta_{n+1}\mathbf{g}_{\mathbf{z,0}} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{x}}_{\mathbf{0}} \\ \dot{\mathbf{z}}_{\mathbf{0}} \end{pmatrix} \Bigg], \end{aligned} \tag{6.27d}$$

where we initialize

$$\dot{\mathbf{x}}_{\mathbf{0}} = \mathbf{d}_{\mathbf{x}}, \qquad \dot{\mathbf{z}}_{\mathbf{0}} = \mathbf{d}_{\mathbf{z}} \quad \text{and} \quad \dot{\mathbf{p}} = \mathbf{d}_{\mathbf{p}} \tag{6.28}$$

using the sensitivity direction $\mathbf{d}$. Note that all model function derivatives needed in (6.27d) can be efficiently computed as directional derivatives. Furthermore, the IND predictor $\dot{\mathbf{y}}_{\mathbf{n+1}}^{\mathbf{P}}$ and the IND corrector constant $\dot{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}}$ can be computed via modified divided differences in the same way as it is described in Section 5.3.1 for the predictor and corrector constant of the nominal trajectory. Having a closer look at the derived scheme (6.27), we see that it represents again a predictor-corrector scheme. More precisely, it is the BDF predictor-corrector scheme for the directional

formulation of the forward variational DAE (6.8) when the discretization scheme (including iteration matrices) of the computation of the nominal trajectory is used. Note that this equivalence between the derivative of the nominal discretization scheme and the discretization scheme for the variational ODE/DAE holds not only for BDF methods, but for all linear discretization schemes [Boc87].
We give the iterative forward IND approach in algorithmic form in two variants: Simultaneous with the nominal integration (see Algorithm 6.1) and deferred for a later determination of forward sensitivities (see Algorithm 6.2), assuming that the complete discretization scheme of the nominal integration as well as the nominal solution at the integration gridpoints is somehow available. Note that the algorithms, although here described for one sensitivity direction, can easily be modified to compute several directional sensitivities in parallel, which is also used in practice to improve efficiency.
The first order iterative forward IND scheme is equal to the "staggered corrector method" that was proposed later by Feehery et al. [FTB97], provided that the staggered corrector method uses the same number of Newton iterations for the solution of the corrector equation in the variational DAE as used for the nominal solution. Otherwise the IND principle would be violated.

Algorithm 6.1: First order iterative forward IND scheme (simultaneous)

Input: t_0, t_f, h_0, initial values $\mathbf{y_0}$, parameter $\mathbf{p}$, sensitivity direction $\mathbf{d}$.
Output: Nominal solution $\mathbf{y_N}$, forward directional sensitivity $\dot{\mathbf{y}}_\mathbf{N} = \mathcal{W} \cdot \mathbf{d}$.
set $k_0 = 1$, $n = 0$;
initialize nominal integration with $\mathbf{y_0}$, $\mathbf{p}$ and sensitivities with $\mathbf{d}$ (6.28);
while t_f *not reached* **do**
  compute $\mathbf{y_{n+1}}$ by nominal integration step for h_n, k_n including s_n
    Newton-like iterations using matrix M_n to solve the corrector equation;
  if *step accepted* **then**
    `// corrector equation successfully solved,`
    `// error estimation accepted`
    compute IND predictor (6.27a) and IND corrector constant (6.27b);
    using $\mathbf{y_{n+1}}$, h_n and $\mathbf{M_n^{-1}}$, make s_n iterations (6.27c)/(6.27d)
      for the IND corrector equation to obtain $\dot{\mathbf{y}}_\mathbf{n+1}$;
    $t_{n+1} = t_n + h_n$;
    determine h_{n+1} and k_{n+1} for next step;
    $n = n + 1$;
  else
    `// corrector equation solving failed or`
    `// error estimation too large`
    update Jacobian approximation $\mathbf{M_n}$ according to monitor strategy or reduce stepsize h_n;
  end
end
$N = n$;

Algorithm 6.2: First order iterative forward IND scheme (deferred)

Input: Discretization scheme stored from nominal integration (stepsizes h_i, orders k_i, trajectory values $\mathbf{y_i}$, number of Newton iterations s_i, used iteration matrix factorizations $\mathbf{M_i^{-1}}$), number of integration steps N, sensitivity direction $\mathbf{d}$.

Output: Directional sensitivity $\dot{\mathbf{y}}_\mathbf{N} = \mathcal{W} \cdot \mathbf{d}$.

set $n = 0$, initialize sensitivities with $\mathbf{d}$ (6.28);

while $n < N$ **do**

  get t_n, h_n, k_n from stored discretization scheme;

  compute IND predictor (6.27a) and IND corrector constant (6.27b) ;

  get s_n, $\mathbf{M_n^{-1}}$ and $\mathbf{y_{n+1}}$ from stored scheme;

  using $\mathbf{y_{n+1}}$, h_n and $\mathbf{M_n^{-1}}$, make s_n iterations (6.27c)/(6.27d) for the IND corrector equation to obtain $\dot{\mathbf{y}}_\mathbf{n+1}$;

  $n = n + 1$;

end

Direct forward IND

In the direct forward approach we assume that during the nominal integration we have solved the corrector equation (6.22) exactly in every integration step, i.e., $\mathbf{f_n^{BDF}}(\mathbf{y_{n+1}}; \mathbf{x_{n+1}^{CC}}, \mathbf{p}, \mathbf{x_0}, \mathbf{z_0}) = \mathbf{0}$. Then we use the implicit function theorem to describe the dependency of $\mathbf{y_{n+1}}$ on the other quantities

$$\frac{\partial \mathbf{y_{n+1}}}{\partial \mathbf{v}} = -\left[\frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{y_{n+1}}}\right]^{-1} \frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{v}} = -\mathbf{J_n^{-1}} \frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{v}}, \tag{6.29}$$

for $\mathbf{v} \in \{\mathbf{x_{n+1}^{CC}}, \mathbf{p}, \mathbf{x_0}, \mathbf{z_0}\}$.

We summarize over all dependencies to obtain $\dot{\mathbf{y}}_\mathbf{n+1}$

$$\begin{aligned}
\dot{\mathbf{y}}_\mathbf{n+1} &= -\mathbf{J_n^{-1}} \left[\frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{x_{n+1}^{CC}}} + \frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{p}}\dot{\mathbf{p}} + \frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{x_0}}\dot{\mathbf{x}}_\mathbf{0} + \frac{\partial \mathbf{f_n^{BDF}}}{\partial \mathbf{z_0}}\dot{\mathbf{z}}_\mathbf{0}\right] \\
&= -\mathbf{J_n^{-1}} \left[\begin{pmatrix} \mathbf{A_{n+1}}\dot{\mathbf{x}}_\mathbf{n+1}^\mathbf{CC} \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{A_{p,n+1}}(\mathbf{x_{n+1}^{CC}} + \alpha_{0,n}^C \mathbf{x_{n+1}}) - h_n \mathbf{f_{p,n+1}} \\ \mathbf{g_{p,n+1}} - \theta_{n+1}\mathbf{g_{p,0}} \end{pmatrix} \dot{\mathbf{p}} \right. \\
&\quad \left. + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -\theta_{n+1}\mathbf{g_{x,0}} & -\theta_{n+1}\mathbf{g_{z,0}} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{x}}_\mathbf{0} \\ \dot{\mathbf{z}}_\mathbf{0} \end{pmatrix}\right].
\end{aligned} \tag{6.30}$$

Using the direct approach, the corresponding implicit BDF discretization scheme of the integration step is differentiated and not the actual predictor-corrector method. Hence in this approach no predictor dependency occurs and no Newton-like iterations are made for the IND equation. Instead, the building and decomposition of the actual Jacobian $\mathbf{J_n}$ is needed in every step, which in most cases renders the direct approach more expensive than the iterative one. The other model derivatives can be obtained efficiently by directional forward derivatives. The algorithmic form of the direct forward IND approach is given for the simultaneous version in Algorithm 6.3 and for

the deferred version in Algorithm 6.4. Note that in the simultaneous version, the computed and decomposed Jacobian can be used as iteration matrix for the next nominal integration step. Additionally, note that (6.30) can be understood as the direct solution of the corrector equation of the forward variational DAE, since it is a linear equation. This is very similar to the strategy used by Caracotsios and Stewart [CS85] in their DASSL based code DDASAC. Later this approach was referred to as the staggered direct method [FTB97].

Algorithm 6.3: First order direct forward IND scheme (simultaneous)

Input: t_0, t_f, h_0, initial values $\mathbf{y_0}$, parameter $\mathbf{p}$, sensitivity direction $\mathbf{d}$
Output: Nominal solution $\mathbf{y_N}$, forward directional sensitivity $\dot{\mathbf{y}}_\mathbf{N}$.
set $k_0 = 1$, $n = 0$;
initialize nominal integration with $\mathbf{y_0}$ and $\mathbf{p}$ and sensitivities with $\mathbf{d}$ (6.28);
while t_f *not reached* **do**
  compute $\mathbf{y_{n+1}}$ by nominal integration step for h_n, k_n including s_n Newton-like iterations using matrix $\mathbf{M_n}$ to solve the corrector equation;
  if *step accepted* **then**
    `// corrector equation successfully solved,`
    `// error estimation accepted`
    compute IND corrector constant (6.27b);
    using $\mathbf{y_{n+1}}$ and h_n, compute $\mathbf{J_n}$ from (5.44),
      factorize $\mathbf{J_n}$ and solve system (6.30) to obtain $\dot{\mathbf{y}}_\mathbf{n+1}$;
    $\mathbf{M_{n+1}^{-1}} = \mathbf{J_n^{-1}}$, $t_{n+1} = t_n + h_n$;
    determine h_{n+1} and k_{n+1} for next step;
    $n = n + 1$;
  else
    `// corrector equation solving failed or`
    `// error estimation too large`
    update Jacobian approximation $\mathbf{M_n}$ according to monitor strategy or reduce stepsize h_n;
  end
end
$N = n$;

Remark 6.1 (Forward IND schemes versus solution of the forward variational DAE) The presented forward IND schemes lead to a very efficient way to compute approximations for the forward sensitivities. Compared to other codes that are constructed around the solution of the forward variational DAE in addition to the nominal DAE they offer by construction a natural exploitation of the inherent problem structure. This structure exploitation has to be taken into account additionally when using a non IND-based approach. It should be noted that the convergence of the forward IND schemes to the true sensitivities is assured with the same order as for the nominal solution, due to their interpretation as discretization scheme for the forward

Algorithm 6.4: First order direct forward IND scheme (deferred)

Input: Discretization scheme stored from nominal integration (stepsizes h_i, orders k_i, trajectory values $\mathbf{y_i}$), sensitivity direction $\mathbf{d}$
Output: Directional sensitivity $\dot{\mathbf{y}}_\mathbf{N}$.
initialize sensitivities with $\mathbf{d}$;
$n = 0$;
while $n < N$ **do**
  get t_n, h_n, k_n from stored discretization scheme;
  compute IND corrector constant (6.27b);
  get $\mathbf{y_{n+1}}$ from stored scheme;
  using $\mathbf{y_{n+1}}$ and h_n, compute $\mathbf{J_n}$ from (5.44),
    factorize $\mathbf{J_n}$ and solve system (6.30) to obtain $\dot{\mathbf{y}}_\mathbf{n+1}$;
  $n = n + 1$;
end

variational DAE. And hence also the intermediate quantities of the forward IND sweep converge against the trajectory of the solution of the forward variational DAE.

Forward IND using finite differences

If a lower precision of the computed sensitivities is sufficient, a third approach to compute first order forward sensitivities is possible. Since the integrator code `DAESOL-II` allows the storage and reuse of the discretization scheme, a later "replay" of the integration for different initial values and parameter is possible. Also a simultaneous computation of several trajectories with one discretization scheme, determined either by the first trajectory or by all trajectories together, is supported. Therefore, also a finite difference based IND approach can be used to compute sensitivities. In this approach, the original trajectory and trajectories for slightly perturbed initial values and parameter are computed using the same discretization scheme, and afterwards the finite differences are calculated. Since the same discretization scheme is used for all trajectories, the principle of IND is fulfilled. As in the case of ordinary functions, one can only expect an accuracy of roughly the square-root of the machine precision, even for the optimal choice of the perturbation size. On the other hand, no additional derivatives of the model functions are needed in addition to the ones needed anyway during nominal integration. Depending on the effort for derivative evaluation this might lead to significant savings in computational time, provided some accuracy of the sensitivities can be spent. This approach was later also used by Støren and Hertzberg [SH99] in their DASSL based code DASSP. Note that using this approach a computation of adjoint sensitivities is not possible.

6.4.2 Adjoint sensitivity generation

After the derivation of the well-known first order forward IND schemes for BDF methods, we now develop in the same framework the new first order adjoint IND schemes first presented in

[AB08]. Like in the forward case, we present two slightly different schemes based on the actual predictor-corrector method and on the corresponding implicit scheme, respectively.

Iterative adjoint IND

We start with the development of the iterative adjoint IND scheme. Like the iterative forward scheme it is based on the actual predictor-corrector scheme for the nominal trajectory. We assume that the nominal integration has been performed and that the discretization scheme including stepsizes, orders, iteration matrices (respectively their factorizations), iteration numbers as well as the nominal solution at the integration gridpoints has been stored. Note that, due to the monitor strategy used within the Newton-like method (cf. Section 5.3.3), the number of different iteration matrices is significantly smaller than the overall number of accepted integration steps.

For a given adjoint direction $\bar{\mathbf{y}} \in \mathbb{R}^{n_x+n_z}$ the sensitivity information is propagated backwards through the sequence of elementary operations representing the nominal integration scheme. The elementary operations are here also given by (6.25). Following the common AD notation, we denote the adjoint quantity corresponding to the nominal quantity $\mathbf{v}$ by $\bar{\mathbf{v}}$. The adjoint IND sweep starts by setting all intermediate adjoint quantities to zero and initializing $\bar{\mathbf{y}}_{\mathbf{N}} := \bar{\mathbf{y}}$. Then we apply the reverse mode of AD to the elemental representation of the nominal integration scheme to propagate the sensitivity information backward through the integration process. With the partial derivatives (6.26) of the elementary functions we obtain the propagation rules (not displaying the arguments)

$$\bar{\mathbf{y}}_{\mathbf{n-i}} \mathrel{+}= \alpha_{i,n}^{P} \bar{\mathbf{y}}_{\mathbf{n+1}}^{\mathbf{P}}, \quad 0 \le i \le k_n, \tag{6.31a}$$

$$\bar{\mathbf{x}}_{\mathbf{n+1-i}} \mathrel{+}= \alpha_{i,n}^{C} \bar{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}}, \quad 1 \le i \le k_n, \tag{6.31b}$$

$$\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})} \mathrel{+}= \bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l+1})}, \tag{6.31c}$$

$$\boldsymbol{\Delta}\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})} = \bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l+1})}, \tag{6.31d}$$

$$(\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})})^T \mathrel{+}= -\boldsymbol{\Delta}\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})\,T} \mathbf{M}_{\mathbf{n}}^{-1}\, \mathbf{J}_{\mathbf{n}}, \tag{6.31e}$$

$$\bar{\mathbf{p}}^T \mathrel{+}= -\boldsymbol{\Delta}\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})\,T} \mathbf{M}_{\mathbf{n}}^{-1} \begin{pmatrix} \mathbf{A}_{\mathbf{p,n+1}}(\mathbf{x}_{\mathbf{n+1}}^{\mathbf{CC}} + \alpha_{0,n}^{C}\mathbf{x}_{\mathbf{n+1}}) - h_n \mathbf{f}_{\mathbf{p,n+1}}, \\ \mathbf{g}_{\mathbf{p,n+1}} - \theta_{n+1}\mathbf{g}_{\mathbf{p,0}} \end{pmatrix}, \tag{6.31f}$$

$$(\bar{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}})^T \mathrel{+}= -\boldsymbol{\Delta}\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})\,T} \mathbf{M}_{\mathbf{n}}^{-1}\, \mathbf{A}_{\mathbf{n+1}}, \tag{6.31g}$$

$$(\bar{\mathbf{x}}_{\mathbf{0}}^T, \bar{\mathbf{z}}_{\mathbf{0}}^T) \mathrel{+}= -\boldsymbol{\Delta}\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})\,T} \mathbf{M}_{\mathbf{n}}^{-1} \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -\theta_{n+1}\mathbf{g}_{\mathbf{x,0}} & -\theta_{n+1}\mathbf{g}_{\mathbf{z,0}} \end{pmatrix}. \tag{6.31h}$$

The model function derivatives in (6.31e) and (6.31f) can efficiently be evaluated together by one adjoint directional derivative of $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{A}\dot{\mathbf{x}}$. With (6.31) we can write the iterative backward IND scheme in algorithmic form, leading to Algorithm 6.5.
During the computations, systems of the type $\boldsymbol{\mu} = \mathbf{M}_{\mathbf{n}}^{-T}\bar{\mathbf{y}}_{\mathbf{n+1}}$ have to be solved. Note that in most direct linear algebra packages (e.g., ATLAS [WPD01], UMFPACK [Dav04]) a given factorization of $\mathbf{M}_{\mathbf{n}}$ can be reused for the solution of systems incorporating the transposed matrix $\mathbf{M}_{\mathbf{n}}^T$, such that in the adjoint IND scheme neither an explicit transposition nor a new factorization is needed.

Algorithm 6.5: First order iterative adjoint IND scheme

Input: Discretization scheme stored from nominal integration (stepsizes, orders, trajectory values, iteration matrix factorizations, iteration counts), adjoint sensitivity direction $\bar{\mathbf{y}}$.

Output: Adjoint directional sensitivity $\bar{\mathbf{y}}_\mathbf{0}, \bar{\mathbf{p}}$.

initialize sensitivities with $\bar{\mathbf{y}}_\mathbf{N} = \bar{\mathbf{y}}$;

$n = N - 1$;

while $n \geq 0$ **do**

  $\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{s_n})} = \bar{\mathbf{y}}_{\mathbf{n+1}}$;

  get $\mathbf{y_{n+1}}$, t_n, h_n, k_n, $\mathbf{M}_\mathbf{n}^{-1}$, s_n from stored discretization scheme;

  for $l = s_n - 1 : 0$ **do**

    knowing (6.31d), solve $\boldsymbol{\mu} = \mathbf{M}_\mathbf{n}^{-T} \bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l+1})}$;

    using $\boldsymbol{\mu}$, compute respectively increment $\bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{l})}, \bar{\mathbf{p}}, \bar{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}}, \bar{\mathbf{x}}_\mathbf{0}, \bar{\mathbf{z}}_\mathbf{0}$ by applying (6.31c), (6.31e), (6.31f), (6.31g) and (6.31h);

  end

  $\bar{\mathbf{y}}_{\mathbf{n+1}}^{\mathbf{P}} = \bar{\mathbf{y}}_{\mathbf{n+1}}^{(\mathbf{0})}$;

  propagate corrector constant dependency backwards using (6.31b);

  propagate predictor dependency backwards using (6.31a);

  `// all contributions of` $\mathbf{y_n}$ `taken into account,`

  `// value of` $\bar{\mathbf{y}}_\mathbf{n}$ `is final`

  $n = n - 1$;

end

Direct adjoint IND

To derive the first order direct adjoint IND scheme, we suppose again that the nominal integration has been performed. Furthermore, the discretization scheme including stepsizes, orders and trajectory values at the integration grid of the nominal integration has been stored. Other than in the iterative reverse scheme, we assume that during nominal integration all corrector equations have been solved exactly.

Like in the direct forward approach the corresponding implicit integration scheme is differentiated and hence the predictor dependency and the Newton-like iterations are not taken into account. Instead, the implicit function theorem is used to propagate the sensitivities backwards. With the partial derivatives (6.29) of the trajectory values $\mathbf{y_{n+1}}$ and using the reverse mode of AD we obtain

the propagation rules

$$(\bar{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}})^T = -\bar{\mathbf{x}}_{\mathbf{n+1}}^T \begin{pmatrix} \mathbb{I}_{n_x} & \mathbf{0} \end{pmatrix} \mathbf{J}_{\mathbf{n}}^{-1} \begin{pmatrix} \mathbf{A_{n+1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{6.32a}$$

$$\bar{\mathbf{p}}^T \mathrel{+}= -\bar{\mathbf{x}}_{\mathbf{n+1}}^T \begin{pmatrix} \mathbb{I}_{n_x} & \mathbf{0} \end{pmatrix} \mathbf{J}_{\mathbf{n}}^{-1} \begin{pmatrix} \mathbf{A_{p,n+1}}(\mathbf{x}_{\mathbf{n+1}}^{\mathbf{CC}} + \alpha_{0,n}^C \mathbf{x_{n+1}}) - h_n \mathbf{f_{p,n+1}} \\ \mathbf{g_{p,n+1}} - \theta_{n+1} \mathbf{g_{p,0}} \end{pmatrix}, \tag{6.32b}$$

$$(\bar{\mathbf{x}}_{\mathbf{0}}^T, \bar{\mathbf{z}}_{\mathbf{0}}^T) \mathrel{+}= -\bar{\mathbf{x}}_{\mathbf{n+1}}^T \begin{pmatrix} \mathbb{I}_{n_x} & \mathbf{0} \end{pmatrix} \mathbf{J}_{\mathbf{n}}^{-1} \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -\theta_{n+1} \mathbf{g_{x,0}} & -\theta_{n+1} \mathbf{g_{z,0}} \end{pmatrix}. \tag{6.32c}$$

Thus, in every step the building and decomposition of the Jacobian $\mathbf{J}_{\mathbf{n}}^{-1}$ is needed. All other model function derivatives in (6.32b) are obtained as adjoint directional derivatives. Algorithm 6.6 shows the algorithmic form of the direct adjoint IND sweep. Note that the algorithm is given for an input direction $\bar{\mathbf{x}} \in \mathbb{R}^{n_x}$ with respect to the differential variables only. However, using the implicit function theorem and the algebraic equations at the final time, an adjoint sensitivity direction containing a nonzero algebraic part can be expressed by an adjoint direction only containing a differential part. Note furthermore, that in practice the algorithms should be implemented with the possibility to propagate several adjoint sensitivities at once, as this increases the efficiency significantly, e.g., by savings during the derivative evaluations of the model functions.

Algorithm 6.6: First order direct adjoint IND scheme

Input: Discretization scheme stored from nominal integration (stepsizes, orders, trajectory values), adjoint sensitivity direction $\bar{\mathbf{x}}$.

Output: Adjoint directional sensitivities $\bar{\mathbf{y}}_{\mathbf{0}}$, $\bar{\mathbf{p}}$.

initialize sensitivities $\bar{\mathbf{x}}_{\mathbf{N}} = \bar{\mathbf{x}}$;

$n = N - 1$;

while $n \geq 0$ **do**

- get $\mathbf{y_{n+1}}$, t_n, h_n, k_n from stored discretization scheme;
- solve $\boldsymbol{\mu} = \mathbf{J}_{\mathbf{n}}^{-T} \begin{pmatrix} \bar{\mathbf{x}}_{\mathbf{n+1}} \\ \mathbf{0} \end{pmatrix}$;
- using $\boldsymbol{\mu}$, increment $\bar{\mathbf{x}}_{\mathbf{n+1}}^{\mathbf{CC}}$, $\bar{\mathbf{p}}$, $\bar{\mathbf{x}}_{\mathbf{0}}$, $\bar{\mathbf{z}}_{\mathbf{0}}$ by applying (6.32a), (6.32b), (6.32c);
- propagate corrector constant dependency backwards using (6.31b);
- `// all contributions of` $\mathbf{x_n}$ `taken into account,`
- `// value of` $\bar{\mathbf{x}}_{\mathbf{n}}$ `is final`
- $n = n - 1$;

end

Remark 6.2 (Adjoint IND schemes versus solution of adjoint variational DAE)
The presented adjoint IND schemes are to the best of our knowledge the first and only ones implemented for implicit methods in general as well as for LMMs. All other codes supporting adjoint sensitivity generation are based on the solution of the adjoint variational DAE. Compared to them the IND schemes we developed offer the possibility to reuse matrix factorizations and trajectory values from the nominal integration and have no need to interpolate trajectory values.

Hence in comparison this usually leads to a significantly better performance of the IND approach, which we confirm numerically on several IVP examples in Section 9.2 on page 232. Also only when using IND the sensitivity approximations represent the exact derivative of the numerical solution of the nominal IVP.

It should be noted that we have convergence of the numerical sensitivities computed by the adjoint IND schemes to the true adjoint sensitivities with the same order as for the nominal solution when the stepsizes of the nominal integration tend to zero. However the intermediate quantities in the adjoint IND scheme will in general not converge to the trajectory values of the solution of the adjoint variational DAE. This is different from the case of the forward IND schemes described in Remark 6.1. The adjoint IND scheme does for LMMs in general not correspond to a consistent discretization scheme of the adjoint variational DAE, and hence we cannot expect convergence. The only exception here would be the case of a method with fixed order and equidistant stepsizes.

Another way to understand this discrepancy is to ignore for the moment a possible parameter dependency and analyze the meaning of the intermediate adjoint quantities. The final value of an adjoint intermediate quantity in the IND schemes holds the derivative information about how the result of the numerical scheme at the final time depends on the single trajectory value of the numerical solution at the corresponding integration gridpoint. In a predictor-corrector LMM, however, no trajectory point, except the initial value, determines alone the progress of the numerical solution over time. Even when we consider the truly implicit BDF method this is only the case for values of gridpoints, where the BDF order of the following integration step is equal to one. In all other cases more than one historical trajectory point contributes to a new trajectory value and they determine the progress of the numerical solution together. Hence in a general LMM only in the adjoint quantity corresponding to the initial value the full derivative information on the dependency of the final numerical values is accumulated.

In the solution trajectory of the adjoint variational DAE on the other hand every trajectory point carries in a sense the full sensitivity information of the nominal solution values (or a function depending on them) at the final time. This is in analogy to that any trajectory point of the analytical nominal solution fully determines the development of the analytical nominal solution over time (uniqueness of the solution is here always assumed).

As a result of these considerations we can expect that the lower the maximum order of the used LMM (or the maximum number of historical trajectory values contributing to a new one) is, the closer the intermediate adjoint quantities will be to the trajectory of the adjoint variational DAE. This behavior is illustrated for a simple test problem in Example 6.3 that follows below.

Note that this difference from the solution of the adjoint variational DAE is not a drawback or shortcoming of the adjoint IND scheme as we are only interested here in the adjoint sensitivity information over the whole integration horizon. However, for certain strategies that use in their derivation the properties of the solution trajectory of the adjoint variational DAE, such as strategies for a posteriori estimation of the global integration error or strategies for the control of the global error by choosing a suitable discretization scheme during integration [BR01, CP04, LV07, TB09], we might have to take this difference into account if we want to use them in connection with adjoint IND schemes.

Example 6.3 (Adjoint IND scheme versus solution of adjoint variational DAE)
We consider the ODE-IVP

$$\begin{aligned}
\dot{x}_1(t) &= x_1(t) \\
\dot{x}_2(t) &= x_2(t) + x_1(t)x_1(t) \\
\dot{x}_3(t) &= x_3(t) + x_1(t)x_2(t) \\
\dot{x}_4(t) &= x_4(t) + x_1(t)x_3(t) + x_2(t)x_2(t) \\
\dot{x}_5(t) &= x_5(t) + x_1(t)x_4(t) + x_2(t)x_3(t) \\
t \in [0,1], &\quad \mathbf{x}(0) = (1, 1, 0.5, 0.5, 0.25)^T,
\end{aligned} \tag{6.33}$$

with the solution $\mathbf{x}(t) = (e^t, e^{2t}, 0.5\,e^{3t}, 0.5\,e^{4t}, 0.25\,e^{5t})^T$. We compute the sensitivity of the component $x_5(1)$ at the final time with respect to the initial values using a directional adjoint sensitivity. We do this by the first order iterative adjoint IND scheme presented above, that is implemented in our integrator code `DAESOL-II`, for different choices of $k_{\max}$, the maximum BDF order during nominal integration. Additionally, we compute the sensitivity by solving the corresponding adjoint variational ODE. The resulting intermediate adjoint values obtained in the IND scheme and the adjoint trajectory computed from the variational ODE are depicted in Figure 6.2 on the facing page.

We see that for a maximum BDF order of $k_{\max} = 1$, intermediate adjoint values of the BDF scheme and the solution of the adjoint variational ODE practically coincide, while larger deviations occur when a higher BDF order is allowed. Note that these deviations will in general also not vanish if the stepsizes in the nominal integration are forced to be very small (or alternatively, if a very high integration accuracy is demanded). The more regular "offset" between the two approaches that is observed for higher BDF orders can in principle be reduced in exchange for significantly increased computational costs by scaling the adjoint IND values with the BDF Jacobian of the corresponding step. The peaks and the "oscillating" behavior however cannot be eliminated in this way and these effects are especially present in regions where order and significant stepsize changes occurred during the integration.

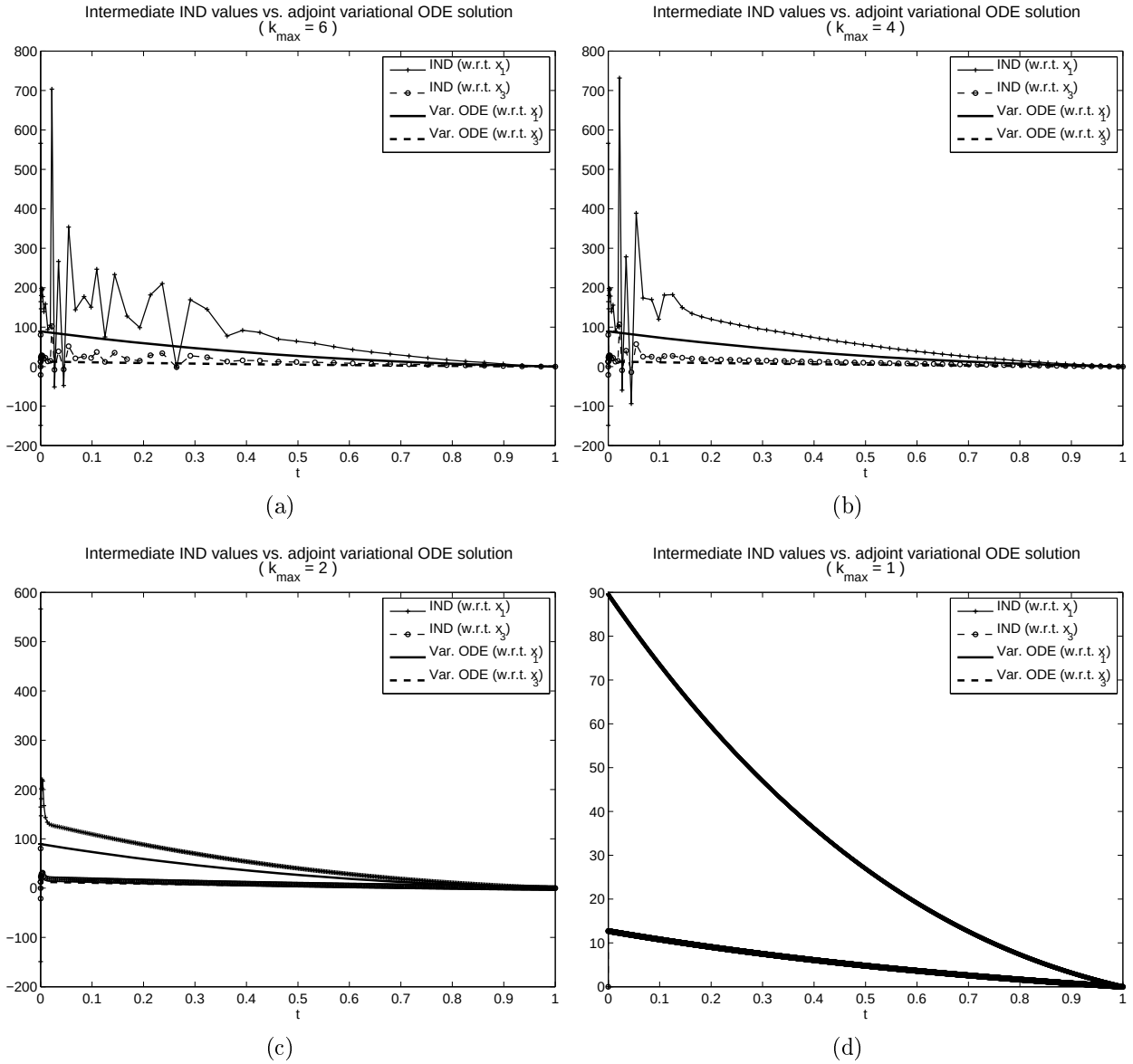

Figure 6.2: Comparison of the intermediate values of the first order iterative adjoint IND scheme for different choices of the maximum BDF order $k_{\max}$ during integration and the solution of the corresponding adjoint variational equation for the ODE-IVP (6.33). The lower the maximum BDF order the more similar the intermediate values in the adjoint IND scheme and the solution of the adjoint variational ODE become.

6.5 Generation of sensitivities of arbitrary order

After presenting efficient IND-based schemes for the computation of first order forward and adjoint sensitivities in the last section we now proceed to the question, how the m-th order sensitivity tensor

$$\mathcal{W}^{(m)}(\tau;\tau_0,\mathbf{y_0},\mathbf{p}) := \frac{\mathrm{d}^m\mathbf{y}(\tau;\tau_0,\mathbf{y_0},\mathbf{p})}{\mathrm{d}(\mathbf{y_0},\mathbf{p})^m} \in \mathbb{R}^{(n_x+n_z)\times(n_x+n_z+n_p)^m} \tag{6.34}$$

or certain parts or contractions thereof can be computed. Here $\mathbf{y}(\tau) = (\mathbf{x}(\tau)^T, \mathbf{z}(\tau)^T)^T$ is again the solution of the DAE-IVP (6.1). The efficient computation of higher-order sensitivities is of vital importance not only for the exact-Hessian optimization algorithm presented in this thesis, but also in the areas of robust optimization and optimal experimental design or in the analysis of dynamic systems in general. However, normally not the complete sensitivity tensor is needed, but only a subtensor with respect to a subset of variables, or only a contraction of the tensor in target and source space. An example for the last would be the directional derivative of a gradient, corresponding to a reduced second order derivative tensor that is often needed in optimization algorithms. Hence we will address more specifically the task of developing schemes for the computation of higher-order forward directional sensitivities

$$\dot{\mathbf{w}}^{(m)}(\tau,\mathbf{d};\tau_0,\mathbf{y_0},\mathbf{p}) := \frac{\mathrm{d}^m\mathbf{y}(\tau;\tau_0,\mathbf{y_0},\mathbf{p})}{\mathrm{d}(\mathbf{y_0},\mathbf{p})^m}\,\mathbf{d}^m \in \mathbb{R}^{n_x+n_z} \tag{6.35}$$

of order m for a direction $\mathbf{d} \in \mathbb{R}^{n_x+n_z+n_p}$. Here we understand the multiplication with $\mathbf{d}^m$ as tensor contractions. The second kind of schemes we develop computes higher-order forward/adjoint sensitivities

$$\dot{\bar{\mathbf{w}}}^{(m)}(\tau,\bar{\mathbf{y}},\mathbf{d};\tau_0,\mathbf{y_0},\mathbf{p}) \;:=\; \bar{\mathbf{y}}^T\frac{\mathrm{d}^{m+1}\mathbf{y}(\tau;\tau_0,\mathbf{y_0},\mathbf{p})}{\mathrm{d}(\mathbf{y_0},\mathbf{p})^{m+1}}\,\mathbf{d}^m \in \mathbb{R}^{n_x+n_z+n_p} \tag{6.36}$$

of order $k+1$ for a forward direction $\mathbf{d}$ and an adjoint direction $\bar{\mathbf{y}}$. From suitable sets of these directional sensitivities any part or also the whole higher-order sensitivity tensor can be obtained, either directly or using exact interpolation, as explained in the Section 2.4.3. Before we derive in the following the forward directional variational DAE-IVP that can be used to compute the forward directional sensitivity (6.35) of order m we state the set partition based version [Joh02] of Faà di Bruno's formula [FdB57], which is needed in the derivations.

Lemma 6.4 (Higher-order chain rule)
Let $\psi : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ and $\chi : \mathbb{R} \to \mathbb{R}^{n_1}$ be sufficiently smooth. Then it holds for $t \in \mathbb{R}$ and $m \in \mathbb{N}$ that

$$\frac{\mathrm{d}^m\psi(\chi(t))}{\mathrm{d}t^m} = \sum_{\pi\in\Pi(m)} \psi^{|\pi|}(\chi(t))\prod_{i=1}^{m}(\chi^{(i)}(t))^{r_i}, \tag{6.37}$$

where $\Pi(m)$ is the set of all partitions of $\{1,2,\ldots,m\}$, $|\pi|$ the number of blocks in a partition π and r_i the number of blocks within π containing exactly i elements. The superscripts $|\pi|$ and (i) denote here the derivative degree of ψ and χ, respectively.

With the help of this lemma and the definition

$$\mathbf{v}^{\mathbf{d},(j)}(\tau) := \begin{pmatrix} \frac{\mathrm{d}^j \mathbf{y}(\tau)}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^j} \cdot \mathbf{d}^j \\ \frac{\mathrm{d}^j \mathbf{p}}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^j} \cdot \mathbf{d}^j \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{w}}_{\mathbf{x}}^{(j)}(\tau,\mathbf{d}) \\ \dot{\mathbf{w}}_{\mathbf{z}}^{(j)}(\tau,\mathbf{d}) \\ \delta_{1j}\,\mathbf{d_p} \end{pmatrix} \in \mathbb{R}^{n_x+n_z+n_p}, \tag{6.38}$$

where δ_{ij} is the Kronecker-delta, we can derive the higher-order forward directional variational DAE for the computation of forward sensitivities of type (6.35).

Higher-order forward directional variational DAE

We obtain the higher-order forward directional variational DAE-IVP of order m by forming the m-th derivative of the nominal IVP (6.1) with respect to initial values and parameter in a direction $\mathbf{d} = (\mathbf{d}_{\mathbf{x}}^T, \mathbf{d}_{\mathbf{z}}^T, \mathbf{d}_{\mathbf{p}}^T)^T \in \mathbb{R}^{n_x+n_z+n_p}$. We can write the forward directional variational DAE of order m (only indicating the time dependency) with the help of (6.37) and (6.38) as

$$\frac{\mathrm{d}^m(\mathbf{A}(\tau)\frac{\partial \mathbf{x}(\tau)}{\partial \tau})}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^m} \cdot \mathbf{d}^m = \sum_{\pi \in \Pi(m)} \frac{\partial^{|\pi|}\mathbf{f}(\tau)}{\partial(\mathbf{x},\mathbf{z},\mathbf{p})^{|\pi|}} \prod_{j=1}^{m} (\mathbf{v}^{\mathbf{d},(j)}(\tau))^{r_j} \tag{6.39a}$$

$$0 = \sum_{\pi \in \Pi(m)} \frac{\partial^{|\pi|}\mathbf{g}(\tau)}{\partial(\mathbf{x},\mathbf{z},\mathbf{p})^{|\pi|}} \prod_{j=1}^{m} (\mathbf{v}^{\mathbf{d},(j)}(\tau))^{r_j} - \theta(\tau)\frac{\partial^m \mathbf{g}(\tau_0)}{\partial(\mathbf{x},\mathbf{z},\mathbf{p})^m} \cdot \mathbf{d}^m \tag{6.39b}$$

with the initial values

$$\dot{\mathbf{w}}^{(j)}(\tau_0,\mathbf{d}) = \delta_{1j} \begin{pmatrix} \mathbf{d_x} \\ \mathbf{d_z} \end{pmatrix}. \tag{6.39c}$$

Also here the multiplications with $\mathbf{d}^j$ respectively $\mathbf{v}^{\mathbf{d},(j)}$ are to be understood as tensor contractions. From the Leibniz rule we obtain for the left-hand term in (6.39a)

$$\begin{aligned} \frac{\mathrm{d}^m(\mathbf{A}(\tau)\frac{\partial \mathbf{x}(\tau)}{\partial \tau})}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^m} \cdot \mathbf{d}^m &= \sum_{j=0}^{m} \left(\frac{\mathrm{d}^j \mathbf{A}(\tau)}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^j} \left(\frac{\mathrm{d}^{m-j}\frac{\partial \mathbf{x}(\tau)}{\partial \tau}}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^{m-j}} \cdot \mathbf{d}^{m-j} \right) \right) \cdot \mathbf{d}^j \\ &= \sum_{j=0}^{m} \left(\frac{\mathrm{d}^j \mathbf{A}(\tau)}{\mathrm{d}(\mathbf{x_0},\mathbf{z_0},\mathbf{p})^j} \frac{\partial \dot{\mathbf{w}}^{(m-j)}(\tau,\mathbf{d})}{\partial \tau} \right) \cdot \mathbf{d}^j. \end{aligned} \tag{6.40}$$

From the formulas (6.39) and (6.40) we learn several things. First, we observe that for $m = 1$ we indeed obtain the directional forward variational DAE-IVP given in (6.8), in this case for one sensitivity direction. Besides that, the m-th order equation is linear. Furthermore, we see that the equation contains in general besides the nominal solution $\mathbf{y}(\tau)$ also terms involving $\dot{\mathbf{w}}^{(j)}(\tau,\mathbf{d})$ for all $1 \leq j \leq m$. Hence the m-th order sensitivity cannot be computed "isolated", but only in connection with the directional sensitivities in direction $\mathbf{d}$ of all lower orders $1 \leq j < m$, which also have to be computed by forming and solving the corresponding equations. This can be done

like in the first order case either simultaneously or in a staggered way. Note that a general higher-order analogon for the adjoint variational DAE cannot be obtained straightforward.

Until today to the best of our knowledge only very few integration codes are available at all that are able to generate second order sensitivity information for ODEs or DAEs in an automatic way. These are the IND-based BDF code `DAESOL-II` presented in this thesis, supporting second order directional forward and forward/adjoint sensitivities, the earlier IND-based BDF code `DAESOL`, which has been extended by Bauer [Bau99] for the computation of second order forward sensitivities, as well as the latest versions of the BDF codes CVODES and IDAS, computing directional second order forward/adjoint sensitivities from a forward/adjoint variational DAE system presented by Özyurt and Barton [OB05] for ODEs. Alternatively, the nominal IVP could always be augmented manually by the corresponding first and second order forward variational DAE problem and this augmented system then can be solved together using an arbitrary integrator code. However, then usually the structure of the problem is not adequately exploited, which leads to a very high integration effort. There exists so far one implementation besides `DAESOL-II` that can generate arbitrary order forward sensitivities. It is described by Barrio [Bar06] and is based on an explicit Taylor series integration method. Hence it cannot be efficiently applied to stiff systems. The generation of arbitrary order forward/adjoint sensitivities is until now only supported by `DAESOL-II`.

In the following we develop the first numerical schemes based on IND that allow the computation of arbitrary order directional forward and forward/adjoint sensitivities. The key to the derivation of flexible efficient schemes that work not only for the second order but also for arbitrary orders is the combination of the IND principle with the AD technique of univariate Taylor Coefficient (TC) propagation explained in Section 2.4.3. Note that we will only develop the higher-order IND schemes here for higher-order analoga of the iterative IND schemes. However, also a version corresponding to the direct schemes can be derived, using Taylor coefficient propagation rules for implicitly defined functions, which are described in [Ked80, WSW10]. As we will address in the comparison of the effort of the different IND schemes, the direct schemes are more efficient than the iterative schemes only in very special cases, which do not occur in the framework of the thesis. Hence we skip the derivation, which can be done however in complete analogy to the derivation of the iterative schemes presented here.

6.5.1 Higher-order forward schemes

To derive the higher-order (iterative) forward IND scheme of order m, we use again the elemental representation of the discretization scheme of the nominal IVP given by the Equations (6.25). Other than in the first order scheme we now propagate a univariate Taylor polynomial of order m through this elemental representation to propagate the derivative information through the integration process. In other words, we perform a forward TC sweep of order m for the elemental representation of the integration process. The nominal integration constitutes in this framework the propagation of the zero order Taylor coefficients, which can be done independently from the propagation of the coefficients of higher order. Hence the nominal integration process can remain

unchanged and we use the trajectory values as zero order Taylor coefficients in the calculations of the forward TC sweep. We initialize, like we would for an ordinary function, the TCs for the sensitivity propagation process based on the direction $\mathbf{d}$ as

$$\begin{aligned} \mathbf{X_0} &= \begin{bmatrix} \mathbf{x_0} & \mathbf{d_x} & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n_x \times (m+1)}, \\ \mathbf{Z_0} &= \begin{bmatrix} \mathbf{z_0} & \mathbf{d_z} & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n_z \times (m+1)}, \\ \mathbf{P} &= \begin{bmatrix} \mathbf{p} & \mathbf{d_p} & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n_p \times (m+1)}. \end{aligned} \tag{6.41}$$

We then denote the intermediate Taylor coefficients belonging to the gridpoint τ_n with $\mathbf{Y_n} = \begin{bmatrix} \mathbf{y_{n,0}} & \mathbf{y_{n,1}} & \cdots & \mathbf{y_{n,m}} \end{bmatrix}$, where $\mathbf{y_{n,0}} \equiv \mathbf{y_n}$.

Because the operations (6.25a)-(6.25c) in the integration, that do not involve the evaluation of the model functions, are all just additions or scalar multiplications, the TC propagation rules to obtain the corresponding values $\mathbf{Y_n^P}$ and $\mathbf{X_n^{CC}}$ are very easy to implement and the additional effort for them compared to a nominal integration scales linearly with m. Also the decomposition of $\mathbf{M_n}$ in (6.25d) can be reused, only the number of right-hand side vectors for which the system has to be solved increases from 1 to $m+1$ in the higher-order forward scheme. To understand this we remember that the solution of a linear system for a fixed matrix also only constitutes a series of scalar multiplications, divisions and additions operating on the right-hand side data. The main numerical effort is therefore to be expected in the propagation through $\mathbf{f_n^{BDF}}$, involving the usually nonlinear model functions of the DAE.

The (simultaneous) forward IND-TC scheme for an arbitrary order m is given in algorithmic form by Algorithm 6.7. Note that this scheme can also be derived in an deferred version as well as be extended for the parallel propagation of several directional sensitivities. In this case, the zero order coefficients, i.e., the nominal trajectory, which are identical for all directions, need to be computed and stored only once. Note also that the operations performed in the first order forward IND-TC scheme are equal to the operations of the "ordinary" iterative first order IND scheme, i.e., both schemes are the same.

By the use of an m-th order forward IND-TC sweep we clearly obtain the (scaled) m-order directional derivatives of the numerical solution within machine precision. However the relation between the intermediate TCs in this scheme and the solution trajectory of the m-th order forward directional variational DAE-IVP (6.39) approximating the directional forward sensitivities (6.35) is not obvious. Hence we formulate the following proposition.

Proposition 6.5 (Forward IND-TC sweep versus directional variational DAE)
Let $\mathbf{d} = (\mathbf{d_x}^T, \mathbf{d_z}^T, \mathbf{d}_p^T)^T \in \mathbb{R}^{n_x+n_z+n_p}$ be a forward sensitivity direction. Denote the TC at gridpoint τ_n that has been computed using the higher-order forward IND scheme of order m given in Algorithm 6.7 with $\mathbf{Y_n}$. Furthermore define for a given TC $\mathbf{Y}$ the scaled quantities $\tilde{\mathbf{Y}} = \begin{bmatrix} 0!\,\mathbf{y_0} & 1!\,\mathbf{y_1} & 2!\,\mathbf{y_1} & \cdots & m!\,\mathbf{y_m} \end{bmatrix}$. Finally denote with

$$\dot{\mathbf{W}}^{(j)}(\tau_n, \mathbf{d}) := \left[\mathbf{y_n}(\tau_n) \quad \dot{\mathbf{w}}_\mathbf{n}^{(1)}(\tau_n, \mathbf{d}) \quad \cdots \quad \dot{\mathbf{w}}_\mathbf{n}^{(j)}(\tau_n, \mathbf{d}) \right]$$

the grouping of the nominal trajectory values and the forward sensitivities up to order j at the gridpoint τ_n that have been obtained by the solution of the forward directional variational DAE-IVPs (6.39) up to order j using the same BDF discretization scheme as for the solution of the

Algorithm 6.7: Arbitrary order forward IND-TC scheme (simultaneous)

Input: t_0, t_f, h_0, initial values $\mathbf{y_0}$, parameter $\mathbf{p}$, sensitivity direction $\mathbf{d}$, sensitivity order m
Output: Taylor coefficients $\mathbf{Y_N}$ containing nominal solution $\mathbf{y_N}$ and scaled forward sensitivities $\frac{1}{j!}\dot{\mathbf{w}}_{\mathbf{N}}^{(j)}$ for $1 \leq j \leq m$.

set $k_0 = 1$, $n = 0$;
initialize with Taylor coefficients (6.41) according to initial values and sensitivity direction $\mathbf{d}$;
while t_f *not reached* **do**
  compute $\mathbf{y_{n+1}}$ by nominal integration step for h_n, k_n including s_n Newton-like iterations using matrix $\mathbf{M_n}$ to solve the corrector equation;
  if *step accepted* **then**
    `// corrector equation successfully solved,`
    `// error estimation accepted`
    propagate Taylor polynomials through predictor (6.25a) and corrector constant (6.25b) computation ;
    using $\mathbf{y_{n+1}}$, h_n and $\mathbf{M_n^{-1}}$, propagate Taylor coefficients through s_n Newton-like iterations by propagation through (6.25c)/(6.25d) to obtain $\mathbf{Y_{n+1}}$;
    $t_{n+1} = t_n + h_n$;
    determine h_{n+1} and k_{n+1} for next step;
    $n = n + 1$;
  else
    `// corrector equation solving failed or`
    `// error estimation too large`
    update Jacobian approximation $\mathbf{M_n}$ according to monitor strategy or reduce stepsize h_n;
  end
end
$N = n$;

nominal DAE-IVP.

Then it holds

$$\dot{\mathbf{W}}^{(m)}(\tau_n, \mathbf{d}) = \tilde{\mathbf{Y}}_{\mathbf{n}} \qquad \forall n \in \mathbb{N}. \tag{6.42}$$

In other words, the higher-order forward IND scheme of order m is equivalent to solving the set of scaled directional forward variational DAEs up to order m simultaneously using the same BDF discretization scheme as for the computation of the nominal trajectory.

Proof:

Note that for $m = 0$ the claim is trivial and as the first order forward TC based IND scheme is equal to the ordinary iterative forward IND scheme, the case $m = 1$ is also true. To prove the general case, we first apply the BDF discretization scheme to the directional variational DAE (6.39). With the abbreviations $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(j)} := \dot{\mathbf{w}}^{(j)}(\tau_{n+1}, \mathbf{d})$ (analogously for the other quantities and functions),

$\mathbf{v} := (\mathbf{x_0}, \mathbf{z_0}, \mathbf{p})$ and the higher-order sensitivity corrector constant $\dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x,CC}(j)} := \sum_{i=1}^{k_n} \alpha_{i,n}^C \dot{\mathbf{w}}_{\mathbf{n+1-i}}^{\mathbf{x}(j)}$ we obtain for the BDF step n with order k_n the scheme

$$\begin{aligned} 0 &= \mathbf{A_{n+1}} \left(\alpha_{0,n}^C \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m)} + \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x,CC}(m)} \right) + \sum_{j=1}^{m} \left(\frac{\mathrm{d}^j \mathbf{A_{n+1}}}{\mathrm{d}(\mathbf{x_0}, \mathbf{z_0}, \mathbf{p})^j} \left(\alpha_{0,n}^C \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m-j)} + \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x,CC}(m-j)} \right) \right) \cdot \mathbf{d}^j \\ &\quad - h_n \sum_{\pi \in \Pi(m)} \frac{\partial^{|\pi|} \mathbf{f_{n+1}}}{\partial \mathbf{v}^{|\pi|}} \prod_{j=0}^{m} (\mathbf{v}_{\mathbf{n+1}}^{\mathbf{d},(j)})^{r_j} \qquad (6.43a) \\ 0 &= \sum_{\pi \in \Pi(m)} \frac{\partial^{|\pi|} \mathbf{g_{n+1}}}{\partial \mathbf{v}^{|\pi|}} \prod_{j=0}^{m} (\mathbf{v}_{\mathbf{n+1}}^{\mathbf{d},(j)})^{r_j} - \theta(\tau_{n+1}) \frac{\partial^m \mathbf{g_0}}{\partial \mathbf{v}^m} \cdot \mathbf{d}^m \qquad (6.43b) \end{aligned}$$

for the computation of the new sensitivity value $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}$. We define the right-hand side of the system now as a function

$$\mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}(m)}(\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}; \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x,CC}(m)}, \dot{\mathbf{W}}_{\mathbf{n}}^{(m-1)}, \ldots, \dot{\mathbf{W}}_{\mathbf{n+1-k_n}}^{(m-1)}, \mathbf{p}, \mathbf{x_0}, \mathbf{z_0}), \qquad (6.44)$$

which depends besides on the value $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}$ to be computed also on k_n historical values of the m-th order sensitivity, the current and k_n historical values of the directional sensitivities of all lower orders and also on the initial values and parameter.

To see now the equivalence between the m-th order forward IND-TC sweep and the solution of these systems up to order m we first analyze the Jacobian of the system (6.43) with respect to the values $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}$ that are to be computed. Applying the higher-order chain rule (6.37) also to the terms involving the derivatives of the matrix $\mathbf{A_{n+1}}$ and considering that the partition $\tilde{\pi} := \{12 \ldots m\} \in \Pi(m)$ for which $|\tilde{\pi}| = 1$, $r_1 = \ldots = r_{m-1} = 0$ and $r_m = 1$ is the only one leading to terms containing $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}$ in the summations above, we can write the system as

$$\begin{aligned} 0 &= \alpha_{0,n}^C \mathbf{A_{n+1}} \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m)} + \left(\frac{\partial \mathbf{A_{n+1}}}{\partial(\mathbf{x},\mathbf{z})} (\alpha_{0,n}^C \mathbf{x_{n+1}} + \mathbf{x}_{\mathbf{n+1}}^{\mathbf{CC}}) \right) \begin{pmatrix} \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m)} \\ \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{z}(m)} \end{pmatrix} \\ &\quad - h_n \frac{\partial \mathbf{f_{n+1}}}{\partial(\mathbf{x},\mathbf{z})} \begin{pmatrix} \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m)} \\ \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{z}(m)} \end{pmatrix} + \mathcal{R}^{\mathbf{x}} \qquad (6.45a) \\ 0 &= \frac{\partial \mathbf{g_{n+1}}}{\partial(\mathbf{x},\mathbf{z})} \begin{pmatrix} \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{x}(m)} \\ \dot{\mathbf{w}}_{\mathbf{n+1}}^{\mathbf{z}(m)} \end{pmatrix} + \mathcal{R}^{\mathbf{z}}, \qquad (6.45b) \end{aligned}$$

where the terms $\mathcal{R}^{\mathbf{x}}$ and $\mathcal{R}^{\mathbf{z}}$ do not depend on m-th order terms $\dot{\mathbf{w}}_{\mathbf{i}}^{(m)}$. Hence we see that for any order m the Jacobian in the BDF predictor-corrector scheme for the solution of the variational DAE of order m is the same as the Jacobian (5.44) for the computation of the nominal trajectory. Hence it is sensible to use the same iteration matrix for the treatment of the nominal trajectory and the variational DAEs of higher order.

It remains to be shown that for an arbitrary integration step n it holds that if the historical values and lower order sensitivities $\dot{\mathbf{W}}_{\mathbf{n+1}}^{(m-1)}$, $\dot{\mathbf{W}}_{\mathbf{n}}^{(m)}$, ..., $\dot{\mathbf{W}}_{\mathbf{n+1-k_n}}^{(m)}$ fulfill the induction assumption

(6.42), then also the newly computed sensitivity value $\dot{\mathbf{w}}_{\mathbf{n+1}}^{(m)}$ does. The proposition then follows by induction over the sensitivity order m and the integration steps n from the fact that the initial Taylor coefficients (6.41) for the higher-order forward IND scheme and the initial sensitivities (6.39c) of the forward variational DAE-IVP up to order m trivially fulfill (6.42).
First we note that as we use the same discretization scheme for nominal and variational problem, the computation of the predictor and corrector constant for the variational DAEs uses the same coefficients as for the nominal trajectory. As a result, the corresponding values $\mathbf{Y}_{\mathbf{n}}^{\mathbf{P}}$ and $\mathbf{X}_{\mathbf{n+1}}^{\mathbf{CC}}$ computed by Taylor propagation through (6.25a) and (6.25b) still fulfill the analogon of (6.42) for intermediate quantities. Likewise we can argue for the Newton iterates and the corresponding quantities in the forward IND-TC scheme obtained by Taylor propagation through (6.25c). Hence, it is sufficient to show that the m-th order coefficient of the result of the Taylor propagation through $\mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}}$ is equal to $\frac{1}{m!}\mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}(m)}$ in the solution of the variational DAE. Comparing (6.22) and (6.43) shows, however, that $\mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}(m)}$ can be understood as the m-th order directional derivative of $\mathbf{f}_{\mathbf{n}}^{\mathbf{BDF}}$, such that the last claim follows from the induction assumption and Proposition 2.13 on page 40.

□

Summarizing, we see that the combination of IND and forward TC propagation allows the derivation of efficient schemes for the generation of arbitrary order forward sensitivities and eliminates the tedious and error-prone process of forming and coding the corresponding directional variational DAEs for an arbitrary order. Furthermore the IND-TC approach encapsulates the complex dependencies between the sensitivities of different order in the propagation process through the model functions, rendering the scheme on the integrator code level quite simple. This ensures also an efficient exploitation of the problem structure, as by propagation through the model functions the sensitivities of all orders can here be treated simultaneously without blowing up the system size. This has to be achieved manually in case of using the approach involving the variational DAEs by adapting the integration scheme to use a staggered computation of the different orders. Even then this staggered computation of the different orders leads to more overhead and a higher effort compared to the IND-TC approach. Nevertheless we obtain by using the forward IND-TC scheme approximations of the forward sensitivities at every gridpoint that are guaranteed to converge against the true forward sensitivities with the same order as the nominal trajectory.

6.5.2 Higher-order forward/adjoint schemes

In this section we present how the combination of IND and TC propagation can be used to obtain approximations of forward/adjoint sensitivities $\dot{\bar{\mathbf{w}}}^{(m)}(\tau_0, \bar{\mathbf{y}}, \mathbf{d})$ of an arbitrary order. To derive the forward/adjoint IND-TC scheme of order m for the forward direction $\mathbf{d}$ and the adjoint direction $\bar{\mathbf{y}}$, we use once more the elemental representation of the discretization scheme of the nominal IVP given by the Equations (6.25). Like in the first order (iterative) adjoint scheme, we assume that the nominal integration has been performed and the discretization scheme, i.e., stepsizes, BDF orders of the steps, used iteration matrices and iteration counts, has been stored. Other than for the first order adjoint schemes we additionally assume that not only the values of the nominal trajectories at the gridpoints have been stored, but also the intermediate Taylor

coefficients $\mathbf{Y_n}$ of a forward IND-TC sweep of order m for the forward direction $\mathbf{d}$. Based on this information and the already derived first order adjoint scheme, a higher-order adjoint IND-TC sweep can be obtained to compute forward/adjoint sensitivities. For this we initialize the adjoint Taylor coefficients $\bar{\mathbf{Y}} = \begin{bmatrix}\bar{\mathbf{y}} & \mathbf{0} & \dots & \mathbf{0}\end{bmatrix} \in \mathbb{R}^{n_y \times (m+1)}$ based on the adjoint direction. Afterwards the operations of the first order adjoint sweep given by (6.31) are performed in Taylor arithmetic of order m, as explained in Section 2.4.3. Doing this we obtain, analogously to Proposition 2.18 for ordinary functions, scaled forward/adjoint sensitivities. The algorithmic form of the adjoint IND-TC sweep for an arbitrary order m is given in Algorithm 6.8. We keep in mind that the order m here refers to the order of the propagated adjoint Taylor polynomials, but the derivative degree occurring in $\dot{\tilde{\mathbf{w}}}^{(m)}$ is $m+1$. Also this algorithm can be extended to propagate several forward and adjoint directions simultaneously. Like in the iterative first order adjoint IND scheme, we can reuse the factorization of the iteration matrices from the solution of the nominal problem.

Algorithm 6.8: Arbitrary order forward/adjoint IND-TC scheme

Input: Discretization scheme stored from a forward IND-TC sweep of order m (stepsizes, orders, iteration matrix factorizations, iteration counts, Taylor coefficients), adjoint sensitivity direction $\bar{\mathbf{y}}$.

Output: Adjoint Taylor coefficients $\bar{\mathbf{Y}}_\mathbf{0}, \bar{\mathbf{P}}$ containing scaled forward/adjoint sensitivities $\frac{1}{j!}\dot{\tilde{\mathbf{w}}}^{(j)}(\tau_\mathrm{e}, \bar{\mathbf{y}}, \mathbf{d})$, $0 \le j \le m$.

initialize sensitivities with $\bar{\mathbf{Y}}_\mathbf{N} = \begin{bmatrix}\bar{\mathbf{y}} & \mathbf{0} & \dots & \mathbf{0}\end{bmatrix}$;
$n = N - 1$;
while $n \ge 0$ **do**
  $\bar{\mathbf{Y}}^{(\mathbf{s_n})}_{\mathbf{n+1}} = \bar{\mathbf{Y}}_{\mathbf{n+1}}$;
  get $\mathbf{Y_{n+1}}$, t_n, h_n, k_n, $\mathbf{M_n^{-1}}$, s_n from stored discretization scheme;
  for $i = s_n - 1 : 0$ **do**
    knowing (6.31d), solve $\mathbf{\Lambda} = \mathbf{M_n^{-T}}\bar{\mathbf{Y}}^{(\mathbf{i+1})}_{\mathbf{n+1}}$;
    using $\mathbf{\Lambda}$, compute respectively increment $\bar{\mathbf{Y}}^{(\mathbf{i})}_{\mathbf{n+1}}, \bar{\mathbf{P}}, \bar{\mathbf{X}}^{\mathbf{CC}}_{\mathbf{n+1}}, \bar{\mathbf{X}}_\mathbf{0}, \bar{\mathbf{Z}}_\mathbf{0}$ by applying (6.31c), (6.31e), (6.31f), (6.31g) and (6.31h) in Taylor arithmetic of order m;
  end
  $\bar{\mathbf{Y}}^{\mathbf{P}}_{\mathbf{n+1}} = \bar{\mathbf{Y}}^{(\mathbf{0})}_{\mathbf{n+1}}$;
  propagate corrector constant dependency backwards using (6.31b);
  propagate predictor dependency backwards using (6.31a);
  `// all contributions of` $\mathbf{y_n}$ `taken into account,`
  `// value of` $\bar{\mathbf{Y}}_\mathbf{n}$ `is final`
  $n = n - 1$;
end

Remark 6.6 (Interpretation of intermediate adjoint Taylor coefficients)
Similar to the first order adjoint IND case, we have convergence of the (rescaled) propagated adjoint Taylor coefficients obtained by the forward/adjoint IND-TC scheme to the corresponding

analytical forward/adjoint sensitivities. However, the intermediate adjoint Taylor coefficients of order greater than zero cannot be interpreted as an approximation to an forward/adjoint sensitivity of type (6.36). This is here not caused by the application of the IND approach, but a more general property which would also occur if a corresponding forward/adjoint variational DAE would be derived and solved, as, e.g., done for the second-order case in [OB05]. This can be understood by realizing that for an intermediate gridpoint τ_n with $n > 0$ in general the corresponding stored Taylor coefficients at this gridpoint obtained by the preceding forward IND-TC scheme of order m will not have the form $\mathbf{Y_n} = \begin{bmatrix}\mathbf{y_n} & \hat{\mathbf{d}} & \mathbf{0} & \dots & \mathbf{0}\end{bmatrix}$. Only in this case the adjoint Taylor coefficients at τ_n with order greater than zero could be interpreted as a forward/adjoint sensitivity $\dot{\bar{\mathbf{w}}}^{(m)}(\tau_\mathrm{e}, \bar{\mathbf{y}}, \hat{\mathbf{d}}; \tau_n, \mathbf{y_n}, \mathbf{p})$. It should be emphasized that analogously to the first order case even this is only true for the IND-TC scheme, if the integration process would have been restarted in τ_n or the truly implicit scheme of order 1 had been used in this step. Note furthermore that the zero order components of the adjoint Taylor coefficients are identical to the intermediate values of an iterative first order adjoint IND sweep and hence the observations in Remark 6.2 are also valid for them.

6.6 Comparison of the different IND-based schemes

After presenting the different first and higher-order forward and adjoint IND-based schemes for sensitivity generation, we analyze in this section the computational effort and the memory demand of the different IND approaches. We compare the approaches among themselves and also to the solution of the nominal IVP as well as to the repetition of the nominal integration based on a previously stored discretization scheme, a so-called integration replay.

6.6.1 Computational effort

The two main computational tasks in the solution of the nominal IVP and also in the IND sweeps are the computations related to the interpolation process for predictor and corrector polynomial on the one hand and the solution of the corrector equations on the other hand. For the solution of the nominal IVP additional computations, related to the error control as well as to the stepsize and order strategy, are needed.

Considering the first task, suppose the predictor and corrector coefficients are known. Then the computational costs to form the predictor and the corrector constant via (6.25a) and (6.25b) are theoretically identical for the nominal solution and a replay of the integration. They grow linearly with the number of states n_y and the number of integration steps N.

In practice, some more effort is needed during the nominal solution for the computation of the interpolation coefficients, the update of the modified divided differences scheme, the computation of higher-order modified differences for error estimation and the scaling. The complexity of these additional computations is either bounded by a constant, or grows linearly with n_y and the number of integration steps, including rejected steps. For large scale systems usually this additional effort is not significant compared to the overall effort.

The effort for the computation of the corresponding quantities in the sensitivity propagation using the first order forward and adjoint IND schemes by the operation (6.27a) and (6.27b) (respectively their adjoint variants) is equal to the one for the nominal solution, if one directional sensitivity is computed. Otherwise it scales linearly with the number of sensitivity directions.

The effort for the operations related to the predictor and corrector constant quantities in the higher-order forward and forward/adjoint IND-TC schemes additionally grows linearly with the order m.

Considering the second task, it can be said that for large scale systems the solution of the corrector equations is the most time-consuming part. It can be divided into the cost of building the iteration matrix, the matrix factorizations and the cost for the actual solution of the linear systems based on the factorized matrix.

During the nominal solution, this effort is given by the needed decompositions of the iteration matrix and the rebuilds of the Jacobian (5.44) according to the monitor strategy. The complexity of one of these decompositions depends strongly on the sparsity structure of the iteration matrix. In the worst case of a dense matrix, it will grow with n_y^3. A matrix rebuild brings the additional expenses of computing the complete Jacobians of the model functions. Again, the sparsity structures of the Jacobians determine here the effort. The worst case assumption of a dense matrix and the complexity bounds from AD theory (see, e.g., [Gri00]) lead to the theoretical bound of approximately $2.5\,n_y$ times the time of a model function evaluation. Fortunately, usually not in every step a decomposition or rebuild is needed. Furthermore, in every Newton-like iteration occurring in the nominal integration, the solution of a linear system and one evaluation of the model functions are needed. This is also true for an integration replay. In the worst case of dense matrices, the complexity of the solution of the linear systems is n_y^2.

During integration replays and iterative IND-based forward and adjoint sweeps of any order, no decompositions or rebuilds of the iteration matrix are needed. In the first order schemes for every Newton-like iteration and for every sensitivity direction one linear system solution and one directional forward or adjoint directional derivative of the model functions is needed. Here, using AD theory, the effort for one model derivative can be bounded in the forward mode by 2.5 times a function evaluation, in the adjoint mode by a factor of 4. In the higher-order schemes, the solution of m linear systems is needed per Newton iteration and sensitivity direction, as well as the propagation of a Taylor polynomial of degree m through the model functions. The effort for this scales with m^2 times the effort of a model function evaluation.
In the first order direct IND schemes, for every integration step one matrix rebuild, one linear system solution and one directional model function derivative are needed. This shows that the direct schemes are only of interest if the cost of a matrix factorization is not too high compared to a linear system solution and a model function derivative.
In the deferred forward IND schemes, the backward IND schemes and the replay, there are additional costs for storing and loading the components of the discretization scheme. For now we assume that all these values can be kept in the main memory, such that this kind of effort is probably negligible.

In Table 6.1 we sum up the number of different operations needed in the different approaches where we refer to the deferred versions of the IND schemes. If we assume that for large scale systems

	Integration	Replay	IND iterative	IND direct	IND-TC order m
effort for error estimation, step-size and order control, rejected steps	yes	no	no	no	no
matrix rebuilds and decompositions	according to monitor strategy, usually $<< N$	0	0	N	0
model function evaluations	$\sum_{i=0}^{N-1} s_n$	$\sum_{i=0}^{N-1} s_n$	0	0	0
model function derivatives	0	0	$\sum_{i=0}^{N-1} s_n$	N	0
TC propagations trough model function	0	0	0	0	$\sum_{i=0}^{N-1} s_n$
solutions of linear systems	$\sum_{i=0}^{N-1} s_n$	$\sum_{i=0}^{N-1} s_n$	$\sum_{i=0}^{N-1} s_n$	N	$m \sum_{i=0}^{N-1} s_n$

Table 6.1: Comparison of the number of different types of operations needed to perform a nominal integration, an integration replay, a first order iterative respectively direct deferred forward or backward IND sweep for one sensitivity direction, as well as the forward or adjoint IND-TC sweep of order m for one respectively one pair of sensitivity directions. The operation counts are given in terms of the overall number of (accepted) integration steps N and the number of Newton-like iterations in each integration step s_n.

the model function and especially the model derivative evaluation dominates the integration process, we obtain immediately a conservative theoretically upper bound for the cost of a first order directional sensitivity using the iterative IND schemes: a directional first order forward sensitivity costs at most 2.5 nominal integrations, and a directional first order adjoint sensitivity at most 4 nominal integrations. However, in practice better values are to be expected, as for large scale problems the matrix factorization during nominal integration are also an expensive task. Note that if we sacrifice some accuracy and use a replay to obtain a first order forward sensitivity using the finite differences IND approach, the cost can obviously be bounded by 2 nominal integrations. But also in this case a better practical performance is expected. We confirm these estimations numerically on a scalable test problem in Section 9.1 on page 225.

6.6.2 Memory usage

To analyze the overall needed storage for the presented approaches, we split the memory consumption into two main parts: the temporary storage needed for the current nominal integration, replay or IND sweep, and the memory needed to store the discretization scheme for a later replay or IND sweep.

For the nominal integration the memory used during integration consists of the memory needed for the interpolation process, the solution of the corrector equation and for stepsize and order control. The latter is bounded by a small constant and can be neglected. The memory needed for the interpolation process and the solution of the linear systems in the Newton-like iterations, except the iteration matrix and the model function Jacobians, grows linearly in $n_y + n_p$ and can be bounded by $C := [2(k_{max} + 2) + 6]n_y + n_p$. The storage for the iteration matrix and the model function Jacobians (that need to be kept to perform the second step in the monitor strategy (cf. Section 5.3.3)) is in both cases bounded by two times the nonzero entries of the Jacobians plus n_y, which means in the worst case of dense matrices it is equal to $n_y^2 + n_y$.
A replay does not consume additional memory compared to the nominal integration, as one can use the actual iteration matrix factorization in the stored discretization scheme.
Deferred iterative first order forward and adjoint IND sweeps for one direction also consume less memory for the actual computation than a nominal integration. Again, the matrix factorizations can be used by referencing the stored scheme. The needed storage for intermediate sensitivity quantities grows linearly in $n_y + n_p$ and as the trajectory point y_{n+1} depends on at most $k_n + 1$ earlier values during a sweep only the corresponding number of sensitivity values has to be kept at a time. This leads in the end more or less to the same bound C on memory consumption as in the nominal case, however without the need for matrix storage. This is not the case if we perform a simultaneous first order forward IND sweep. Here we have approximately to double the linear bound C of the nominal integration. On the other hand, no storing of the discretization scheme is needed. Note that for all IND sweeps the memory consumption grows also linearly in the number of sensitivity directions treated in parallel. For the higher-order IND-TC sweeps the memory consumption grows also linearly with the order m.

The memory consumption for storing the discretization scheme depends in the first place on what is actually stored on the tape. Depending on whether a replay should be performed, a direct or iterative first order IND sweep or a higher-order forward or forward/adjoint IND-TC sweep, different information is required for each of these operations (see Table 6.2).
A replay needs the used stepsizes, orders, iteration matrices and iteration counts, while a deferred first order direct forward or direct adjoint sweep needs the used stepsizes, orders and the nominal trajectory values and parameter. The deferred first order iterative IND sweeps and the higher-order forward IND-TC sweep also need the iteration matrices used during nominal integration and the number of Newton-like iterations in each integration step. Finally, the higher-order adjoint IND-TC sweep of order m needs additionally the stored trajectory Taylor coefficients up to order m.
This means that to allow for a later replay, the size of the stored discretization scheme grows

linearly with the number of accepted integration steps N plus the needed storage for the iteration matrices, which depends on their sparsity structure and their overall number. For a direct IND sweep it grows linearly with $N + Nn_y + n_p$, and for the iterative IND sweeps and the higher-order forward IND-TC sweep linearly in $N + Nn_y + n_p$ plus the needed storage for the iteration matrices, which again depends on their sparsity structure and their overall number. Finally, for the higher-order adjoint IND-TC sweep of order m it grows linearly with $N + Nmn_y + mn_p$ plus the matrix storage.

	Replay	IND direct	IND iterative & fwd. IND-TC	fwd/adj IND-TC
stepsizes, orders	yes	yes	yes	yes
nominal trajectory values and parameter	no	yes	yes	yes
trajectory Taylor coefficients	no	no	no	yes
used iteration matrices and number of Newton-like iterations	no	no	yes	yes

Table 6.2: Overview over the types of information that has be stored on the tape to enable the different kinds of deferred operations.

Remark 6.7 (Checkpointing)
As we have seen, to perform a replay or a deferred IND sweep, some information of the nominal integration or a previous IND-TC sweep has to be stored on the tape, which for large scale problems and long time horizons might become quite large. It should be noted here that applying IND in the presented way to derive the sensitivity generation schemes instead of treating the integrator as a black box leads automatically to a "vertical checkpointing". In contrast to the checkpointing strategies commonly used for integration codes that operate on the time horizon (horizontal checkpointing), here we end up with a hierarchical scheme: The intermediate values of the evaluations of the model functions $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{A}$ are not stored, but recalculated as needed in an adjoint IND sweep. This reduces the overall memory consumption in IND-based schemes drastically compared to the application of an black-box AD approach. In the context of a multiple shooting method also the partition of the whole time horizon into subintervals leads to a so-called "natural checkpointing". For some problems, for which the tape does not fit into the main memory any more, the investigation of other checkpointing strategies, as, e.g., described by Walther [Wal00], may be interesting. For example, the factorizations of the iteration matrices are very expensive, in a sequential checkpointing scheme these could preferably be kept and some parts of the nominal trajectory might be dropped and recomputed on demand.

6.7 Other sensitivity related topics

After presenting and comparing the different IND-based approaches for first and higher-order sensitivity generation we now address in this section some more specific topics related to sensitivity generation and, if applicable, their implementation in the packages `DAESOL-II` and `SolvIND`.

6.7.1 Exact interpolation

The presented higher-order forward and forward/adjoint IND schemes allow the efficient computation of univariate directional sensitivities of arbitrary order. Sometimes however, also entries or parts of the higher-order sensitivity tensor might be needed that are not directly computable as such an univariate directional sensitivity. To obtain them we can use the method of exact interpolation presented in Section 2.4.3 for ordinary functions. The idea is directly transferable to the IND-TC context and can be used here to compute any entry of a sensitivity tensor of a given order based on a suitable set of propagated rays, as described in the Propositions 2.19 and 2.21. To simplify higher-order sensitivity generation as well as the usage of exact interpolation for the user, `SolvIND` provides a ray manager that computes, based on a given set of directions and a given set of multi-indices describing the desired multivariate derivatives, the minimal set of needed rays and builds the corresponding initial Taylor coefficients for the IND-TC sweep.
Additionally, the ray manager can be used to execute the exact interpolation after the forward or forward/adjoint IND sweep has been performed. It takes the propagated forward or adjoint Taylor coefficients and interpolates the desired multivariate sensitivities based on the multi-indices passed earlier. Note that multivariate sensitivities of different orders can be computed simultaneously by only one execution of an IND-TC scheme.

6.7.2 Tape management

As mentioned in Section 6.6, to enable, e.g., a later replay, or any IND-based adjoint sweep, specific parts of the discretization scheme and of the trajectory information have to be stored on a tape and need to be retrieved later. `DAESOL-II` contains a tape management system that allows the creation of an arbitrary number of distinct tapes (only limited by the available memory), as well as the possibility to store, access, overwrite or delete each of them independently. Also the kind of information to be stored on a tape can be selected. This allows the efficient use of `DAESOL-II` also in the context of adaptive optimization algorithms, including the lifted methods presented in this thesis and other (direct) multiple shooting based methods. In the context of these methods it is desirable, sometimes also essential, that several distinct tapes can be kept. In general one wants to keep here at least one tape for each subinterval of the partitioned time horizon of the underlying problem. Then one can perform, for example, efficient forward and adjoint sensitivity sweeps over the whole time horizon. In an adaptive framework it might also be of interest to reuse some of the tapes for replay and sensitivity computation during several optimization iterations, while others should be updated and overwritten. If the memory is not large enough, some of the tapes might be considered to be deleted and recomputed later. `DAESOL-II` allows all the needed

operations and furthermore supports these scenarios by keeping additional information on the quantities available on a specific tape as well as the tape size.

6.7.3 Continuous representation of sensitivities

The IND-based sensitivity generation schemes presented above aim mainly at the efficient computation of the derivative of the solution of the nominal IVP at the end time τ_e of the integration horizon with respect to initial values and parameter. In certain cases however, as already addressed in Section 5.3.9 for the values of the nominal trajectory, it is highly interesting to obtain also derivatives of trajectory values at timepoints inside the integration horizon with respect to initial values and parameter. For reason of stability and efficiency this should be possible without forcing the integration gridpoints to contain these points.
For the forward sensitivities computed by forward IND(-TC) schemes this can be achieved, because as described in Remark 6.1 and Proposition 6.5 they can be understood as solution trajectories of forward variational DAE-IVPs obtained by a BDF-discretization scheme. Hence we can obtain an error controlled continuous representation of the forward sensitivities in exactly the same way as for the nominal trajectory values in Section 5.3.9 on page 145, i.e., by using the interpolation polynomial of the underlying BDF method to interpolate sensitivity values at arbitrary timepoints in the integration horizon. In the case of higher-order forward sensitivities it is also possible to use exact interpolation on these interpolated sensitivities, if desired.
Note again that no corresponding natural (and hence error controlled) interpolation procedure exists in the IND framework for the adjoint sensitivities, i.e., the computation of the derivatives of the trajectory values at τ_e with respect to the values at an arbitrary time inside integration horizon. The only possibility here is to start the integration process at this timepoint. This is due to the fact that the adjoint IND(-TC) schemes cannot be interpreted as an BDF-method (or another consistent LMM) for the solution of the adjoint variational DAE, as mentioned in Remark 6.2.

6.7.4 Sensitivity injection

In the optimization context it is often needed to compute the derivatives of functions which do not only depend on the system states $\mathbf{y}(\tau_e)$ at the endpoint of the integration horizon, but also on system states inside the integration interval. Imagine here for example least-square objective terms penalizing the deviation of the state of the system from measurements or also point constraints on the system states at some given timepoints. The evaluation of these functions and also of their derivatives with respect to initial values and parameter of the DAE-IVP can be performed based on the continuous representation of the nominal trajectory and the forward sensitivities described in the Sections 5.3.9 and 6.7.3. This however is not always desirable, because in this way the first and higher-order adjoint IND(-TC) schemes could not be employed, but only forward IND schemes. One possibility to use adjoint schemes would be to choose each of the necessary timepoints as an endpoint of an integration process and compute the adjoint sensitivities according to the function's derivative. This however will become prohibitively costly already for a moderate number of intermediate timepoints treated this way.

Hence we present now a possibility to "inject" additional adjoint directions respectively adjoint Taylor coefficients at arbitrary timepoints during an adjoint IND(-TC) sweep. Based on this approach, we can compute first and higher-order derivatives of functions depending on the systems states at arbitrary timepoints by executing a first order adjoint IND scheme or a higher-order forward/adjoint IND-TC scheme, respectively, only once.
The injection itself is again based on the combination of IND and AD. Suppose the nominal integration has been performed, and the discretization scheme has been determined. Then the trajectory value at an arbitrary point $\tau_I \in [\tau_n, \tau_{n+1}]$ can be computed by an evaluation of the corrector polynomial of integration step n, leading to

$$\mathbf{y}(\tau_I) = \sum_{i=0}^{k_n} \alpha_{i,n}^I(\tau_I)\mathbf{y_{n+1-i}}, \tag{6.46}$$

where the coefficients $\alpha_{i,n}^I$ only depend on the relative position of τ_I with respect to the gridpoints $\tau_{n+1}, \ldots, \tau_{n+1-k_n}$. The same holds for the case of the computation of forward sensitivities of any order. To obtain now adjoint injection rules that comply with the IND principle, we consider this interpolation again as an elementary function. Then we apply the reverse AD mode to obtain the rules

$$\bar{\mathbf{y}}_{\mathbf{n+1-i}} \mathrel{+}= \alpha_{i,n}^I(\tau_I)\bar{\mathbf{y}}(\tau_I), \qquad 0 \le i \le k_n. \tag{6.47}$$

The value of $\bar{\mathbf{y}}(\tau_I)$ is here determined by the (adjoint) derivative of the dependent function with respect to $\mathbf{y}(\tau_I)$ in the first order case, or the adjoint Taylor coefficient propagated through the function in the framework of a higher-order forward/adjoint scheme.
Note that the injection has to occur before any of the values $\bar{\mathbf{y}}_{\mathbf{n+1-i}}$, $0 \le i \le k_n$ is itself used in the computation of other values in the adjoint IND(-TC) sweep. As on the other hand for efficiency only $k_{\max}+2$ intermediate values $\bar{\mathbf{y}}_{\mathbf{i}}$ are kept in the adjoint sweep at a time, the injection cannot be done too much in advance (e.g., at the start of the adjoint sweep). Hence the injection is performed in precisely that moment, where the value $\bar{\mathbf{y}}_{\mathbf{n+1}}$ would have been accumulated completely in the ordinary adjoint sweep. The whole procedure for the forward/adjoint IND-TC scheme is described in Algorithm 6.9 on the next page.
The adjoint injection can be used in `DAESOL-II` simply by passing a list of timepoints where the injection should occur and registering a call-back function, that delivers the value $\bar{\mathbf{y}}(\tau_I)$ for the corresponding τ_I when called by the integrator as the injection takes place. By this mechanism, the $\bar{\mathbf{y}}(\tau_I)$ can either be computed on demand when the function is called, or be precomputed before the adjoint sweep is started and then simply passed to the integrator, depending on the user's needs in the specific context.

Example 6.8 (Hessian of Lagrangian depending on states at arbitrary timepoints)
We illustrate a possible use of the continuous sensitivity output and adjoint injection in `DAESOL-II` using an example from optimization. Imagine a scalar objective function

$$\psi(\mathbf{y}(\tau_{a_1}), \ldots, \mathbf{y}(\tau_{a_l}), \mathbf{y}(\tau_e)), \quad \tau_{a_i} \in [\tau_0, \tau_e] \text{ for } 1 \le i \le l_1,$$

and a set of point constraints

$$\mathbf{h_j}(\mathbf{y}(\tau_{b_j})) \ge \mathbf{0} \text{ with } \tau_{b_j} \in [\tau_0, \tau_e] \text{ for } 1 \le j \le l_2.$$

Algorithm 6.9: Adjoint sensitivity injection (forward/adjoint IND-TC version)

Input: Discretization scheme stored from a forward IND-TC sweep of order m (stepsizes, orders, iteration matrix factorizations, iteration counts, Taylor coefficients), adjoint sensitivity direction $\bar{\mathbf{y}}$, injection times $\tau_{I_j} \in [\tau_0, \tau_e]$, $1 \leq j \leq l$ (sorted), adjoint Taylor coefficients to inject $\bar{\mathbf{Y}}^{\mathbf{I}}_{\mathbf{j}}$, $1 \leq j \leq l$.

Output: Adjoint Taylor coefficients $\bar{\mathbf{Y}}_{\mathbf{0}}, \bar{\mathbf{P}}$ containing scaled forward/adjoint sensitivities respecting injected sensitivities.

initialize sensitivities with $\bar{\mathbf{Y}}_{\mathbf{N}} = \begin{bmatrix} \bar{\mathbf{y}} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}$;
$n = N - 1$, $j = l$;
while $n \geq 0$ **do**
  get τ_n, k_n from stored discretization scheme;
  while $\tau_{I_j} >= \tau_n$ **do**
    compute coefficients $\alpha^I_{i,n}(\tau_{I_j})$, $1 \leq i \leq k_n$;
    for $i = 0 : k_n$ **do**
      $\bar{\mathbf{Y}}_{\mathbf{n+1-i}} \mathrel{+}= \alpha^I_{i,n}(\tau_{I_j}) \bar{\mathbf{Y}}^{\mathbf{I}}_{\mathbf{j}}$;
    end
    $j = j - 1$;
  end
  $\bar{\mathbf{Y}}^{(\mathbf{s_n})}_{\mathbf{n+1}} = \bar{\mathbf{Y}}_{\mathbf{n+1}}$;
  get $\mathbf{Y_{n+1}}$, h_n, $\mathbf{M}^{-1}_{\mathbf{n}}$, s_n from stored discretization scheme;
  for $i = s_n - 1 : 0$ **do**
    propagate through Newton-like iteration;
  end
  $\bar{\mathbf{Y}}^{\mathbf{P}}_{\mathbf{n+1}} = \bar{\mathbf{Y}}^{(\mathbf{0})}_{\mathbf{n+1}}$;
  propagate corrector constant dependency backwards using (6.31b);
  propagate predictor dependency backwards using (6.31a);
  `// all contributions of` $\mathbf{y_n}$ `taken into account,`
  `// value of` $\bar{\mathbf{Y}}_{\mathbf{n}}$ `is final`
  $n = n - 1$;
end

Here $\mathbf{y}(\tau; \tau_0, \mathbf{y_0}, \mathbf{p})$ is given as the solution of a corresponding DAE-IVP on the integration horizon $[\tau_0, \tau_e]$.

Then using the continuous output and adjoint injection capabilities of `DAESOL-II` we can compute the Hessian of the Lagrangian $\mathcal{L} := \psi - \sum_{j=1}^{l_2} \boldsymbol{\mu}_{\mathbf{j}}{}^T \mathbf{h_j}$ with respect to $(\mathbf{y_0}, \mathbf{p})$ using only one forward IND-TC sweep of order 1 with the unit directions $\mathbf{e}^{\mathbf{n_y+n_p}}_{\mathbf{j}}$, $1 \leq j \leq n_y + n_p$, as forward sensitivity directions, followed by only one adjoint IND-TC sweep. The workflow for this using `DAESOL-II` might then be as follows.

1. Register a plug-in with the output grid containing the set of timepoints $\{\tau_{a_1}, \dots, \tau_{a_l}, \tau_{b_1}, \dots, \tau_{b_k}\}$

in ascending order that stores the propagated forward Taylor Coefficients $\mathbf{Y}(\tau_I)$ at the corresponding timepoints when called from the integrator.

2. Let the integrator perform a forward IND-TC sweep of order 1 initialized with the directions $\mathbf{e}_\mathbf{j}^{\mathbf{n_y+n_p}}$, $1 \leq j \leq n_y + n_p$ (with storage of the whole discretization scheme) to obtain forward Taylor coefficients at τ_e and the output grid.

3. Using the forward Taylor coefficients, perform a forward/reverse TC propagation through the objective function ψ with adjoint direction $\bar{y}_\psi = 1$ to obtain the adjoint Taylor coefficient $\bar{\mathbf{Y}}(\tau_\mathrm{e})$ for the initialization of the adjoint IND-TC sweep, as well as the adjoint Taylor coefficients $\bar{\mathbf{Y}}(\tau_{a_i})$ to inject at the τ_{a_i}. Likewise perform forward/reverse TC propagations through the constraint functions $\mathbf{h_j}$ with adjoint directions $\boldsymbol{\mu}_\mathbf{j}$ to obtain the adjoint Taylor coefficients $\bar{\mathbf{Y}}(\tau_{b_i})$ to inject at the τ_{b_i}.

4. Register the adjoint injection grid (in this case equal to the output grid in 1.) and the call-back routine delivering $\bar{\mathbf{Y}}(\tau_{a_i})$, or $\bar{\mathbf{Y}}(\tau_{b_i})$, respectively, when called from the integrator with the corresponding gridpoint.

5. Let the integrator perform an adjoint IND-TC sweep of order 1 based on the discretization scheme stored in 2., propagating $n_y + n_p$ adjoint TCs, each initialized with the adjoint TC $\bar{\mathbf{Y}}(\tau_\mathrm{e})$.

6. The propagated adjoint TCs $\bar{\mathbf{Y}}_\mathbf{0}^\mathbf{i}, \bar{\mathbf{P}}^\mathbf{i}$, $1 \leq 1 \leq n_y + n_p$ now contain each in the zero order coefficients the gradient of the Lagrangian. Furthermore, in its first order coefficients the i-th TC contains the $\mathbf{y_0}$-part and the $\mathbf{p}$-part, respectively, of i-th column of the Hessian of the Lagrangian with respect to $(\mathbf{y_0}, \mathbf{p})$.

6.7.5 Error control of forward sensitivities

The presented forward IND(-TC) schemes for sensitivity generation are all based on the same discretization scheme that is used by the solution process for the nominal DAE-IVP. In this solution process the discretization scheme normally is determined adaptively by the error estimation and the stepsize and order control strategies presented in Chapter 5 for the nominal trajectory only. Hence the produced discretization scheme will only lead to error controlled nominal trajectory values, not necessarily to error controlled sensitivities. For the forward sensitivities we have only the convergence results stated above for the case that the stepsizes tend to zero.
Usually, this is not a big problem in practice. First, by the use of IND we always obtain the exact sensitivity of the numerical nominal solution. Furthermore, practical observations show that in most applications the loss of accuracy lies in the region of one order of magnitude, which often is tolerable. However, there are situations where a high accuracy of the computed forward sensitivities is desired or where the determination of the discretization scheme based only on the nominal solution will lead to a blow-up for the sensitivities and/or completely wrong sensitivity approximations. The latter is for example the case if some of the possibly instable modes of the system's dynamic are not triggered during the solution of the nominal DAE-IVP for the specific

set of initial values and parameter, while the solution of the forward variational would do. A very simple academic example where this behavior might occur is the Dahlquist equation.

Example 6.9
Consider the simple ODE-IVP

$$\dot{x}(\tau) = px(\tau), \quad p \in \mathbb{R}, \quad x(0) = x_0, \quad t \in [0,1].$$

Its nominal solution is $x(\tau) = x_0 e^{p\tau}$. The first order variational DAE-IVP is then given by

$$\dot{x}_{x_0}(\tau) = px_{x_0}(\tau), \quad p \in \mathbb{R}, \quad x_{x_0}(0) = 1, \quad t \in [0,1],$$

with the solution $x_{x_0}(\tau) = e^{p\tau}$. If we now choose $x_0 = 0$ and perform a first order forward IND sweep, we will obtain approximations for x_{x_0} that are completely useless. This is caused by the fact that the solution process indeed computes the correct solution $x(\tau) \equiv 0$ and that the error estimation in this case tells us that the local error in each step is equal to zero. Hence the stepsize strategy will continuously and rapidly increase the used stepsizes. This leads to a discretization scheme that is perfect for the solution of the nominal IVP but of course disastrous for the approximation of the sensitivities. This behavior is also shown in practice in Section 9.3 on page 242.

The interpretation of the forward IND(-TC) scheme as BDF discretization scheme for the variational IVPs here again is the key to a possible remedy. The computed forward sensitivities are then to be understood as approximations of the solution trajectory of the corresponding variational IVP computed by a BDF method. Hence in principle the same mechanisms described earlier for the estimation of the local error in the approximation of the nominal solution can be applied to them. As the error estimation is mainly based on the modified divided differences and the increments of the Newton-like method, which need to be computed anyway also for the sensitivity approximations, this leads only to a relatively small increase of the effort and needs moderate changes in the original integration code. Once a local error estimation has been established for the sensitivity approximation, analogously the presented stepsize and order strategies can be used to obtain a suitable new stepsize and order for the computation of the next sensitivity approximation. In this way, a suitable discretization scheme for the computation of the sensitivity approximations can be generated. It should be noted here that it is advisable to modify the scaling factors for the forward sensitivity trajectories at least according to the norm of the corresponding sensitivity direction. This is sensible, because if we multiply the sensitivity direction by 2 this will change the first order forward sensitivities by the factor 2, the second order forward sensitivities by the factor 2^2 and so on.
To obey to the IND principle, i.e., to use the same discretization scheme for the nominal and sensitivity trajectories, we use a modified simultaneous iterative forward IND(-TC) sweep that is described in its first order version in Algorithm 6.10 on the next page. Here in an integration step the Newton-like iterations and the corresponding contraction estimations are performed simultaneously for nominal and forward sensitivity trajectories. If the iterations do not converge for any of the trajectories, then the corrector equation solution is considered as failed. Otherwise

the local error is estimated for all trajectories, and again the step is rejected, if any of the error estimations is too large. Otherwise the next step is planned for each trajectory separately and the most pessimistic estimation will be used in the next integration step for all trajectories. In case that the step has failed, the monitor strategy respectively the presented strategies for stepsize reduction are applied, based on the trajectories that caused the step failure. Also here the most pessimistic estimation for the stepsize will then be used for the computation of all trajectories during the repetition of the step.

Algorithm 6.10: Error control for forward sensitivities

Input: t_0, t_f, h_0, initial values $\mathbf{y_0}$, parameter $\mathbf{p}$, set of sensitivity directions $\mathbf{D}$
Output: Nominal solution $\mathbf{y_N}$, forward sensitivities $\dot{\mathbf{Y}}_\mathbf{N} = \mathcal{W} \cdot \mathbf{D}$.
set $k_0 = 1$, $n = 0$;
initialize nominal integration with $\mathbf{y_0}$, $\mathbf{p}$ and sensitivities with $\mathbf{D}$ (6.28);
while t_f *not reached* **do**
  compute interpolation coefficients using h_n and k_n;
  compute nominal and IND predictor and corrector constant;
  compute $\mathbf{y_{n+1}}$ and $\dot{\mathbf{Y}}_\mathbf{n+1}$ by s_n Newton-like iterations using matrix M_n to solve the nominal respectively IND corrector equation;
  if *Newton-like method converged for all trajectories* **then**
    Perform error estimation separately for all trajectories;
    if *error check passed for all trajectories* **then**
      $t_{n+1} = t_n + h_n$;
      determine stepsize and order for next step separately for all trajectories;
      choose the smallest of proposed stepsizes as h_{n+1} and the corresponding order as k_{n+1} for the next step;
      $n = n + 1$;
    else
      compute reduced stepsizes for all trajectories for which the error check failed and choose smallest of them as h_n;
    end
  else
    update Jacobian approximation $\mathbf{M_n}$ according to monitor strategy or compute reduced stepsizes for all trajectories for which Newton-like method didn't converge and choose smallest of them as h_n;
  end
end
$N = n$;

Using this approach we obtain a commonly generated discretization scheme for both the nominal trajectory and the forward sensitivities. This allows an error controlled computation of both the nominal solution and of forward sensitivities of any order. A numerical example for the application of this strategy is given in Section 9.3 on page 242. Note that compared to a normal simultaneous iterative forward IND(-TC) sweep, the effort will usually be higher. The first reason for this is,

of course, that in general a finer discretization grid will be used. However, even in the case that the common discretization scheme has in the end been determined only by the nominal solution, i.e., is equal to the one without error control of the sensitivities, the effort will be higher. This is caused for one part by the additional effort for the error estimation on the sensitivity trajectories. For the other part a step rejection is now more expensive. Compared to the usual approach, here also for a rejected step the Newton-like iterations for the sensitivity trajectories have been performed, including the computation of the model function derivatives, or the corresponding TC propagation, respectively. Hence in practice it is advisable to enable the error control for the forward sensitivities only if necessary.

6.7.6 Global error estimation

The stepsize selection and error control strategies described in Section 5.3.4 are all based on the estimation of the local integration error, described in Section 5.3.2. They are usually an effective and efficient mean to obtain an error controlled numerical solution of the underlying IVP problem with high performance. However, in some cases they might not be sufficient. One occasion might be an IVP for which the global error of the solution at the end of the integration horizon is not in the range of the prescribed local integration tolerance, because of the accumulation of the errors made in each step as well as error propagation. In this case it would be interesting to have (at least) an estimation of the global error. Another example is the case where not the error of the solution of the IVP itself but the error in a quantity depending on the solution is of interest. Here, a so-called goal-oriented error estimation, and maybe also error control, is desired.
There exist a number of proposed strategies for the global error estimation for ODE and DAE solutions in literature, refer, e.g., to [Ske86, Joh88, Est95, JV98, MSTZ03, CP04, ALW07, LV07, TB09] and references therein. Mostly the strategies are developed for finite element, collocation or one-step methods. Approaches for BDF methods in the ODE case can be found in [CP04, TB09]. The two main ideas repeatedly mentioned in literature are, on the one hand, the solution of an IVP approximating the global error in parallel with the computation of the original IVP solution. On the other hand, there is the usage of adjoint sensitivity information in connection with a local error or defect estimation to compute a global error estimate, which is partly inspired by the analogous approach in PDE numerics [BR01]. For a global error estimation of the IVP solution both approaches deliver, in general, reliable results (cf. [LV07]). The adjoint approach is favorable when a goal-oriented a posteriori error estimation is desired as the complexity does not depend on the dimension of the IVP in this case.
To allow both an efficient global error estimation of the solution of the IVP as well as an efficient goal-oriented error analysis, the following adjoint based strategy for an a posteriori error estimation is implemented in `DAESOL-II`. We assume that a nominal integration for a given tolerance has been performed using N integration steps and that the local error estimates $\boldsymbol{\varepsilon}_{\mathbf{loc,i}}$, $1 \leq i \leq N$, have been stored on the tape. Furthermore, we denote the (scalar) quantity of interest with $\phi(\mathbf{y_N})$ which is assumed to depend directly on the IVP solution $\mathbf{y_N}$ at the end time. To compute an error estimation for $\phi(\mathbf{y_N})$ we perform a first order adjoint IND sweep with adjoint direction $\frac{\partial \phi}{\partial \mathbf{y_N}}$ which yields the adjoint quantities $\bar{\mathbf{y}}_\mathbf{i}$, $N \geq i \geq 0$. Then, the goal-oriented error estimate $\boldsymbol{\varepsilon}_\phi$ is

simply obtained as

$$\varepsilon_\phi := \sum_{i=1}^{N} \varepsilon_{\mathbf{loc,i}}{}^T \bar{\mathbf{y}}_\mathbf{i}, \tag{6.48}$$

where $\varepsilon_{\mathbf{loc,i}}$ is the local error estimation in the i-th step of the integration, as presented in Section 5.3.4. Note that we only use quantities that are computed anyway during a nominal integration and a first order adjoint sweep, respectively. Hence the error estimation does not cause additional numerical effort besides the storage of the local errors and the summation. The approach can be extended directly to quantities of interest that depend on intermediate values of the solution by combination with the idea of adjoint sensitivity injection described in Section 6.7.4.
In some sense, our approach is quite similar to the approach presented by Cao and Petzold [CP04], as both are based on the use of adjoint sensitivity information in combination with the local error estimation of the nominal IVP solution. However, in our approach we do not use the solution of the adjoint variational differential equation (6.13) but the intermediate adjoint sensitivity quantities of the presented adjoint IND sweep. Hence, it is also directly applicable to DAEs, in contrast to the approach of Cao and Petzold. Furthermore, by the use of higher-order adjoint sweeps in principle a global error estimation of forward sensitivities can be obtained as well. Because the intermediate adjoint quantities are the exact derivatives of the numerical IVP solution w.r.t. the corresponding trajectory value (see Remark 6.2), they can be interpreted as the sensitivities of the numerical solution w.r.t. perturbations or errors in these trajectory values. In this sense, our approach can be understood as a first order estimate for the error in the quantity of interest based on a condition estimate of the numerical integration scheme. This is similar to the approaches of estimating the influences of round-off error in numerical algorithms presented, e.g., in [Lin76, Stu80] and numerous later works.
The efficiency of our approach is demonstrated numerically on several test problems in Section 9.4 on page 245.

6.7.7 Time and control transformations of function and derivatives

In general we like to be able to solve the following relaxed index one DAE-IVP type, that occurs, e.g., in a multistage multiple shooting setup in stage i

$$\begin{aligned} \hat{\mathbf{A}}(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p_p})\, \dot{\mathbf{x}}(t) - p_{h_i} \cdot \hat{\mathbf{f}}(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p_p}) &= \mathbf{0}, \qquad \mathbf{x}(t_0) = \mathbf{x_0}, \quad (6.49a) \\ \hat{\mathbf{g}}(t, \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t), \mathbf{p_p}) - \hat{\theta}(t)\, \hat{\mathbf{g}}(t_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{u}(t_0), \mathbf{p_p}) &= \mathbf{0}, \qquad \mathbf{z}(t_0) = \mathbf{z_0}, \quad (6.49b) \end{aligned}$$

where $t \in [t_0, t_\mathrm{e}] \subset [\sum_{j=1}^{i-1} p_{h_j}, \sum_{j=1}^{i} p_{h_j}]$. This problem formulation is obtained from the formulation (6.1), which we have considered throughout this whole chapter, by the time transformation $t = \sum_{j=1}^{i-1} p_{h_j} + p_{h_i} \cdot \tau$ and by assuming a finite-dimensional parametrization of the control functions $\mathbf{u}(t) : \mathbb{R} \to \mathbb{R}^{n_u}$ by control parameter $\mathbf{u}(t) \equiv \mathbf{u}(\mathbf{p_q}; \tau)$. Therefore, it is sensible to derive the numerical strategies and to implement the corresponding integration routines only for the problem type (6.1), as from both the mathematical and the implementational point of view the integration routine does not need to know about the possible transformation. Instead, when the

integrator demands an evaluation, derivative computation or the propagation of TCs through one of the model function $\mathbf{f}$, $\mathbf{g}$ or $\mathbf{A}$ "living" in the integrator context, suitable transformations have to be performed before and after the corresponding model function $\hat{\mathbf{A}}, \hat{\mathbf{f}}, \hat{\mathbf{g}}$ formulated in the user context are called. In `SolvIND` these transformations are handled by the `TEvaluator` module and derived classes. The transformations are fully transparent for the integrator, such that all integrator codes in the `SolvIND` suite can simply be written for the IVP class (6.1) and nevertheless handle problems of type (6.49). In this way, the transformation code has only to be written once and can be maintained and extended in one dedicated module, which is a clear advantage compared, e.g., to the `MUSCOD-II`[DLS01] package. Furthermore, this architecture allows to switch easily between different control parametrization and to add new ones without the need to change the formulation of the model functions $\hat{\mathbf{A}}, \hat{\mathbf{f}}, \hat{\mathbf{g}}$ or to make changes to the integration code itself.

The transformations that are needed for an ordinary evaluation of a model function are straightforward. For the evaluation at $(\tau, \mathbf{y}, \mathbf{p})$ we first have to translate the integrator time τ to the "physical" time t

$$t = \sum_{j=1}^{i-1} p_{h_j} + p_{h_i} \cdot \tau \tag{6.50a}$$

and to evaluate the actual control values for the given parametrization

$$\mathbf{u} = \mathbf{u}(\mathbf{p_q}; \tau). \tag{6.50b}$$

Then the user-space model function $\hat{\mathbf{A}}, \hat{\mathbf{f}}$ or $\hat{\mathbf{g}}$ can be evaluated at $(t, \mathbf{y}, \mathbf{u}, \mathbf{p_p})$. In the case of the differential right hand side $\hat{\mathbf{f}}$ the result then has to be scaled afterwards by the actual time transformation factor p_{h_i}

$$\mathbf{f}(\tau, \mathbf{y}, \mathbf{p}) = p_{h_i} \cdot \hat{\mathbf{f}}(t, \mathbf{y}, \mathbf{u}, \mathbf{p_p}). \tag{6.50c}$$

In the case of a derivative evaluation or a TC propagation, additionally the derivative directions and the obtained derivatives or the input TCs and the propagated TCs have to be transformed.

For the forward case it is straightforward to transform τ-directions/TCs and $\mathbf{p_h}$-directions/TCs into t-directions/TCs by applying the forward AD/TC propagation rules presented in Chapter 2 to (6.50a). Likewise we use the actual definition of (6.50b) to transform $\mathbf{p_q}$-directions/TCs into $\mathbf{u}$-directions/TCs. Based on the transformed directions or TCs the derivatives of the user-defined model functions $\hat{\mathbf{A}}, \hat{\mathbf{f}}$ or $\hat{\mathbf{g}}$ can be evaluated. For the differential right hand side the obtained derivative or the propagated TCs have to be transformed afterwards based on (6.50c).

An analogous process has to be performed to evaluate an adjoint directional derivative: In the case of the differential right hand side after an evaluation of the function itself, first the adjoint directions have to be transformed. This is not needed for the other model functions. Then the adjoint derivatives of the user given model function can be computed, and afterwards the obtained adjoint derivatives have to be transformed. If we denote the adjoint direction with $\bar{\boldsymbol{\lambda}}$, the evaluation of the adjoint derivative, e.g., for the differential right hand side starts after the

corresponding function evaluation with

$$p_{h_i} \mathrel{+}= \bar{\boldsymbol{\lambda}}^T \hat{\mathbf{f}}(t, \mathbf{y}, \mathbf{u}, \mathbf{p_p}) \qquad \text{(only for } \mathbf{f}\text{)} \tag{6.51a}$$

$$\hat{\bar{\boldsymbol{\lambda}}} = p_{h_i} \cdot \bar{\boldsymbol{\lambda}}. \qquad \text{(only for } \mathbf{f}\text{)} \tag{6.51b}$$

Then the adjoint derivative of $\hat{\mathbf{f}}(t, \mathbf{y}, \mathbf{u}, \mathbf{p_p})$ with adjoint direction $\hat{\bar{\boldsymbol{\lambda}}}$ is evaluated. Afterwards the obtained derivatives with respect controls and time are transformed using

$$\bar{\mathbf{q}}^T \mathrel{+}= \bar{\mathbf{u}}^T \frac{\partial \mathbf{u}(\mathbf{p_q}; \tau)}{\partial \mathbf{q}} \tag{6.51c}$$

$$\bar{\tau} \mathrel{+}= \bar{\mathbf{u}}^T \frac{\partial \mathbf{u}(\mathbf{p_q}; \tau)}{\partial \tau}, \tag{6.51d}$$

and

$$\bar{p}_{h_j} \mathrel{+}= \bar{t}, \qquad (1 \leq j \leq i-1) \tag{6.51e}$$

$$\bar{p}_{h_i} \mathrel{+}= \bar{t} \cdot \tau \tag{6.51f}$$

$$\bar{\tau} \mathrel{+}= \bar{t} \cdot p_{h_i}. \tag{6.51g}$$

Finally, we have obtained the adjoint derivative $(\bar{\tau}, \bar{\mathbf{y}}, \bar{\mathbf{p}})$ of the integrator-space model function $\mathbf{f}(\tau, \mathbf{y}, \mathbf{p})$ in direction $\bar{\mathbf{y}}$. To perform an adjoint TC propagation, we can apply basically the same scheme and replace the operations (6.51) by the corresponding operations in Taylor arithmetics, as explained earlier.

6.7.8 Sensitivity propagation across switching events

In this section we consider the extension of the presented first and higher-order IND(-TC) schemes to the case of IVPs where the model dynamic is possibly subject to implicitly defined state and parameter dependent discontinuities and/or non-differentiabilities. We call these events switches. Although this type of problems is not the focus of the applications in this thesis, the treatment of these switching events is important in many fields of application. We consider here the relaxed index 1 DAE-IVP

$$\begin{aligned} \dot{\mathbf{x}}(\tau) - \mathbf{f}(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p}, \operatorname{sgn}(\sigma(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p}))) &= 0 \qquad (6.52a) \\ \mathbf{g}(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p}, \operatorname{sgn}(\sigma(\tau, \mathbf{x}(\tau), \mathbf{z}(\tau), \mathbf{p})) - \theta(\tau)\, \mathbf{g}(\tau_0, \mathbf{x_0}, \mathbf{z_0}, \mathbf{p}) &= 0 \qquad (6.52b) \\ \mathbf{x}(\tau_0) = \mathbf{x_0}, \qquad \mathbf{z}(\tau_0) = \mathbf{z_0}, \qquad \tau \in [\tau_0, \tau_\mathrm{e}], & \end{aligned}$$

where we call $\sigma : [\tau_0, \tau_\mathrm{e}] \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}$ the switch function and assume in the following that it is sufficiently smooth, but at least continuously differentiable in all variables. Note that this formulation can be easily extended to the case of a vector-valued switch function. However, also in this case one generally assumes that only one switch function becomes zero at the same time, such that we restrict our analysis to the case of one switch function. Also note that for notational simplicity we only consider the case $\mathbf{A} \equiv \mathbb{I}$ here, while the derivation for the general

case can be made analogously. The roots of the switch function define the timepoints where switching events occur. By the implicit function theorem we can interpret the switching time τ_s as implicitly defined by the values $(\tau_0, \mathbf{y_0}, \mathbf{p})$.
We assume for now that only one switching event occurs, and that this event occurs at time τ_s. Then we denote the limits of the differential variables at τ_s from the left and right with

$$\mathbf{x}^-(\tau_s; \tau_0, \mathbf{y_0}, \mathbf{p}) := \lim_{\epsilon \searrow 0} \mathbf{x}(\tau_s - \epsilon; \tau_0, \mathbf{y_0}, \mathbf{p}) \tag{6.53}$$

and

$$\mathbf{x}^+(\tau_s; \mathbf{\Delta_x}; \tau_s, \mathbf{y}^-, \mathbf{p}) := \lim_{\epsilon \searrow 0} \mathbf{x}(\tau_s + \epsilon; \mathbf{\Delta_x}; \tau_s, \mathbf{x}^-, \mathbf{p}), \tag{6.54}$$

where

$$\mathbf{\Delta_x} : [\tau_0, \tau_e] \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_y}, \qquad (\tau, \mathbf{x}^-(\tau), \mathbf{p}) \mapsto \mathbf{x}^+(\tau), \tag{6.55}$$

describes a possible jump in the differential variables occurring at the switching event

$$\mathbf{x}^+(\tau_s; \mathbf{\Delta_x}) := \mathbf{x}^-(\tau_s) + \mathbf{\Delta_x}(\tau, \mathbf{x}^-(\tau), \mathbf{p}). \tag{6.56}$$

We assume here that the jump does not explicitly depend on the algebraic variables and also that a corresponding jump $\mathbf{\Delta_z}$ in the algebraic variables always conserves the consistency of the algebraic equations. Then the states after the switching event are fully determined by the differential variables and we write for the algebraic variables at τ_s

$$\mathbf{z}^-(\tau_s; \tau_0, \mathbf{y_0}, \mathbf{p}) := \lim_{\epsilon \searrow 0} \mathbf{z}(\tau_s - \epsilon; \tau_0, \mathbf{y_0}, \mathbf{p}) \tag{6.57}$$

and

$$\mathbf{z}^+(\tau_s; \mathbf{\Delta_z}; \tau_0, \mathbf{y_0}, \mathbf{p}) := \mathbf{z}(\tau_s; \tau_s, \mathbf{x}^+(\tau_s; \mathbf{\Delta_x}; \tau_0, \mathbf{y_0}, \mathbf{p}), \mathbf{p}). \tag{6.58}$$

We denote furthermore the model functions before the switch with $\mathbf{f^1}$ and $\mathbf{g^1}$, and after the switch with $\mathbf{f^2}$ and $\mathbf{g^2}$.

In the following, we will consider the task of generating first and higher-order derivatives of the solutions of the IVP (6.52). For the question of how the implicitly defined switching time τ_s can be determined efficiently in a numerical integration scheme refer to [Kir06, BP04, Ehr05] and references therein. The proof of the differentiability of the solution of (6.52) has been given by Bock in [Boc87] for the ODE case, and an extension of the theorem for the index 1 DAE case is stated in [BP04]. Both versions require that the switching is consistent, i.e., that

$$\frac{\mathrm{d}\sigma^-}{\mathrm{d}\tau}(\tau_s, \mathbf{x}^-, \mathbf{z}^-) \cdot \frac{\mathrm{d}\sigma^+}{\mathrm{d}\tau_s}(\tau_s, \mathbf{x}^+, \mathbf{z}^+) > 0, \tag{6.59}$$

where σ^- and σ^+ are the one-sided derivatives of the switch function. Consistent switching hence requires that the switch function changes its sign at τ_s.

Note that the sensitivities of the algebraic variables after the switch are uniquely determined by the sensitivities of the differential variables and by the consistency conditions. For educational purposes we now derive the formulas for the full first order sensitivities

$$\frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{x_0}} \text{ and } \frac{\mathrm{d}\mathbf{x}(\tau_\mathrm{e})}{\mathrm{d}\mathbf{z_0}}$$

in the "classical" way. Similar derivations are presented in [Boc87, Mom01, Kir06] for the case of ODEs and in [BP04] for index 1 DAEs.

We start by defining the abbreviations

$$\begin{aligned} \mathbf{f}^- &:= \mathbf{f^1}(\tau_s, \mathbf{x}^-, \mathbf{z}^-, \mathbf{p}) & \mathbf{f}^+ &:= \mathbf{f^2}(\tau_s, \mathbf{x}^+, \mathbf{z}^+, \mathbf{p}) \\ \sigma &:= \sigma(\tau_s, \mathbf{x}^-, \mathbf{z}^-, \mathbf{p}) & \mathbf{\Delta} &:= \mathbf{\Delta}(\tau_s, \mathbf{x}^-, \mathbf{p}) \\ \dot{\mathbf{z}}^- &:= \dot{\mathbf{z}}(\tau_s, \mathbf{x}^-, \mathbf{z}^-, \mathbf{p}) & \mathbf{x_e} &:= \mathbf{x}(\tau_\mathrm{e}; \tau_s, \mathbf{x}^+, \mathbf{p}), \end{aligned}$$

where $\dot{\mathbf{z}}$ can be obtained by applying the implicit function theorem to the consistency conditions. Then we have

$$\begin{aligned} \frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{x_0}} &= \frac{\partial \mathbf{x}^-}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}} + \frac{\partial \mathbf{\Delta}}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\left(\frac{\partial \mathbf{x}^-}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}}\right) \\ &= \left(\mathbf{f}^- + \frac{\partial \mathbf{\Delta}}{\partial \tau_s} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}} \end{aligned} \tag{6.60}$$

and

$$\begin{aligned} \frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{z_0}} &= \frac{\partial \mathbf{x}^-}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}} + \frac{\partial \mathbf{\Delta}}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\left(\frac{\partial \mathbf{x}^-}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}}\right) \\ &= \left(\mathbf{f}^- + \frac{\partial \mathbf{\Delta}}{\partial \tau_s} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}}. \end{aligned} \tag{6.61}$$

From the implicit function theorem, applied to σ, we obtain

$$\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} = -\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- + \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1} \underbrace{\left(\frac{\partial \sigma}{\partial \mathbf{x}^-}\frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}} + \frac{\partial \sigma}{\partial \mathbf{z}^-}\frac{\partial \mathbf{z}^-}{\partial \mathbf{x_0}}\right)}_{\left(\frac{\partial \sigma}{\partial \mathbf{x}^-} + \frac{\partial \sigma}{\partial \mathbf{z}^-}((\mathbf{g}_z^-)^{-1}\mathbf{g}_x^-)\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}}} \tag{6.62}$$

$$\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} = -\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- + \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1} \underbrace{\left(\frac{\partial \sigma}{\partial \mathbf{x}^-}\frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}} + \frac{\partial \sigma}{\partial \mathbf{z}^-}\frac{\partial \mathbf{z}^-}{\partial \mathbf{z_0}}\right)}_{\left(\frac{\partial \sigma}{\partial \mathbf{x}^-} + \frac{\partial \sigma}{\partial \mathbf{z}^-}((\mathbf{g}_z^-)^{-1}\mathbf{g}_x^-)\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}}}. \tag{6.63}$$

Furthermore, we have

$$\frac{\partial \mathbf{x_e}}{\partial \tau_s} = -\frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\mathbf{f}^+. \tag{6.64}$$

Combing all this leads to

$$\begin{aligned}
\frac{\mathrm{d}\mathbf{x_e}}{\mathrm{d}\mathbf{x_0}} &= \frac{\partial \mathbf{x_e}}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{x_0}} \\
&\overset{(6.64)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left(-\mathbf{f}^+\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{x_0}}\right) \\
&\overset{(6.60)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left[\left(-\mathbf{f}^+ + \mathbf{f}^- + \frac{\partial \mathbf{\Delta}}{\partial \tau_s} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{x_0}} + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}}\right] \\
&\overset{(6.62)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left[\left(\mathbf{f}^+ - \mathbf{f}^- - \frac{\partial \mathbf{\Delta}}{\partial \tau_s} - \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- + \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1}\right. \\
&\qquad \left.\cdot\left(\frac{\partial \sigma}{\partial \mathbf{x}^-} + \frac{\partial \sigma}{\partial \mathbf{z}^-}((\mathbf{g}_\mathbf{z}^-)^{-1}\mathbf{g}_\mathbf{x}^-)\right) + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\right]\frac{\partial \mathbf{x}^-}{\partial \mathbf{x_0}}
\end{aligned} \tag{6.65}$$

and

$$\begin{aligned}
\frac{\mathrm{d}\mathbf{x_e}}{\mathrm{d}\mathbf{z_0}} &= \frac{\partial \mathbf{x_e}}{\partial \tau_s}\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{z_0}} \\
&\overset{(6.64)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left(-\mathbf{f}^+\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \frac{\mathrm{d}\mathbf{x}^+}{\mathrm{d}\mathbf{z_0}}\right) \\
&\overset{(6.61)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left[\left(-\mathbf{f}^+ + \mathbf{f}^- + \frac{\partial \mathbf{\Delta}}{\partial \tau_s} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\frac{\mathrm{d}\tau_s}{\mathrm{d}\mathbf{z_0}} + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}}\right] \\
&\overset{(6.63)}{=} \frac{\partial \mathbf{x_e}}{\partial \mathbf{x}^+}\left[\left(\mathbf{f}^+ - \mathbf{f}^- - \frac{\partial \mathbf{\Delta}}{\partial \tau_s} - \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- + \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1}\right. \\
&\qquad \left.\cdot\left(\frac{\partial \sigma}{\partial \mathbf{x}^-} + \frac{\partial \sigma}{\partial \mathbf{z}^-}((\mathbf{g}_\mathbf{z}^-)^{-1}\mathbf{g}_\mathbf{x}^-)\right) + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\right]\frac{\partial \mathbf{x}^-}{\partial \mathbf{z_0}}.
\end{aligned} \tag{6.66}$$

Finally, we have obtained the relation

$$\mathcal{W}_\mathbf{x}^\mathbf{x}(\tau_\mathrm{e};\tau_0) = \mathcal{W}_\mathbf{x}^\mathbf{x}(\tau_\mathrm{e};\tau_s)\ \mathcal{U}_\mathbf{x}^\mathbf{x}(\tau_s)\ \mathcal{W}_\mathbf{x}^\mathbf{x}(\tau_s;\tau_0) \tag{6.67}$$

and

$$\mathcal{W}_\mathbf{z}^\mathbf{x}(\tau_\mathrm{e};\tau_0) = \mathcal{W}_\mathbf{x}^\mathbf{x}(\tau_\mathrm{e};\tau_s)\ \mathcal{U}_\mathbf{x}^\mathbf{x}(\tau_s)\ \mathcal{W}_\mathbf{z}^\mathbf{x}(\tau_s;\tau_0) \tag{6.68}$$

with the update matrix

$$\begin{aligned}
\mathcal{U}_\mathbf{x}^\mathbf{x}(\tau_s) := \left[\left(\mathbf{f}^+ - \mathbf{f}^- - \frac{\partial \mathbf{\Delta}}{\partial \tau_s} - \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- - \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1}\right. \\
\left.\cdot\left(\frac{\partial \sigma}{\partial \mathbf{x}^-} + \frac{\partial \sigma}{\partial \mathbf{z}^-}((\mathbf{g}_\mathbf{z}^-)^{-1}\mathbf{g}_\mathbf{x}^-)\right) + \left(\mathbb{I} + \frac{\partial \mathbf{\Delta}}{\partial \mathbf{x}^-}\right)\right].
\end{aligned} \tag{6.69}$$

Likewise, the sensitivities with respect to the parameter can be obtained as

$$\mathcal{W}_{\mathbf{p}}^{\mathbf{x}}(\tau_e;\tau_0) = \mathcal{W}_{\mathbf{x}}^{\mathbf{x}}(\tau_e;\tau_s) \quad \left(\mathcal{U}_{\mathbf{x}}^{\mathbf{x}}(\tau_s) \;\; \mathcal{W}_{\mathbf{p}}^{\mathbf{x}}(\tau_s;\tau_0) \quad + \mathcal{U}_{\mathbf{p}}^{\mathbf{x}}(\tau_s)\right) + \mathcal{W}_{\mathbf{p}}^{\mathbf{x}}(\tau_e;\tau_s), \tag{6.70}$$

with the parameter related update matrix

$$\mathcal{U}_{\mathbf{p}}^{\mathbf{x}}(\tau_s) := \left[\left(\mathbf{f}^+ - \mathbf{f}^- - \frac{\partial \boldsymbol{\Delta}}{\partial \tau_s} - \frac{\partial \boldsymbol{\Delta}}{\partial \mathbf{x}^-}\mathbf{f}^-\right)\left(\frac{\partial \sigma}{\partial \tau_s} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- - \frac{\partial \sigma}{\partial \mathbf{z}^-}\dot{\mathbf{z}}^-\right)^{-1} \cdot \left(\frac{\partial \sigma}{\partial \mathbf{p}}\right) + \frac{\partial \boldsymbol{\Delta}}{\partial \mathbf{p}}\right]. \tag{6.71}$$

We observe that already the derivation of the update matrices for the full first order sensitivities is lengthy and leads to complex formulas that are not easy to implement efficiently (and error-free). In case of few needed directional sensitivities the efficiency can be increased significantly by employing forward or adjoint directional derivatives wherever possible in the equations above. The derivation of second (or even higher-order) updates in this classical way however is a tedious and error-prone work. Mombaur [Mom01] presents the lengthy formulas for second order updates in the ODE case, but refrains from their implementation due to the implementational and computational complexity they give rise to.

As second and higher-order sensitivity update formulas computed in the classical way are worthless in practice we develop now the first strategy for the computation of first and higher order sensitivities based on the propagation of TCs across the switching event. This allows to obtain an efficient formulation of schemes for the generation of arbitrary order sensitivities also in the presence of switches.
We start by recalling the chain of dependencies between the quantities involved in a switching event in the practical computation. First, we remember that by means of the implicit function theorem the switch time τ_s is uniquely determined by the initial values and parameter. In this sense, we can interpret the switch function as a function of the switch time and initial values and parameter $\tilde{\sigma}(\tau_s, \mathbf{y_0}, \mathbf{p}) := \sigma(\tau_s, \mathbf{y}(t_s; \mathbf{y_0}, \mathbf{p}), \mathbf{p})$. Based on the switch time the corresponding states $\mathbf{y}^-$ at τ_s can be determined. The sensitivity of the $\mathbf{y}^-$ is then composed of their "ordinary" dependence of initial values and parameter and their dependence on the switching time. From $\mathbf{y}^-$ one then obtains $\mathbf{y}^+$ by application of the state jump $\boldsymbol{\Delta}$. Afterwards $\mathbf{y}(\tau_e)$ can be computed based on the new initial time τ_s and new initial values $\mathbf{y}^+$. The sensitivities of $\mathbf{y}(\tau_e)$ consist then of the "ordinary" dependency on the initial value $\mathbf{y}^+$ and the parameter as well as the initial time τ_s. Before we are able to use this chain of dependencies for the formulation of a forward TC propagation scheme, we first need the following results concerning the propagation of TCs through an implicitly defined function as well as the computation of the TCs of an ODE solution at the start or end time of the integration horizon based on the corresponding right hand side function.

Lemma 6.10 (TC propagation through implicitly defined functions)
We consider a function $\boldsymbol{\sigma} \in \mathcal{C}^k(\mathbb{R}^{n_v+n_u}, \mathbb{R}^{n_v})$, with k sufficiently large, that implicitly defines the

value $\mathbf{v} \in \mathbb{R}^{n_v}$ in terms of $\mathbf{u} \in \mathbb{R}^{n_u}$ by the equation

$$\boldsymbol{\sigma}(\mathbf{v}, \mathbf{u}) \equiv \mathbf{0}. \tag{6.72}$$

This means we assume silently that $\boldsymbol{\sigma}_{\mathbf{v}}$ is regular. Let $\mathbf{u_0}, \ldots, \mathbf{u_k}$ be the TCs of a Taylor polynomial $\mathbf{u}(s)$ of the truly independent variables and denote with $\mathbf{v_0}, \ldots, \mathbf{v_k}$ the TCs of the Taylor polynomial of the dependent variable $\mathbf{v}(\mathbf{u}(s))$ that we desire to compute. Finally, denote with $\boldsymbol{\sigma_0}, \ldots, \boldsymbol{\sigma_k}$ the TCs resulting from a forward propagation of the TCs $\mathbf{v_0}, \ldots, \mathbf{v_k}$ and $\mathbf{u_0}, \ldots, \mathbf{u_k}$ through the function $\boldsymbol{\sigma}$.

Then the TCs of the Taylor polynomial corresponding to $\mathbf{v}(\mathbf{u}(s))$ can be obtained from the TCs of the input Taylor polynomial $\mathbf{u}(s)$ for any $k \geq 1$ by the following iterative scheme.

1. Compute $\mathbf{v_0} = \mathbf{v}(\mathbf{u_0})$ and $\mathbf{v_1} = -\left(\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\mathbf{v}}(\mathbf{v_0}, \mathbf{u_0})\right)^{-1} \frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\mathbf{u}}(\mathbf{v_0}, \mathbf{u_0})\mathbf{u_1}$
2. For $j = 2, \ldots k$ do
   a) Perform a forward TC propagation of order j through $\boldsymbol{\sigma}$ using input coefficients $\hat{\mathbf{v}} := \begin{bmatrix}\mathbf{v_0} & \cdots & \mathbf{v_{j-1}} & 0\end{bmatrix}$ and $\hat{\mathbf{u}} := \begin{bmatrix}\mathbf{u_0} & \cdots & \mathbf{u_{j-1}} & 0\end{bmatrix}$ to obtain $\hat{\boldsymbol{\sigma}}_{\mathbf{j}}$.
   b) Solve $\mathbf{v_j} = -\left(\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\mathbf{v}}(\mathbf{v_0}, \mathbf{u_0})\right)^{-1} \left(\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}\mathbf{u}}(\mathbf{v_0}, \mathbf{u_0})\mathbf{u_j} + \hat{\boldsymbol{\sigma}}_{\mathbf{j}}\right)$

Proof:

For a proof of a slightly more general version of the Lemma refer to Wagner et al. [WSW10].

□

Lemma 6.11 (TCs of ODE solution at start time)
Let $\epsilon > 0$, $\mathbf{f} \in \mathcal{C}^k([\hat{t} - \epsilon, \hat{t} + \epsilon] \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p}, \mathbb{R}^{n_x})$ and $\mathbf{x}(t)$ be the solution of the problem

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{p}), \qquad t \in]\hat{t} - \epsilon, \hat{t} + \epsilon[, \qquad \mathbf{x}(\hat{t}_0) = \hat{\mathbf{x}}_{\mathbf{0}}. \tag{6.73}$$

Let furthermore $\hat{t}(s) = \sum_{i=0}^{k} \hat{t}_i s^i$, $\hat{\mathbf{x}}(s) = \sum_{i=0}^{k} \hat{\mathbf{x}}_{\mathbf{i}} s^i$ and $\mathbf{p}(s) = \sum_{i=0}^{k} \mathbf{p_i} s^i$ be Taylor polynomials of order k in s and denote with $\mathbf{x_0}, \ldots, \mathbf{x_k}$ the Taylor coefficients of the expansion of the solution

$$\mathbf{x}(\hat{t}_0; \hat{t}(s), \hat{\mathbf{x}}(s), \mathbf{p}(s)) = \mathbf{x_0} + \mathbf{x_1} s + \ldots + \mathbf{x_k} s^k + \mathcal{O}(s^{k+1}) \tag{6.74}$$

at time $\hat{t}_0$. Then the TCs $\mathbf{x_i}$ can be computed, e.g., by a Taylor series expansion with respect to s of the representation

$$\mathbf{x}(t; \hat{t}(s), \hat{\mathbf{x}}(s), \mathbf{p}(s)) = \hat{\mathbf{x}}(s) + \int_{\hat{t}(s)}^{t} \mathbf{f}(\tau; \mathbf{x}(\tau, \hat{t}(s), \hat{\mathbf{x}}(s), \mathbf{p}(s)), \mathbf{p}(s)) d\tau \tag{6.75}$$

around $s = 0$ and its evaluation in t_0. The first coefficients are then given by the recurrence

$$\mathbf{x_0} = \hat{\mathbf{x}}_{\mathbf{0}} \tag{6.76a}$$

$$\mathbf{x_1} = \hat{\mathbf{x}}_{\mathbf{1}} - \mathbf{f}(\hat{t}_0, \hat{\mathbf{x}}_{\mathbf{0}}, \mathbf{p_0}) \cdot \hat{t}_1 \tag{6.76b}$$

$$\mathbf{x_2} = \hat{\mathbf{x}}_{\mathbf{2}} - \mathbf{f}(\hat{t}_0, \hat{\mathbf{x}}_{\mathbf{0}}, \mathbf{p_0}) \cdot 2\hat{t}_2 - \hat{t}_1 \frac{\partial \mathbf{f}(\hat{t}_0, \hat{\mathbf{x}}_{\mathbf{0}}, \mathbf{p_0})}{\partial(t, \mathbf{x}, \mathbf{p})} \begin{pmatrix} \hat{t}_1 \\ \hat{\mathbf{x}}_{\mathbf{1}} + \mathbf{x_1} \\ 2\mathbf{p_1} \end{pmatrix} \tag{6.76c}$$

Proof:

Given $\mathbf{f}$ is sufficiently smooth, the integral in (6.75) can be differentiated with respect to both its integrand and its limits. The application of basic calculus rules then leads to the given formulas of the coefficients. It should be noted that in practice the TCs of the solution can be efficiently obtained with the help of an AD-tool and adapted model functions or, at least for lower orders, by directional derivatives of $\mathbf{f}$.

□

Remark 6.12 (TCs of ODE solution at end time)
The preceding Lemma can be used to immediately obtain the TCs of the ODE solution at the end time when the end time is given by a Taylor polynomial $\hat{t}(s)$ by simply multiplying the $\hat{t}_i, 1 \leq i \leq k$ by -1 in the formulas above.

Based on the described dependencies in the computations occurring at a switching event, the TC propagation rules through the implicit function theorem and for the computation of the start/end time TCs of an ODE solution, we formulate now in Algorithm 6.11 on page 202 and Algorithm 6.12 on page 202 a sensitivity generation scheme for arbitrary order directional forward sensitivities for the case that switching events (may) occur. In the first order case Algorithm 6.12 can also be used to compute the update matrices described earlier by using unit directions as input TCs. However, if only a small number (compared to the overall number of variables) of directional sensitivities is needed, it is much more efficient to propagate these directional sensitivities directly across the switch. Note that by application of the reverse mode of AD on the instruction level we can obtain higher-order forward/adjoint schemes for sensitivity propagation across switching events.

Example 6.13 (First and second order sensitivities of a bouncing ball)
Consider a ball thrown away in a gravitational field which bounces on the floor after some time and is reflected. We describe the system by the 4 differential states $\mathbf{x} = (p^x, v^x, p^y, v^y)$ describing the balls horizontal and vertical position and velocity. The initial equations of motions are then given by

$$\frac{\partial p^x(\tau)}{\partial \tau} = v^x \qquad \frac{\partial v^x(\tau)}{\partial \tau} = 0$$
$$\frac{\partial p^y(\tau)}{\partial \tau} = v^y \qquad \frac{\partial v^y(\tau)}{\partial \tau} = -g,$$

with initial values $\mathbf{x^1}(\tau_0) = (p_0^x, v_0^x, p_0^y, v_0^y)$. The parameter g represents here the vertical acceleration of the ball due to the gravitational field.
The event that the ball hits the floor is then characterized by the root of the switch function $\sigma = p^y$. We describe the reflexion of the ball with the jump function $\Delta(t, v^y, r) = (0, 0, 0, -(1+r)v^y)^T$, where r is a damping factor. We further assume that the floor consists of sensor elements, that with each contact activate or deactivate, respectively, a force field. If activated, it causes a horizontal acceleration a of the ball. We assume that at time t_0 this force field is deactivated. Combining

$\mathbf{p} = (g, r, a)$ we then have the two model functions for the system

$$\mathbf{f^1}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} v^x \\ 0 \\ v^y \\ -g \end{pmatrix} \qquad \mathbf{f^2}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} v^x \\ a \\ v^y \\ -g \end{pmatrix}.$$

In this relative simple setup we can still determine the solution analytically. Assume that τ_e is chosen for simplicity in a way that the only one ground contact takes place at τ_s. Then we have as solution $\mathbf{x^1}(\tau)$ on the first part ($\tau_0 \leq \tau \leq \tau_s$)

$$\begin{aligned} p^x(\tau) &= p_0^x + v_0^x(\tau - \tau_0), & v^x(\tau) &= v_0^x, \\ p^y(\tau) &= p_0^y + v_0^y(\tau - \tau_0) - \frac{1}{2}g(\tau - \tau_0)^2, & v^y(\tau) &= v_0^y - g(\tau - \tau_0). \end{aligned} \tag{6.77}$$

From this we can determine the impact time, where $\sigma = p^y(\tau) = 0$, as

$$\tau_s = \frac{g\tau_0 + v_0^y + \sqrt{(v_0^y)^2 + 2gp_0^y}}{g}. \tag{6.78}$$

Denoting the states of $\mathbf{x^2}(\tau_s)$ at the switch time (after the application of the jump) with an subscript $\cdot_s$, we obtain as solution $\mathbf{x^2}(\tau)$ on the second part ($\tau_s \leq \tau \leq \tau_\mathrm{e}$)

$$\begin{aligned} p^x(\tau) &= p_s^x + v_0^x(\tau - \tau_s) + \frac{1}{2}a(\tau - \tau_s)^2, & v^x(\tau) &= v_s^x + a(\tau - \tau_s), \\ p^y(\tau) &= p_s^y + v_s^y(\tau - \tau_s) - \frac{1}{2}g(\tau - \tau_s)^2, & v^y(\tau) &= v_s^y - g(\tau - \tau_s). \end{aligned} \tag{6.79}$$

By inserting the expressions for τ_s and the solutions of the first part at τ_0, as well as the state jump, we obtain expressions for the solution $\mathbf{x^2}(\tau_\mathrm{e})$ at the end time in terms of initial values and parameter, which can then be differentiated analytically to obtain the desired sensitivities of the solutions. The calculation of these expressions is straight-forward, but technically and leads to large and complex terms, such that we state here only the numerical values of some sensitivities of interest. For the computations to follow, we assume a setup described by the values

$$\begin{aligned} \tau_0 &= 0 & p_0^x &= 0 & v_0^x &= 1 \\ \tau_\mathrm{e} &= 3 & p_0^y &= 1.8 & v_0^y &= 8 \\ g &= 10 & r &= 0.9 & a &= 2. \end{aligned} \tag{6.80}$$

Then the switch time is obtained as $\tau_s = 1.8$. The solution at τ_s before the jump is given by $\mathbf{x^1}(\tau_s) = (\frac{9}{5}, 1, 0, -10)^T$ and at the end time by $\mathbf{x^2}(\tau_\mathrm{e}) = (\frac{111}{25}, \frac{17}{5}, \frac{18}{5}, -3)^T$. The complete first order sensitivities that we obtain from the analytical representation of the complete solution are

$$\mathcal{W}(\tau_\mathrm{e}) \equiv \frac{\partial \mathbf{x^2}(\tau_\mathrm{e})}{\partial(\mathbf{x_0}, \mathbf{p})} = \begin{pmatrix} 1 & 3 & -\frac{6}{25} & -\frac{54}{125} & \frac{243}{625} & 0 & \frac{18}{25} \\ 0 & 1 & -\frac{1}{5} & -\frac{9}{25} & \frac{81}{250} & 0 & \frac{6}{5} \\ 0 & 0 & \frac{69}{50} & \frac{351}{250} & -\frac{2529}{2500} & 12 & 0 \\ 0 & 0 & \frac{19}{10} & \frac{63}{25} & -\frac{1329}{500} & 10 & 0 \end{pmatrix}. \tag{6.81}$$

The second order sensitivities with respect to p_0^y, v_0^y and g are then given by

$$\frac{\partial^2 \mathbf{x^2}(\tau_e)}{\partial (p_0^y)^2} = \begin{pmatrix} \frac{11}{250} \\ \frac{1}{50} \\ -\frac{209}{500} \\ -\frac{19}{100} \end{pmatrix}, \qquad \frac{\partial^2 \mathbf{x^2}(\tau_e)}{\partial (v_0^y)^2} = \begin{pmatrix} \frac{351}{6250} \\ -\frac{9}{1250} \\ -\frac{6669}{12500} \\ \frac{171}{2500} \end{pmatrix}, \qquad \frac{\partial^2 \mathbf{x^2}(\tau_e)}{\partial g^2} = \begin{pmatrix} \frac{-15309}{625000} \\ -\frac{8019}{125000} \\ -\frac{632529}{1250000} \\ -\frac{1539}{250000} \end{pmatrix}. \tag{6.82}$$

We will now show how the (first and) second order sensitivities of the solution at the end time with respect to p_0^y, v_0^y and g can be computed by the forward IND-TC propagation scheme presented in the Algorithms 6.11 and 6.12. Analogously to Example 2.20 on page 48, we initialize the TCs $\mathbf{X^0}$ and $\mathbf{P}$ for the different directional sensitivities at the begin of the integration process according to the corresponding rays to

$$\mathbf{X^0_{p_y^0}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{9}{5} & 2 & 0 \\ 8 & 0 & 0 \end{bmatrix}, \mathbf{P_{p_y^0}} = \begin{bmatrix} 10 & 0 & 0 \\ \frac{9}{10} & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \qquad \mathbf{X^0_{v_y^0}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{9}{5} & 0 & 0 \\ 8 & 2 & 0 \end{bmatrix}, \mathbf{P_{v_y^0}} = \begin{bmatrix} 10 & 0 & 0 \\ \frac{9}{10} & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix},$$
$$\mathbf{X^0_g} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{9}{5} & 0 & 0 \\ 8 & 0 & 0 \end{bmatrix}, \mathbf{P_g} = \begin{bmatrix} 10 & 2 & 0 \\ \frac{9}{10} & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}. \tag{6.83}$$

We then use the ordinary second order forward IND-TC sweep including switch detection, as described in Algorithm 6.11, to obtain the TCs $\mathbf{X}^-$ at the switch time $\tau_s = 1.8$ as

$$\mathbf{X^-_{p_y^0}} = \begin{bmatrix} \frac{9}{5} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \\ -10 & 0 & 0 \end{bmatrix}, \qquad \mathbf{X^-_{v_y^0}} = \begin{bmatrix} \frac{9}{5} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & \frac{18}{5} & 0 \\ -10 & 2 & 0 \end{bmatrix}, \qquad \mathbf{X^-_g} = \begin{bmatrix} \frac{9}{5} & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -\frac{81}{25} & 0 \\ -10 & -\frac{18}{5} & 0 \end{bmatrix}. \tag{6.84}$$

Next we compute the switch time TCs $\boldsymbol{\tau_s}$ as described in Algorithm 6.12. Using Lemma 6.10 with $\mathbf{v} = \tau_s$, $\mathbf{u} = ((\mathbf{x}^-)^T, \mathbf{p}^T)^T$ we first obtain with $\mathbf{f}^- = (1, 0, -10, -10)^T$ (skipping the arguments)

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{v}} = \frac{\partial \sigma}{\partial \tau} + \frac{\partial \sigma}{\partial \mathbf{x}^-}\mathbf{f}^- = -10 \quad \text{and} \quad \frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{u}} = \frac{\mathrm{d}\sigma}{\mathrm{d}(\mathbf{x}^-, \mathbf{p})} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{6.85}$$

such that

$$\left(\frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{v}}\right)^{-1} \frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{u}} = \begin{pmatrix} 0 & 0 & -\frac{1}{10} & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{6.86}$$

With Lemma 6.10 we then obtain

$$\tau_{s,1}^{p_y^0} = -(-\frac{1}{10}) \cdot 2 = \frac{1}{5}, \quad \tau_{s,1}^{v_y^0} = -(-\frac{1}{10}) \cdot \frac{18}{5} = \frac{9}{25}, \quad \tau_{s,1}^{g} = -(-\frac{1}{10}) \cdot (-\frac{81}{25}) = -\frac{81}{250}. \tag{6.87}$$

To compute the $\tau_{s,2}$ we perform a forward TC propagation sweep of second order to compute $\hat{\sigma}_2$. As inputs we use the zeroth and first order coefficients of $\boldsymbol{\tau}_\mathbf{s}$, which we just have computed as well as the ones of $\mathbf{X}^-$ and $\mathbf{P}$. The second order coefficients are set to zero for all inputs. By using the formulas (6.76a)-(6.76c) of Lemma 6.11 (and negating the τ_s) we then obtain

$$\hat{\sigma}_2^{p_y^0} = -\frac{1}{5}, \quad \hat{\sigma}_2^{v_y^0} = \frac{9}{125}, \quad \hat{\sigma}_2^{g} = \frac{8019}{12500}, \tag{6.88}$$

and hence finally the switch time TCs

$$\boldsymbol{\tau}_\mathbf{s}^{\mathbf{p_y^0}} = \begin{bmatrix} \frac{9}{5} & \frac{1}{5} & -\frac{1}{50} \end{bmatrix}, \qquad \boldsymbol{\tau}_\mathbf{s}^{\mathbf{v_y^0}} = \begin{bmatrix} \frac{9}{5} & \frac{9}{25} & \frac{9}{1250} \end{bmatrix}, \qquad \boldsymbol{\tau}_\mathbf{s}^{\mathbf{g}} = \begin{bmatrix} \frac{9}{5} & -\frac{81}{250} & \frac{8019}{125000} \end{bmatrix}. \tag{6.89}$$

Using again the formulas in Lemma 6.11 based on $-\boldsymbol{\tau}_\mathbf{s}$, $\mathbf{X}^-$, $\mathbf{P}$ we then adapt the TCs of the solution $\mathbf{x}^-$ at the switch time before the application of the jump according to its end time dependency. This leads to the adapted TCs $\mathbf{X}^{\mathbf{J}-}$ given by

$$\mathbf{X}_{\mathbf{p_y^0}}^{\mathbf{J}-} = \begin{bmatrix} \frac{9}{5} & \frac{1}{5} & -\frac{1}{50} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ -10 & -2 & \frac{1}{5} \end{bmatrix}, \quad \mathbf{X}_{\mathbf{v_y^0}}^{\mathbf{J}-} = \begin{bmatrix} \frac{9}{5} & \frac{9}{25} & \frac{9}{1250} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ -10 & -\frac{1}{5} & -\frac{9}{125} \end{bmatrix}, \quad \mathbf{X}_{\mathbf{g}}^{\mathbf{J}-} = \begin{bmatrix} \frac{9}{5} & -\frac{81}{250} & \frac{8019}{125000} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ -10 & -\frac{9}{25} & \frac{81}{12500} \end{bmatrix}. \tag{6.90}$$

We then propagate $\boldsymbol{\tau}_\mathbf{s}$, $\mathbf{X}_{\mathbf{J}-}$, $\mathbf{P}$ through the jump based on the jump function $\boldsymbol{\Delta}$ to obtain the TCs $\mathbf{X}_{\mathbf{J}+}$ after the jump as

$$\mathbf{X}_{\mathbf{p_y^0}}^{\mathbf{J}+} = \begin{bmatrix} \frac{9}{5} & \frac{1}{5} & -\frac{1}{50} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 9 & \frac{9}{5} & -\frac{9}{50} \end{bmatrix}, \qquad \mathbf{X}_{\mathbf{v_y^0}}^{\mathbf{J}+} = \begin{bmatrix} \frac{9}{5} & \frac{9}{25} & \frac{9}{1250} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 9 & \frac{36}{25} & \frac{81}{1250} \end{bmatrix}, \qquad \mathbf{X}_{\mathbf{g}}^{\mathbf{J}+} = \begin{bmatrix} \frac{9}{5} & -\frac{81}{250} & \frac{8019}{125000} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 9 & \frac{81}{250} & -\frac{729}{125000} \end{bmatrix}. \tag{6.91}$$

Finally we adapt the $\mathbf{X}^{\mathbf{J}+}$ according to the start time dependency of $\mathbf{x}^+$. We compute the corrected TCs $\mathbf{X}^{\mathbf{J}+}$ by using again Lemma 6.11 based on $\boldsymbol{\tau}_\mathbf{s}$, $\mathbf{X}^{\mathbf{J}+}$, $\mathbf{P}$ as well as the value and Jacobian of $\mathbf{f}^+$ at the switch after the application of the jump

$$\mathbf{f}^+ = \begin{pmatrix} 1 \\ 2 \\ 9 \\ -10 \end{pmatrix}, \qquad \frac{\partial \mathbf{f}^+}{\partial(t, \mathbf{x}, \mathbf{p})} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}. \tag{6.92}$$

which leads to

$$\mathbf{X}_{\mathbf{p_y^0}}^{+} = \begin{bmatrix} \frac{9}{5} & 0 & \frac{1}{25} \\ 1 & -\frac{2}{5} & \frac{1}{25} \\ 0 & -\frac{9}{5} & -\frac{19}{50} \\ 9 & \frac{19}{5} & -\frac{19}{50} \end{bmatrix}, \quad \mathbf{X}_{\mathbf{v_y^0}}^{+} = \begin{bmatrix} \frac{9}{5} & 0 & \frac{81}{625} \\ 1 & -\frac{18}{25} & -\frac{9}{625} \\ 0 & -\frac{81}{25} & -\frac{1539}{1250} \\ 9 & \frac{126}{25} & \frac{171}{1250} \end{bmatrix}, \quad \mathbf{X}_{\mathbf{g}}^{+} = \begin{bmatrix} \frac{9}{5} & 0 & \frac{6561}{62500} \\ 1 & \frac{81}{125} & -\frac{8019}{62500} \\ 0 & \frac{729}{250} & -\frac{124659}{125000} \\ 9 & -\frac{729}{250} & -\frac{1539}{125000} \end{bmatrix}. \tag{6.93}$$

The $\mathbf{X}^+$ are then used as start values for an ordinary second order forward IND-TC sweep from τ_s to τ_e. The resulting TCs containing the desired (scaled) sensitivities at the end time are then given by

$$\mathbf{X}^{\mathbf{e}}_{\mathbf{p}^0_{\mathbf{y}}} = \begin{bmatrix} \frac{111}{25} & -\frac{12}{25} & \frac{11}{125} \\ \frac{17}{5} & -\frac{2}{5} & \frac{1}{25} \\ \frac{18}{5} & \frac{69}{25} & -\frac{209}{250} \\ -3 & \frac{19}{5} & -\frac{19}{50} \end{bmatrix}, \quad \mathbf{X}^{\mathbf{e}}_{\mathbf{v}^0_{\mathbf{y}}} = \begin{bmatrix} \frac{111}{25} & -\frac{108}{125} & \frac{351}{3125} \\ \frac{17}{5} & -\frac{18}{25} & -\frac{9}{625} \\ \frac{18}{5} & \frac{351}{125} & -\frac{6669}{6250} \\ -3 & \frac{126}{25} & \frac{171}{1250} \end{bmatrix}, \quad \mathbf{X}^{\mathbf{e}}_{\mathbf{g}} = \begin{bmatrix} \frac{111}{25} & \frac{486}{625} & -\frac{15309}{312500} \\ \frac{17}{5} & \frac{81}{125} & -\frac{8019}{62500} \\ \frac{18}{5} & -\frac{2529}{1250} & -\frac{632529}{625000} \\ -3 & -\frac{1329}{250} & -\frac{1539}{125000} \end{bmatrix}. \tag{6.94}$$

Performing exact interpolation by dividing the first and second order coefficients of the TCs at the end time by 2 and comparing them with (6.82) and (6.81) we see that by following the propagation scheme described in Algorithms 6.11 and 6.12 we have computed the desired elements of the sensitivity Hessian tensor along with the corresponding columns of the Wronskian matrix. We further observe that all needed model function derivatives can be computed efficiently either by directional derivatives or (IND-)TC propagation. Finally, it should be noted that if we use this scheme for a first order sweep the actual performed operations are identical to the operation needed when using the formulas (6.67), (6.68) and (6.70) to compute first order sensitivities, only that now all needed derivatives are evaluated as directional derivatives. Hence also for the computation of first order sensitivities the application of the newly proposed scheme does normally lead to an improved performance, as no computation of the complete update matrices is needed.

Algorithm 6.11: Arbitrary order forward IND-TC scheme including switches

Input: τ_0, τ_e, h_0, initial values $\mathbf{y_0}$, parameter $\mathbf{p}$, sensitivity direction $\mathbf{d}$, sensitivity order m
Output: Taylor coefficients $\mathbf{Y_N}$ containing nominal solution $\mathbf{y_N}$ and scaled forward sensitivities $\frac{1}{j!}\dot{\mathbf{w}}_\mathbf{N}^{(j)}$ for $1 \leq j \leq m$.

set $k_0 = 1$, $n = 0$;
initialize with Taylor coefficients (6.41) according to initial values and sensitivity direction $\mathbf{d}$;
while τ_e *not reached* **do**
  compute $\mathbf{y_{n+1}}$ by nominal integration step;
  if *step accepted* **then**
    propagate Taylor polynomials to obtain $\mathbf{Y_{n+1}}$;
    perform switch detection;
    if *switch detected* **then**
      determine switch time τ_s;
      compute trajectory values $\mathbf{y}^-$ and TCs $\mathbf{Y}^-$ at τ_s from continuous representation;
      propagate TCs across switch event as described in Algorithm 6.12;
      restart integration at τ_s with $\mathbf{Y}^+$ and possibly switched model functions;
    end
    $t_{n+1} = t_n + h_n$;
    determine h_{n+1} and k_{n+1} for next step;
    $n = n + 1$;
  else
    update Jacobian approximation $\mathbf{M_n}$ according to monitor strategy or reduce stepsize h_n;
  end
end
$N = n$;

Algorithm 6.12: Forward TC propagation across switching event

Input: Switch time τ_s, Taylor coefficients $\mathbf{Y}^-$ and $\mathbf{P}$, sensitivity order m
Output: Taylor coefficients $\mathbf{Y}^+$ for further propagation after the switch event.

using τ_s, $\mathbf{y_0}^-$ and $\mathbf{p_0}$ compute $\mathbf{f}^- = \frac{\partial \mathbf{x}^-(\tau_s)}{\partial t}$ as well as $\frac{\partial \mathbf{z}^-(\tau_s)}{\partial t}$ by using the implicit function theorem on $\mathbf{g}$;

compute $\tau_{s,1}, \ldots, \tau_{s,m}$ by application of Lemma 6.10 (intertwined with Lemma 6.11) with $\mathbf{v} = \tau_s$ and $\mathbf{u} = (\mathbf{y}^-, \mathbf{p})$ to the switch function σ;

adapt $\mathbf{Y}^-$ according to the end time dependency of $\mathbf{y}^-$ using Lemma 6.11 to obtain adapted TCs $\mathbf{X^{J-}}$ and along with them $\mathbf{Z^{J-}}$ by application of Lemma 6.10 to $\mathbf{g}$;

propagate $\boldsymbol{\tau_s}$, $\mathbf{Y^{J-}}$ and $\mathbf{P}$ through the jump using $\boldsymbol{\Delta}$ and obtain $\mathbf{Y^{J+}}$;

adapt $\mathbf{Y^{J+}}$ according to the start time dependency of $\mathbf{y}^+$ using Lemma 6.11 to obtain adapted TCs $\mathbf{X}^+$ and along with them $\mathbf{Z}^+$ by application of Lemma 6.10 to $\mathbf{g}$;

7 A lifted exact-Hessian SQP method for OCPs with DAEs

In Chapter 4 we have shown how the lifting idea can be used to obtain an exact-Hessian Sequential Quadratic Programming (SQP) method for the solution of Nonlinear Programs (NLPs) with a certain internal structure of the problem functions. Furthermore, we have presented in Chapter 6 how Taylor Coefficient (TC) propagation in connection with the principle of Internal Numerical Differentiation (IND) can be used to derive efficient schemes for the computation of higher-order directional sensitivities of Initial Value Problem (IVP) solutions for Differential Algebraic Equations (DAEs). In this chapter we will now describe how both ideas can be combined and adapted to create an efficient Lifted Partially Reduced Sequential Quadratic Programming (L-PRSQP) algorithm with exact Hessians for the solution of Optimal Control Problems (OCPs) involving DAEs in the framework of direct multiple shooting.

7.1 The fundamentals of the method

We consider for now a (single stage) OCP for index 1 DAEs of the type (1.1) with a Mayer term cost functional and possibly coupled (multi-)point constraints, which includes interior point and boundary constraints as special cases. Furthermore, we assume that we have introduced the length of the horizon as parameter, using the time transformation described in Section 1.1, and that this is already accounted for in the model functions, such that the OCP has the following form

$$
\begin{aligned}
&\min_{\mathbf{u}(\cdot),\mathbf{x}(\cdot),\mathbf{z}(\cdot),\mathbf{p}} && c\big(\mathbf{x}(1),\mathbf{z}(1),\mathbf{p}\big) \\
&\text{s.t.} \\
\mathbf{A}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p})\cdot\dot{\mathbf{x}}(t) &= \mathbf{f}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}), && t\in T=[0,1] \\
\mathbf{0} &= \mathbf{g}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}) \\
\mathbf{0} &\leq \mathbf{h}^{\text{cont}}(t,\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{p}) \\
\mathbf{0} &\left\{\begin{matrix}=\\ \leq\end{matrix}\right\} \mathbf{h}^{\text{point}}(\{t_i,\mathbf{x}(t_i),\mathbf{z}(t_i)|t_i\in T, 1\leq i\leq n_{\text{pct}}\},\mathbf{p}),
\end{aligned}
$$

where the point constraints shall also contain the fixation of the initial values $\mathbf{x}(0)=\mathbf{x_0}$.
We apply now the direct multiple shooting approach presented in Section 1.2.3 to transform this OCP into a NLP. In this process, we make use of the relaxed formulation of the algebraic equations (cf. Section 5.3.7). For notational convenience we assume that the discretization grids of the control functions and the continuous path and control constraints are chosen identical with

the multiple shooting grid, defined by the timepoints $0 = t_0 < t_1 < \ldots < t_i < \ldots < t_{n_{\mathrm{ms}}} = 1$. Furthermore, we assume that the timepoints, in which the point conditions are defined, are also part of the shooting grid. In the end, this leads to the following structured NLP

$$\begin{aligned}
\min_{\mathbf{w_0^x}, \mathbf{w_0^z}, \mathbf{u_0}, \ldots, \mathbf{w_{n_{\mathrm{ms}}}^x}, \mathbf{w_{n_{\mathrm{ms}}}^z}, \mathbf{u_{n_{\mathrm{ms}}}}, \mathbf{p}} \quad & c\big(\mathbf{w_{n_{\mathrm{ms}}}^x}, \mathbf{w_{n_{\mathrm{ms}}}^z}, \mathbf{p}\big) && \text{(7.2a)} \\
\text{s.t.} \quad & && \\
\mathbf{0} &= \mathbf{w_{i+1}^x} - \mathbf{x}(t_{i+1}; t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p}), \quad 0 \le i \le n_{\mathrm{ms}} - 1 && \text{(7.2b)} \\
\mathbf{0} &= \mathbf{g}(t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p}), \quad 0 \le i \le n_{\mathrm{ms}} && \text{(7.2c)} \\
\mathbf{0} &\le \mathbf{h}^{\mathrm{cont}}(t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p}), \quad 0 \le i \le n_{\mathrm{ms}} && \text{(7.2d)} \\
\mathbf{0} &\begin{Bmatrix} = \\ \le \end{Bmatrix} \mathbf{h}^{\mathrm{point}}(\{t_i, \mathbf{w_i^x}, \mathbf{w_i^z} | 1 \le i \le n_{\mathrm{ms}}\}, \mathbf{p}), && \text{(7.2e)}
\end{aligned}$$

where we use here and in the following for notational simplicity the same symbols for the functions as in the continuous OCP before, even if, e.g., they only depend now indirectly on the control parameter. We now like to solve this NLP using a structure exploiting exact-Hessian SQP method based on the lifting approach presented in Chapter 4.

7.1.1 The structure of the QP subproblem

To see how we can use the lifting approach in this case, we first have a closer look at the structure of the QP subproblems arising in the solution of (7.2) using an SQP method.

In each step of the SQP method, the following subproblem must be solved for the determination of the step in the variables and for the determination of the new multiplier values (cf. Section 3.3.1). Note that in the following sometimes superfluous dependencies, e.g., for the cost functional or for the point constraints, are not eliminated for notational convenience.

$$\min_{\mathbf{\Delta w^x}, \mathbf{\Delta w^z}, \mathbf{\Delta u}, \mathbf{\Delta p}} \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta w_0^z} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^x} \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^z} \\ \mathbf{\Delta u_{n_{\mathrm{ms}}}} \\ \mathbf{\Delta p} \end{pmatrix}^T \mathbf{B} \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta w_0^z} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^x} \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^z} \\ \mathbf{\Delta u_{n_{\mathrm{ms}}}} \\ \mathbf{\Delta p} \end{pmatrix} + \boldsymbol{\nabla} c^T \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta w_0^z} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^x} \\ \mathbf{\Delta w_{n_{\mathrm{ms}}}^z} \\ \mathbf{\Delta u_{n_{\mathrm{ms}}}} \\ \mathbf{\Delta p} \end{pmatrix} \qquad \text{(7.3a)}$$

s.t.

$$\mathbf{0} = \mathbf{d}_i + \mathbf{\Delta w^x_{i+1}} - \frac{\partial \mathbf{x}(t_{i+1}; t_i, \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p})}{\partial(\mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p})} \begin{pmatrix} \mathbf{\Delta w^x_i} \\ \mathbf{\Delta w^z_i} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}, \quad 0 \leq i \leq n_{\text{ms}} - 1 \tag{7.3b}$$

$$\mathbf{0} = \mathbf{g_i} + \frac{\partial \mathbf{g}(t_i, \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p})}{\partial \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p}} \begin{pmatrix} \mathbf{\Delta w^x_i} \\ \mathbf{\Delta w^z_i} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}, \quad 0 \leq i \leq n_{\text{ms}} \tag{7.3c}$$

$$\mathbf{0} \leq \mathbf{h^{\text{cont}}_i} + \frac{\partial \mathbf{h}^{\text{cont}}(t_i, \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p})}{\partial(\mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p})} \begin{pmatrix} \mathbf{\Delta w^x_i} \\ \mathbf{\Delta w^z_i} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}, \quad 0 \leq i \leq n_{\text{ms}} \tag{7.3d}$$

$$\mathbf{0} \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{h}^{\text{point}}(\ldots) + \sum_{i=0}^{n_{\text{ms}}} \frac{\partial \mathbf{h}^{\text{point}}(\ldots)}{\partial(\mathbf{w^x_i}, \mathbf{w^z_i})} \begin{pmatrix} \mathbf{\Delta w^x_i} \\ \mathbf{\Delta w^z_i} \end{pmatrix} + \frac{\partial \mathbf{h}^{\text{point}}(\ldots)}{\partial \mathbf{p}} \mathbf{\Delta p}, \tag{7.3e}$$

where we define

$$\begin{aligned} \mathbf{\Delta w^x} &:= (\mathbf{\Delta w^x_0}, \ldots, \mathbf{\Delta w^x_{n_{\text{ms}}}}) \\ \mathbf{\Delta w^z} &:= (\mathbf{\Delta w^z_0}, \ldots, \mathbf{\Delta w^z_{n_{\text{ms}}}}) \\ \mathbf{\Delta u} &:= (\mathbf{\Delta u_0}, \ldots, \mathbf{\Delta u_{n_{\text{ms}}}}) \\ \mathbf{d}_i &:= \mathbf{w^x_{i+1}} - \mathbf{x}(t_{i+1}; t_i, \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p}) \\ \mathbf{g_i} &:= \mathbf{g}(t_i, \mathbf{w^x_i}, \mathbf{w^z_i}, \mathbf{u_i}, \mathbf{p}) \end{aligned}$$

and abbreviate analogously for the other functions. As a result, the corresponding KKT Matrix has a characteristic structure that is depicted in Figure 7.1 on the following page.
The classical condensing approach [BP84] would first compute all the quantities, especially the complete Hessian blocks and constraint Jacobians, in the above QP and then eliminates the steps in the differential nodes $\mathbf{\Delta w^x_i}, 1 \leq i \leq n_{\text{ms}}$ from the problem by using the linearized continuity conditions (7.3b). Note that in the case of fixed initial values, $\mathbf{\Delta w^x_0}$ can also be eliminated trivially. The QP has now been condensed to a smaller problem in $\mathbf{\Delta w^z}$, $\mathbf{\Delta u}$ and $\mathbf{\Delta p}$, from which afterwards the step $\mathbf{\Delta w^x}$ (and also the new multiplier of the continuity conditions) can be expanded.
Note furthermore that in the more general case of nonlinearly coupled multi-point constraints or a cost functional that does not decouple properly on the different shooting intervals, the structure of the Hessian would not be block-sparse, but dense. While this would not make a difference for the lifted approach presented here, it would render the classical condensing approach more complicated. Here either some reformulations would be needed to regain some structure, or in the worst case the complete full-space Hessian would have to be approximated.

In principle, we could now employ the lifted exact-Hessian SQP approach from Chapter 4 in a straightforward way to solve the NLP (7.2). The function to be lifted in this context is the

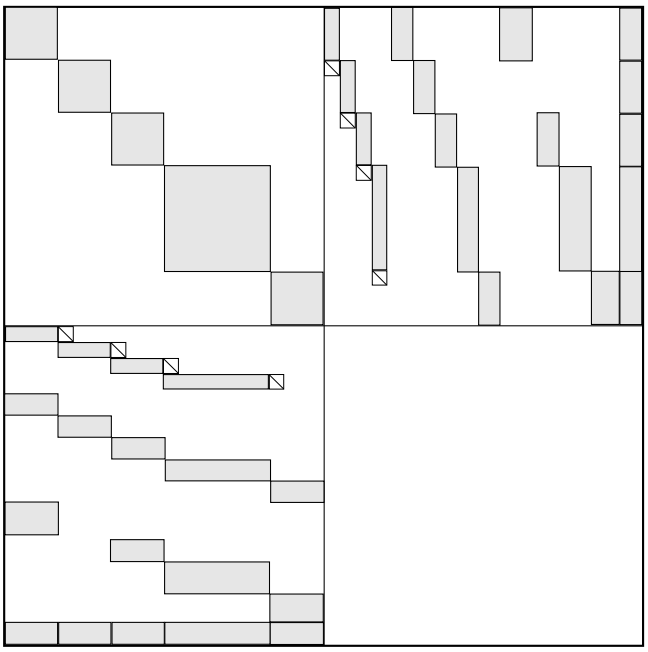

Figure 7.1: The characteristic structure of the (full-space) KKT-Matrix of the QPs arising in the direct multiple shooting approach, if we assume the decoupling of the cost functional and only linearly coupled multi-point constraints. Note that free global parameter possibly have to be "localized" on the intervals to obtain above structure. The Hessian (upper left part) is block-sparse and also the constraints (lower left part) are typically to a large extent block sparse. From top to bottom we see here the structure of the linearized continuity conditions, the linearized consistency conditions, the linearized decoupled node constraints and the linearized coupled node constraints.

combined evaluation of the Lagrange gradient and of the discretized path and control constraints (7.2d), the point constraints (7.2e) and consistency conditions (7.2c). The nodes in the lifting sense are the differential multiple shooting node values and the corresponding multiplier of the continuity conditions. By applying the lifted Newton method to this function we then obtain a lifted exact-Hessian SQP method that computes the quantities of the condensed QP directly by univariate forward/reverse TC propagation in directions of the control parameter, parameter and the algebraic variables at the shooting nodes. Unfortunately, for large scale systems with a lot of algebraic states and in comparison only a few control parameter, this would lead to an inefficient approach, as the resulting QP would still be very large. Hence the aim is also to eliminate the steps in the algebraic shooting node variables from the QP, such that it is reduced in size to the "true" degrees of freedom. How this can be achieved is presented in the next section.

7.1.2 Partial reduction technique for DAEs

As mentioned before, the lifting idea can be used straightforward to compute directly a condensed QP in the steps of control parameter, parameter and algebraic variables. The reason why it is not possible to obtain easily a reduced QP only in the control and parameter steps are the possible

discontinuities in the algebraic state trajectories and the fact that the algebraic states are only defined implicitly through the other variables. This prohibits to formulate corresponding explicit matching conditions, which would be needed for the direct application of the lifting approach to eliminate the steps in the algebraic variables in a fully automated way.

However, the partial reduction technique for DAEs presented in [Lei99] in context of the classical condensing can fortunately be transferred to the context of our lifted SQP method presented in Section 4.2.3 to overcome this problem. The partial reduction strategy for the classical condensing makes use of the index 1 assumption on the underlying DAE model and uses the linearized consistency conditions (7.3c) to eliminate the algebraic steps from the problem. Recall that due to the index 1 assumption the Jacobian of $\mathbf{g}$ with respect to the algebraic variables is invertible. Hence we can reformulate (7.3c) to

$$\mathbf{\Delta w_i^z} = -\left(\frac{\partial \mathbf{g}(t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})}{\partial \mathbf{z}}\right)^{-1} \left[\mathbf{g_i} + \frac{\partial \mathbf{g}(t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})}{\partial (\mathbf{x}, \mathbf{u}, \mathbf{p})} \begin{pmatrix} \mathbf{\Delta w_i^x} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}\right], \quad 0 \leq i \leq n_{\mathrm{ms}} \tag{7.4}$$

This relationship can be used to eliminate the steps in the algebraic variables from the QP in terms of the steps in differential variables, control parameter and parameter. One then arrives at a partially reduced QP of the form

$$\min_{\mathbf{\Delta w^x}, \mathbf{\Delta w^z}, \mathbf{\Delta u}, \mathbf{\Delta p}} \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{ms}}^x} \\ \mathbf{\Delta u_{n_{ms}}} \\ \mathbf{\Delta p} \end{pmatrix}^T \widetilde{\mathbf{B}} \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{ms}}^x} \\ \mathbf{\Delta u_{n_{ms}}} \\ \mathbf{\Delta p} \end{pmatrix} + \widetilde{\mathbf{\nabla} c}^T \begin{pmatrix} \mathbf{\Delta w_0^x} \\ \mathbf{\Delta u_0} \\ \vdots \\ \mathbf{\Delta w_{n_{ms}}^x} \\ \mathbf{\Delta u_{n_{ms}}} \\ \mathbf{\Delta p} \end{pmatrix} \tag{7.5a}$$

s.t.

$$\mathbf{0} = \widetilde{\mathbf{d}}_i + \mathbf{\Delta w_{i+1}^x} - \frac{\partial \mathbf{x}(t_{i+1}; t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})}{\partial (\mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})} \mathcal{V}_\mathbf{i} \begin{pmatrix} \mathbf{\Delta w_i^x} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}, \quad 0 \leq i \leq n_{\mathrm{ms}} - 1 \tag{7.5b}$$

$$\mathbf{0} \leq \widetilde{\mathbf{h}_\mathbf{i}^{\mathrm{cont}}} + \frac{\partial \mathbf{h}^{\mathrm{cont}}(t_i, \mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})}{\partial (\mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})} \mathcal{V}_\mathbf{i} \begin{pmatrix} \mathbf{\Delta w_i^x} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix}, \quad 0 \leq i \leq n_{\mathrm{ms}} \tag{7.5c}$$

$$\mathbf{0} \left\{\begin{matrix} = \\ \leq \end{matrix}\right\} \widetilde{\mathbf{h}^{\mathrm{point}}}(\ldots) + \sum_{i=0}^{n_{\mathrm{ms}}} \frac{\partial \mathbf{h}^{\mathrm{point}}(\ldots)}{\partial (\mathbf{w_i^x}, \mathbf{w_i^z}, \mathbf{u_i}, \mathbf{p})} \tilde{\mathcal{V}}_\mathbf{i} \begin{pmatrix} \mathbf{\Delta w_i^x} \\ \mathbf{\Delta u_i} \\ \mathbf{\Delta p} \end{pmatrix} + \frac{\partial \mathbf{h}^{\mathrm{point}}(\ldots)}{\partial \mathbf{p}} \mathbf{\Delta p}, \tag{7.5d}$$

where we define

$$
\begin{aligned}
\mathbf{G}_{\mathbf{v},\mathbf{i}} &:= \frac{\partial \mathbf{g}(t_i, \mathbf{w}_\mathbf{i}^\mathbf{x}, \mathbf{w}_\mathbf{i}^\mathbf{z}, \mathbf{u_i}, \mathbf{p})}{\partial \mathbf{v}} \quad \text{for } \mathbf{v} \in \{\mathbf{x}, \mathbf{z}, \mathbf{u}, \mathbf{p}\} \\
\widetilde{\mathbf{d}}_i &:= \mathbf{d}_i - \frac{\partial \mathbf{x}(t_{i+1}; t_i, \mathbf{w}_\mathbf{i}^\mathbf{x}, \mathbf{w}_\mathbf{i}^\mathbf{z}, \mathbf{u_i}, \mathbf{p})}{\partial \mathbf{w}_\mathbf{i}^\mathbf{z}} \mathbf{G_{z,i}}^{-1} \mathbf{g_i} \\
\widetilde{\mathbf{h}_\mathbf{i}^{\text{cont}}} &:= \mathbf{h}_\mathbf{i}^{\text{cont}} - \frac{\partial \mathbf{h}^{\text{cont}}(t_i, \mathbf{w}_\mathbf{i}^\mathbf{x}, \mathbf{w}_\mathbf{i}^\mathbf{z}, \mathbf{u_i}, \mathbf{p})}{\partial \mathbf{w}_\mathbf{i}^\mathbf{z}} \mathbf{G_{z,i}}^{-1} \mathbf{g_i} \\
\widetilde{\mathbf{h}^{\text{point}}}(\ldots) &:= \mathbf{h}^{\text{point}}(\ldots) - \sum_{i=0}^{n_{\text{ms}}} \frac{\partial \mathbf{h}^{\text{point}}(\ldots)}{\partial \mathbf{w}_\mathbf{i}^\mathbf{z}} \mathbf{G_{z,i}}^{-1} \mathbf{g_i} \\
\mathbf{V}_{\mathbf{x},\mathbf{i}} &:= \begin{pmatrix} \mathbb{I} \\ -\mathbf{G_{z,i}}^{-1} \mathbf{G_{x,i}} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \; \mathbf{V}_{\mathbf{u},\mathbf{i}} := \begin{pmatrix} \mathbf{0} \\ -\mathbf{G_{z,i}}^{-1} \mathbf{G_{u,i}} \\ \mathbb{I} \\ \mathbf{0} \end{pmatrix}, \\
\mathbf{V}_{\mathbf{p},\mathbf{i}} &:= \begin{pmatrix} \mathbf{0} \\ -\mathbf{G_{z,i}}^{-1} \mathbf{G_{p,i}} \\ \mathbf{0} \\ \mathbb{I} \end{pmatrix}, \widetilde{\mathbf{V}_{\mathbf{p},\mathbf{i}}} := \begin{pmatrix} \mathbf{0} \\ -\mathbf{G_{z,i}}^{-1} \mathbf{G_{p,i}} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \\
\mathcal{V}_\mathbf{i} &:= \begin{pmatrix} \mathbf{V_{x,i}} & \mathbf{V_{u,i}} & \mathbf{V_{p,i}} \end{pmatrix}, \; \tilde{\mathcal{V}}_\mathbf{i} := \begin{pmatrix} \mathbf{V_{x,i}} & \mathbf{V_{u,i}} & \widetilde{\mathbf{V_{p,i}}} \end{pmatrix}.
\end{aligned}
$$

The structure of this QP is still similar to the full-space QP as we can see in Figure 7.2.

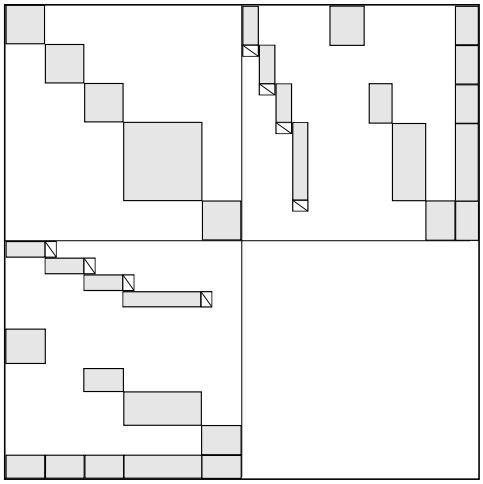

Figure 7.2: The characteristic structure of the KKT-Matrix of the QPs arising in the direct multiple shooting approach, after applying the partial reduction technique for DAEs to eliminate the algebraic variables from the problem. The transformed Hessian (upper left part) is still block-sparse and also the remaining constraints (lower left part) have very much the same block-sparse structure as before.

The quantities of this "preprocessed" QP can be computed by directional derivatives according to the direction matrices $\mathbf{V_{v,i}}$. Afterwards, the steps in the differential variables can again be

eliminated by classical condensing using the conditions (7.5b) to arrive at a small QP in $\mathbf{\Delta u}$ and $\mathbf{\Delta p}$ with a structure depicted in Figure 7.3. Once these steps are computed, $\mathbf{\Delta w^x}$ and $\mathbf{\Delta w^z}$ can be expanded based on (7.5b) and (7.4), as well as the multiplier values with the exception of the multiplier of the consistency conditions (7.3c) (see [Lei99] for more details).

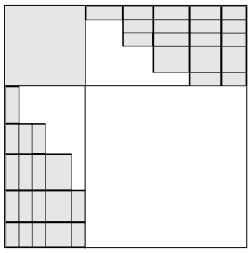

Figure 7.3: The characteristic structure of the KKT-Matrix of the QPs arising in the direct multiple shooting approach, after applying the partial reduction technique for DAEs and condensing to eliminate the differential and algebraic states from the problem. The condensed Hessian (upper left part) is now dense and the remaining condensed constraints (lower left part) have a stair-like shape.

If one compares now the quantities of this preprocessed QP and the quantities that occur during the computations of the lifted SQP, one can observe that the partial reduction and elimination of the algebraic steps can be achieved in the lifted SQP simply by substituting the original residuals $\mathbf{d}_i$ by the modified residuals $\widetilde{\mathbf{d}}_i$ in the suitable computations (and analogously for the constraints and the cost function) and by modifying the directional derivatives computed in the lifting algorithm according to the $\mathbf{V_{v,i}}$. With these modification of our lifted SQP approach we can directly compute the quantities of the condensed QP in $\mathbf{\Delta u}$ and $\mathbf{\Delta p}$ using directional derivatives only in the "true" degrees of freedom $\mathbf{u}$ and $\mathbf{p}$ (plus one)).

7.1.3 Computing the condensed QP using lifting and IND-TC propagation

We apply in our lifted SQP algorithm univariate first and second order (IND-)TC propagation for the computation of the directional derivatives of the problem functions and the directional sensitivities of the IVP solutions. These approaches were presented and developed in the Chapters 2 and 6, respectively. We now address shortly how (IND-)TC propagation is used in the multiple shooting context to perform the evaluation of the residual and the reduced function as well as of the derivatives of the reduced function that occur in the lifted Newton algorithm in Chapter 4.

First let us recall once more that the lifted exact-Hessian SQP is based on the lifted Newton method (Algorithm 4.4), where the function that is lifted is the combined evaluation of the Lagrange gradient and of the constraints (7.2d) and (7.2e) (cf. Section 4.2.3). The constraints are in

the following summarized as $\mathbf{h}$. The nodes that are introduced in the lifting context are here the differential multiple shooting node values $\mathbf{w}_\mathbf{i}^\mathbf{x}$ (also called primal nodes) and the multiplier $\boldsymbol{\lambda}_\mathbf{i}$ (also called dual nodes) of the corresponding continuity conditions (7.2b). The multiplier associated with the constraints $\mathbf{h}$ we denote in the following with $\boldsymbol{\mu}$. With $\mathbf{d}_i$ and $\bar{\mathbf{d}}_i$ we denote the primal and dual node residuals, respectively.

The different operations of the lifting framework are then performed as follows.

- To update the node residuals and to evaluate in parallel the reduced function (cf. Algorithm 4.3), in this case the constraints and the Lagrange gradient, a sequence of IVP solutions and node function evaluations is performed forward through the time horizon. This is followed by corresponding first order adjoint (IND-)TC sweeps backwards through the time horizon. This is described exemplarily in Algorithm 7.1 for the ODE case.

Algorithm 7.1: L-PRSQP: Residual and function evaluation

for $i = 0 : n_{\text{ms}} - 1$ **do**
  Solve IVP on interval $[t_i, t_{i+1}]$ for initial values $\mathbf{w}_\mathbf{i}^\mathbf{x}$, and parameter $\mathbf{u_i}$, $\mathbf{p}$ to obtain $\mathbf{x_{i+1}} := \mathbf{x}(t_{i+1}; t_i, \mathbf{w}_\mathbf{i}^\mathbf{x}, \mathbf{u_i}, \mathbf{p})$. During integration store the discretization scheme and the trajectory values on tape;
  Update $\mathbf{d}_i := \mathbf{w}_\mathbf{i+1}^\mathbf{x} - \mathbf{x_{i+1}}$;

for $i = 0 : n_{\text{ms}}$ **do**
  Compute the constraint residuals $\mathbf{h_i}$ and the adjoint derivatives of the constraints $\bar{\mathbf{h}}_\mathbf{i}$ for the adjoint derivative direction $\boldsymbol{\mu}_\mathbf{i}$;
  Compute the cost functional contribution of the node i and its gradient $\bar{\mathbf{c}}_\mathbf{i}$ with respect to the variables in node i;

for $i = n_{\text{ms}} - 1 : 0$ **do**
  Update the dual node residuals $\bar{\mathbf{d}}_i := \boldsymbol{\lambda}_\mathbf{i} - \bar{\mathbf{h}}_\mathbf{i+1} - \bar{\mathbf{c}}_\mathbf{i+1} - \bar{\mathbf{x}}_\mathbf{i+1}$ (assume $\bar{\mathbf{x}}_{\mathbf{n}_{\text{ms}}} \equiv 0$ and only consider the $\mathbf{x}$-parts of the adjoint derivatives here);
  Perform a first order adjoint IND sweep through the interval $[t_i, t_{i+1}]$ with adjoint sensitivity direction $\boldsymbol{\lambda}_\mathbf{i}$ to obtain $\bar{\mathbf{x}}_\mathbf{i}$;

Update the Lagrange gradient by adding the corresponding $\mathbf{u_i}$ and $\mathbf{p}$ parts of $\bar{\mathbf{x}}_\mathbf{0}$ as well as of $\bar{\mathbf{h}}_\mathbf{i}$ and $\bar{\mathbf{c}}_\mathbf{i}$ for $0 \leq i \leq n_{\text{ms}}$;

- To compute the derivatives of the reduced function w.r.t. the degrees of freedom, first order forward sweeps through the IVP solutions and the node function evaluations are employed, followed by second order reverse sweeps. The number of forward sensitivity directions increases here from $n_u + n_p$ on the first interval to $(n_{\text{ms}} - 1)n_u + n_p$ on the last interval. Accordingly, the number of adjoint TCs to be propagated by (IND-)TC decreases from the last to the first interval from $(n_{\text{ms}} - 1)n_u + n_p$ to $n_u + n_p$.

- The directional derivative of the reduced function in direction of the (primal and dual) node residual vector $(\mathbf{d}^T, \bar{\mathbf{d}}^T)^T$ is computed in a similar way by a first order forward (IND-)TC

sweep, followed by a second order adjoint IND-TC/TC sweep through the IVP solutions and the node function evaluations. Here only one forward and adjoint direction is needed.

- Finally, the steps $\boldsymbol{\Delta p}$, $\boldsymbol{\Delta u}$ and $\boldsymbol{\Delta \mu}$ in the parameter, the control parameter and the constraint multiplier, respectively, that are computed by the solution of the condensed QP, have to be expanded to the steps $\boldsymbol{\Delta w^x}$, $\boldsymbol{\Delta \lambda}$ and $\boldsymbol{\Delta w^z}$ in the differential node variables, the corresponding multiplier and the algebraic node variables, respectively. This can be achieved by one additional first order forward sweep through the problem functions combined with a following second order adjoint (IND-)TC sweep backwards through the time horizon for one sensitivity direction.

Note that usually only in the update of the node and constraint residuals an ordinary IVP solution with determination of the discretization grid and tape storing has to be performed. All other IND operations are performed based on the stored discretization schemes and use the taped information. This leads to a significantly increased performance by avoiding, e.g., additional matrix factorizations and step rejections. Furthermore, it ensures that the numerically computed derivative information is consistent with the numerical function evaluation.

7.1.4 The basic L-PRSQP algorithm

We now formulate our lifted exact-Hessian SQP method for DAE-OCPs, employing the partial reduction technique for DAEs and (IND-)TC propagation for the computation of the needed derivatives and sensitivities.

The basic algorithm then reads as (compare also the lifted Newton Algorithm 4.4 on page 83):

1. **Initialization.** Set $\mathbf{u}$ and $\mathbf{p}$ to their initial values. Initialize $\mathbf{w^x}$ and $\mathbf{w^z}$ either to given values or by a zero order forward sweep (the sequential solution of the corresponding initial values problems). Initialize the multiplier of the discretized path and control constraints and of the point constraints to zero. Initialize the multiplier of the continuity conditions by a first order adjoint sweep.

2. **Residual and function evaluation.** Evaluate the current primal (node) residuals as well as the constraint residuals and the cost functional by sequential zero order forward sweeps (IVP solutions) on the multiple shooting intervals. Evaluate the dual (multiplier) residuals and the Lagrange gradient by adjoint sweeps (cf. Section 7.1.3). In this context also evaluate the Jacobians of the consistency conditions with respect to the algebraic variables in the gridpoints and store them (usually these are quite sparse matrices).

3. **Termination check.** If the maximum number of iterations is exceeded or the termination criterion is fulfilled, then STOP.

4. **Computation of condensed QP quantities.** Compute the constraint Jacobians by sequential first order forward sweeps in the directions of the control parameter and parameter. Perform afterwards second order reverse sweeps to obtain the Hessian and the Lagrange gradient. Finally compute the "correction" term for the constraint residuals and the Lagrange gradient by a first order forward and following second adjoint sweep in the direction of the node and multiplier residuals.

5. **QP solution.** Compute and set the simple step bounds for control parameter and parameter. Assemble the QP data and pass them to QP solver. Initialize the active set of the QP solver with the one of the previous QP solution. Solve the QP.

6. **Step expansion.** Expand the computed step in control parameter and parameter to the step in the differential and algebraic nodes. Compute the step in the constraint multipliers for the discretized path/control constraints and point constraints from the multipliers of the QP solution. Expand this multiplier step to the step in the multiplier of the continuity conditions by a following second order adjoint sweep.

7. **Step application.** Apply the computed step in variables and multipliers, then go to 2.

7.2 Further aspects

In order to apply our proposed algorithm successfully for the efficient solution of practical applications, there are some additional aspects to consider. The most important are addressed in the following.

7.2.1 Termination criterion

An important question in practice is when to stop the algorithm. Usually, if we let the algorithm iterate freely, at one point the algorithm will mainly work to compensate quasi-random discretization and round-off errors introduced, e.g., by the IVP solution on varying integration grids. In this case, iterating further will not improve the solution significantly but only lead to unnecessary computational effort. Sometimes the solution is also needed only with a lower accuracy. Hence in general it is sensible to define a suitable termination criterion for the algorithm.

We use in our SQP algorithm the so-called KKT tolerance, that goes back to Powell and has been later used also by other authors, e.g., [CS84, Lei99]. The KKT tolerance is defined by

$$\texttt{kktTol} := |\boldsymbol{\nabla} c(\boldsymbol{\xi})^T \boldsymbol{\Delta\xi}| + \sum_{i=1}^{n_{\text{eq}}} |\lambda_i^{\text{eq}} h_i^{\text{eq}}(\boldsymbol{\xi})| + \sum_{i=1}^{n_{\text{ineq}}} |\mu_i^{\text{ineq}} h_i^{\text{ineq}}(\boldsymbol{\xi})|, \tag{7.6}$$

where $\boldsymbol{\xi}$ combines all NLP variables, $\mathbf{h}^{\text{eq}}$ and $\mathbf{h}^{\text{ineq}}$ subsume all equality and inequality constraints of the problem, respectively, and $\boldsymbol{\lambda}^{\text{eq}}$ and $\boldsymbol{\mu}^{\text{ineq}}$ are the corresponding multiplier. The KKT tolerance hence combines the possible improvement in the cost functional and the weighted constraint violations. We stop our algorithm as soon as the KKT tolerance decreases below a user given accuracy.

7.2.2 Trust region globalization

As discussed in Section 3.3.5, an exact-Hessian SQP method needs in practice a (trust region) globalization strategy. We use in our algorithm the Trust Region (TR) globalization strategy given in [Lei99]. Although global convergence of the algorithm to a local minimum based on this strategy cannot be proven rigorously, it has been used successfully for several years now in the code `MUSCOD-II` and hence proven reliable.
The TR strategy is based on the modified l_1 penalty function, given by

$$\mathcal{P}(\boldsymbol{\xi}, \nu, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = c(\xi) + \nu \sum_{i=1}^{n_1} \bar{\lambda}_{1,i}^{\text{eq}} |h_{1,i}^{\text{eq}}(\boldsymbol{\xi})| + \sum_{i=1}^{n_{\text{eq}}-n_1} \bar{\lambda}_{2,i}^{\text{eq}} |h_{2,i}^{\text{eq}}(\boldsymbol{\xi})| + \sum_{i=1}^{n_{\text{ineq}}} |\mu_i^{\text{ineq}} \min(0, h_i^{\text{ineq}}(\boldsymbol{\xi}))|, \tag{7.7}$$

where $\boldsymbol{\xi}$ combines all NLP variables, $\mathbf{h}_{\mathbf{1}}^{\text{eq}}$ contains all consistency conditions and $\mathbf{h}_{\mathbf{2}}^{\text{eq}}$ as well as $\mathbf{h}^{\text{ineq}}$ subsume all other equality and the inequality constraints of the problem, respectively. $\boldsymbol{\lambda}^{\text{eq}}$ and $\boldsymbol{\mu}^{\text{ineq}}$ are the corresponding multiplier. As common, the size of the weights must be larger than the absolute values of the corresponding multiplier to ensure exactness and compatibility with the

search direction. As the multiplier of the consistency conditions are not available, a heuristic is used for the choice of ν. For more details on this topic, we refer to [Lei99]. Based on this merit function, we employ a box trust region strategy, where the step in NLP variables in the QP is limited to $\|\boldsymbol{\Delta\xi}\|_\infty \leq \rho$. After the step candidate has been computed, it is tested for improvement of the merit function using the criterion

$$\mathcal{P}(\boldsymbol{\xi} + \boldsymbol{\Delta\xi}, \nu, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) \overset{!}{\leq} \mathcal{P}(\boldsymbol{\xi}, \nu, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) + \epsilon D_{\boldsymbol{\Delta\xi}} \mathcal{P}(\boldsymbol{\xi}, \nu, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}),$$

where, e.g., $\epsilon = 10^{-4}$ and $D_{\boldsymbol{\Delta\xi}}$ describes the directional derivative in direction $\boldsymbol{\Delta\xi}$. The trust region is then adapted as follows. If the criterion is fulfilled in the first attempt and a trust region bound is active, the trust region radius is doubled and the QP solution is repeated. If the test fails, the trust region size is decreased by the factor 2 and the QP solution is repeated. Otherwise, the trust region radius remains unchanged, the step is accepted and the algorithm proceeds.

7.2.3 Infeasible subproblems

In practice, it might happen that in some SQP iteration the QP subproblem becomes infeasible, e.g., due to a very small trust region radius. This is usually detected by the QP solver and the violated constraints are identified and hence can be obtained from the QP solver. To overcome this problem, we employ a constraint relaxation strategy which loosens the linearized constraints of the QP in the following way: For each violated constraint the violation is computed and the corresponding constraint bound is shifted by $1 + \delta$ times the violation, with, e.g., $\delta = 0.1$. Afterwards, the QP solution is repeated and the algorithm proceeds. However, the corresponding SQP iteration is marked as relaxed and after too many relaxed iteration in a row, e.g., 10, the algorithm terminates with an error.

7.2.4 Treatment of node bounds

In principle, simple bounds on the differential and algebraic nodes can be treated as inequality constraints in the multiple shooting gridpoints. Then they are partially reduced and condensed like normal constraints and finally enter the condensed QP subproblem. For larger systems, adding all these inequalities leads to QP subproblems with very many inequality constraints. This in turn complicates their solution and slows the algorithm down significantly.
However, it can observed for practical problems that most of these bounds never become active. This motivates us to use a so-called potentially active bounds strategy proposed also in [Lei99]: Only the node bounds that are marked as potentially active are added to the problem, all other node bounds are ignored. This set of potentially active bounds is empty at the beginning of the solution process. After each QP solution and step expansion it is tested, whether a node bound is violated. If this is the case, the corresponding bound will be added to the set of potentially active bounds (and will remain there for the rest of the solution process). Then the QP solution is repeated based on the new set of potentially active bounds. Note that no complete recalculation of the condensed QP quantities is needed in the case where new node bounds enter the set of potentially active bounds. All information that is needed to construct the corresponding

new condensed constraint is readily available or can be obtained by one additional directional derivative.

7.2.5 Problem scaling

Theoretically, a Newton method (and our exact-Hessian SQP could in principle be understood as such) should be invariant regarding to the problem's scaling. In practice, however, the scaling of a problem seems to have a significant impact on the performance of all types of SQP methods (see, e.g., [CS84]), which probably is often a result of error-amplification due an ill-conditioning. Also the solution of the occurring IVP problems, and here especially the internal error estimation of the integrator, relies on a proper model scaling, or on suitable scale factors (see also Section 5.3.8). In general, it would favorable when a proper scaling would have been performed completely in advance by the modeler creating the problem description. However, as this cannot always be expected and sometimes is also not possible in practice (e.g., due to auto-generated models), we offer in our algorithmic setup the possibility to scale all occurring variables, as well as the constraints and cost functionals.

7.2.6 Free initial values

We like to mention here that the presented algorithm can also be extended straightforward to the case, where a part or all of the initial (differential) states are true degrees of freedom. Even if all initial values are free our lifted SQP approach can be employed, but will probably not give rise to a speedup compared to the classical condensing approach.

7.2.7 Other cost functionals

In the problem description above, we restricted ourselves to the case of a Mayer term cost functional. However, of course other cost functional types can be used in connection with our algorithm.

A Lagrange cost functional can, e.g., simply be introduced by adding the Lagrange term as additional differential state with initial value zero and by adding the value of this state at the final time to the Mayer term.

For nonlinear least-squares problem we offer the possibility to define the least-square residual function that is then evaluated in the gridpoints. This can be used either in connection with the lifted SQP described above, or with the lifted Gauss-Newton method which is also implemented.

An extension of this is a least-square function defined on an arbitrary timegrid, which is often desired in parameter estimation problems or in optimal experimental design. This can be achieved by the use of the continuous forward sensitivity output and the adjoint sensitivity injection facilities of our integrator `DAESOL-II` (cf. Sections 6.7.3 and 6.7.4). Combined with the use of integrator plugins also a continuous least-squares function can be achieved, e.g., based on a numerical quadrature formula.

7.2.8 Multistage problems

The algorithm can be extended in a straightforward manner for the treatment of multistage problems. The forward sweeps are in this case not terminated at the end of the first stage, but instead continued through the stage transition function to the next stage and so on. Similarly, the corresponding adjoint sweeps are performed beginning on the last multiple shooting interval of the last stage and proceeding backwards to the first interval of the first stage, also passing the stage transition functions in the reverse order in this process .

7.3 Comparison with a classical condensing approach

At the end of this chapter, we give a short comparison of the estimated memory and run-time demands of our L-PRSQP algorithm and an exact-Hessian PRSQP algorithm employing classical condensing, such as implemented in `MUSCOD-II` [DLS01].
We assume in the following that we solve a one stage problem with Mayer term cost functional, n_{ms} shooting intervals, n_{constr} decoupled constraints in each timepoint of the multiple shooting grid and fixed initial values. Furthermore, we assume a piecewise constant discretization of the controls, such that we have n_u control parameter for each interval.

To compare now our lifting based strategy with the approach using classical condensing, we first note that for large scale problems usually the costs for derivative/sensitivity computations dominate the costs of the overall solution process. In this particular comparison, the cost for sensitivity generation also constitutes the main difference between the two approaches, as the size of the QP subproblems solved in each step is identical and comparably small.

If we consider now a PRSQP approach based on the classical condensing, the main computational costs here are related to the computation of the Hessians (and Jacobians) of the problem functions, the generation of the sensitivities of the IVP-solutions needed for the condensing, and the condensing of the QP subproblem itself. In our lifting approach, the costs associated with condensing do not exist, and the Hessians and Jacobian only need to be computed in a smaller subspace, given by a set of directions. On the other hand, another directional derivative is needed here for the expansion of the QP solution to the step in the NLP variables (cf. Section 4.2.3). In the end, the complexity of the lifted approach does not depend on the number of states of the problem.

Regarding the memory demands of the two approaches, the classical condensing needs to store the complete Hessian matrix of the Lagrangian and the complete constraint Jacobians w.r.t the differential states, parameter and control parameter as well as the full sensitivity matrices on the multiple shooting intervals. Note that the Hessian of the Lagrange function and the constraint Jacobians are block-sparse, where the degree of sparsity is mainly determined by the number and the dependencies of the coupled constraints. Without nonlinearly coupled constraints (and using "localized" parameter), the Hessian will be block-diagonal, where one block corresponds to one multiple shooting interval.

The Tables 7.1 and 7.2 show the complexity of the computational costs for the classical approach and the lifted approach, respectively, and Table 7.3 on the following page gives a comparison of the memory demand of both approaches for the case of decoupled (or at most linearly coupled multi-point) constraints. For a more detailed explanation on the computational costs and the memory demand arising in the classical approach refer, e.g., to [Lei99].

Computational task	Complexity classical condensing
Hessian + constr. Jac.	$\mathcal{O}(n_{\text{ms}}(n_x + n_p + n_u))$ [IVP]
(fwd/adj IND-TC)	$\mathcal{O}(n_{\text{constr}} n_{\text{ms}}(n_x + n_p + n_u))$ [F]
Hessian + constr. Jac.	$\mathcal{O}(n_{\text{ms}}(n_x + n_p + n_u)^2)$ [IVP]
(Finite diff.)	$\mathcal{O}(n_{\text{constr}} n_{\text{ms}}(n_x + n_p + n_u)^2)$ [F]
Condensing	$\mathcal{O}(n_{\text{ms}}(n_{\text{constr}} + n_x) n_x (n_x + n_p + n_u))$ [O]
QP solution	$\mathcal{O}((n_p + n_{\text{ms}} n_u)^3)$[O]
Step expansion	$\mathcal{O}(n_{\text{ms}} n_x (n_x + n_p + n_u))$[O]
Other calculations	$\mathcal{O}(n_{\text{ms}}(n_{\text{constr}} + n_x + n_p + n_u))$[O]

Table 7.1: Estimated costs of the tasks in one SQP step of methods based on classical condensing, using the partial reduction strategy for DAEs. The costs are stated in terms of IVP solutions on one interval [IVP], nonlinear scalar function evaluations [F] or general multiply-add operations [O], e.g., resulting from matrix-matrix or matrix-vector products. Only the leading complexity terms are given.

Computational task	Complexity lifting
condensed Hessian +	$\mathcal{O}(n_{\text{ms}}(\frac{n_{\text{ms}}+1}{2} n_u + n_p))$ [IVP]
condensed constr.Jac.	$\mathcal{O}(n_{\text{ms}}(\frac{n_{\text{ms}}+1}{2} n_u + n_p))$ [F]
QP solution	$\mathcal{O}((n_p + n_{\text{ms}} n_u)^3)$[O]
Step expansion	$\mathcal{O}(n_{\text{ms}})$[IVP]
Other calculations	$\mathcal{O}(n_{\text{ms}}(n_{\text{constr}} + n_x + n_p + n_u))$[O]

Table 7.2: Estimated costs of the tasks in one SQP step of methods based on lifting, using the partial reduction strategy for DAEs. The costs are stated in terms of IVP solutions on one interval [IVP], nonlinear scalar function evaluations [F] or general multiply-add operations [O], e.g., resulting from matrix-matrix or matrix-vector products. Only the leading complexity terms are given.

These comparisons demonstrate clearly that for large scale problems with few degrees of freedom the L-PRSQP approach based on lifting outperforms the classical approach by far. For these problems, the classical approach will also often be computational infeasible. Consider for example a problem with $n_x = 10000$ differential states and $n_u = 15$ controls, that is discretized with piecewise constant controls on a grid with $n_{\text{ms}} = 20$ gridpoints. The resulting storage capacity needed in the classical approach, only for the Hessian blocks, is then $20 \cdot 10015 \cdot 10015 \cdot 8$ bytes or, approximately, 14.95 GB, which is nowadays near the limits of workstation memory (especially, if we consider that also some memory is needed for IVP solution and sensitivity generation).

Quantity	Memory classical condensing	Memory lifting
Hessian blocks	$\mathcal{O}(n_{\mathrm{ms}}(n_x+n_u+n_p)^2)$	—
condensing matrices	$\mathcal{O}(n_{\mathrm{ms}}n_x(n_x+n_u+n_p))$	—
constraint Jacobians	$\mathcal{O}((n_{\mathrm{constr}}+n_x)n_{\mathrm{ms}}(n_x+n_u+n_p))$	—
Jacobians $\mathbf{G_{z,i}}$	$\mathcal{O}(n_{\mathrm{ms}}n_z^2)$ (sparse, worst case)	
condensed Hessian	$\mathcal{O}((n_{\mathrm{ms}}n_u+n_p)^2)$	
condensed constr. Jacobian	$\mathcal{O}(n_{\mathrm{constr}}n_{\mathrm{ms}}(n_u+n_p))$	
other QP data and variables, residuals,etc.	$\mathcal{O}(n_{\mathrm{ms}}(n_{\mathrm{constr}}+n_x+n_z+n_u+n_p))$	

Table 7.3: Estimated memory demand of exact-Hessian SQP methods based on classical condensing and on lifting, respectively, using the the partial reduction strategy for DAEs. Only the leading complexity terms are given.

Furthermore, in each SQP step the equivalent of more than $(21{\cdot}20{\cdot}0.5{\cdot}15{+}20{\cdot}10000){\cdot}10 = 2031500$ IVP solutions (provided the second order forward/adjoint IND-TC/TC approaches presented in this thesis is used for the Hessian computation, more than 2 billions if the Hessian is computed by finite differences or pure forward IND as, e.g., in `MUSCOD-II`) has to be performed, which is usually not possible within a reasonable timeframe without using a large parallel computer. The L-PRSQP approach needs in this case a storage capacity in the order of several megabytes and the equivalent of about $21{\cdot}20{\cdot}0.5{\cdot}15{\cdot}10 = 31500$ IVP solutions and will hence still be computationally feasible on an ordinary desktop PC.

8 Numerical examples for lifting-based optimization

In this chapter we illustrate with the help of several numerical examples the properties and advantages of the lifted optimization methods derived in Chapter 4 in more detail.
The first two sections show applications of the lifted Gauss-Newton and the lifted SQP algorithm, respectively, to a toy example from optimal control. In the third section we illustrate how the lifting idea can be used to adapt a given numerical simulation code for the efficient solution of a large scale parameter estimation problem with `LiftOpt`.

8.1 A Gauss-Newton toy example

For this first toy example we consider the tutorial optimal control problem for a one-dimensional dynamical system from [DBDW06]

$$\min_{x(\cdot),u(\cdot)} \quad \int_0^3 |x(t)|^2 + |u(t)|^2 dt \tag{8.1a}$$

s.t.

$$\dot{x}(t) = x(t)\,(x(t)+1) + u(t) \tag{8.1b}$$
$$x(0) = x_0 \tag{8.1c}$$
$$x(3) = 0 \tag{8.1d}$$
$$|x(t)| \leq 1 \tag{8.1e}$$
$$|u(t)| \leq 1. \tag{8.1f}$$

The objective is to minimize the absolute value of the state over the whole time horizon while penalizing the control. Note that the system cannot be controlled and "blows up" if the state goes beyond $x_b \approx 0.619$.
To solve this infinite-dimensional problem, we employ a direct multiple shooting discretization and divide, as explained in Section 1.2.3, the time horizon into a grid with 30 equal subintervals of length 0.1 and discretize the controls to be piecewise constant on each of these intervals.
The integral objective is approximated as a sum, where the function evaluations are made at the grid points. The state constraints are enforced also only at the grid points. To solve the system dynamics we use Euler's method with time steps equal to the subintervals. Doing this, we obtain a finite-dimensional NLP which we solve using the lifted Gauss-Newton algorithm, proposed in Section 4.2.1 on page 86 as well as with the non-lifted version, both implemented in `LiftOpt`. We employ here the full-step methods without globalization strategies. The lifting is

done by introducing the 30 state values at the grid points as intermediate values. The arising QP subproblems are solved using `qpOASES` [Fer07].

In this example, we employ two different initialization strategies when using the lifted approach. To begin with, we apply the lifted Gauss-Newton approach and use a function evaluation to initialize the intermediate values. Additionally, we use the fact that we want to minimize the absolute values of the states and set all intermediate values to zero. The convergence criterion is based on the sum of the Euclidean norm of the step in the controls, the Euclidean norm of the constraint violations and, in the lifted case, the Euclidean norm of the residual vector. The tolerance was chosen to be 10^{-6}.

In Table 8.1 we show a comparison of the results for different initial values x_0 of the dynamical system. The controls are initialized to zero on all subintervals in every case which means that with growing initial state the problems become more difficult to solve. We observe that although the lifted approach performs in most cases already slightly better than the non-lifted one, we can still improve the performance considerably by using a priori information in node initialization, an advantage well known from the context of direct multiple shooting [Boc87]. Furthermore, we observe that a reasonable initialization of the nodes makes the optimization more robust against bad initial guesses of the controls. This allows a quick solution, even when in the non-lifted or automatically initialized lifted algorithm the initial guess for the controls would lead to a blow up of the system.

x_0	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (zero init.)
0.02	5	5	4
0.03	6	5	4
0.04	6	6	4
0.05	7	6	4
0.06	8	7	5
0.07	9	7	5
0.08	10	8	5
0.09	13	10	5
0.10	17	13	5
0.20	err_{nan}	err_{nan}	6
0.30	err_{nan}	err_{nan}	7

Table 8.1: Results of the Gauss-Newton approaches for the optimal control example described in Section 8.1. Shown are the number of iterations needed until convergence for different initial states x_0 of the dynamical system. err_{nan} denotes that the run was not successful, because the system "blew up" during integration at some iterate, such that the QP solver quits due to "nan"-values. Compared are here the non-lifted Gauss-Newton approach, the lifted Gauss-Newton approach with automatic initialization of the intermediate values by system integration and the lifted Gauss-Newton approach when started with nodes initialized to zero.

8.2 An SQP toy example

To test the lifted SQP algorithm, proposed in Section 4.2.3 on page 89, we consider again the optimal control problem (8.1). We use the same setup as in the previous Gauss-Newton case with small modifications. The gradient of the Lagrangian is evaluated using the adjoint mode of automatic differentiation. To lift the system we introduce along with the system states at the gridpoints the corresponding adjoint values which leads to 60 intermediate values. Again we apply and compare the proposed (full-step) exact-Hessian SQP algorithm in three variants: (i) the non-lifted version, (ii) the lifted version using automatic node initialization and (iii) the lifted version using a zero initialization. The results are shown in Table 8.2. First, we observe that, compared to the Gauss-Newton methods, the SQP versions sometimes lag slightly behind, especially the unlifted version when we start at some distance from the solution. In this case, the Gauss-Newton approximation leads to faster convergence than the exact Hessian with its bad initial multiplier guesses. As we use here an exact-Hessian and undamped method, in one case we cannot avoid the bad luck to run into an area where the Hessian is not positive definite, leading the QP solver `qpOASES` [Fer07] to quit the iterations. When started closer to the solution by zero initialization, we see that the SQP method converges faster than Gauss-Newton due to the better local convergence properties. Besides that, comparing the lifted and non-lifted versions of the SQP we again see a better performance of the lifted versions and again the zero initialization leads to much faster convergence and to a more robust behavior.

x_0	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (zero init.)
0.02	6	5	3
0.03	7	5	3
0.04	8	6	3
0.05	9	6	3
0.06	10	6	3
0.07	12	7	4
0.08	15	8	4
0.09	19	11	4
0.10	25	err_{pd}	4
0.20	err_{nan}	err_{nan}	4
0.30	err_{nan}	err_{nan}	4

Table 8.2: Results of the SQP approaches for the optimal control example described in Section 8.2. Shown are the number of iterations needed until convergence for tolerance $\text{tol} = 10^{-6}$ and different initial states x_0 of the dynamical system. err_{nan} denotes that the run was not successful, because the system "blew up" during integration at some iterate, such that the QP solver quits due to "nan"-values. err_{pd} denotes that the run was not successful, because at some point the Hessian became indefinite, leading to an exit of the convex QP solver. Compared are here the non-lifted SQP approach, the lifted SQP approach with automatic initialization of the intermediate values by system integration and the lifted SQP approach when started with nodes initialized to zero.

8.3 Lifting a simulation code for the shallow water equation

In this section we illustrate the possibilities of the lifting idea for the extension of user given simulation/evaluation code. As application example we choose a parameter estimation example involving a shallow water equation model describing wave propagation in a basin that we found on the internet [Cop] in search for some truly large scale "user function" for use within `LiftOpt`. The model is described by a system of hyperbolic Partial Differential Equations (PDEs) and the corresponding equations are given by

$$\begin{aligned} \partial_t u(t,x,y) &= -g\partial_x h(t,x,y) - bu(t,x,y) \\ \partial_t v(t,x,y) &= -g\partial_y h(t,x,y) - bv(t,x,y) \\ \partial_t h(t,x,y) &= -H\left[\partial_x u(t,x,y) + \partial_y v(t,x,y)\right], \end{aligned}$$

where u, v are the horizontal and vertical water velocities, h is the deviation of the water surface from the mean water height H, $g \approx 9.81$ is the gravitational constant and b the viscous drag. As "true" values for b and H we use $b = 2$ and $H = 0.01$. For the numerical test we assume a quadratic basin corresponding to $\Omega = (0, 0.2) \times (0, 0.2)$ and consider the time horizon $t \in [0, 1]$. Furthermore, we assume that the basin is bounded by walls that reflect the incoming waves. We use an equidistant discretization in space of 30-by-30 gridpoints, finite differences in space, and for time-stepping we use the stepsize $dt = 10^{-4}$ with an explicit Euler scheme, resulting in $3 \cdot 30 \cdot 30 \cdot 10^4 = 27 \cdot 10^6$ internal variables. In our scenario we start with a plain surface and add at start time a splash of height 0.01 and radius 0.03. The numerical solution is depicted for component h in Figure 8.1 on the next page. During system simulation we take measurements only of component h every 100th time step.

The least-squares objective function we use in the parameter estimation to determine b and H is the quadratic deviation of h from the measured data in Euclidean norm, summed up over all 90000 measurements. As constraints we impose that neither b nor H should become negative. The convergence criterion is based on the sum of the Euclidean norm of the step in the controls, the Euclidean norm of the constraint violations and, in the lifted case, the Euclidean norm of the residual vector. The tolerance was chosen to be 10^{-6}. We apply the lifted and non-lifted Gauss-Newton approach to solve the problem. The lifting is done by introducing the values of h at the measurement times as node values, leading to overall 90000 node values.

The test was performed on a Linux machine with a 3.0 GHz Pentium D CPU, 3 GB RAM and GCC compiler version 4.3.2. The needed derivatives are computed by automatic differentiation using the tool ADOL-C [GJU96] in version 2.0, that has been coupled to `LiftOpt`. Linear algebra operations are performed using the ATLAS [WPD01] library and the QP subproblems are solved using `qpOASES` [Fer07].

In the lifted case, we test using automatic node initialization as well as using the measurement data for node initialization. The results for different initial guesses of b and H are displayed in Table 8.3 on page 224. The average time needed for one iteration in the unlifted case is 8.86s, while one lifted iteration takes on average 11.81s. Note that this difference in the effort for one lifted versus one unlifted iteration will usually be smaller for problems with a larger number of

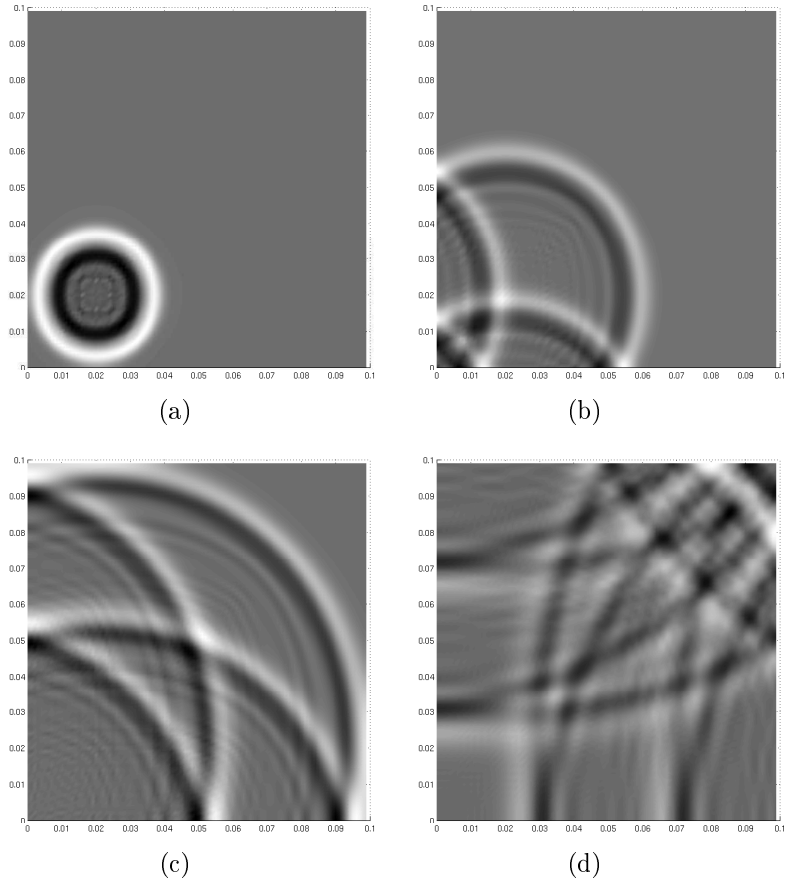

Figure 8.1: Numerical solution for component h of the shallow water equation model described in Section 8.3 for the true values $b = 2$ and $H = 0.01$, depicted at timepoints (a): $t = 0.1$, (b): $t = 0.25$, (c): $t = 0.5$ and (d): $t = 1.0$.

degrees of freedom, due to a comparatively smaller overhead.

We observe that lifting does not improve the performance much if we start close to the true parameter values. On the other hand, if we start at more distance from the solution, the lifted approach again leads to a significantly faster convergence, especially for perturbations in the parameter H. When the lifted approach is initialized with the measurement data, the performance is even better, although only a part of the system state, i.e., h, is measured here.

b	H	# iterations unlifted	#iterations lifted (autom. init.)	#iterations lifted (meas init.)
0.5	0.01	5	5	4
5	0.01	6	5	4
15	0.01	17	7	6
30	0.01	27	7	6
2	0.005	31	9	5
2	0.02	38	12	5
2	0.1	44	13	8
0.2	0.001	33	12	7
1	0.005	47	10	5
4	0.02	56	10	5
1	0.02	44	9	6
20	0.001	24	10	6

Table 8.3: Results of the parameter estimation example for the shallow water equation model described in Section 8.3. Shown are the number of iterations needed until convergence for tolerance tol $= 10^{-6}$ and different sets of initial parameter guesses. Compared are here the non-lifted Gauss-Newton approach, the lifted Gauss-Newton approach with automatic initialization of the intermediate values by system simulation and the lifted approach when using the measurement data for node initialization. The "true" parameter values are $b = 2$ and $H = 0.01$. The average time needed for one unlifted iteration is 8.86s versus 11.81s for one lifted iteration.

9 Numerical examples for sensitivity related strategies

In this chapter we demonstrate the performance and efficiency of the sensitivity related strategies presented in Chapter 6 on several test setups by the use of our integrator code `DAESOL-II` together with our integrator package `SolvIND`.
We begin in the first section with an analysis of the different IND-based strategies on a scalable test example. In the second section we compare on several examples from an IVP testset our new adjoint IND schemes for sensitivity generation with the alternative, commonly used approach of solving the adjoint variational equation. The third section shows the practical applicability of our error control strategy for forward sensitivities. In the fourth section we demonstrate the effectivity of our global error estimation strategy by a comparison with an alternative approach from literature on a series of test examples.

9.1 Comparison of the different IND-based strategies for sensitivity generation

We compare in this section how the different IND-based sensitivity generation strategies presented in Chapter 6 perform numerically on a scalable ODE test problem from chemical engineering, which is explained shortly in the following.

9.1.1 The SMB model

The example we consider here is the MODICON variant of the Simulated Moving Bed (SMB) chromatography process with two species and six columns. It is described by a general rate Partial Differential Equation (PDE) model which is explained in more detail in [TED+07]. For both species $i = 1, 2$ the general rate model considers three phases: the instationary phase c_i, the liquid stationary phase $c_{p,i}$ and the adsorbed stationary phase $q_{p,i}$. The general rate model consists of

$$\partial_t c_i = \mathrm{Pe}_i^{-1} \partial_z^2 c_i - \partial_z c_i - \mathrm{St}_i \left(c_i - c_{\mathrm{p},i}|_{r=1} \right), \qquad (t,z) \in (0,T) \times (0,1), \qquad (9.1)$$

$$\partial_t \left((1-\epsilon_\mathrm{p}) q_{\mathrm{p},i} + \epsilon_\mathrm{p} c_{\mathrm{p},i} \right) = \eta_i \left(r^{-2} \partial_r \left(r^2 \partial_r c_{\mathrm{p},i} \right) \right), \qquad (t,r) \in (0,T) \times (0,1), \qquad (9.2)$$

and the boundary conditions

$$\partial_z c_i(t,0) = \mathrm{Pe}_i \left(c_i(t,0) - c_\mathrm{in}(t) \right), \qquad \partial_z c_i(t,1) = 0, \qquad (9.3)$$

$$\partial_r c_{\mathrm{p},i}(t,0) = 0, \qquad \partial_r c_{\mathrm{p},i}(t,1) = \mathrm{Bi}_i \left(c_i(t,z) - c_{\mathrm{p},i}(t,1) \right), \qquad (9.4)$$

with positive constants ϵ_p (porosity), η_i (nondimensional diffusion coefficient), Pe_i (Péclet number), St_i (Stanton number), and Bi_i (Biot number). The two stationary phases are coupled by an algebraic condition, the nonlinear Bi-Langmuir isotherm equation

$$q_{\mathrm{p},i} = \frac{H_i^1 c_{\mathrm{p},i}}{1 + k_1^1 c_{\mathrm{p},1} + k_2^1 c_{\mathrm{p},2}} + \frac{H_i^2 c_{\mathrm{p},i}}{1 + k_1^2 c_{\mathrm{p},1} + k_2^2 c_{\mathrm{p},2}}, \tag{9.5}$$

with non-negative constants H_i^j (Henry coefficients) and k_i^j (isotherm parameter). The PDE has essentially only one spatial dimension, as the dynamics inside the particles can be eliminated [Gu95].

For the discretization in space a higher-order Nodal Discontinuous Galerkin method is used, which is described by Hesthaven and Warburton [HW08]. This approach leads to a large, structured ODE system in time. For more details on the discretization we refer to Potschka et al. [PBE+08].

9.1.2 The test setup

For the numerical tests we use a series of different discretization orders and different numbers of discretization points. The resulting number of variables in the IVP is $n = 24m(l+1)+13$, where l is the discretization order and m the number of discretization elements per column. The total size splits up into $n_x = 24m(l+1)+7$ differential states, $n_{p_p} = 1$ parameter, $n_{p_q} = 4$ control parameter and $n_{p_h} = 1$ parameter for the length of the time horizon. The iteration matrix in the BDF method is block sparse. Enlarging the number of elements increases the number of blocks, whereas increasing the order leads to a larger blocksize. This is sketched in Figure 9.1.
An analysis of the eigenvalues of the Jacobian of the right-hand side function of the ODE model of the SMB process shows that with increasing problem size, i.e., a finer discretization in space, more and more eigenvalues move closer to the imaginary axis. This situation is demonstrated in Figure 9.2. Hence, stability problems for higher-order BDF methods are to be expected and have also been confirmed numerically in tests for the maximum orders $k_{\max} = 4, 5$ and 6. Therefore, we restrict the maximum BDF order to $k_{\max} = 3$ for our tests.

The computations are performed using `DAESOL-II/SolvIND` on a desktop computer with an Intel Pentium D CPU with 3.0 GHz and 3.8 GB of RAM, running under the Ubuntu 8.04 64-bit operating system. The code was compiled with GCC version 4.3.2. The generation of the model derivatives is done using the tool ADOL-C [GJU96] in version 2.1.0. UMFPACK [Dav04] in version 5.0.2 is used for sparse matrix operations. We run a series of tests for orders $r = 2, \ldots, 12$ and with $m = 2, \ldots, 24$ elements for each order. The number of differential states lies between $n_x = 151$ and $n_x = 7495$ and the maximum number of nonzero elements in the iteration matrix is $n_{\mathrm{nz}} = 160249$. As tolerance for the integration we choose $\texttt{tol} = \texttt{atol} = 10^{-6}$.

For each combination of discretization order and elements, we perform a nominal integration on the time horizon $[0, 10]$ as base reference for comparison and one nominal integration with storage of the scheme including trajectory values and matrices. Based on the stored discretization scheme,

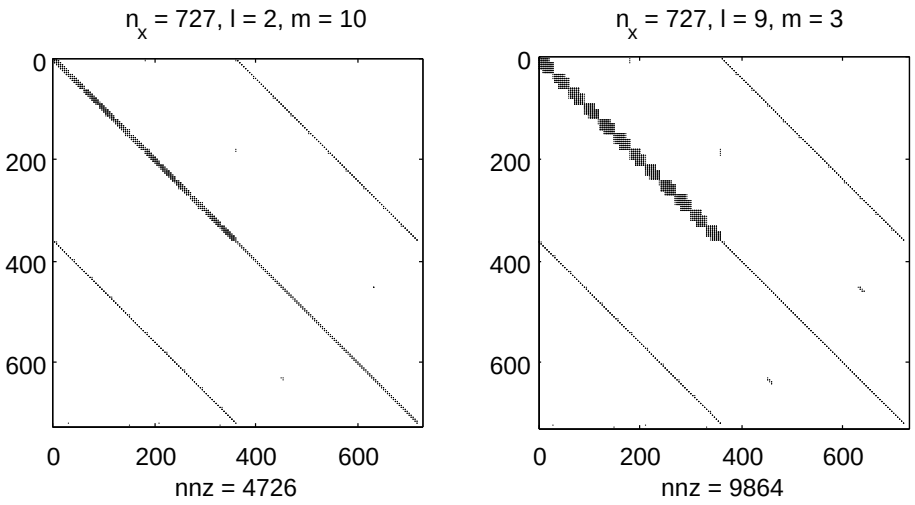

Figure 9.1: Sparsity structure of the iteration matrix in the BDF method for the SMB example for the case of $l = 2, m = 10$ and the case $l = 9, m = 3$, both leading to a problem of size $n_x = 727$. We see that a larger discretization order results in larger blocks of nonzero entries and in a larger number of nonzero elements in the iteration matrix.

we perform a deferred first order iterative forward IND sweep and a first order iterative adjoint IND sweep, each for one sensitivity direction. The direct schemes are not considered here due to the high effort that would be needed in this example to build and factorize the iteration matrices in every IND step. Then we perform a second order forward IND-TC sweep and a second order forward/adjoint IND-TC sweep, also for one directional sensitivity in each case. Finally, we perform an integration replay based on the stored discretization scheme with disturbed initial values and parameter to generate a forward sensitivity by IND using finite differences. For each of these actions, a number of timings and statistics is taken. The results are depicted in Figure 9.3 on page 230 and Figure 9.4 on page 231.

For the comparison, we choose the number of the nonzero elements of the iteration matrix n_{nz} as problem size. This quantity correlates directly to the costs for the matrix operations and the derivative evaluation, which together normally dominate the overall effort.

In Figure 9.3 we see that the numbers of integration steps, of Newton-like iterations and of needed matrix decompositions grow at first with the problem size, but then reach an upper limit. Due to the monitor strategy only about every 20 integration steps a new iteration matrix is used and overall only 2 Jacobian evaluations (for matrix rebuilds) are needed. The combination of the monitor strategy and the exploitation of the good sparsity structure of the problem leads in the end to a linear relation between the total integration time and the problem size. In the nominal integration, half of the time is spent for model function and model derivative evaluation

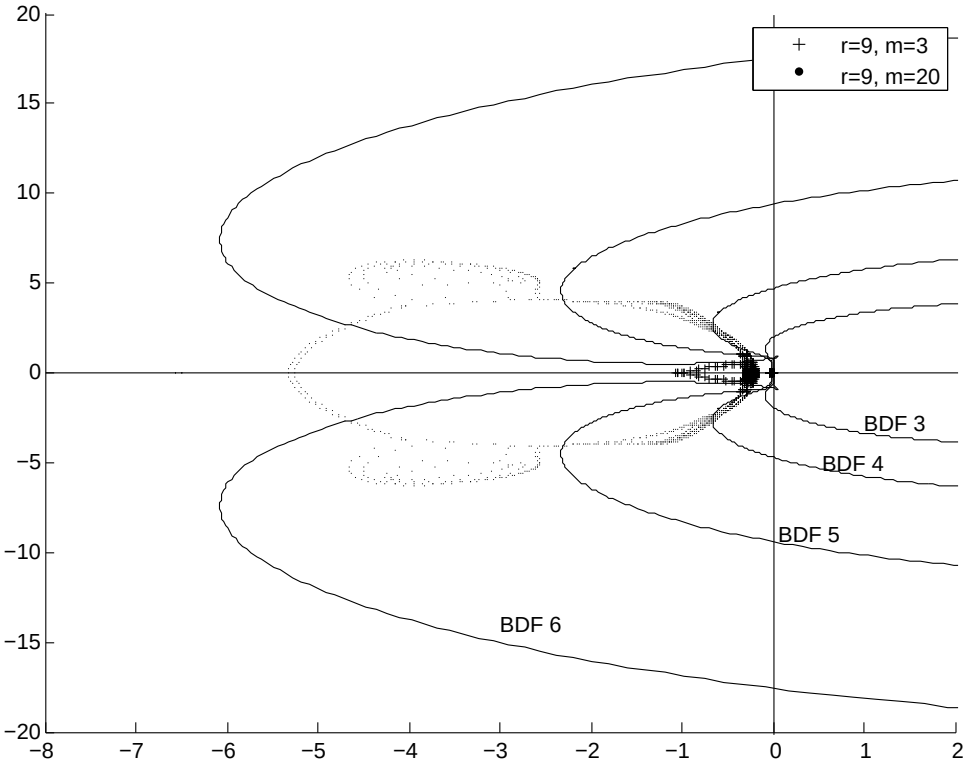

Figure 9.2: The stability regions of the (equidistant) BDF methods for orders 3 to 6 and a subset of the eigenvalues of the right-hand side Jacobian of the SMB process for the cases of order $r = 9$ and $m = 3$ and $m = 20$, respectively, scaled by the factor 0.001. The stability regions are the areas outside the solid boundary lines. We observe that for larger problem sizes the (already scaled) eigenvalues do not lie inside the stability regions for BDF orders larger than 3. This leads to stability problems with higher-order BDF methods, enforcing very small stepsizes regardless of the local discretization error. Therefore, a maximum BDF order of $k_{\max} = 3$ is advisable for the numerical solution of the problem.

and about one third for matrix factorizations and the solution of the linear systems. The average ratios between the time needed for the nominal integration and the time needed for different other operation are given in Table 9.1.

We observe that an integration replay costs only about a half of a nominal integration, as no additional matrix decomposition and derivative evaluations are needed in this case. The ratios of the iterative first and second order forward IND(-TC) sweeps are significantly below the bounds that given in AD theory for ordinary functions, as also here no additional matrix factorization and Jacobian evaluations are needed. Also the first order adjoint IND sweep and the second order forward/adjoint IND-TC sweep are a very efficient mean for sensitivity generation and their ratios are still below the theoretical bounds for ordinary functions.

However, as we can see in Figure 9.4 on page 231, especially for the adjoint sweeps there is still room for larger performance improvements. The key to obtain an even better performance is in this case not an improvement of the IND schemes themselves, but in speeding-up the generation of the model function derivatives, as they cause in average 87 percent of the effort in the first order

Average ratio between nominal integration and ...	
integration replay	0.53
forward IND sweep	1.38
adjoint IND sweep	3.73
2nd order fwd IND-TC sweep	2.19
2nd order fwd/adj IND-TC sweep	10.67

Table 9.1: The time needed for different integrator operations compared to the time for the nominal integration of the SMB example.

adjoint sweep and about 94 percent in the second order forward/adjoint IND-TC sweep. If we keep in mind that the number of model function evaluations in the nominal integration is equal to the number of model function derivatives needed for a first order iterative adjoint IND sweep, we see that the effort for the subtask of model function derivative evaluation does not respect the theoretical bounds. In theory, the derivative evaluation in the first order adjoint IND sweep should take between 3 and 4 times the time of the function evaluation in the nominal integration. Instead, the average ratio for the SMB example is 12.40. Hence, bringing the effort of the model derivative evaluation nearer to its theoretical bounds will improve the performance of our adjoint IND(-TC) schemes tremendously. Of course, to a smaller extent this would also improve the performance of the forward IND(-TC). Therefore, an investigation of other means of AD-based model derivative generation besides operator overloading, such as a source code transformation that respects the model-inherent structure including matrix-vector operations, would be very interesting for large scale problems.

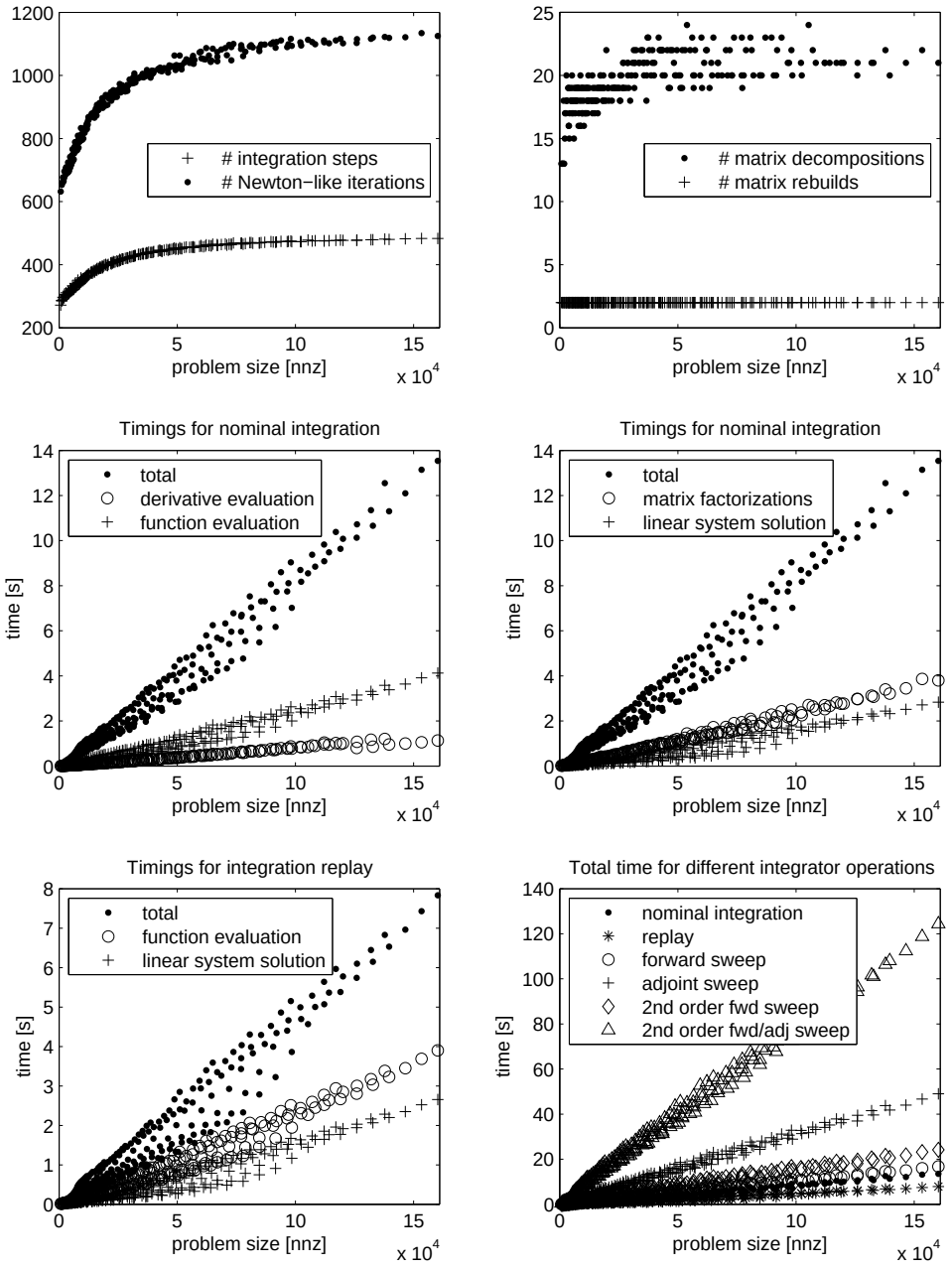

Figure 9.3: This figure shows several statistics for the SMB example in relation to the problem size (nonzero elements of the iteration matrix). The upper left figure shows the number of integration steps and the overall number of Newton-like iterations. Both numbers grow at first with the problem size but then remain below an upper bound. The same holds for the number of required decompositions of the iteration matrix. The number of needed iteration matrix rebuilds is independent of the problem size (upper right). The middle row shows that the total time for the nominal integration increases linearly with the problem size, where the evaluation of model functions and derivatives takes about half of the time (middle left) and matrix factorization and the solution of linear systems about one third (middle right). The lower left figure shows the timings for an integration replay, where the model function evaluations take about half the time. The lower right figure shows a comparison of the overall times needed for the different integrator operations.

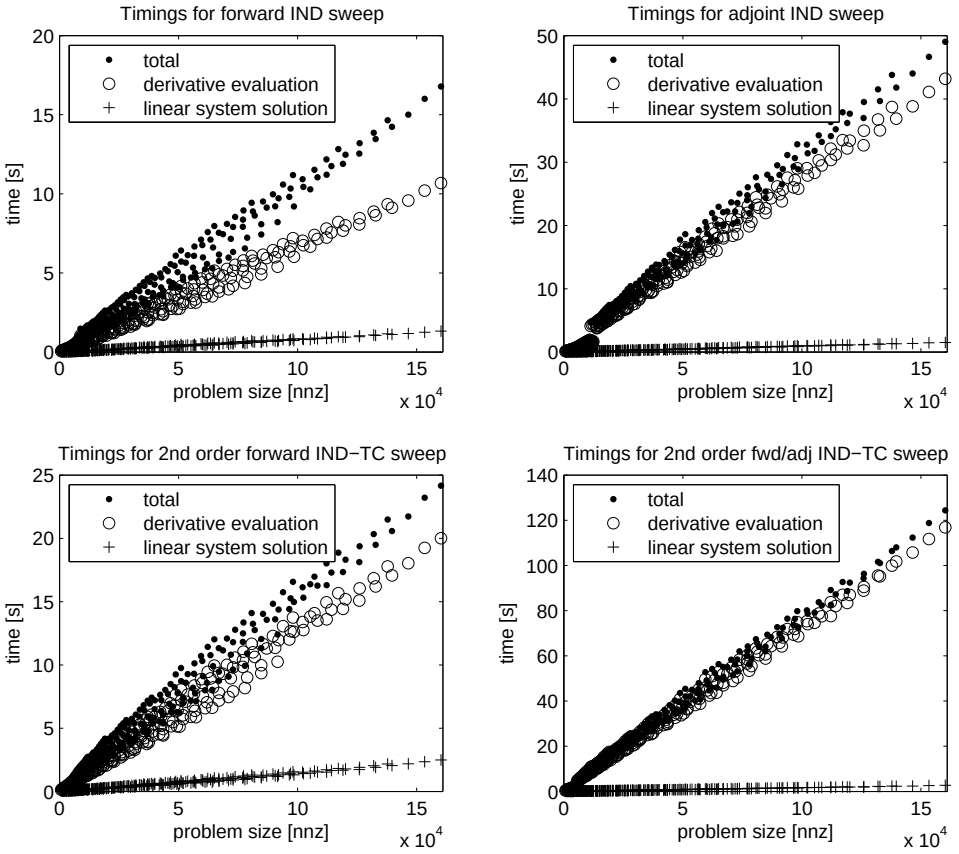

Figure 9.4: This figure shows the timings for different kinds of IND sweeps for the SMB example: a first order iterative forward IND sweep (upper left), a first order iterative adjoint IND sweep (upper right), a second order forward IND-TC sweep (lower left) and a second order forward/adjoint IND-TC sweep (lower right). Displayed are in each case the overall time needed, as well as the time needed for evaluation of the model function derivatives and for the solution of the linear systems. We observe that the derivative evaluation is in each case the most time consuming subtask, taking in average about 66 and 82 percent of the time in the forward IND(-TC) sweeps, respectively, and about 87 and 94 percent in the adjoint IND(-TC) schemes, respectively.

9.2 Adjoint IND versus solution of adjoint variational equation

In this section we compare the IND-based (first order) iterative adjoint scheme developed in this thesis against the commonly used approach of solving the adjoint variational equation numerically. More specifically, we compare the performance of the implementation of our strategies in `DAESOL-II` with the performance of the widely used integrator codes CVODES and IDAS from the SUNDIALS [HBG+05] integrator suite. These are the quasi standard codes for a not IND-based generation of adjoint sensitivity information of ODE/DAE-IVPs. A comparison of the strategies related to nominal IVP solution and (first order) forward sensitivity generation implemented in DAESOL-II with other integrator codes can be found in [Bau99]. As test problems we choose three examples from the IVP testset of the university of Bari [MI08]: two nonlinear ODE-IVPs and an index 1 DAE-IVP.
The computations are performed using `DAESOL-II` and CVODES/IDAS from the SUNDIALS suite in version 2.6.0. The platform is a desktop computer with an Intel Pentium D CPU with 3.0 GHz and 3.8 GB of RAM, running under the Ubuntu 8.04 64-bit operating system. The code was compiled with GCC version 4.3.2. For all codes the required model functions and model derivatives are evaluated via the `SolvIND` evaluator layer to obtain comparable results. For the generation of the model function derivatives `SolvIND` uses the tool ADOL-C in version 2.1.0.

In the following we explain each of the problems and present the setup as well as the results of the corresponding numerical tests.

9.2.1 The HIRES problem

The HIRES problem is a stiff IVP consisting of 8 nonlinear ODEs. It was first presented by Schäfer [Sch75] and describes the so-called High Irradiance Responses of photomorphogenesis on the basis of phytochrome by a chemical reaction system of 8 species. The IVP is defined by

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} -k_1 x_1(t) + k_2 x_2(t) + k_6 x_3(t) + o_{k_s} \\ k_1 x_1(t) - (k_2 + k_3) x_2(t) \\ -(k_1 + k_6) x_3(t) + k_2 x_4(t) + k_5 x_5(t) \\ k_3 x_2(t) + k_1 x_3(t) - (k_2 + k_4) x_4(t) \\ -(k_1 + k_5) x_5(t) + k_2 (x_6(t) + x_7(t)) \\ -k_+ x_6(t) x_8(t) + k_4 x_4(t) + k_1 x_5(t) - k_2 x_6(t) + k^* x_7(t) \\ k_+ x_6(t) x_8(t) - (k_2 + k_- + k^*) x_7(t) \\ -k_+ x_6(t) x_8(t) + (k_2 + k_- + k^*) x_7(t) \end{pmatrix} \tag{9.6}$$

$$t \in [0, 321.8122], \quad \mathbf{x}(0) = (1, 0, 0, 0, 0, 0, 0, 5.7 \cdot 10^{-3})^T,$$

where the parameter, taken from [HW96], are given by

$k_1 = 1.71$	$k_3 = 8.32$	$k_5 = 0.035$	$k_+ = 280$	$k^* = 0.69$
$k_2 = 0.43$	$k_4 = 0.69$	$k_6 = 8.32$	$k_- = 0.69$	$o_{k_s} = 7 \cdot 10^{-4}$.

The numerical reference solution of the IVP is taken from [MI08] and is given by

$$\mathbf{x}^*(321.8122) = \begin{pmatrix} 0.7371312573325668 \cdot 10^{-3} \\ 0.1442485726316185 \cdot 10^{-3} \\ 0.5888729740967575 \cdot 10^{-4} \\ 0.1175651343283149 \cdot 10^{-2} \\ 0.2386356198831331 \cdot 10^{-2} \\ 0.6238968252742796 \cdot 10^{-2} \\ 0.2849998395185769 \cdot 10^{-2} \\ 0.2850001604814231 \cdot 10^{-2} \end{pmatrix}. \tag{9.7}$$

The numerical reference values for the Wronskian $\mathcal{W}_{\text{ref}}$ are generated by CVODES with all tolerances set to 10^{-15}.

We solve the problem using `DAESOL-II` and CVODES, respectively, for the series of tolerances $\texttt{tol}_i = \texttt{atol}_i = 10^{-\frac{4+i}{4}}$, $1 \leq i \leq 44$, and initial stepsize 10^{-2}. Besides the nominal solution we compute adjoint sensitivities for $n_{\text{adjDir}} = 1, 2, 3$ and the Wronskian $\mathcal{W}$ using $n_{\text{adjDir}} = n_x = 8$ adjoint directions. For both codes, checkpointing is disabled and we use dense direct linear algebra for matrix and vector operations. In CVODES the tolerance for the solution of the adjoint variational ODE was set to $\texttt{bTol} = 10 \cdot \texttt{tol}$.
From the obtained solution and the reference solution we compute the global error $\boldsymbol{\varepsilon}$ at the end of the time horizon and the number of significant digits $\text{scd} = -\log_{10}(\|\boldsymbol{\varepsilon}\|_\infty)$ of the solution. For the comparison of the sensitivities we compute the deviation from the reference values by $\Delta W = \|\mathcal{W} - \mathcal{W}_{\text{ref}}\|_\infty$.

Based on this setup, we compare the performance of both integrator codes for adjoint sensitivity generation. Figure 9.5 on page 235 shows the results of the comparison. We observe that the stepsize and error control strategy in `DAESOL-II` needs about half of the integration steps of CVODES to reach a given number of significant digits for the nominal solution. Concerning the deviation from the reference Wronskian, we see that both codes show a similar improvement in the approximation with increasing accuracy of the nominal solution. `DAESOL-II` shows here slightly higher "turbulences". However, this is to be expected, as, by construction, there exists no explicit error control for the adjoint IND scheme. Nevertheless, the Wronskian approximation is in nearly all cases as good as the one computed by CVODES.
We recall now that by construction of our adjoint IND scheme the number of required matrix decompositions and Jacobian evaluations in `DAESOL-II` remains the same for the pure nominal integration and the combination of nominal integration with the computation of an arbitrary number of directional adjoint sensitivities. We observe that for lower integration accuracies, considering only the nominal integration, `DAESOL-II` and CVODES need about the same number of decompositions and Jacobian evaluations, whereas for higher integration accuracies, the monitor strategy of `DAESOL-II` seems to work better than the corresponding mechanism in CVODES as significantly fewer decompositions/evaluations are needed. Furthermore, in CVODES the number of decompositions and evaluations grows approximately linearly with the number of directional

adjoint sensitivities that are computed. The number of directional adjoint model function derivatives grows in `DAESOL-II` strictly linearly with the number of adjoint sensitivities, in CVODES approximately linearly, while the absolute number needed in CVODES is slightly higher than in `DAESOL-II`.
In the end, all this leads to an overall computational time of CVODES for the nominal integration combined with the adjoint sensitivity generation that is significantly larger than that of `DAESOL-II`. For lower integration accuracies, `DAESOL-II` is about 10 times faster and for very high accuracies still about 1.5 to 3 times. For this example, the latter is due to the fact that for very high accuracies `DAESOL-II` needs more Newton-like iterations in the nominal integration than CVODES. Furthermore, these are relatively expensive compared to the matrix factorizations due to the small size of the example.

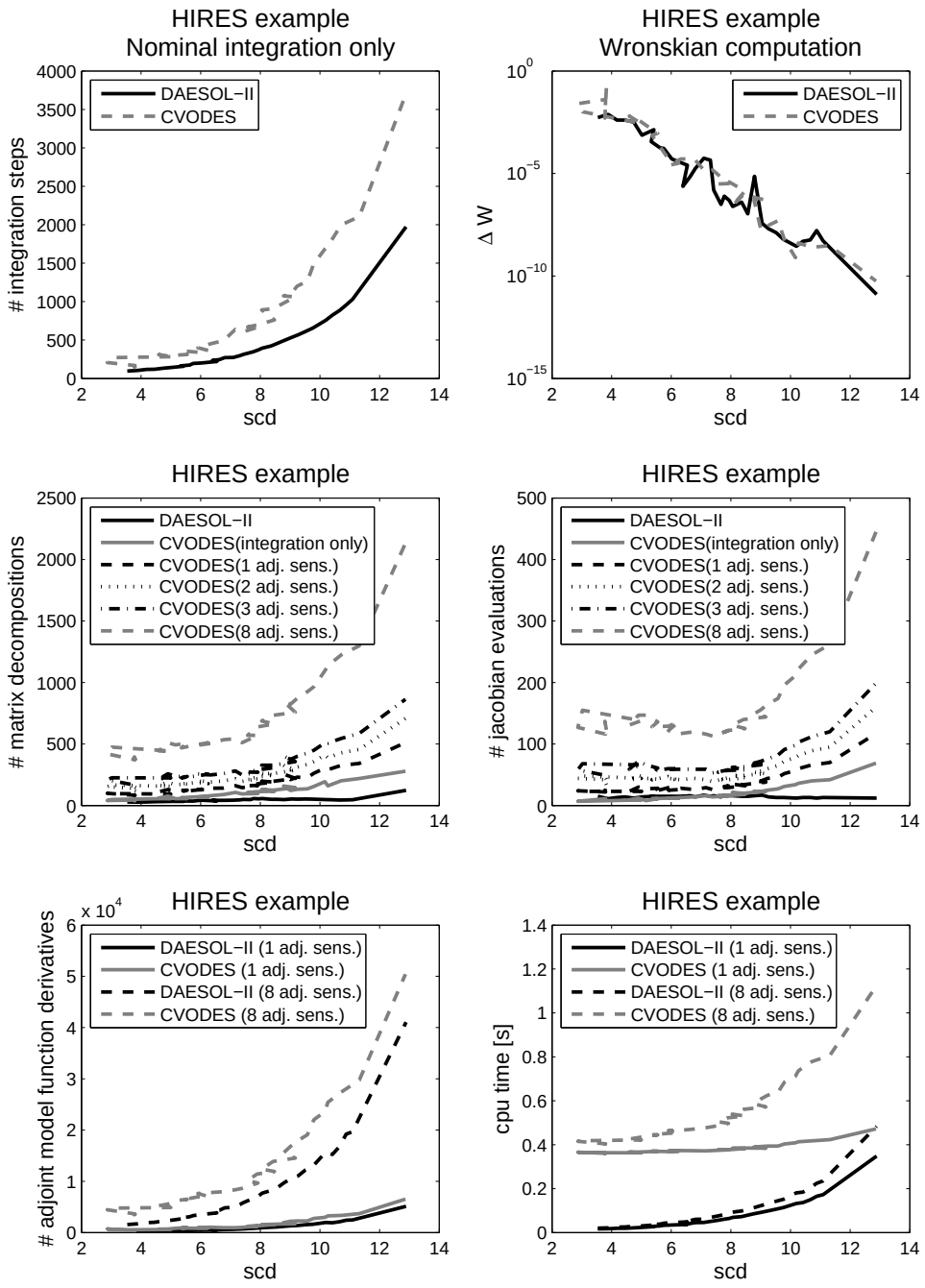

Figure 9.5: The numerical results of applying `DAESOL-II` and CVODES to the HIRES example. The upper row shows on the left the total number of integration steps needed for the nominal integration in relation to the accuracy of the solution expressed by the number of significant digits. The upper right figure shows the sup-norm of the deviation of the numerically computed Wronskian approximation from the reference Wronskian. The middle row shows the number of needed decompositions of the iteration matrix (left) and of the needed Jacobian evaluations (right). Both quantities are depicted for `DAESOL-II` (where they are independent of the number of adjoint sensitivities) and for CVODES for the cases of a pure nominal integration as well as the combination of nominal integration with the computation of 1, 2, 3 adjoint sensitivities and the Wronskian (8 adj. sens.). The lower row shows for both codes the number of needed directional adjoint derivatives of the model functions (left) as well as the overall computational time (right). Each quantity is depicted for the combination of the nominal integration with the computation of one adjoint sensitivity and the computation of the Wronskian, respectively.

9.2.2 The Pleiades problem

The Pleiades problem is a nonstiff ODE-IVP problem with 28 equations. The formulation and the setup are taken from [HNW93]. The problem originates from a celestial mechanics problem of seven stars in the plane, described by the coordinates $\mathbf{x_x} \in \mathbb{R}^7$, $\mathbf{x_y} \in \mathbb{R}^7$ and masses $m_i := i$, $1 \leq i \leq 7$. The equations of motion of the system are derived from the law of gravity, where the gravitational constant is assumed as equal to one. Combining $\mathbf{x} := (\mathbf{x_x}^T, \mathbf{x_y}^T)^T$ and formulating the problem as a first order ODE system leads to the equations

$$\mathrm{d}\begin{pmatrix}\mathbf{x}(t)\\ \dot{\mathbf{x}}(t)\end{pmatrix}/\mathrm{d}t = \begin{pmatrix}\dot{\mathbf{x}}(t)\\ \mathbf{f}(\mathbf{x})\end{pmatrix}.$$

Here the acceleration is given by $\mathbf{f}(\mathbf{x}(t)) = \begin{pmatrix}\mathbf{f^x}(\mathbf{x}(t))\\ \mathbf{f^y}(\mathbf{x}(t))\end{pmatrix}$, where the components of the functions are defined by

$$f_i^x(\mathbf{x}(t)) = \sum_{j\neq i} m_j \frac{x_{x,j} - x_{x,i}}{((x_{x,j} - x_{x,i})^2 + (x_{y,j} - x_{y,i})^2)^{\frac{3}{2}}}$$
$$f_i^y(\mathbf{x}(t)) = \sum_{j\neq i} m_j \frac{x_{y,j} - x_{y,i}}{((x_{x,j} - x_{x,i})^2 + (x_{y,j} - x_{y,i})^2)^{\frac{3}{2}}}.$$

We solve the problem on the time horizon $t \in [0, 3]$ and for the initial values

$$\mathbf{x_x}(0) = (3, 3, -1, -3, 2, -2, 2)^T, \qquad \mathbf{x_y}(0) = (3, -3, 2, 0, 0, -4, 4)^T,$$
$$\dot{\mathbf{x}}_\mathbf{x}(0) = (0, 0, 0, 0, 0, 1.75, -1.5)^T, \qquad \dot{\mathbf{x}}_\mathbf{y}(0) = (0, 0, 0, -1.25, 1, 0, 0)^T.$$

The numerical reference solution as given in [MI08] is

$$\mathbf{x}_\mathbf{x}^*(3) = \begin{pmatrix} 0.3706139143970502 \cdot 10^0 \\ 0.3237284092057233 \cdot 10^1 \\ -0.3222559032418324 \cdot 10^1 \\ 0.6597091455775310 \cdot 10^0 \\ 0.3425581707156584 \cdot 10^0 \\ 0.1562172101400631 \cdot 10^1 \\ -0.7003092922212495 \cdot 10^0 \end{pmatrix}, \qquad \mathbf{x}_\mathbf{y}^*(3) = \begin{pmatrix} -0.3943437585517392 \cdot 10^1 \\ -0.3271380973972550 \cdot 10^1 \\ 0.5225081843456543 \cdot 10^1 \\ -0.2590612434977470 \cdot 10^1 \\ 0.1198213693392275 \cdot 10^1 \\ -0.2429682344935824 \cdot 10^0 \\ 0.1091449240428980 \cdot 10^1 \end{pmatrix},$$

$$\dot{\mathbf{x}}_\mathbf{x}^*(3) = \begin{pmatrix} 0.3417003806314313 \cdot 10^1 \\ 0.1354584501625501 \cdot 10^1 \\ -0.2590065597810775 \cdot 10^1 \\ 0.2025053734714242 \cdot 10^1 \\ -0.1155815100160448 \cdot 10^1 \\ -0.8072988170223021 \cdot 10^0 \\ 0.5952396354208710 \cdot 10^0 \end{pmatrix}, \qquad \dot{\mathbf{x}}_\mathbf{y}^*(3) = \begin{pmatrix} -0.3741244961234010 \cdot 10^1 \\ 0.3773459685750630 \cdot 10^0 \\ 0.9386858869551073 \cdot 10^0 \\ 0.3667922227200571 \cdot 10^0 \\ -0.3474046353808490 \cdot 10^0 \\ 0.2344915448180937 \cdot 10^1 \\ -0.1947020434263292 \cdot 10^1 \end{pmatrix}.$$

The numerical reference values for the Wronskian $\mathcal{W}_{\text{ref}}$ are generated by CVODES with all tolerances set to 10^{-15}.

We solve the problem using DAESOL-II and CVODES, respectively, for the series of tolerances $\texttt{tol}_i = \texttt{atol}_i = 10^{-\frac{4+i}{4}}$, $1 \leq i \leq 44$, and initial stepsize 10^{-2}. Besides the nominal solution we compute adjoint sensitivities for $n_{\text{adjDir}} = 1, 2, 3$ and the Wronskian $\mathcal{W}$ using $n_{\text{adjDir}} = n_x = 28$ adjoint directions. For both codes, checkpointing is disabled and we use dense direct linear algebra for matrix and vector operations. In CVODES the tolerance for the solution of the adjoint variational ODE was set to $\texttt{bTol} = 10 \cdot \texttt{tol}$.
From the obtained solution and the reference solution we compute the global error $\boldsymbol{\varepsilon}$ at the end of the time horizon and the number of significant digits $\text{scd} = -\log_{10}(\|\boldsymbol{\varepsilon}\|_\infty)$ of the solution. For the comparison of the sensitivities we compute the deviation from the reference values by $\Delta W = \|\mathcal{W} - \mathcal{W}_{\text{ref}}\|_\infty$.

Based on this setup, we compare the performance of both integrator codes for adjoint sensitivity generation. Figure 9.6 on the next page shows the results of the comparison. We observe that the stepsize error control strategy in DAESOL-II needs in general fewer integration steps than CVODES to reach a given number of significant digits for the nominal solution. Concerning the deviation of the reference Wronskian, we see that both codes deliver a similarly accurate approximation of the reference Wronskian, while the deviation of DAESOL-II is in general slightly larger.
We recall again that by construction of the adjoint IND scheme the number of required matrix decompositions and Jacobian evaluations in DAESOL-II remains the same for the pure nominal integration and the nominal integration combined with the computation of an arbitrary number of directional adjoint sensitivities. We observe that already for the nominal integration DAESOL-II needs significantly less matrix decompositions and Jacobian evaluations than CVODES. Additionally, in CVODES the number of needed decompositions and evaluations grows approximately linearly with the number of directional adjoint sensitivities that are computed. The number of directional adjoint model function derivatives grows in DAESOL-II strictly linearly with the number of adjoint sensitivities, in CVODES approximately linearly, while the absolute number needed in CVODES is about 2 times that of DAESOL-II.
In the end, all this leads to an overall computational time of CVODES for the nominal integration combined with the adjoint sensitivity generation that is significantly larger than that of DAESOL-II. For one adjoint sensitivity direction CVODES needs about 1.25 times the time of DAESOL-II. Concerning the computation of the complete Wronskian DAESOL-II is about 5 times faster than CVODES.

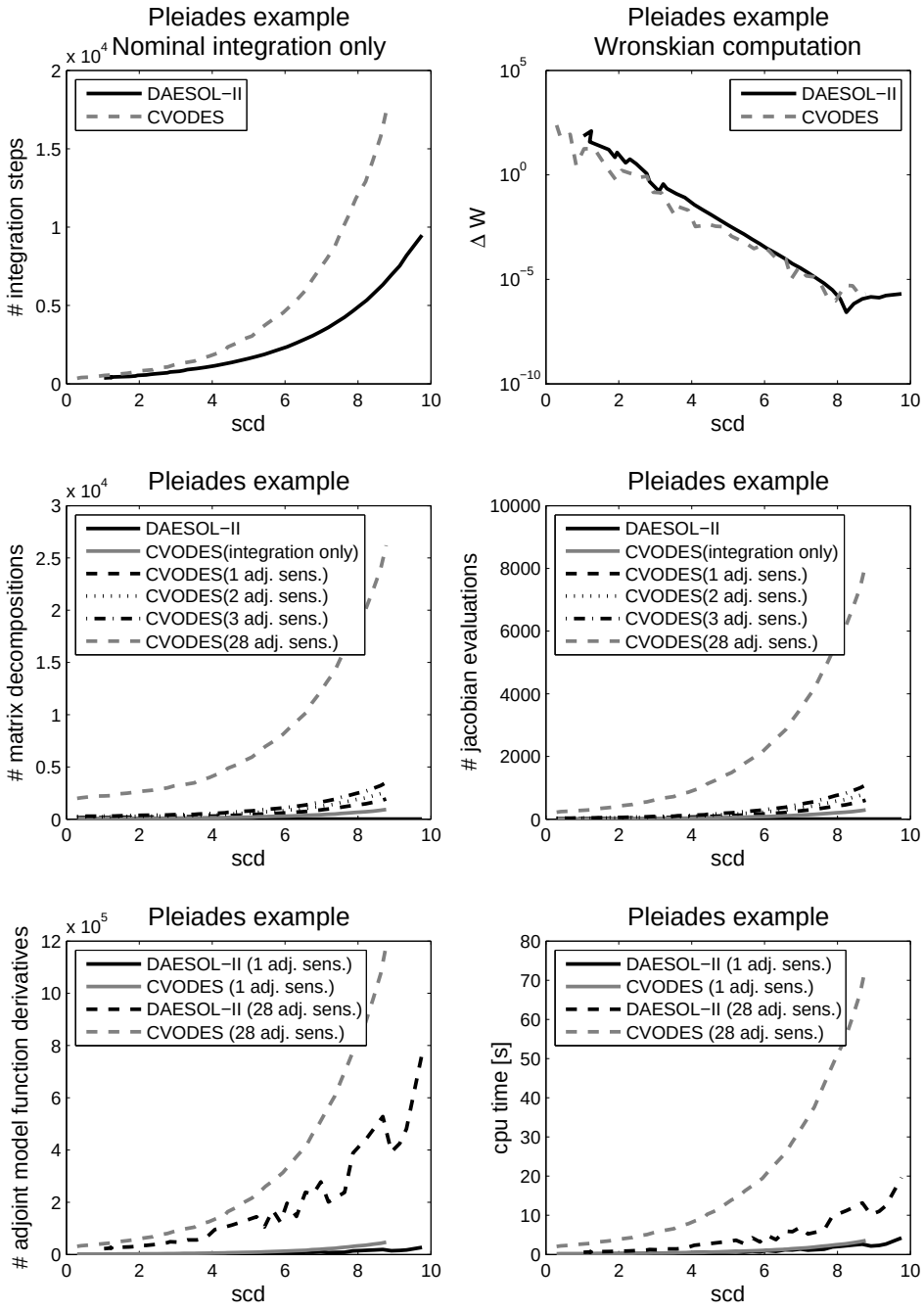

Figure 9.6: The numerical results of applying `DAESOL-II` and CVODES to the Pleiades example. The upper row shows on the left the total number of integration steps needed for the nominal integration in relation to the accuracy of the solution expressed by the number of significant digits. The upper right figure shows the sup-norm of the deviation of the numerically computed Wronskian approximation from the reference Wronskian. The middle row shows the number of needed decompositions of the iteration matrix (left) and of the needed Jacobian evaluations (right). Both quantities are depicted for `DAESOL-II` (where they are independent of the number of adjoint sensitivities) and for CVODES for the cases of a pure nominal integration as well as the combination of nominal integration with the computation of 1, 2, 3 adjoint sensitivities and the Wronskian (28 adj. sens.). The lower row shows for both codes the number of needed directional adjoint derivatives of the model functions (left) as well as the overall computational time (right). Each quantity is depicted for the combination of the nominal integration with the computation of one adjoint sensitivity and the computation of the Wronskian, respectively.

9.2.3 The chemical Akzo Nobel problem

The chemical Akzo Nobel problem is a stiff DAE-IVP of index 1 in five differential and one algebraic equation. The setup and parameter values of the problem are presented in [Sto98]. The problem originates from the Akzo Nobel laboratories and describes a chemical process in which 2 species are mixed to obtain a third one, while carbon dioxide is added continuously. Due to commercial reasons, the species themselves are not disclosed. The overall number of species involved in the reaction is 6. The DAE system is given by

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} -2r_1 + r_2 - r_3 - r_4 \\ -0.5r_1 - r_4 - 0.5r_5 + F_{\text{in}} \\ r_1 - r_2 + r_3 \\ -r_2 + r_3 - 2r_4 \\ r_2 - r_3 + r_5 \end{pmatrix}, \tag{9.8a}$$

$$0 = K_s x_1(t) x_4(t) - z_1(t). \tag{9.8b}$$

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

$$r_1 = k_1 x_1(t)^4 \sqrt{x_2(t)}, \qquad r_3 = \frac{k_2}{K} x_1(t) x_5(t), \qquad r_5 = k_4 z_1(t)^2 \sqrt{x_2(t)},$$
$$r_2 = k_2 x_3(t) x_4(t), \qquad r_4 = k_3 x_1(t) x_4(t)^2, \qquad F_{\text{in}} = A\left(\frac{p_{\text{CO}_2}}{H} - x_2(t)\right),$$

and the parameter values by

$$k_1 = 18.7, \qquad k_4 = 0.42, \qquad K_s = 115.83,$$
$$k_2 = 0.58, \qquad K = 34.4, \qquad p_{\text{CO}_2} = 0.9,$$
$$k_3 = 0.09, \qquad A = 3.3, \qquad H = 737.$$

We solve the problem on the time horizon $t \in [0, 180]$ and for the initial values

$$\mathbf{x}(0) = (0.444, 0.00123, 0, 0.007, 0)^T, \qquad \mathbf{z}(0) = K_s x_1(0) x_4(0). \tag{9.9}$$

The numerical reference solution taken from [MI08] is

$$\mathbf{x}(180) = \begin{pmatrix} 0.1150794920661702 \cdot 10^{0} \\ 0.1203831471567715 \cdot 10^{-2} \\ 0.1611562887407974 \cdot 10^{0} \\ 0.3656156421249283 \cdot 10^{-3} \\ 0.1708010885264404 \cdot 10^{-1} \end{pmatrix},$$
$$\mathbf{z}(180) = 0.4873531310307455 \cdot 10^{-2}. \tag{9.10}$$

The numerical reference values for the Wronskian $\mathcal{W}_{\text{ref}}$ are generated by IDAS with all tolerances set to 10^{-15}.

We solve the problem using DAESOL-II and IDAS, respectively, for the series of tolerances $\texttt{tol}_i = \texttt{atol}_i = 10^{-\frac{4+i}{4}}$, $1 \leq i \leq 44$, and initial stepsize $h_0 = \texttt{tol}_i$. Besides the nominal solution we compute adjoint sensitivities for $n_{\text{adjDir}} = 1, 2, 3$ and the Wronskian $\mathcal{W}$ using $n_{\text{adjDir}} = n_y = 6$ adjoint directions. For both codes, checkpointing is disabled and we use dense direct linear algebra for matrix and vector operations. In IDAS the tolerance for the solution of the adjoint variational DAE was set to $\texttt{bTol} = 10 \cdot \texttt{tol}$.
From the obtained solution and the reference solution we compute the global error $\boldsymbol{\varepsilon}$ at the end of the time horizon and the number of significant digits $\text{scd} = -\log_{10}(\|\boldsymbol{\varepsilon}\|_\infty)$ of the solution. For the comparison of the sensitivities we compute the deviation from the reference values by $\Delta W = \|\mathcal{W} - \mathcal{W}_{\text{ref}}\|_\infty$.

Based on this setup, we compare the performance of both integrator codes for adjoint sensitivity generation. Figure 9.6 on page 238 shows the results of the comparison. We observe that the stepsize error control strategy in DAESOL-II needs in general significantly fewer integration steps than IDAS to reach a given number of significant digits for the nominal solution. Concerning the deviation of the reference Wronskian, we see that both codes deliver very similarly accurate approximations of the reference Wronskian for higher integration accuracies. For lower accuracies, the deviations show some fluctuations for DAESOL-II which is not entirely surprising because, by construction, there is no explicit error control in the adjoint IND scheme.
We recall again that by construction of our adjoint IND scheme the number of required matrix decompositions and Jacobian evaluations in DAESOL-II remains the same for the pure nominal integration and the nominal integration combined with the computation of an arbitrary number of directional adjoint sensitivities. We observe that already for the nominal integration DAESOL-II needs significantly less matrix decompositions and Jacobian evaluations than IDAS. Additionally, in IDAS the number of needed decompositions and evaluations grows approximately linearly with the number of directional adjoint sensitivities that are computed. The number of required directional adjoint model function derivatives grows in DAESOL-II strictly linearly with the number of adjoint sensitivities, in IDAS it grows approximately linearly, while the absolute number in IDAS is slightly larger than in DAESOL-II.
In the end, all this leads to an overall computational time of IDAS for the nominal integration combined with the adjoint sensitivity generation that is significantly larger than for DAESOL-II. For one adjoint sensitivity direction IDAS needs at least 2 times the time of DAESOL-II. Again this difference is smallest for very high integration accuracies, as here DAESOL-II needs in this example more Newton-like iterations than IDAS in the nominal integration. When computing the complete Wronskian DAESOL-II is at least 4 times faster than IDAS.

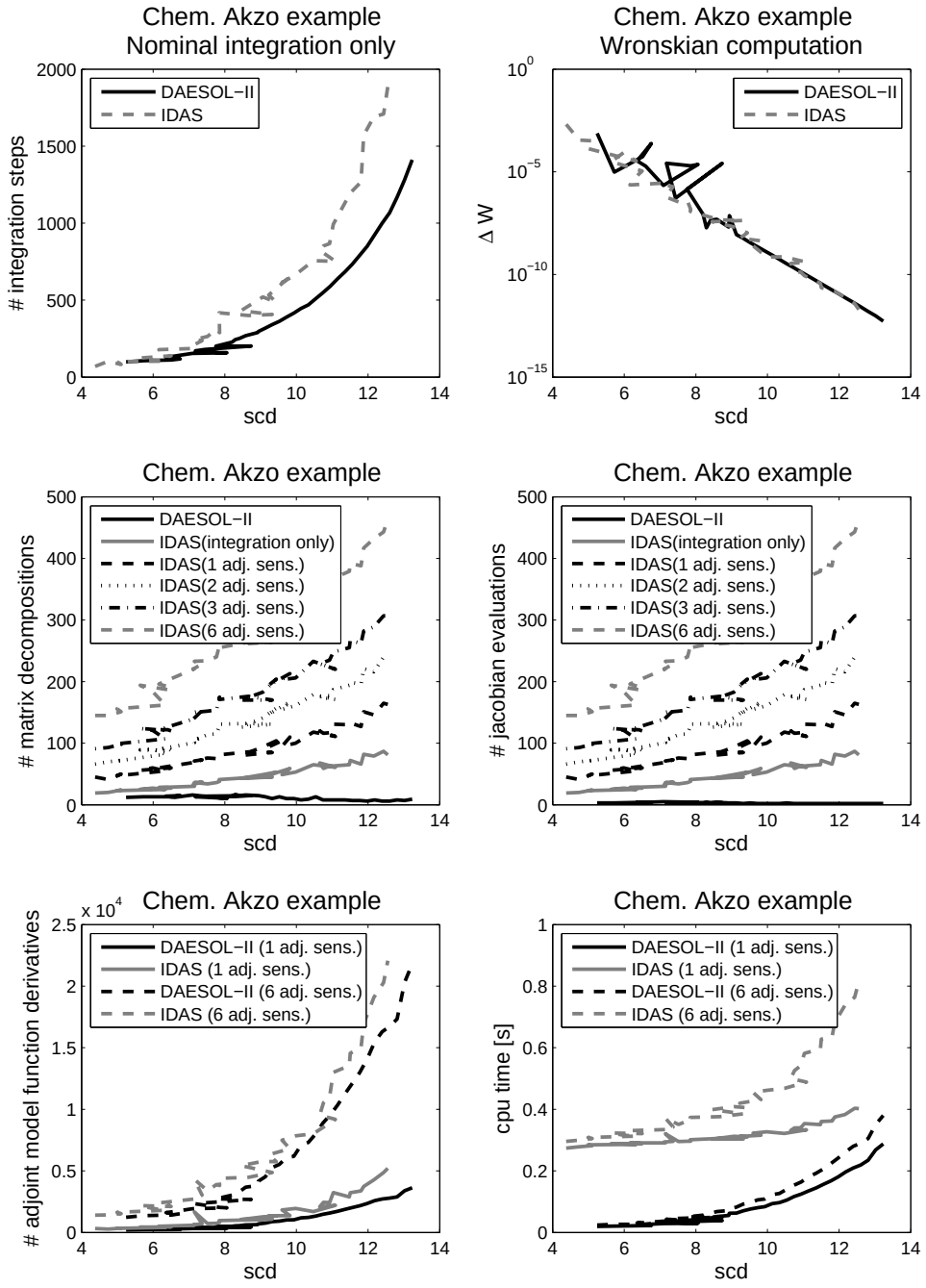

Figure 9.7: The numerical results of applying `DAESOL-II` and IDAS to the chemical Akzo Nobel example. The upper row shows on the left the total number of integration steps needed for the nominal integration in relation to the accuracy of the solution expressed by the number of significant digits. The upper right figure shows the sup-norm of the deviation of the numerically computed Wronskian approximation from the reference Wronskian. The middle row shows the number of needed decompositions of the iteration matrix (left) and of the needed Jacobian evaluations (right). Both quantities are depicted for `DAESOL-II` (where they are independent of the number of adjoint sensitivities) and for IDAS for the cases of a pure nominal integration as well as the combination of nominal integration with the computation of 1, 2, 3 adjoint sensitivities and the Wronskian (6 adj. sens.). The lower row shows for both codes the number of needed directional adjoint derivatives of the model functions (left) as well as the overall computational time (right). Each quantity is depicted for the combination of the nominal integration with one adjoint sensitivity direction and the computation of the Wronskian, respectively.

9.2.4 Test summary

The results of the numerical tests on the three relatively small but numerically challenging examples from the IVP testset of the university of Bari can be summarized as follows. Concerning the solution of the nominal IVP, it can be seen that the stepsize and monitor-strategy implemented in `DAESOL-II` leads, in general, to a smaller number of integration steps and to fewer iteration matrix decompositions and Jacobian evaluations compared to the SUNDIALS codes. For these examples, this advantage is, to some part, compensated for very high integration accuracies by a larger number of Newton-like iterations needed in `DAESOL-II` as these are relatively expensive for these rather small problems. With respect to the adjoint sensitivity generation, the adjoint IND approach implemented in `DAESOL-II` shows its strength compared to the solution of the adjoint variational equation by not needing additional matrix decompositions nor Jacobian evaluations and no additional interpolation of the system states. All this is required by the SUNDIALS codes. As a result, `DAESOL-II` is significantly faster on all presented examples, even more if only a medium integration accuracy is needed. The advantages of `DAESOL-II` become more explicit the larger the problem size becomes and the more directional adjoint sensitivities are required. A theoretical drawback of the adjoint IND scheme, although it did not show up in these examples, might be that by construction there is no explicit error control for the adjoint sensitivities, as they are the exact derivatives of numerical scheme for the nominal IVP solution. Furthermore, for the iterative adjoint IND scheme without checkpointing, that we used here, the iteration matrices used in the nominal solution need to be stored, leading to a slightly increased memory demand compared to the SUNDIALS codes.
Note that these results transfer directly to the task of computing second order sensitivities by the forward/adjoint IND-TC scheme in `DAESOL-II` and by the solution of the corresponding variational ODE/DAE [OB05], respectively, which is implemented in the SUNDIALS solvers. First note that the coding of this variational equation is cumbersome and error-prone. Furthermore, in SUNDIALS the states as well as the forward sensitivities have to be interpolated during the backward integration of the problem and additional matrix factorizations and Jacobian evaluations are needed, like in the first order case. Hence, `DAESOL-II` can be expected to perform significantly better than the SUNDIALS codes also for the task of second order sensitivity generation, because the number of needed model function derivatives is, as in the first order case, similar in both codes.

9.3 Error control for forward sensitivities

In this section we illustrate at hand of a numerical example the problem that may arise for the sensitivity computation by IND-based schemes, if the discretization scheme is determined only based only on the nominal IVP solution. Furthermore, we show that the strategy we propose in Section 6.7.5 on page 185 offers an efficient remedy for this problem.

Consider first the simple ODE-IVP

$$\dot{x}(t) = px(t), \qquad x(0) = x_0, \qquad t \in [0,1], \quad p \in \mathbb{R}, \tag{9.11}$$

with analytical solution $x(t) = x_0 \cdot e^{pt}$. If we solve this problem numerically for the initial value $x_0 = 0$, then the error and stepsize control in `DAESOL-II` will choose ever increasing stepsizes, as the estimated (and in this case also the true) local error is equal to zero. This is perfect for the nominal IVP solution, as it minimizes the overall effort. However, if we compute based on this discretization scheme the directional forward sensitivity w.r.t. the initial value x_0, then the obtained value will be a very bad approximation for the analytical sensitivity $W_{x_0}(t) = e^{pt}$. This is due to the fact that in this singular case the error estimation for the nominal problem is completely invalid for the sensitivity problem. Choosing the discretization grid based on both the nominal problem and the sensitivity, as described in Algorithm 6.10 on page 187, leads to a good approximation of the analytical sensitivity, at the expense of more integration steps: 47 ($p = 2$) respectively 74 ($p = -10$) compared to 22 are needed in this case. This is depicted in Figure 9.8. When using Algorithm 6.10 the discretization scheme is identical to the one we obtain if we only solve the nominal IVP with $x_0 = 1$.

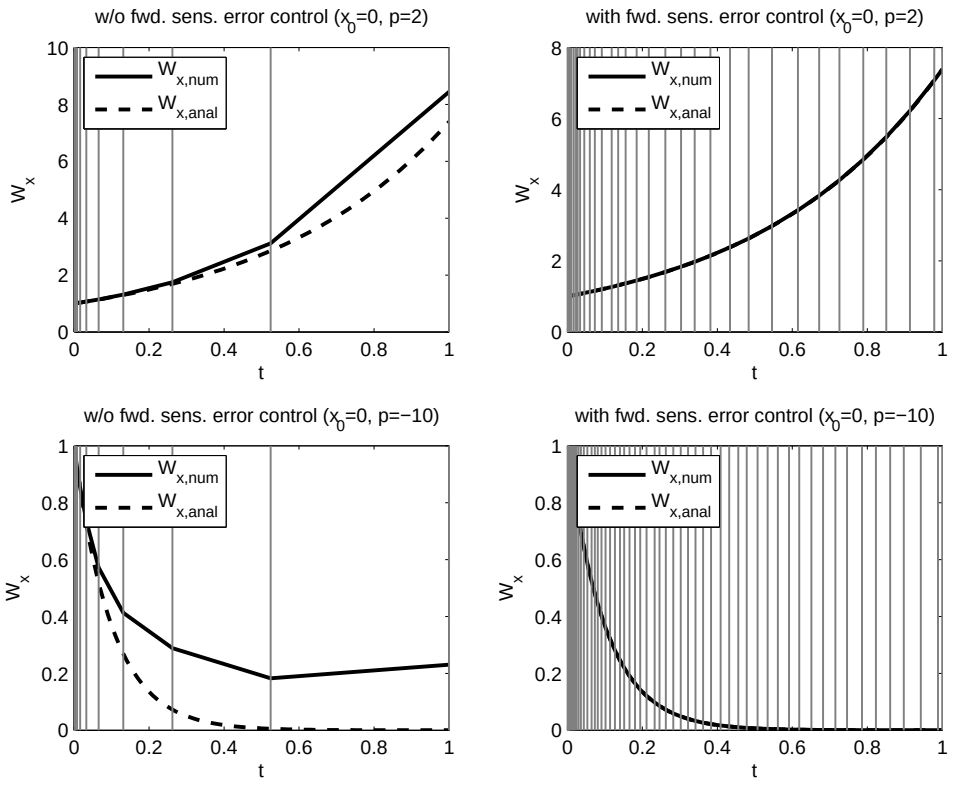

Figure 9.8: The figure shows the analytical and the numerically computed sensitivity of IVP (9.11) with respect to the initial value x_0, which is obtained by first order iterative forward IND. The upper row shows the results for a value of $p = 2$, the lower row for $p = -10$. In the left column the discretization scheme is only determined by the nominal IVP solution process, while in the right column it is determined by Algorithm 6.10, i.e., respecting also the sensitivities. The vertical bars in all the figures indicate the timepoints of the used discretization grid.

This approach is efficient not only for the previous illustrative, somewhat exotic example, but also for more commonly occurring problems. Consider, e.g., the damped harmonic oscillator, described by the ODE-IVP

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} x_2(t) \\ -2p_1x_2(t) - p_2^2x_1(t) \end{pmatrix}, \qquad \mathbf{x}(0) = (2,0)^T, \quad p_1 = 0.1,\ p_2 = 1, \qquad t \in [0,10]. \tag{9.12}$$

For the choice of parameter the analytical solution is then given by

$$\mathbf{x}(t) = \begin{pmatrix} Ce^{-p_1t}\cos(\omega t + \arctan(\phi)); \\ -p_1x_1(t) + Ce^{-p_1t}(-p_2\sin(\omega t + \arctan(\phi))) \end{pmatrix}. \tag{9.13}$$

where $\omega := \sqrt{p_2^2 - p_1^2}$, $\phi := -\frac{x_2(0)+p_1x_1(0)}{\omega x_1(0)}$ and $C := x_1(0)\sqrt{1+\phi^2}$.

We now compute the nominal IVP solution, as well as the Wronskian matrix by iterative forward IND sweeps using `DAESOL-II` using the tolerances `tol` = `atol` = 10^{-i} for $i = 3, 5, 7$. This is done once using the discretization scheme only determined by the nominal IVP solution, and once using a discretization scheme that is based on both the nominal IVP solution and the sensitivities. We compare the results with the analytical values and compute the errors in sup-norm. The results are displayed in Table 9.2 below. We observe that by using Algorithm 6.10 we obtain the Wronskian matrix with about the same order of accuracy that is obtained in the ordinary case for the nominal solution. As a side-effect the nominal solution is in exchange now computed with a higher accuracy as before. Both observations are characteristical for the approach. As the numerical IVP solution and the numerical sensitivities are not decoupled in this approach, but the computed sensitivities are also here the exact derivatives of the computed numerical solution, it is reasonable that enforcing a higher accuracy of the sensitivities leads to a higher accuracy of the underlying numerical solution. Finally, we observe that the activation of forward sensitivity error control leads to a moderate increase of the computational costs.

`tol`	sens. err. ctrl.	error $\mathbf{x}$	error $\mathcal{W}$	# steps	# decomp.	# eval.
10^{-3}	off	$5.56 \cdot 10^{-3}$	$9.84 \cdot 10^{-2}$	47	2	1
	on	$1.01 \cdot 10^{-3}$	$1.79 \cdot 10^{-3}$	56	2	1
10^{-5}	off	$1.36 \cdot 10^{-4}$	$1.97 \cdot 10^{-3}$	72	2	1
	on	$1.98 \cdot 10^{-5}$	$3.06 \cdot 10^{-4}$	92	2	1
10^{-7}	off	$3.09 \cdot 10^{-6}$	$5.36 \cdot 10^{-5}$	115	2	1
	on	$4.95 \cdot 10^{-7}$	$9.00 \cdot 10^{-6}$	147	3	1

Table 9.2: Comparison of the accuracy and the computation cost of the computation of the nominal solution $\mathbf{x}$ and Wronskian $\mathcal{W}$ for the example of the damped harmonic oscillator (9.12) with and without activated error control for the forward sensitivities. Displayed are the error in the solution and the Wronskian, and the number of needed integration steps as well as the number of needed matrix decompositions and Jacobian evaluations.

9.4 Global error estimation

In this section we present a set of 7 ODE and 1 DAE-IVP examples that is used to test our global error estimation strategy, presented in Section 6.7.6 on page 188, and implemented in `DAESOL-II`. We compare our strategy to the one proposed by [CP04] for BDF methods. The latter strategy has been tested on the 7 ODE examples in [TB09], which enables us to make a proper comparison of both approaches.

The setups for the IVPs are as follows.

- Example 1. The Dahlquist equation

$$\dot{x}(t) = \lambda x(t), \quad t \in [0, T], \quad x(0) = x_0. \tag{9.14}$$

  It is solved for the three setups
  a) $\lambda = 1$, $x_0 = 10^{-4}$, $T = 10$,
  b) $\lambda = -1$, $x_0 = 1$, $T = 1$,
  c) $\lambda = -20$, $x_0 = 1$, $T = 1$.

  The analytical solution is given by $x(t) = x_0 \cdot e^{\lambda t}$.

- Example 2.

$$\dot{x}(t) = -(0.25 + \sin(\pi t))x(t)^2, \quad t \in [0, 1], \quad x(0) = 1. \tag{9.15}$$

  The analytical solution is given by $x(t) = \pi/(\pi + 1 + 0.25\pi t - \cos(\pi t))$.

- Example 3.

$$\begin{aligned} \dot{x}_1(t) &= \frac{1}{2(1-t)} x_1(t) - 2t x_2(t) \\ \dot{x}_2(t) &= \frac{1}{2(1-t)} x_2(t) + 2t x_1(t) \\ t \in [0, 10], &\quad \mathbf{x}(0) = (1, 0)^T. \end{aligned} \tag{9.16}$$

  The analytical solution is given by $\mathbf{x}(t) = \sqrt{1+t} \begin{pmatrix} \cos(t^2) \\ \sin(t^2) \end{pmatrix}$.

- Example 4.

$$\begin{aligned} \dot{x}_1(t) &= -x_2(t) \\ \dot{x}_2(t) &= -x_1(t) \\ t \in [0, 10], &\quad \mathbf{x}(0) = (2 \cdot 10^{-4}, 0)^T. \end{aligned} \tag{9.17}$$

  The analytical solution is given by $\mathbf{x}(t) = 10^{-4} \begin{pmatrix} e^{-t} + e^t \\ e^{-t} - e^t \end{pmatrix}$.

- Example 5.

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= -x_1(t) \\ t \in [0, 50], \quad \mathbf{x}(0) &= (0, 1)^T. \end{aligned} \tag{9.18}$$

  The analytical solution is given by $\mathbf{x}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$.

- Example 6.

$$\begin{aligned} \dot{x}_1(t) &= x_1(t) \\ \dot{x}_2(t) &= x_2(t) + x_1(t)x_1(t) \\ \dot{x}_3(t) &= x_3(t) + x_1(t)x_2(t) \\ \dot{x}_4(t) &= x_4(t) + x_1(t)x_3(t) + x_2(t)x_2(t) \\ \dot{x}_5(t) &= x_5(t) + x_1(t)x_4(t) + x_2(t)x_3(t) \\ t \in [0, 1], \quad \mathbf{x}(0) &= (1, 1, 0.5, 0.5, 0.25)^T. \end{aligned} \tag{9.19}$$

  The analytical solution is given by $\mathbf{x}(t) = \begin{pmatrix} e^t \\ e^{2t} \\ 0.5\, e^{3t} \\ 0.5\, e^{4t} \\ 0.25\, e^{5t} \end{pmatrix}$.

- Example 7.

$$\begin{aligned} \dot{x}(t) &= -L(x(t) - \sin(\pi t)) + \pi \cos(\pi t)) \\ t \in [0, 1], \quad L = 50, \quad x(0) &= 0. \end{aligned} \tag{9.20}$$

  The analytical solution is given by $x(t) = \sin(\pi t)$.

- Example 8.

  As DAE example we use the chemical Akzo Nobel problem, described in Section 9.2.3 on page 239.

We solve every IVP for the 8 different tolerances $\texttt{tol} = 10^{-2}, \ldots, 10^{-10}$. During the nominal solution the local error estimates for each (successful) integration step are stored. Afterwards, we perform a first order adjoint IND sweep, initialized with the set of unit directions, corresponding to the components of the solution. During the adjoint sweep, the global error estimates are accumulated using Formula (6.48). Finally, we compare each global error estimate $\tilde{\boldsymbol{\varepsilon}}$ with the corresponding real global error $\boldsymbol{\varepsilon}$ of the numerical solution at the end of the integration horizon. To do this, we compute the so-called effectivity index of the estimation, given by

$$I := \frac{\|\tilde{\boldsymbol{\varepsilon}}\|_2}{\|\boldsymbol{\varepsilon}\|_2}. \tag{9.21}$$

Clearly, in theory a value of $I = 1.0$ would be desirable. However, in practice values between $I = 0.5$ and $I = 2.0$ represent already very good estimations, and also somewhat larger deviations are actually good enough for the practical application in adaptive algorithms.

The effectivity indices obtained by solving the above examples and estimating the global error using (6.48) are given in Table 9.3 and compared to the results of the approach of Cao and Petzold stated in [TB09], which have been obtained using IDAS. We observe that our approach based on the intermediate values of the adjoint IND sweep delivers for the ODE examples in general a more precise and more reliable error estimation as the effectivity indices lie closer to 1.0. Exceptions are some singular overestimations of the error in case of Example 1 for large integration tolerances. For Example 8 we observe that our approach transfers directly to (index 1) DAEs with good results.

Example/solver		tol							
		10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
1a	IDAS	7.13	7.23	7.09	9.12	8.95	8.54	16.72	9.18
	DAESOL-II	1.30	0.81	0.51	0.88	1.48	0.69	1.29	1.81
1b	IDAS	13.41	3.22	2.96	129.2	6.74	2.45	8.67	5.73
	DAESOL-II	4.75	28.19	3.00	3.24	1.87	0.92	15.67	3.34
1c	IDAS	0.61	1.59	0.46	0.45	2.08	7.63	10.12	10.36
	DAESOL-II	86.33	86.79	3.42	6.74	3.65	3.04	2.75	2.04
2	IDAS	5.63	9.27	106.9	19.36	72.67	13.98	16.38	0.31
	DAESOL-II	2.75	0.83	2.38	4.97	1.98	6.93	3.13	2.42
3	IDAS	13.58	13.02	13.66	13.00	11.59	10.92	10.77	11.35
	DAESOL-II	2.67	2.68	2.54	2.73	2.67	2.72	2.90	2.42
4	IDAS	7.13	7.25	7.08	6.45	8.68	12.10	15.70	12.45
	DAESOL-II	1.30	0.77	0.73	0.94	2.00	1.50	1.66	1.88
5	IDAS	4.13	8.89	15.04	7.98	1.45	7.63	8.64	4.16
	DAESOL-II	3.16	2.75	2.89	2.73	2.72	2.58	2.71	2.67
6	IDAS	6.14	10.54	14.31	8.09	12.94	4.62	8.35	13.86
	DAESOL-II	0.86	2.33	1.90	1.89	2.26	2.70	2.63	2.53
7	IDAS	0.003	0.04	0.002	0.008	$1 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$7 \cdot 10^{-5}$	$1 \cdot 10^{-5}$
	DAESOL-II	6.86	12.35	14.31	7.70	2.05	4.48	3.96	2.61
8	DAESOL-II	1.07	1.08	6.11	14.54	0.99	2.39	2.38	2.38

Table 9.3: The table shows the effectivity indices defined by (9.21) obtained by the solution of Examples 1 to 7 of this section for different tolerances using Formula (6.48) implemented in DAESOL-II and the approach of Cao and Petzold implemented in IDAS, respectively, to estimate the global error at the end of the integration horizon. For the Example 8 only the results of our approach are given, as the approach in [CP04] is not directly applicable to DAEs. It can be observed that the approach implemented in DAESOL-II leads in general to more accurate and more reliable estimations, with the exception of some significant overestimations of the error for Example 1 and large integration tolerances. Also for the DAE example DAESOL-II delivers good estimates.

10 Optimal control of a distillation column

In this Chapter, we use the new adjoint based exact-Hessian SQP method proposed in Chapter 7 to solve a real world application problem. We use here the implementation of our algorithm in the C++ code `DynamicLiftOpt`, together with `SolvIND/DAESOL-II` for the evaluation of the model functions and their derivatives as well as the solution of the Initial Value Problems (IVPs) and the corresponding sensitivity generation.
As application example we choose the optimal control of a high purity binary distillation column. The model is taken from [Die02] and describes a pilot plant distillation column of the "Institut für Systemdynamik und Regelungstechnik" of the University of Stuttgart. The numerical setup of the optimization problem is taken from [SBPD+07].
In the following, we first give a brief description of the model of the distillation column. Afterwards, we present the setup of the optimal control problem. Finally, we present the results obtained by the numerical solution of the problem using `DynamicLiftOpt` and, for comparison, `MUSCOD-II`.

10.1 Description of the distillation column

The distillation column we consider here is used for the separation of a (binary) mixture of methanol and n-propanol. It consists of 40 bubble cap trays and has an overall height of 7m and a diameter of 10cm. The reboiler is heated electrically and the overhead vapor is totally condensed in a water cooled condenser that is open to atmosphere. The preheated feed stream enters the column at the feed tray as saturated liquid.
In our case, the inputs that can be used to control the process are given by the heat input q to the reboiler and the volumetric reflux flow l_{vol}. The basic aim is to fullfill given high purity requirements for the distillate d_{vol} and the bottom product b_{vol} at any time. The scheme of the distillation column is depicted in Figure 10.1 on the next page.

As the DAE model of the column is described in detail in [Die02], we give here only a short summary. We number the $n_{\text{trays}} = 40$ bubble cap trays from bottom to top with $k = 1, \ldots, 40$, where the tray with number $n_{\text{feed}} = 20$ is the feed tray. The index $k = 0$ denotes in the following the reboiler and the index $n_{\text{trays}} + 1 = 41$ the condenser.
On each tray k, we denote the molar fluxes of the liquid phase with l_k and of the vapor phase with v_k. The corresponding temperatures are given by θ_k. The molar vapor flux out of the reboiler is denoted by v_0 and the molar flux of the liquid bottom product stream with b. The molar liquid reflux from the condenser to the top tray is given by $l_{n_{\text{trays}}+1}$ and the distillate stream with d. The

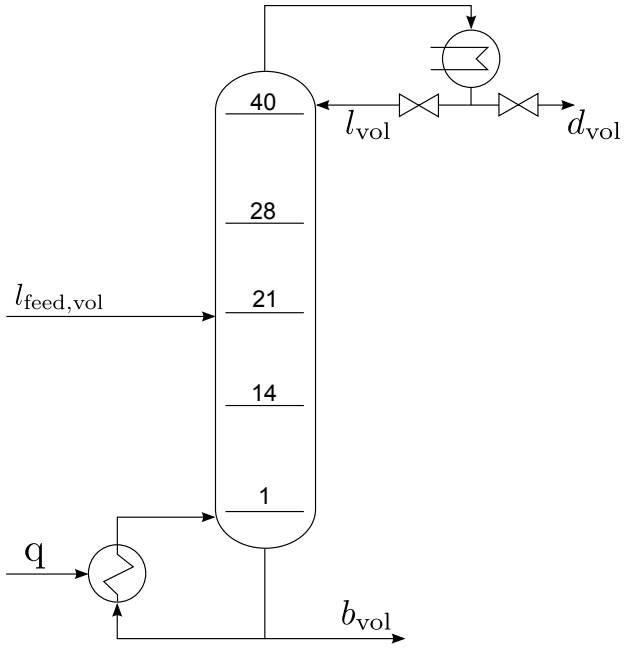

Figure 10.1: Flowsheet of the binary distillation column

liquid molar feed stream entering at the feed tray we denote with l_{feed}. The molar fluxes in the distillation column are shown in Figure 10.2 on the facing page.
Furthermore, we denote with c_k^l the liquid methanol concentrations. Note that, as we have a binary distillation column, the concentrations of n-propanol can then be determined by the closing conditions $c_{\text{n-propanol},k} = 1 - c_k^l$. The pressures in reboiler, condenser and on the trays are denoted with p_k, the liquid volume holdups with ν_k^v, the molar holdups with ν_k and the methanol concentration in the vapor phase with c_k^v.
We assume in the following that the liquid volume holdups of reboiler and condenser are constant as well as the pressures in the reboiler, on the trays and in the condenser. The condenser pressure is fixed to the outside pressure p_{top} and the pressure loss between two trays is constant, i.e.,

$$p_k := p_{k+1} + \Delta p_k.$$

With these assumptions a stiff nonlinear DAE model can be derived as follows.

- The mass balances for the molar holdup on the trays give

$$\dot{\nu}_k = v_{k-1} - v_k + l_{k+1} - l_k, \tag{10.1a}$$

for $k \in \{1, \ldots, n_{\text{feed}} - 1, n_{\text{feed}} + 1, \ldots, n_{\text{trays}}\}$. For the feed tray we have with $k = n_{\text{feed}}$

$$\dot{\nu}_k = v_{k-1} - v_k + l_{k+1} - l_k + l_{\text{feed}}. \tag{10.1b}$$

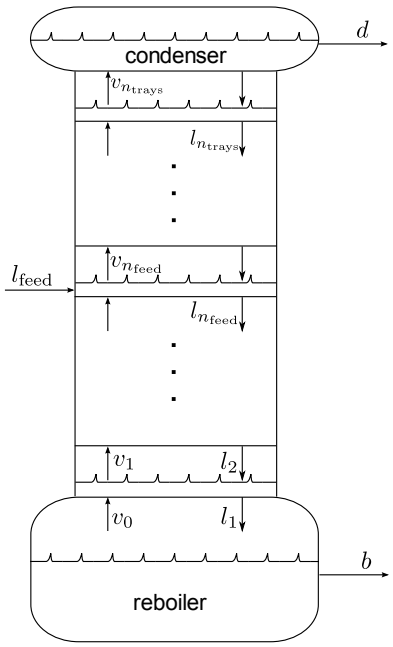

Figure 10.2: Molar flows in the binary distillation column

- Mass conservation for methanol requires in the reboiler

$$\dot{c}_0^l \nu_0 + c_0^l \dot{\nu}_0 = -v_0 c_0^v + l_1 c_1^l - b c_0^l \tag{10.2a}$$

and in the condenser (with $k = n_{\text{trays}}$)

$$\dot{c}_{k+1}^l \nu_{k+1} + c_{k+1}^l \dot{\nu}_{k+1} = v_k c_k^v - d c_{k+1}^l - l_{k+1} c_{k+1}^l. \tag{10.2b}$$

For the feed tray ($k = n_{\text{feed}}$) we have

$$\dot{c}_k^l \nu_k + c_k^l \dot{\nu}_k = v_{k-1} c_{k-1}^v - v_k c_k^v + l_{k+1} c_{k+1}^l - l_k c_k^l + l_{\text{feed}} c_{\text{feed}}^l, \tag{10.2c}$$

where c_{feed}^l is the concentration of methanol in the feed flow. Finally, on the rest of the trays ($k \in \{1, \ldots, n_{\text{feed}} - 1, n_{\text{feed}} + 1, \ldots, n_{\text{trays}}\}$) it holds

$$\dot{c}_k^l \nu_k + c_k^l \dot{\nu}_k = v_{k-1} c_{k-1}^v - v_k c_k^v + l_{k+1} c_{k+1}^l - l_k c_k^l. \tag{10.2d}$$

- If we denote the liquid and vapor stream enthalpies as $h_k^l := h^l(c_k^l, \theta_k)$ and $h_k^v := h^v(c_k^v, \theta_k, p_k)$, respectively, the enthalpy balance gives us in the reboiler

$$\dot{\nu}_0 h_0^l + \nu_0 \left(\frac{\partial h_0^l}{\partial c_0^l} \dot{c}_0^l + \frac{\partial h_0^l}{\partial \theta_0} \dot{\theta}_0 \right) = q - q_{\text{loss}} - v_0 h_0^v + l_1 h_1^l - b h_0^l, \tag{10.3a}$$

where q_{loss} accounts for possible heat losses. For the top tray we have with $k = n_{\text{trays}}$

$$\dot{\nu}_k h_k^l + \nu_k \left(\frac{\partial h_k^l}{\partial c_k^l} \dot{c}_k^l + \frac{\partial h_k^l}{\partial \theta_k} \dot{\theta}_k \right) = v_{k-1} h_{k-1}^v - v_k h_k^v + l_{k+1} h^l(c_{k+1}^l, \theta_{\text{cond}}, p_{k+1}) - l_k h_k^l, \quad (10.3b)$$

where θ_{cond} is the temperature in the condenser and hence also of the liquid reflux l_{k+1} to the top tray. On the feed tray ($k = n_{\text{feed}}$) we obtain

$$\dot{\nu}_k h_k^l + \nu_k \left(\frac{\partial h_k^l}{\partial c_k^l} \dot{c}_k^l + \frac{\partial h_k^l}{\partial \theta_k} \dot{\theta}_k \right) = v_{k-1} h_{k-1}^v - v_k h_k^v + l_{k+1} h_{k+1}^l - l_k h_k^l + l_{\text{feed}} h^l(c_{\text{feed}}^l, \theta_{\text{feed}}, p_{\text{feed}}), \quad (10.3c)$$

and for the rest of the trays with $k \in \{1, \ldots, n_{\text{feed}} - 1, n_{\text{feed}} + 1, \ldots, n_{\text{trays}} - 1\}$

$$\dot{\nu}_k h_k^l + \nu_k \left(\frac{\partial h_k^l}{\partial c_k^l} \dot{c}_k^l + \frac{\partial h_k^l}{\partial \theta_k} \dot{\theta}_k \right) = v_{k-1} h_{k-1}^v - v_k h_k^v + l_{k+1} h_{k+1}^l - l_k h_k^l. \quad (10.3d)$$

- Hydrodynamics: The liquid flow l_k out of each tray is determined based on the "Francis weir formula" and given for $k = 1, \ldots, n_{\text{trays}}$ by

$$l_k \mu(c_k^l, \theta_k) = w_k (\nu_k^v - \nu_k^{\text{ref}})^{\frac{3}{2}}, \quad (10.4)$$

where $\mu(c_k^l, \theta_k)$ is the molar volume of the liquid mixture, ν_k^{ref} the reference volume and w_k are parameter to be defined.

- The temperatures on each tray $k = 0, \ldots, n_{\text{trays}}+1$ are implicitly defined by Dalton's formula

$$p_k - p_1^s(\theta_k) c_k^l - p_2^s(\theta_k)(1 - c_k^l) = 0, \quad (10.5a)$$

where the partial pressures $p_j^s(\theta_k)$, $j = 1, 2$, are defined by the Antoine equation as

$$p_j^s(\theta) = \exp\left(\pi_{a,j}^A - \frac{\pi_{b,j}^A}{\theta + \pi_{c,j}^A} \right) \quad (10.5b)$$

and the coefficients are given as follows.

	j	$\pi_{a,j}^A$	$\pi_{b,j}^A$[K]	$\pi_{c,j}^A$[K]
methanol	1	23.480	3626.6	-34.29
n-propanol	2	22.437	3166.4	-80.15

Some of the quantities in the equations above can be computed directly in terms of other quantities such that they can be eliminated:

- The vapor concentrations c_k^v are computed using parameter α_k to account for non-ideality of the trays and unmodeled effects for $k = 1, \dots, n_{\text{trays}}$ as

$$c_k^v = \alpha_k \frac{p_1^s(\theta_k)}{p_k} c_k^l + (1 - \alpha_k) c_{k-1}^v \tag{10.6}$$

  and for $k = 0$ as $c_0^v = \frac{p_1^s(\theta_0)}{p_0} c_0^l$.

- The molar volumes $\mu(c_k^l, \theta_k)$ of the liquid mixture are given by

$$\mu(c^l, \theta)(c^l, \theta) := c^l \mu_1(\theta) + (1 - c^l)\mu_2(\theta), \tag{10.7a}$$

  where the molar volumes of the undiluted components $\mu_j(\theta)$ are computed by

$$\mu_j(\theta) := \frac{1}{\pi_{a,j}^v} \exp_{\pi_{b,j}^v}(1 + \exp_{(1-\theta/\pi_{c,j}^v)}(\pi_{d,j}^v)). \tag{10.7b}$$

  Here the coefficients are given as

	j	$\pi_{a,j}^v$ [kmol l^{-1}]	$\pi_{b,j}^v$	$\pi_{c,j}^v$[K]	$\pi_{d,j}^v$
methanol	1	2.288	0.26850	512.4	0.2453
n-propanol	2	1.235	0.27136	536.4	0.2400

- The molar feed flow l_{feed} can be obtained from the volumetric one $l_{\text{feed,vol}}$ by

$$l_{\text{feed,vol}} = \mu(c_{\text{feed}}^l, \theta_{\text{feed}}) l_{\text{feed}}. \tag{10.8}$$

- Analogously, the molar liquid reflux $l_{n_{\text{trays}}+1}$ can be obtained from the volumetric one l_{vol} by

$$l_{n_{\text{trays}}+1} = \mu(c_{n_{\text{trays}}+1}^l, \theta_{n_{\text{trays}}+1}) l_{\text{vol}}. \tag{10.9}$$

- The vapor flux v_0 can be computed from (10.3a).

- We assume that the volume ν_0^v and $\nu_{n_{\text{trays}}+1}^v$ of reboiler and condenser, respectively, are fixed. Then ν_0 and $\nu_{n_{\text{trays}}+1}$ can be eliminated via

$$0 = \dot{\nu}_k^v = \mu(c_k^l, \theta_k)\dot{\nu}_k + \left(\frac{\partial \mu(c_k^l, \theta_k)}{\partial c^l} \dot{c}^l + \frac{\partial \mu(c_k^l, \theta_k)}{\partial \theta} \dot{\theta} \right) \nu_k, \tag{10.10a}$$

  for $k = 0, n_{\text{trays}} + 1$ and the mass conservation in the reboiler is given by

$$\dot{\nu}_0 = -v_0 + l_1 - b \tag{10.10b}$$

  and in the condenser by

$$\dot{\nu}_{n_{\text{trays}}} = v_{n_{\text{trays}}} - l_{n_{\text{trays}}+1} - d. \tag{10.10c}$$

- Finally the liquid and vapor enthalpies $h^l(c^l,\theta)$ and $h^v(c^v,\theta,p)$ are given by

$$h^l(c^l,\theta) := c^l h^{l,1}(\theta) + (1-c^l)h^{l,2}(\theta), \tag{10.11a}$$
$$h^v(c^v,\theta,p) := c^v h^{v,1}(\theta,p) + (1-c^v)h^{v,2}(\theta,p). \tag{10.11b}$$

Here the pure liquid enthalpies $h^{l,j}(\theta)$ are determined by

$$h^{l,j}(\theta) := 4.186\frac{\mathrm{J}}{\mathrm{mol}}\left[\pi^h_{j,1}(\theta-\pi^\theta_0) + \pi^h_{j,2}(\theta-\pi^\theta_0)^2 + \pi^h_{j,3}(\theta-\pi^\theta_0)^3\right], \tag{10.11c}$$

where $\pi^\theta_0 = 273.15\mathrm{K}$. The pure vapor enthalpies $h^{v,j}(\theta,p)$ are given by

$$\begin{aligned} h^{v,j}(\theta,p) := h^{l,j}(\theta) + 8.3147\frac{\mathrm{J}}{\mathrm{mol\ K}}\pi^\theta_{c,j}\sqrt{1-\frac{p}{\pi^p_{c,j}}\left(\frac{\theta}{\pi^\theta_{c,j}}\right)^{-3}} \\ \cdot\left[6.09648 - 1.28862\frac{\theta}{\pi^\theta_{c,j}} + 1.016\left(\frac{\theta}{\pi^\theta_{c,j}}\right)^7\right. \\ \left.+\pi^\Omega_j\left(15.6875 - 13.4721\frac{\theta}{\pi^\theta_{c,j}} + 2.615\left(\frac{\theta}{\pi^\theta_{c,j}}\right)^7\right)\right], \end{aligned} \tag{10.11d}$$

where the coefficients used here are given as

	j	$\pi^h_{j,1}$ [K^{-1}]	$\pi^h_{j,2}$ [K^{-2}]	$\pi^h_{j,3}$ [K^{-3}]	$\pi^\theta_{c,j}$ [K]	$\pi^p_{c,j}$ [Pa]	π^Ω_j
methanol	1	18.31	$1.713\cdot10^{-2}$	$6.399\cdot10^{-5}$	512.6	$8.096\cdot10^6$	0.557
n-propanol	2	31.92	$4.490\cdot10^{-2}$	$9.663\cdot10^{-5}$	536.7	$5.166\cdot10^6$	0.612

The values of the remaining system parameter have been estimated based on the real column. The resulting estimates are given by

parameter	value	parameter	value
ν^v_0	8.5 [l]	p_{top}	939 [h Pa]
$\nu^v_{n_{\mathrm{trays}}+1}$	0.17 [l]	$\Delta p_0,\ldots,\Delta p_{n_{\mathrm{feed}}-1}$	2.5 [h Pa]
$\nu^{\mathrm{ref}}_1,\ldots,\nu^{\mathrm{ref}}_{n_{\mathrm{trays}}}$	0.155 [l]	$\Delta p_{n_{\mathrm{feed}}},\ldots,\Delta p_{n_{\mathrm{trays}}}$	1.9 [h Pa]
$\alpha_0,\ldots,\alpha_{n_{\mathrm{feed}}}$	62%	θ_{feed}	71 [°C]
$\alpha_{n_{\mathrm{feed}}+1},\ldots,\alpha_{n_{\mathrm{trays}}}$	35%	θ_{cond}	47.2 [°C]
$w_0,\ldots,w_{n_{\mathrm{trays}}}$	0.166 [$\mathrm{l}^{-1/2}\ \mathrm{s}^{-1}$]	$l_{\mathrm{feed,vol}}$	14.0 [l h^{-1}]
q_{loss}	0.51 [kW]	c^l_{feed}	0.32

For a detailed description of the parameter estimation refer to [Die02].

Based on the considerations above the equations can be summarized as an index 1 DAE system consisting of the 82 differential variables

$$\mathbf{x} = (c_0^l, \ldots, c_{41}^l, \nu_1, \ldots, \nu_{40})$$

and the 122 algebraic variables

$$\mathbf{z} = (l_1, \ldots, l_{40}, v_1, \ldots, v_{40}, \theta_0, \ldots, \theta_{41}).$$

The differential equations are given by (10.1) and (10.2), whereas the algebraic equations are given by (10.3b)-(10.3d), (10.4) and (10.5a).

10.2 Description of the optimal control problem

The setup of the optimization problem is taken from [SBPD+07]. The fundamental control aim is to satisfy high purity constraints for the product concentrations c_0^l and $c_{n_{\text{trays}}+1}^l$. As usual in distillation control the product concentrations are not controlled directly. Instead, the concentrations on trays 14 and 28 are controlled as they are much more sensitive to disturbances than the product concentrations. If they are kept constant at a suitable given setpoint the product purities are usually safely guaranteed for a wide range of process conditions. Since concentrations are difficult to measure, we use here the temperatures on the tray 14 and 28, which are directly coupled to the concentration by Dalton's formula and the Antoine equation. The desired setpoint is chosen as $\boldsymbol{\theta}^{\text{ref}} := (\theta_{28}^{\text{ref}}, \theta_{14}^{\text{ref}})^T := (70\ ^\circ\text{C}, 88\ ^\circ\text{C})^T$ and we denote the corresponding steady-state of the system with $(\mathbf{x_s}^T, \mathbf{z_s}^T)^T$. The corresponding control $\mathbf{u_s}$ is given by $\mathbf{u_s} := (l_{\text{vol}}^s, q^s)^T := (4.1833\ \text{l/h}, 2.4899\ \text{kW})^T$.

The objective function of the Optimal Control Problem (OCP) is chosen as the integral over a least-squares term which penalizes the deviation from the setpoint temperatures and regularizes the control. It can be written as

$$\int_{t_0}^{t_{\text{end}}} \left[\|(\tilde{\boldsymbol{\theta}}\mathbf{z} - \boldsymbol{\theta}^{\text{ref}})\|_2^2 + \|\mathbf{R}(\mathbf{u} - \mathbf{u_s})\|_2^2 \right] dt, \tag{10.12}$$

where $\tilde{\boldsymbol{\theta}}$ stands for the projection matrix that extracts the temperatures at trays 14 and 28 from the vector of algebraic variables $\mathbf{z}$. Furthermore, $\mathbf{R} = \text{diag}(0.05\ ^\circ\text{C h l}^{-1}, 0.05\ ^\circ\text{C kW}^{-1})$ is a small diagonal weighting matrix for the regularization term and the time horizon is given by $t_0 = 0$ and $t_{\text{end}} = 5400\text{s}$.
The constraints of the OCP are given by the DAE model of the distillation column presented in the last section, the prescribed initial states of the process as well as the requirement that the bottom product flux and the distillate flux are always non-negative.

In our specific setup, the aim of the optimal control is to drive the process from a disturbed initial state, caused by a malfunction in the heating system, as close as possible to the desired reference

setpoint (and the corresponding steady-state) in the given time. To solve the problem, we employ a direct multiple shooting discretization of the problem. We approximate the control functions as piecewise constant. We use 7 multiple shooting intervals where the length of the first six intervals is set to 300s and the last one to 3600s. On the last interval the control is fixed to the setpoint steady-state control $\mathbf{u_s}$. The overall number of variables in the resulting (full-space) optimization problem is 1684.

10.3 Numerical results

We solve the problem for a set of scenarios where the initial state of the system is perturbed due to an abnormal increase in heating of ρ percent compared to the reference setpoint, with $\rho \in \{10, 20, 30, 40, 50\}$. We initialize the values at the first 6 multiple shooting nodes according to this disturbed state and at the last 2 nodes with the desired reference steady-state. Then we use our L-PRSQP algorithm, presented in Chapter 7 and implemented in our C++ package `DynamicLiftOpt`, and the software package `MUSCOD-II` [DLS01] to solve this problem. From the `MUSCOD-II` package, which is based on classical condensing, we employ the SQP variant that uses high-rank block-wise BFGS updates for the Hessian.

The computations are performed on a desktop computer with an Intel Pentium D CPU with 3.0 GHz and 3.8 GB of RAM, running under the Ubuntu 8.04 64-bit operating system. The code was compiled with GCC version 4.3.2. The generation of the model derivatives in `SolvIND` is done using the tool ADOL-C [GJU96] in version 2.1.5. UMFPACK [Dav04] in version 5.0.2 is used for sparse matrix operations within `DAESOL-II`. The occurring QP subproblems were solved using the code `QPOPT` [GMS95] in version 1.0.
As tolerance for the integration we choose `tol` $= 10^{-8}$ and for the optimization `kktTol` $= 10^{-6}$ as termination criterion (see Section 7.2.1).

The Figures 10.3 on page 258 to 10.5 on page 260 show exemplarily the optimal solution computed by `DynamicLiftOpt` for the cases of a disturbance in heating of $\rho = 10, 30$ and 50 percent. We observe that in each case the optimal control is able to drive the system back to the desired setpoint.
A comparison of the objective values of the results computed by `DynamicLiftOpt` and `MUSCOD-II` is given in Table 10.1 on the next page. We observe that both algorithms find essentially the same numerical solutions, except in the case of $\rho = 50$, where `MUSCOD-II` converges accidentally to a suboptimal solution.
The statistics of the computations are given in Table 10.2 on the facing page. It can be observed that `DynamicLiftOpt` needs generally about half the number of SQP iterations to find a solution compared to the BFGS approach in `MUSCOD-II`. Also the number of matrix factorizations during integration and sensitivity generation is significantly smaller, because `DynamicLiftOpt` is able to reuse the discretization scheme for sensitivity sweeps via the capabilities of `SolvIND`/`DAESOL-II`. Furthermore, we see that one L-PRSQP iteration is about 2.5 to 3 times as expensive as a BFGS iteration. Note that this is mainly due to the high computational effort for (first and) second

order adjoint AD/TC-sweeps through the DAE model functions. This bottleneck causes between 74.7 and 75.2 percent of the overall CPU load of the solution process. As already observed in the SMB example in Section 9.1, the actual implementation based on operator overloading does in practice not perform near to the theoretical complexity bounds. Hence, further improvements in this area will directly improve the performance of our algorithm considerably. Assuming adjoint TC propagation through the model functions would be possible near the theoretical bounds, our L-PRSQP algorithm would become competitive against the BFGS approach even in terms of computational time per iteration.

Finally, it can be seen that, compared to the corresponding exact-Hessian approach of `MUSCOD-II`, our L-PRSQP approach is about 30 times faster. This superior performance makes our L-PRSQP approach also feasible for the use in online strategies for closed-loop real-time optimal control. It is the first exact-Hessian algorithm based on direct multiple shooting that allows for reasonable sampling times in the context of practical applications.

	Objective value	
ρ	`DynamicLiftOpt`(L-PRSQP)	`MUSCOD-II`(BFGS)
10	1.821626e+02	1.821693e+02
20	3.215875e+02	3.215885e+02
30	4.798845e+02	4.799006e+02
40	6.313409e+02	6.313454e+02
50	7.688280e+02	8.565571e+02

Table 10.1: Objective value of the numerical solutions of the optimal control problem for the binary distillation column. The values are shown for disturbances of $\rho = 10, 20, 30, 40, 50$. The solutions are computed by the L-PRSQP approach implemented in `DynamicLiftOpt` as well as the high-rank BFGS update version of `MUSCOD-II`. We observe that both algorithms compute the same solutions, except in the case of $\rho = 50$ where `MUSCOD-II` converges accidentally to a suboptimal solution.

	`DynamicLiftOpt`				`MUSCOD-II`				
	L-PRSQP				BFGS				ex. Hess
ρ	#iter	avg. time	#step	#fac	#iter	avg. time	#step	#fac	avg. time
10	8	1:03	15564	492	16	0:23	25658	14266	32:19
20	9	1:05	18475	583	13	0:24	22012	12138	32:56
30	9	1:05	20445	607	17	0:26	33763	16401	34:42
40	10	1:02	19925	601	19	0:24	33081	18060	36:58
50	13	1:05	25365	762	25	0:24	48781	23878	36:49

Table 10.2: Statistics of the numerical solution of the optimal control problem for the binary distillation column using `DynamicLiftOpt` and `MUSCOD-II`. For each disturbance scenario $\rho = 10, 20, 30, 40, 50$ the number of SQP iterations are given, the average time per SQP iteration (in mm:ss) as well as the overall number of integration steps and matrix factorizations in the integrator. For the purpose of comparison additionally the average time per SQP iteration needed by the exact-Hessian approach of `MUSCOD-II` is shown, which is based on a finite-difference approximation of the Hessian.

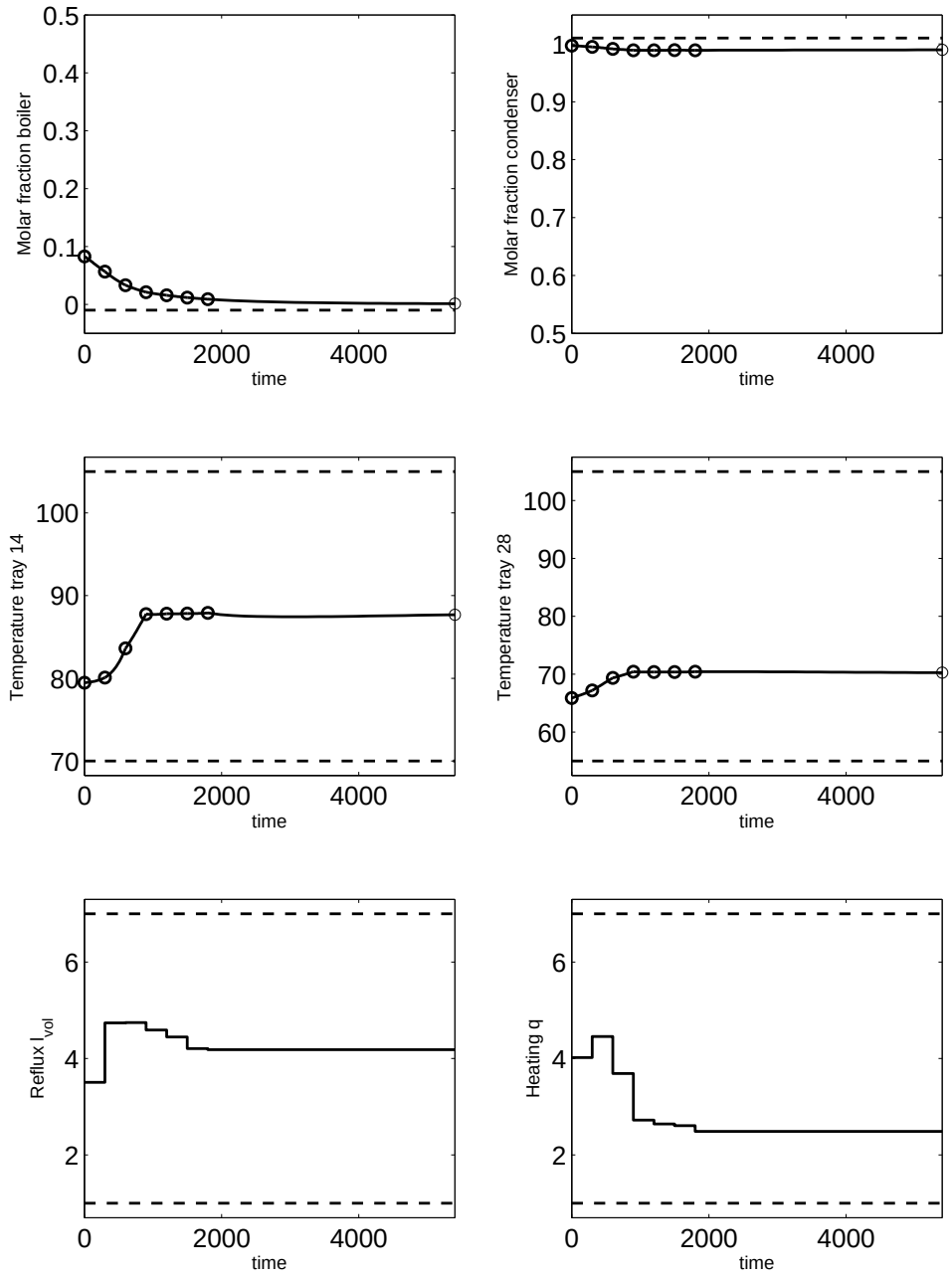

Figure 10.3: Numerical solution of the optimal control problem for the binary distillation column with $\rho = 10$ computed by `DynamicLiftOpt`. The upper row shows the concentrations of methanol in the boiler and the condenser, respectively, corresponding to the purities of the distillate and the bottom product. The middle row shows the temperatures on the "reference" trays 14 and 28. The lower row shows the computed optimal control moves for the volumetric reflux from the condenser to the top tray and for the heating in the reboiler.

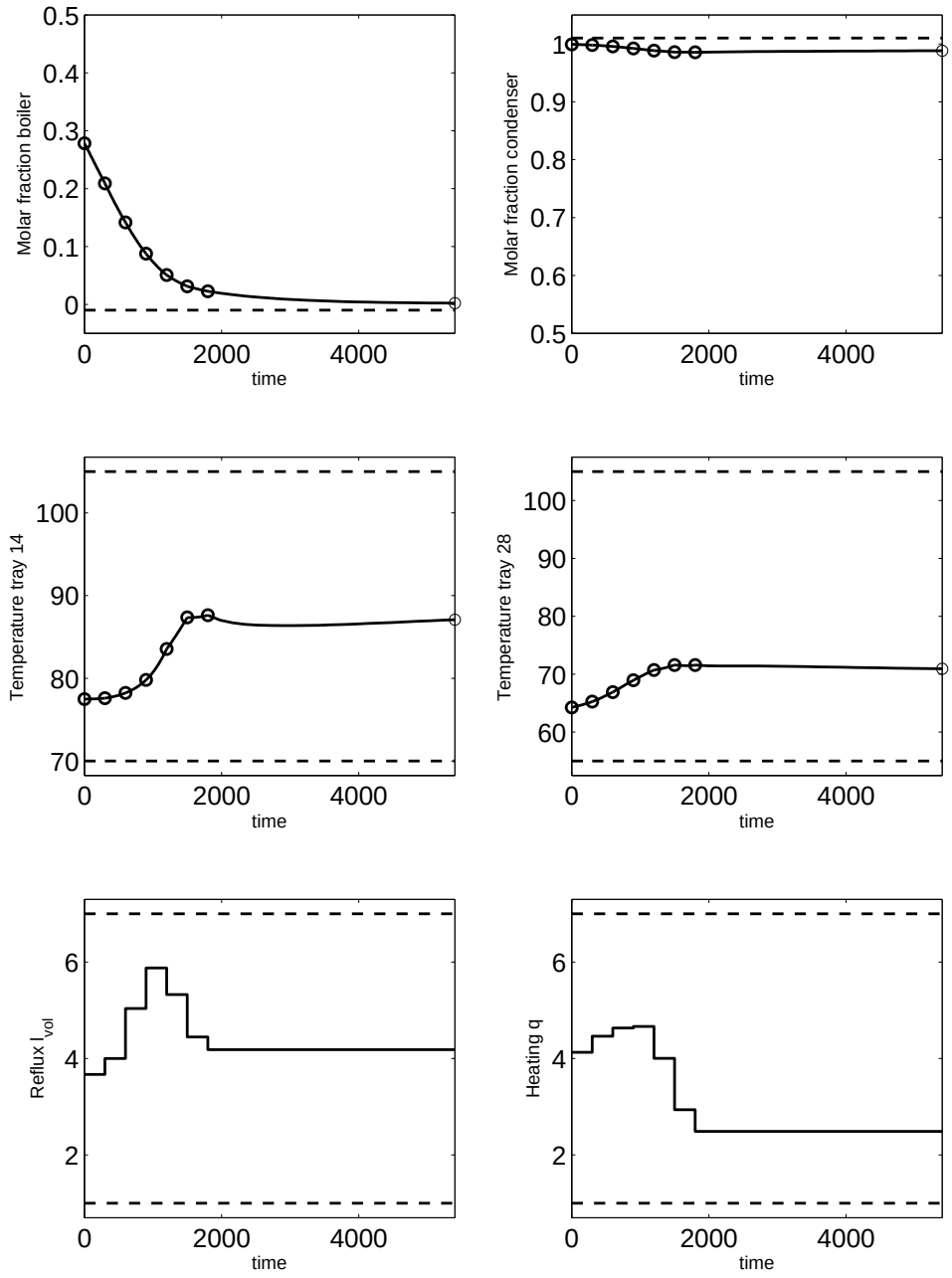

Figure 10.4: Numerical solution of the optimal control problem for the binary distillation column with $\rho = 30$ computed by `DynamicLiftOpt`. The upper row shows the concentrations of methanol in the boiler and the condenser, respectively, corresponding to the purities of the distillate and the bottom product. The middle row shows the temperatures on the "reference" trays 14 and 28. The lower row shows the computed optimal control moves for the volumetric reflux from the condenser to the top tray and for the heating in the reboiler.

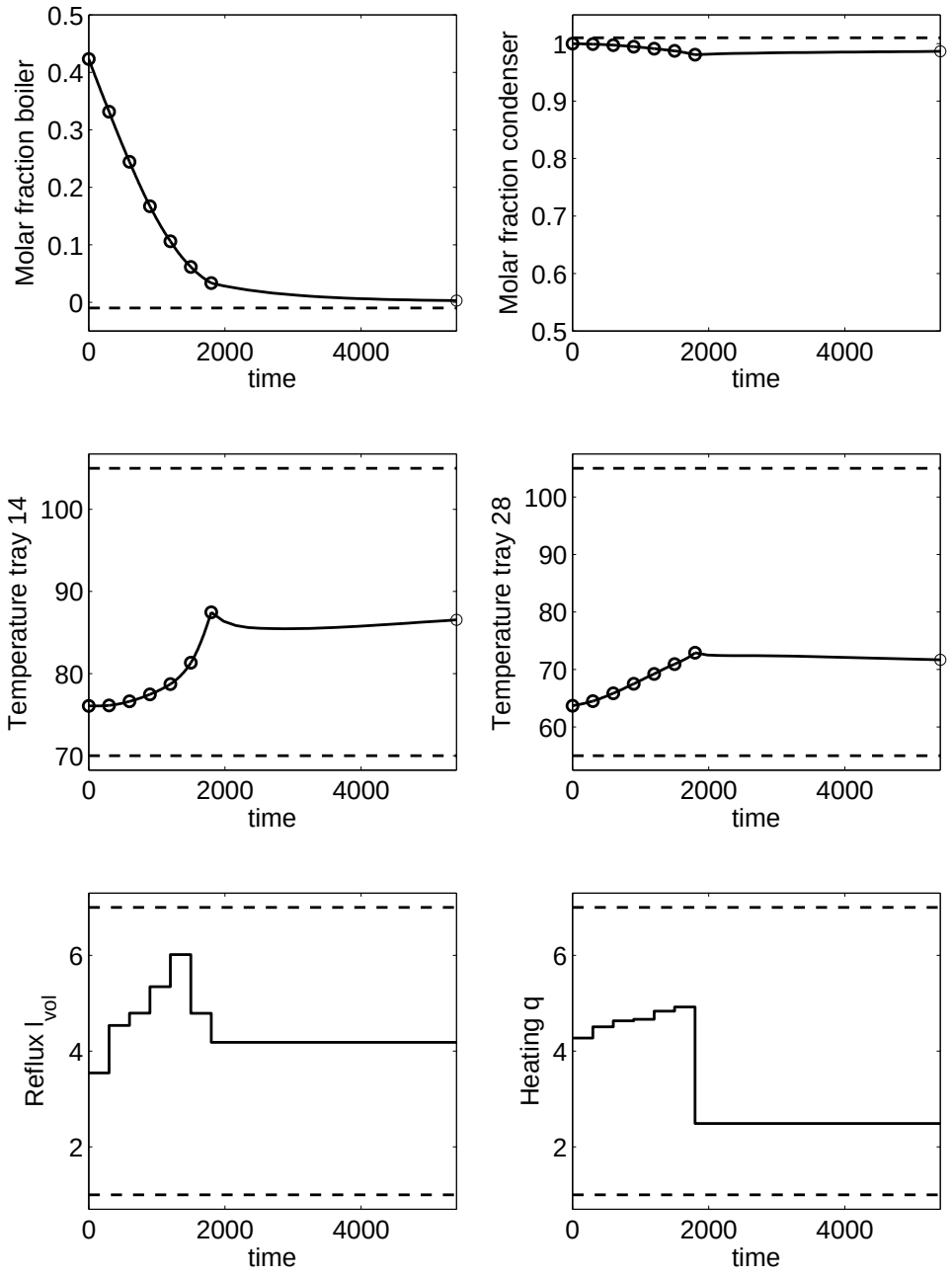

Figure 10.5: Numerical solution of the optimal control problem for the binary distillation column with $\rho = 50$ computed by `DynamicLiftOpt`. The upper row shows the concentrations of methanol in the boiler and the condenser, respectively, corresponding to the purities of the distillate and the bottom product. The middle row shows the temperatures on the "reference" trays 14 and 28. The lower row shows the computed optimal control moves for the volumetric reflux from the condenser to the top tray and for the heating in the reboiler.

11 Summary and Outlook

In the following we give a short summary of the ideas and results presented in this thesis. As there emerge a number of application scenarios and open topics for future research and for extensions of the methods developed in this thesis, we will discuss some of them here. This includes both theoretical aspects as well as aspects related to the numerical packages `DAESOL-II`/`SolvIND` [AKa, AKb] and `LiftOpt`/`DynamicLiftOpt` [AD10, Alb10, Alb] that have been developed in connection with this thesis.

11.1 Summary

This thesis contains several novelties and new ideas for efficient first and higher-order sensitivity generation of IVP solutions on the one hand and for (automatically) structure-exploiting NLP solvers on the other hand. The new approaches from these two fields are merged to obtain a new lifted exact-Hessian SQP method for the solution of Optimal Control Problems (OCPs) involving models consisting of Differential Algebraic Equations (DAEs) of index 1.

Regarding strategies related to numerical IVP solution and sensitivity generation strategies for IVP solutions, we have presented in this thesis

- a brief, heuristic analysis of the impact of the fact that in practice a predictor-corrector scheme is employed in our BDF-method on the stability of the method at the example of the Dahlquist equation. It shows that in the context of our monitor strategy this premature termination of the Newton-like iterations will usually lead to only a moderate loss of stability compared to the "true" implicit method (Section 5.3.3 on page 134).

- the development and implementation of new adjoint IND schemes for implicit Linear Multistep Methods (LMMs) at the example of Backward Differentiation Formula (BDF) methods. They allow the efficient computation of first order adjoint sensitivity information (Section 6.4.2 on page 161). We have shown with the help of several examples from the Bari IVP testset that our adjoint IND approach outperforms the state-of-the-art implementation of the widely used approach based on the solution of the adjoint variational ODE/DAE given by the SUNDIALS solver suite (Section 9.2 on page 232).

- a proposal for a simple a posteriori estimator for the global error of the IVP solution, based on the intermediate quantities of the adjoint IND sweep, that can also be used in the index 1 DAE case (Section 6.7.6 on page 188). We demonstrated that it performs well compared

to a similar approach based on the solution of the adjoint variational equation on a series of examples from the literature (Section 9.4 on page 245).

- the combination of the IND principle with the technique of univariate Taylor Coefficient (TC) propagation, which enabled us to derive the new IND-TC schemes. These allow for the first time the efficient computation of directional sensitivities of DAE-IVP solutions of arbitrary order. Sensitivities can be generated both by a pure forward mode as well as by a forward/adjoint mode that allows, e.g., the efficient computation of Hessian-type sensitivity information of the IVP solution (Section 6.5 on page 168).

- how the forward IND/IND-TC sweeps can be adapted to generate forward sensitivities that are (locally) error controlled. This is done by choosing the integration time grid based on the properties of both the nominal and the variational trajectories (Section 6.7.5 on page 185). We demonstrated on some simple test problems, how the modified algorithm correctly chose a common finer grid to fulfill the error tolerances for the forward sensitivities (Section 9.3 on page 242).

- the description of the propagation of higher-order directional sensitivities across switching events by TC propagation without the need to form complete update matrices (Section 6.7.8 on page 191). This makes the use of higher-order sensitivities in the optimization of problems involving switching events for the first time computationally feasible.

- the creation of the C++ package `SolvIND` [AKa, AKb] (with co-author Christian Kirches) as interface to integrator codes supporting IND, like, e.g., the newly created `DAESOL-II`. `SolvIND` can also be used as building block for dynamic optimization software and facilitates, e.g., semi-automatic ADOL-C support for the generation of first and higher-order model function derivatives as well as the transparent use of time and control transformations that are needed frequently in the solution of OCPs and dynamic optimization in general.

Concerning the development of structure-exploiting NLP solvers using the lifting idea, we have in this thesis

- explained how the multiple shooting idea can be generalized to the lifting idea for nonlinear functions in which cost function and constraints are explicitly computed via intermediate values. We showed that per iteration the complexity of the solution of the lifted problem is the same as for the unlifted problem. In particular, the complexity does not depend on the number of newly introduced intermediate "node" variables (Section 4.1 on page 78).

- demonstrated how by the use of an algorithmic trick the evaluation of a given user problem function can be lifted by the introduction of node values in a minimally invasive way. Furthermore, we have shown how the quantities needed for the forming of the reduced subproblem in each step of a lifted Newton method can be computed efficiently by derivatives of this modified function evaluation (Section 4.1.1 on page 81).

- used the lifting idea to derive automatically structure-exploiting Newton-type NLP solvers in form of a lifted constrained Gauss-Newton and lifted exact-Hessian SQP method. The

developed Gauss-Newton approach can be understood as a generalization of the ideas used in the multiple shooting codes of Schlöder (FIXFIT [Sch88]) and Schäfer (MSOPT [Sch05]), while the lifted exact-Hessian SQP based on adjoint derivative information has never been proposed before. Furthermore, we showed the equivalence of the iterates generated by our algorithm compared to the iterations of a full-space SQP operating in the combined space of original degrees of freedom and node variables (Section 4.2 on page 86).

- given a local convergence analysis and comparison for the lifted and unlifted Newton method on a tutorial problem, to gain a first understanding on why and in which cases the use of a lifted method might be useful or not (Sections 4.3 and 4.4).

- developed and implemented the C++ package `LiftOpt` [Alb]. It implements a lifted Newton, Gauss-Newton and SQP method and can be used to lift user given simulation or function evaluation codes in an easy way for the use in connection with the implemented lifted optimization routines.

- presented the successful application of `LiftOpt` and the proposed lifted Gauss-Newton and lifted SQP on a simple test example as well as for the solution of a large scale NLP. The latter resulted from a parameter estimation problem for a system of hyperbolic PDEs describing a shallow water equation model (Sections 8.1 to 8.3).

Finally, we have merged some of the new ideas from the two areas and

- derived an adjoint-based exact-Hessian SQP method in the framework of Bock's direct multiple shooting [BP84] for the solution of optimal control problems. Here we combined the lifted exact-Hessian SQP and the capability to compute higher-order forward/adjoint directional sensitivities using IND-TC with the partial reduction technique for DAEs of Leineweber [Lei99] (Section 7.1 on page 203).

- showed that the complexity in terms of run-time and memory demand of this algorithm is far superior to exact-Hessian alternatives based on the classical condensing approach. The inherent structure exploitation of our algorithm makes the computational treatment of general large scale systems with an exact-Hessian SQP multiple shooting method for the first time computationally feasible (Section 7.3 on page 216).

- implemented our algorithm in the C++ code `DynamicLiftOpt` and applied it successfully for the solution of an optimal control problem for a practical application from chemical engineering in form of a distillation column. Here we also demonstrated the superior performance of our approach compared to existing exact-Hessian multiple shooting based SQP methods and its competitiveness with update-based methods employing classical condensing (Section 10 on page 249).

11.2 Outlook and future work

The ideas and methods presented above give rise to a variety of application scenarios and extensions, which cannot be addressed completely in the framework of this thesis. We hence propose in the following some directions for interesting future research.

- Parallelization: For the treatment of very large scale systems and/or for further speedup the parallelization of our algorithms would be of interest. Concerning our integrator `DAESOL-II`, it would, for example, be interesting to use parallel linear equation solvers in the Newton-like method.

  Regarding the lifted optimization algorithms, it should be noted that here a parallelization approach similar to the one for classical direct multiple shooting [GB94], i.e., based on the decoupled shooting intervals (or the individual node function evaluations in the general lifting context, respectively), is not possible. This is because, as we have seen, the derivatives need to be propagated sequentially through the node functions (or the shooting intervals, respectively), as the propagated derivatives of the "earlier" node functions are needed as inputs for the propagation through the later ones.

  However, a parallelization could be based on the distribution of the computation of the different directional derivatives to different cores, as each forward/adjoint TC propagation can be performed independently. Considering that for large scale problems the costs are mainly given by the costs for the derivative computations, this might well lead to an efficient parallel approach. Especially, if we recall that the IND-TC schemes offer to reuse the integration grid and the iteration matrices from the nominal integration for all of these parallel sensitivity sweeps and hence no additional integration overhead is to be expected. A similar speedup can be expected if the TC propagation through the model functions could be parallelized, e.g., if a suitable AD tool is developed.

- Adaptive control of the integration accuracy and/or the integration grid: To improve the overall efficiency of optimization algorithms, it would be interesting to develop strategies that adapt the integration accuracy and/or the integration grid according to the actual needs of the optimization routine. The needed accuracy of the sensitivities could be estimated, e.g., based on the contractivity of the Newton-type iterations or the error in a quantity of interest (e.g., cost functional, merit function).

  At the moment, a simple integration grid adaptivity can be realized based on the capabilities of `DAESOL-II` by freezing the integration grid over several optimization iterations and recomputing it with a new tolerance when the estimated error becomes too large. Here the proposed a posteriori global error estimator based on adjoint IND can be used.

  A possible improvement on the integrator level would be the ability to choose the integration grid during the integration based on adjoint sensitivity information from a previous run to ensure that a global error bound on a given quantity of interest is satisfied. This might reduce the overall number of integration steps significantly compared to a strategy that is based on the accuracy of the system states only. While implementations of this idea, based on the

solution of the adjoint variational equations, already exist, these are not trivially transferable to IND-based LMM schemes and hence an interesting subject for further research.

- Nonlinear Model Predictive Control (NMPC): In this thesis we presented the application of our lifted methods in connection with the off-line solution of optimal control problems only. However, lifting offers interesting possibilities also in the context of NMPC and the real-time iteration scheme [Die02, DBS05], as initial value and parameter embedding can be achieved very easily by letting the values enter the problem via lifted node values. Furthermore, it will be interesting to analyze the performance of the lifted exact-Hessian SQP method in online optimization. It allows the treatment of problems using an exact-Hessian SQP method with much smaller sampling intervals than before, at the expense of a slightly prolonged feedback phase (not all QP quantities can be computed without the knowledge of the current system state). As a further extension, our method could also be employed as part of a hierarchical multi-level NMPC scheme [BDKS07, ABK$^+$09].

- Inexact Newton-type methods for optimization: The reduced gradient and constraint information that can be computed cheaply by a second-order forward/adjoint IND-TC sweep can be used in connection with inexact Newton-type methods that are based on update strategies for the Hessian and Jacobian [GW02, DWBK09, Wal08] to further decrease the cost of one NLP iteration and to render the number of directional derivatives needed in each step constant. An alternative approach to achieve this would be, at least for equality constrained problems, the use of lifting in connection with a Krylov-type optimization method, which would also allow the treatment of even larger systems, as it would lead to a completely matrix-free method. Finally, it would be interesting in this context to analyze the possibility of computing so-called high-rank BFGS updates of the Hessian matrix [BP84] more efficiently by using forward/adjoint IND-TC propagation.

- Globalization: It would be interesting, especially in the context of the lifted exact-Hessian SQP method, to develop alternative globalization approaches, e.g., based on natural level functions and the restrictive monotonicity test [BKS00].

- Optimization problems involving higher-order derivatives: The IND-TC schemes presented in this thesis can compute directional sensitivities of IVP solutions of arbitrary order (including the case of models with switching events). Hence the possible application scenarios for them are not limited to an exact-Hessian SQP method for optimal control. They can be used, e.g., to immediately speedup some existing algorithms for robust optimization [KKBS04, DBK06] and Optimal Experimental Design (OED) [Kör02, KKBS04]. Here at least second order sensitivity information is needed, which is nowadays normally computed by the solution of a second order forward variational equation. Furthermore, they allow for the first time the extension of these algorithms for the use of exact-Hessian information, which would correspond to the need of third order sensitivities. Additionally, it would be of course highly interesting to extend the lifted exact-Hessian SQP method presented in this thesis for the use in robust optimization and OED, or even robust OED.

Notation

A general convention throughout this thesis is the use of normal font for scalar quantities and the use of a bold font for vectors and matrices. This holds analogously for scalar and vector-valued functions. Unless indicated otherwise an n-dimensional vector shall always be understood as being equivalent to an $n \times 1$ matrix, i.e., all vectors are "column"-vectors. In the following let $\mathbf{x} \in \mathbb{R}^n$ be a vector and $\mathbf{H} \in \mathbb{R}^{n \times m}$ be a matrix. a and b shall stand for scalar values.

List of mathematical symbols

$\mathbf{H}^T$	Transposed of matrix $\mathbf{H}$
$\Re$	Real part of a complex function or value
$\Im$	Imaginary part of a complex function or value
$a \ll b$	a is much smaller than b
$\mathbf{f_x}$	Short for the Jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$
$\mathbb{I}_n$	Identity matrix or identity operator in $\mathbb{R}^n$
$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of natural numbers (including 0)
$\mathbb{N}^+$	Set of natural numbers (excluding 0)
$\binom{a}{b}$	Binomial coefficients
$\mathcal{C}^0(X, Y)$	Set of functions between spaces X and Y that are (at least) continuous on some open domain of X
$\mathcal{C}^k(X, Y)$	Set of functions between spaces X and Y that are (at least) k-times ($k \in \mathbb{N}^+$) continuously differentiable on some open domain of X
$\mathbf{e_i^n}$	i-th Cartesian unit/basis vector in $\mathbb{R}^n$
$\mathcal{U}_\varepsilon(\mathbf{x})$	Neighborhood around point $\mathbf{x}$ with radius ε.
$u \mathrel{+}= v$	Add-assign operation: $u = u + v$

List of acronyms

AD	Automatic Differentiation
BDF	Backward Differentiation Formula
BFGS	Broyden-Fletcher-Goldfarb-Shanno
DAE	Differential Algebraic Equation
DFP	Davidon-Fletcher-Powell
IND	Internal Numerical Differentiation
IVP	Initial Value Problem
KKT	Karush-Kuhn-Tucker
LICQ	Linear Independence Constraint Qualification
LMM	Linear Multistep Method
NLP	Nonlinear Program
NLSQ	Nonlinear Least-Squares
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
OED	Optimal Experimental Design
PDE	Partial Differential Equation
PRSQP	Partially Reduced Sequential Quadratic Programming
L-PRSQP	Lifted Partially Reduced Sequential Quadratic Programming
QP	Quadratic Program
SQP	Sequential Quadratic Programming
s.t.	such that
TC	Taylor Coefficient
w.r.t.	with respect to

List of Figures

1.1 Illustration of the direct single shooting discretization 15
1.2 Illustration of the direct multiple shooting discretization 18

2.1 Error comparison one-sided vs. central finite differences 24
2.2 Error comparison one-sided finite differences vs. complex step approach 25
2.3 Example for a computational graph of a function evaluation 28

4.1 Comparison of iterates of lifted and non-lifted Newton method on the root finding example 102
4.2 Error and local convergence comparison for lifted and unlifted Newton method on the root finding example 103

5.1 Stability regions of explicit and implicit Euler methods 116
5.2 Stability regions for implicit BDF methods 121
5.3 Stability regions for predictor-corrector BDF methods of order 1 136
5.4 Stability regions for predictor-corrector BDF methods of order 2 137
5.5 Stability regions for predictor-corrector BDF methods of order 5 138

6.1 Dependencies in a BDF step 155
6.2 Comparison of the intermediate values of the adjoint IND scheme and the solution of the adjoint variational equation 167

7.1 Structure of the (full-space) KKT matrix in the direct multiple shooting approach 206
7.2 Structure of the partially-reduced KKT matrix in the direct multiple shooting approach after elimination of the algebraic variables 208
7.3 Structure of the condensed KKT matrix in the direct multiple shooting approach . 209

8.1 Numerical simulation of the shallow water equation model 223

9.1 Sparsity structure of the iteration matrix in the SMB example 227
9.2 Eigenvalues of the model function Jacobian in the SMB example 228
9.3 Results SMB example 1: Integration, replay and comparison 230
9.4 Results SMB example 2: IND sweeps 231
9.5 Comparison of `DAESOL-II` and CVODES on the HIRES example 235
9.6 Comparison of `DAESOL-II` and CVODES on the Pleiades example 238
9.7 Comparison of `DAESOL-II` and IDAS on the chemical Akzo Nobel example 241
9.8 Error control for forward sensitivities on the Dahlquist example 243

10.1 Flowsheet of the binary distillation column 250
10.2 Molar flows in the binary distillation column 251
10.3 Numerical solution of the optimal control problem for the binary distillation column with $\rho = 10$ 258
10.4 Numerical solution of the optimal control problem for the binary distillation column with $\rho = 30$ 259
10.5 Numerical solution of the optimal control problem for the binary distillation column with $\rho = 50$ 260

List of Tables

2.1 General evaluation procedure 27
2.2 Example a for general evaluation procedure 28
2.3 Example for the numerical function evaluation based on an elemental representation 29
2.4 General AD tangent procedure 30
2.5 General first order AD reverse sweep 33
2.6 Example of an abstract AD first order forward and reverse sweep 34
2.7 Example of an actual numerical AD first order forward and reverse sweep 35
2.8 TC forward propagation rules for elemental functions 42

4.1 Cost analysis for lifted, unlifted and full-space Newton method 101

6.1 Cost comparison for different operations related to IVP solution and IND(-TC) . 178
6.2 Overview on tape requirements for different IND operations 180

7.1 Complexity of computational costs for an exact-Hessian SQP based on classical condensing 217
7.2 Complexity of computational costs for an exact-Hessian SQP based on the lifting idea 217
7.3 Comparison of memory demand for an exact-Hessian SQP based on classical condensing and on the lifting idea 218

8.1 Results of the lifted/unlifted Gauss-Newton approach for the solution of a simple optimal control problem 220
8.2 Results of the lifted/unlifted SQP method for the solution of a simple optimal control problem 221
8.3 Results for the solution of the parameter estimation problem for the shallow water equation 224

9.1 Time comparison of different integrator operations in the SMB example. 229
9.2 Accuracy and computational costs of the error control for forward sensitivities in the harmonic oscillator example 244
9.3 Effectivity indices of the global error estimation approach 248

10.1 Comparison of the optimal solutions computed for the binary distillation column example 257
10.2 Computational statistics for the solution of the binary distillation column example 257

List of Algorithms

4.1 Function with output of intermediate variables 79
4.2 Residual function 79
4.3 Modified function 81
4.4 The Lifted Newton Method 83
4.5 Original function evaluation of unlifted function 84
4.6 Function evaluation modified for use in lifted algorithms 84
4.7 Node function 85
6.1 First order iterative forward IND scheme (simultaneous) 158
6.2 First order iterative forward IND scheme (deferred) 159
6.3 First order direct forward IND scheme (simultaneous) 160
6.4 First order direct forward IND scheme (deferred) 161
6.5 First order iterative adjoint IND scheme 163
6.6 First order direct adjoint IND scheme 164
6.7 Arbitrary order forward IND-TC scheme (simultaneous) 172
6.8 Arbitrary order forward/adjoint IND-TC scheme 175
6.9 Adjoint sensitivity injection (forward/adjoint IND-TC version) 184
6.10 Error control for forward sensitivities 187
6.11 Arbitrary order forward IND-TC scheme including switches 202
6.12 Forward TC propagation across switching event 202
7.1 L-PRSQP: Residual and function evaluation 210

Acknowledgments

This thesis was prepared at the Faculty of Mathematics and Computer Sciences of the University of Heidelberg, Germany. I like to thank my advisor Prof. Dr. Dr. h.c. Hans Georg Bock of the Simulation and Optimization Team at the Interdisciplinary Center for Scientific Computing (IWR) for his support of my research, for providing an excellent and motivating work environment and for contributing the initial ideas and many inspiring discussions to my work. Many thanks also go to Dr. Johannes Schlöder from the Simulation and Optimization Team for many interesting and helpful discussions about various topics.
In addition, I wish to thank Prof. Dr. Moritz Diehl from the Optimization in Engineering Center (OPTEC) of the K.U. Leuven, Belgium, for co-supervising my thesis. I am grateful to him for many helpful and inspiring suggestions and the possibility to spend several fruitful and pleasant months of collaboration at the OPTEC.

Furthermore, I like to thank my colleagues Dörte Beigel, Christian Kirches, Simon Lenz and Andreas Schmidt for proofreading parts of this thesis. To them, as well as to many other colleagues from the Simulation and Optimization Team, the IWR and the OPTEC, among them Christian Hoffmann, Peter Kühl, Andreas Potschka, Dr. Sebastian Sager and Leonard Wirsching, go my thanks for many valuable discussions related to research as well as to off-work topics. I am especially grateful to Christian Kirches for his support in the creation of the `SolvIND` integrator suite. I also want to thank our secretary Margret Rothfuß and our system administrator Thomas Klöpfer for their help and their contributions to the friendly atmosphere in the group.

Financial support by the University of Heidelberg and by the German Research Foundation (DFG) in the framework of the International Graduiertenkolleg IGK 710: "Complex processes: Modeling, Simulation and Optimization" and the "Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences" as well as via the Priority Program 1253: "Optimization with Partial Differential Equations" is gratefully acknowledged.

Finally, I want to thank my parents for their continuous support in all its forms and last, but by no means least, Carole for her love, support, encouragement, patience and understanding.

Bibliography

[AB08] J. Albersmeyer and H.G. Bock. Sensitivity Generation in an Adaptive BDF-Method. In Hans Georg Bock, E. Kostina, X.H. Phu, and R. Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 6–10, 2006, Hanoi, Vietnam*, pages 15–24. Springer Verlag Berlin Heidelberg New York, 2008.

[ABK+09] J. Albersmeyer, D. Beigel, C. Kirches, L. Wirsching, H.G. Bock, and J.P. Schlöder. Fast nonlinear model predictive control with an application in automotive engineering. In L. Magni, D.M. Raimondo, and F. Allgöwer, editors, *Lecture Notes in Control and Information Sciences*, volume 384, pages 471–480. Springer Verlag Berlin Heidelberg, 2009.

[ABQ+99] F. Allgöwer, T.A. Badgwell, J.S. Qin, J.B. Rawlings, and S.J. Wright. Nonlinear predictive control and moving horizon estimation – An introductory overview. In P.M. Frank, editor, *Advances in Control, Highlights of ECC'99*, pages 391–449. Springer, 1999.

[AD10] J. Albersmeyer and M. Diehl. The Lifted Newton method and its application in optimization. *SIAM Journal on Optimization*, 20(3):1655–1684, 2010.

[AKa] J. Albersmeyer and C. Kirches. The `SolvIND` user manual. Technical report, IWR, University of Heidelberg, Germany (in preparation).

[AKb] J. Albersmeyer and C. Kirches. The `SolvIND` webpage. http://www.solvind.org.

[Alb] J. Albersmeyer. The `LiftOpt` webpage. http://www.liftopt.org.

[Alb05] J. Albersmeyer. Effiziente Ableitungserzeugung in einem adaptiven BDF-Verfahren. Diploma thesis, Universität Heidelberg, 2005.

[Alb10] J. Albersmeyer. The `LiftOpt` user manual. Technical report, IWR, University of Heidelberg, Germany, 2010.

[ALW07] W. Auzinger, H. Lehner, and E. Weinmüller. Defect-based a posteriori error estimation for index-1 DAEs. Preprint ASC 20-2007, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2007.

[BA83] F. Bashforth and J.C. Adams. *An Attempt To Test The Theories Of Capillary Action By Comparing The Theoretical And Measured Forms Of Drops Of Fluid*. Kessinger Publishing, 1883.

[Bär83] V. Bär. Ein Kollokationsverfahren zur numerischen Lösung allgemeiner Mehrpunktrandwertaufgaben mit Schalt– und Sprungbedingungen mit Anwendungen in der optimalen Steuerung und der Parameteridentifizierung. Diploma thesis, Rheinische Friedrich–Wilhelms–Universität zu Bonn, 1983.

[Bar06] R. Barrio. Sensitivity analysis of ODEs/DAEs using the Taylor series method. *SIAM Journal on Scientific Computing*, 27(6):1929–1947, 2006.

[Bau99] I. Bauer. *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik.* PhD thesis, Universität Heidelberg, 1999.

[BBB+01] T. Binder, L. Blank, H.G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J.P. Schlöder, and O.v. Stryk. Introduction to model based optimization of chemical processes on moving horizons. In M. Grötschel, S.O. Krumke, and J. Rambau, editors, *Online Optimization of Large Scale Systems: State of the Art*, pages 295–340. Springer, 2001.

[BBKS00] I. Bauer, H.G. Bock, S. Körkel, and J.P. Schlöder. Numerical methods for optimum experimental design in DAE systems. *J. Comput. Appl. Math.*, 120(1-2):1–15, 2000.

[BBL+02] C.H. Bischof, H.M. Bücker, B. Lang, A. Rasch, and A. Vehreschild. Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In *Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002)*, pages 65–72, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[BCG93] C.H. Bischof, G.F. Corliss, and A. Griewank. Structured second- and higher-order derivatives through univariate Taylor series. *Optimization Methods and Software*, pages 211–232, 1993.

[BCKM96] C.H. Bischof, A. Carle, P. Khademi, and A. Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3:18–32, 1996.

[BCP96] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations.* SIAM, Philadelphia, 1996. Classics in Applied Mathematics 14.

[BDK04] H.G. Bock, M. Diehl, and E. Kostina. SQP methods with inexact Jacobians for inequality constrained optimization. IWR-Preprint 04-XX, Universität Heidelberg, Heidelberg, 2004.

[BDKS07] H.G. Bock, M. Diehl, E.A. Kostina, and J.P. Schlöder. Constrained Optimal Feedback Control for DAE. In L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes,

and B. van Bloemen Waanders, editors, *Real-Time PDE-Constrained Optimization*, chapter 1, pages 3–24. SIAM, 2007.

[Bel] B.M. Bell. The CppAD homepage. http://www.coin-or.org/CppAD/.

[Bel57] R.E. Bellman. *Dynamic Programming*. University Press, Princeton, N.J., 6th edition, 1957. ISBN 0-486-42809-5 (paperback).

[Ber02] M. Berz. COSY INFINITY Version 8.1 — User's Guide and Reference Manual. Department of Physics and Astronomy MSUHEP-20704, Department of Physics and Astronomy, Michigan State University, 2002.

[Ber05] D.P. Bertsekas. *Dynamic programming and optimal control, Volume 1*. Athena Scientific, Belmont, Mass., 3. ed. edition, 2005.

[Ber07] D.P. Bertsekas. *Dynamic programming and optimal control, Volume 2*. Athena Scientific, Belmont, Mass., 3. ed. edition, 2007.

[BES88] H.G. Bock, E. Eich, and J.P. Schlöder. Numerical solution of constrained least squares boundary value problems in differential-algebraic equations. In K. Strehmel, editor, *Numerical Treatment of Differential Equations*. Teubner, Leipzig, 1988.

[BGHBW03] L.T. Biegler, O. Ghattas, M. Heinkenschloss, and B.v. Bloemen Waanders. *Large-Scale PDE-Constrained Optimization*, volume 30 of *Lecture Notes in Computational Science and Engineering*. Springer, Heidelberg, 2003.

[BH75] A.E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Wiley, New York, 1975.

[Bie84] L.T. Biegler. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Computers and Chemical Engineering*, 8:243–248, 1984.

[Bis] C.H. Bischof. The AutoDiff homepage. `http://www.autodiff.org`.

[BKBC96] C.H. Bischof, P.M. Khademi, A. Bouaricha, and A. Carle. Computation of gradients and Jacobians by dynamic exploitation of sparsity in automatic differentiation. *Optimization Methods and Software*, 7(1):1–39, 1996.

[BKS00] H.G. Bock, E.A. Kostina, and J.P. Schlöder. On the role of natural level functions to achieve global convergence for damped Newton methods. In M.J.D. Powell and S. Scholtes, editors, *System Modelling and Optimization. Methods, Theory and Applications*. Kluwer, 2000.

[Ble86] G. Bleser. Eine effiziente Ordnungs– und Schrittweitensteuerung unter Verwendung von Fehlerformeln für variable Gitter und ihre Realisierung in Mehrschrittverfahren vom BDF–Typ. Diploma thesis, Universität Bonn, 1986.

[BNPvS91] R. Bulirsch, E. Nerz, H.J. Pesch, and O. von Stryk. Combining direct and indirect methods in nonlinear optimal control: Range maximization of a hang glider. Technical Report 313, Technische Universität München, 1991.

[Boc78a] H.G. Bock. Numerical solution of nonlinear multipoint boundary value problems with applications to optimal control. *Zeitschrift für Angewandte Mathematik und Mechanik*, 58:407, 1978.

[Boc78b] H.G. Bock. *Numerische Berechnung zustandsbeschränkter optimaler Steuerungen mit der Mehrzielmethode*. Carl-Cranz-Gesellschaft, Heidelberg, 1978.

[Boc81] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K.H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, volume 18 of *Springer Series in Chemical Physics*, pages 102–125. Springer, Heidelberg, 1981.

[Boc83] H.G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, pages 95–121. Birkhäuser, Boston, 1983.

[Boc87] H.G. Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, volume 183 of *Bonner Mathematische Schriften*. Universität Bonn, Bonn, 1987.

[BP84] H.G. Bock and K.J. Plitt. A Multiple Shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress*, pages 243–247, Budapest, 1984. Pergamon Press. Available at http://www.iwr.uni-heidelberg.de/groups/agbock/FILES/Bock1984.pdf.

[BP04] U. Brandt-Pollmann. *Numerical solution of optimal control problems with implicitly defined discontinuities with applications in engineering*. PhD thesis, Universität Heidelberg, 2004.

[BR01] R. Becker and R. Rannacher. An optimal control approach to error estimation and mesh adaptation in finite element methods. *Acta Numerica 2000*, pages 1–101, 2001.

[BRM97] C.H. Bischof, L. Roh, and A. Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. *Software–Practice and Experience*, 27(12):1427–1456, 1997.

[BS66] R. Bulirsch and J. Stoer. Numerical treatment of ordinary differential equations by extrapolation methods. *Numerische Mathematik*, 8:1–13, 1966.

[BS81] H.G. Bock and J.P. Schlöder. Numerical solution of retarded differential equations with state-dependent time lags. *Zeitschrift für Angewandte Mathematik und Mechanik*, 61:269, 1981.

[BS96] C. Bendtsen and O. Stauning. FADBAD, a flexible C++ package for automatic differentiation. Technical Report IMM–REP–1996–17, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1996.

[BSS94] H.G. Bock, J.P. Schlöder, and V.H. Schulz. Numerik großer Differentiell-Algebraischer Gleichungen – Simulation und Optimierung. In H. Schuler, editor, *Prozeßsimulation*, pages 35–80. VCH Verlagsgesellschaft mbH, Weinheim, 1994.

[Bul71] R. Bulirsch. Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung. Technical report, Carl-Cranz-Gesellschaft, Oberpfaffenhofen, 1971.

[CB89] J.E. Cuthrell and L.T. Biegler. Simultaneous optimization and solution methods for batch reactor profiles. *Computers and Chemical Engineering*, 13(1/2):49–62, 1989.

[CC86] T.F. Coleman and J. Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM J. Alg. Disc. Meth.*, 7(2):221–235, 1986.

[CH52] C.F. Curtiss and J.O. Hirschfelder. Integration of stiff equations. *Proc. Nat. Acad. Sci*, 38:235–243, 1952.

[Cha62] P.E. Chase. Stability properties of predictor-corrector methods for ordinary differential equations. *J. ACM*, 9(4):457–468, 1962.

[CK65] R.L. Crane and R.W. Klopfenstein. A predictor-corrector algorithm with an increased range of absolute stability. *J. ACM*, 12(2):227–241, 1965.

[CL84] M. Crouzeix and F.J. Lisbona. The convergence of variable-stepsize, variable-formula, multistep methods. *SIAM Journal on Numerical Analysis*, 21(3):512–534, 1984.

[CLM87] M. Calvo, F.J. Lisbona, and J. Montijano. On the stability of variable Nordsieck BDF methods. *SIAM Journal on Numerical Analysis*, 24:844–854, 1987.

[CLPP82] R. Chamberlain, C. Lemaréchal, H. C. Pedersen, and M. J. D. Powell. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming*, 16:1–17, 1982.

[CLPS03] Y. Cao, S. Li, L.R. Petzold, and R. Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.

[CM84] T.F. Coleman and J.J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20:187–209, 1984.

[Cop] Wave model and simulation code of D. Copsey. http://www.dcopsey.i12.com/.

[CP04] Y. Cao and L. Petzold. A posteriori error estimation and global error control for ordinary differential equations by the adjoint method. *SIAM J.Sci. Comput.*, 26:359–374, 2004.

[CPR74] A.R. Curtis, M.J.D. Powell, and J.K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.

[Cry72] C.W. Cryer. On the instability of high order backward-difference multistep methods. *BIT*, 12:17–25, 1972.

[CS84] H.-S. Chen and M.A. Stadtherr. Enhancements of the han-powell method for successive quadratic programming. *Computers and Chemical Engineering*, 8:229–234, 1984.

[CS85] M. Caracotsios and W.E. Stewart. Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations. *Computers and Chemical Engineering.*, 9:359–365, 1985.

[CU09] I. Charpentier and J. Utke. Rapsodia: User manual. Technical report, Argonne National Laboratory, 2009.

[CV96] T.F. Coleman and A. Verma. Structure and efficient Jacobian calculation. In M. Berz, Christian Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 149 – 159. SIAM, 1996.

[CV98] T.F. Coleman and A. Verma. ADMAT: An automatic differentiation toolbox for MATLAB. Technical report, Computer Science Department, Cornell University, 1998.

[Dah56] G. Dahlquist. Convergence and stability in numerical integration of ordinary differential equations. *Math. Scand.*, 4:33–53, 1956.

[Dah63] G. Dahlquist. A special stability problem for linear multistep methods. *BIT*, 3:27–43, 1963.

[Dav04] T.A. Davis. Algorithm 832: UMFPACK - an unsymmetric-pattern multifrontal method with a column pre-ordering strategy. *ACM Trans. Math. Software*, 30:196–199, 2004.

[DB83] P. Deuflhard and G Bader. A semi-implicit mid-point rule for stiff systems of ordinary differential equations. *Numerische Mathematik*, 41:373–398, 1983.

[DBDW06] M. Diehl, H.G. Bock, H. Diedam, and P.-B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. *LNCIS*, 2006. (in print).

[DBK06] M. Diehl, H.G. Bock, and E. Kostina. An approximation technique for robust nonlinear optimization. *Mathematical Programming*, 107:213–230, 2006.

[DBS05] M. Diehl, H.G. Bock, and J.P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005.

[Deu74] P. Deuflhard. A Modified Newton Method for the Solution of Ill-conditioned Systems of Nonlinear Equations with Applications to Multiple Shooting. *Numerische Mathematik*, 22:289–311, 1974.

[Die02] M. Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*, volume 920 of *Fortschritt-Berichte VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik*. VDI Verlag, Düsseldorf, 2002. Download also at: http://www.ub.uni-heidelberg.de/archiv/1659/.

[DLS01] M. Diehl, D.B. Leineweber, and A.A.S. Schäfer. MUSCOD-II Users' Manual. IWR-Preprint 2001-25, Universität Heidelberg, 2001.

[DLS+02] M. Diehl, D.B. Leineweber, A.A.S. Schäfer, H.G. Bock, and J.P. Schlöder. Optimization of multiple-fraction batch distillation with recycled waste cuts. *AIChE Journal*, 48(12):2869–2874, 2002.

[DP80] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19 – 26, 1980.

[DWBK09] M. Diehl, A. Walther, H.G. Bock, and E. Kostina. An adjoint-based SQP algorithm with quasi-Newton Jacobian updates for inequality constrained optimization. *Optimization Methods and Software*, 2009.

[Ehr05] T. Ehrlenbach. Effiziente Unstetigkeitsbehandlung bei der Integration linear impliziter differentiell-algebraischer Gleichungssysteme vom Index 1. Diploma thesis, Universität Heidelberg, 2005.

[Eic87] E. Eich. Numerische Behandlung semi-expliziter differentiell-algebraischer Gleichungssysteme vom Index I mit BDF Verfahren. Diploma thesis, Universität Bonn, 1987.

[Eic91] E. Eich. *Projizierende Mehrschrittverfahren zur numerischen Lösung von Bewegungsgleichungen technischer Mehrkörpersysteme mit Zwangsbedingungen und Unstetigkeiten*. PhD thesis, University of Augsburg, 1991.

[Eic93] E. Eich. Convergence results for a coordinate projection method applied to mechanical systems with algebraic constraints. *SIAM Journal on Numerical Analysis*, 30:1467–1482, 1993.

[Enk84] K. Enke. Ein effizientes Rückwärtsdifferentiationsverfahren mit variabler Schrittweite und Ordnung zur Lösung steifer Anfangswertaufgaben. Diploma thesis, Universität Bonn, 1984.

[Est95] D. Estep. A posteriori error bounds and global error control for approximation of ordinary differential equations. *SIAM Journal on Numerical Analysis*, 32(1):1–48, 1995.

[FAC+05] C.A. Floudas, I.G. Akrotirianakis, S. Caratzoulas, C.A. Meyer, and J. Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers and Chemical Engineering*, 29(6):1185–1202, 2005.

[FdB57] Cav. F. Faà di Bruno. Note sur une nouvelle formule du calcul différentiel. *Quaterly J. Pure Appl. Math.*, 1:359–360, 1857.

[Feh69] E. Fehlberg. Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle. *Computing*, 4:93–106, 1969.

[Feh70] E. Fehlberg. Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. *Computing*, 6:61–71, 1970.

[Fer07] H.J. Ferreau. `qpOASES` – an open-source implementation of the online active set strategy for fast model predictive control. In *Proceedings of the Workshop on Nonlinear Model Based Control – Software and Applications, Loughborough*, 2007.

[FL02] Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239, 2002.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester, 2nd edition, 1987. ISBN 0-471-49463-1 (paperback).

[FM90] A.V. Fiacco and G.P. McCormick. Nonlinear programming: Sequential unconstrained minimization techniques. *SIAM publications*, 1990.

[FMT02] R. Franke, M. Meyer, and P. Terwiesch. Optimal control of the driving of trains. *Automatisierungstechnik*, 50(12):606–614, 2002.

[FTB97] W.F. Feehery, J.E. Tolsma, and P.I. Barton. Efficient sensitivity analysis of large-scale differential-algebraic systems. *Applied Numerical Mathematics*, 25:41–54, 1997.

[GB94] J.V. Gallitzendörfer and H.G. Bock. Parallel algorithms for optimization boundary value problems in DAE. In H. Langendörfer, editor, *Praxisorientierte Parallelverarbeitung*. Hanser, München, 1994.

[Gea71] C.W. Gear. *Numerical initial value problems in ordinary differential equations*. Prentice Hall, Englewood Cliffs, NJ, 1971.

[Gea74] C.W. Gear. Asymptotic estimation of errors and derivatives for the numerical solution of ordinary differential equations. *IFIP 74*, pages 447–451, 1974.

[Gea80] C.W. Gear. Runge-Kutta starters for multistep methods. *ACM Trans. Math. Softw.*, 6:263–279, 1980.

[Gea88] C.W. Gear. Differential-algebraic equation index transformations. *SIAM J. Sci. Stat. Comp.*, 9:39–47, 1988.

[GJL+00] P.E. Gill, L.O. Jay, M.W. Leonard, L.R. Petzold, and V. Sharma. An SQP method for the optimal control of large-scale dynamical systems. *Journal of Computational and Applied Mathematics*, 120(1-2):197 – 213, 2000.

[GJU96] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[GMS95] P.E. Gill, W. Murray, and M.A. Saunders. *User's Guide For QPOPT 1.0: A Fortran Package For Quadratic Programming*, 1995.

[GO84] C.W. Gear and O. Osterby. Solving ordinary differential equations with discontinuities. *ACM Trans. on Math. Soft.*, 10(1):23–44, 1984.

[GR91] A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. In *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, 1991.

[GR05] A. Griewank and J. Riehme. Second order derivatives with ADTAGEO. Slides at AD-Workshop Nice, 2005.

[Gri83] R.D. Grigorieff. Stability of multistep-methods on variable grids. *Numerische Mathematik*, 42:359–377, 1983.

[Gri00] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.

[GT74] C.W. Gear and K.W. Tu. The effect of variable mesh size on the stability of multistep methods. *SIAM Journal on Numerical Analysis*, 11:1025–1043, 1974.

[Gu95] T. Gu. Mathematical modelling and scale up of liquid chromatography. *Springer Verlag, New York*, 1995.

[GUG96] U. Geitner, J. Utke, and A. Griewank. Automatic computation of sparse Jacobians by applying the method of Newsam and Ramsdell. In M. Berz et al. eds., editor, *Computational Differentiation, Proceedings of the Second International Workshop*, pages 161–172, Philadelphia, 1996. SIAM.

[GUW00] A. Griewank, J. Utke, and A. Walther. Evaluating higher derivative tensors by forward propagation of univariate Taylor series. *Mathematics of Computation*, pages 1117–1130, 2000.

[GvL96] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[GW74] C.W. Gear and D.S. Watanabe. Stability and convergence of variable order multistep methods. *SIAM Journal on Numerical Analysis*, 11(5):1044–1058, 1974.

[GW02] A. Griewank and A. Walther. On constrained optimization by adjoint based quasi-Newton methods. *Optimization Methods and Software*, 17:869 – 889, 2002.

[Han76] S.P. Han. Superlinearly convergent variable-metric algorithms for general nonlinear programming problems. *Mathematical Programming*, 11:263–282, 1976.

[Han77] S.P. Han. A globally convergent method for nonlinear programming. *JOTA*, 22:297–310, 1977.

[HBG+05] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, September 2005.

[HFH+07] M. Hucka, A.M. Finney, S. Hoops, S.M. Keating, and N. Le Novère. *Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions*, 2007.

[HLR89] E. Hairer, C. Lubich, and M. Roche. *The numerical solution of differential-algebraic systems by Runge-Kutta methods*. Number 1409 in Lecture Notes in Mathematics. Springer, Heidelberg, 1989.

[HNN02] P.D. Hovland, U. Naumann, and B. Norris. An XML-based platform for semantic transformation of numerical programs. In M. Hamza, editor, *Software Engineering and Applications*, pages 530–538, Anaheim, CA, 2002. ACTA Press.

[HNW93] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I*, volume 8 of *Springer Series in Computational Mathematics*. Springer, Berlin, 2nd edition, 1993.

[HP04] L. Hascoët and V. Pascual. TAPENADE 2.1 user's guide. Rapport technique 300, INRIA, Sophia Antipolis, 2004.

[HR71] G.A. Hicks and W.H. Ray. Approximation methods for optimal control systems. *Can. J. Chem. Engng.*, 49:522–528, 1971.

[HSV95] R.F. Hartl, S.P. Sethi, and R.G. Vickson. A survey of the maximum principles for optimal control problems with state constraints. *SIAM Review*, 37:181–218, 1995.

[HV01] M. Heinkenschloss and L.N. Vicente. Analysis of Inexact Trust-Region SQP algorithms. *SIAM Journal on Optimization*, 12(2):283–302, 2001.

[HW96] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II*, volume 14 of *Springer Series in Computational Mathematics*. Springer, Berlin, 2nd edition, 1996.

[HW08] J.S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods*, volume 54 of *Texts in Applied Mathematics*. Springer New York, 2008.

[Inc] Tomlab Optimization Inc. The TOMSYM homepage. http://tomsym.com.

[Iri91] M. Iri. History of automatic differentiation and rounding error estimation. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 1–16. SIAM, Philadelphia, PA, 1991.

[Joh88] C. Johnson. Error estimates and adaptive time-step control for a class of one-step methods for stiff ordinary differential equations. *SIAM Journal on Numerical Analysis*, 25(4):908–926, 1988.

[Joh02] W.P. Johnson. The curious history of Faà di Bruno's formula. *American Mathematical Monthly*, 109:217–234, 2002.

[JS97] H. Jaeger and E.W. Sachs. Global convergence of inexact reduced SQP methods. *Optimization Methods and Software*, 7:83–110, 1997.

[JV98] C.P. Jeannerod and J. Visconti. Global error estimation for index-1 and -2 DAEs. *Numerical algorithms*, 19:111–126, 1998.

[Kar39] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[KB06] S. Kameswaran and L.T. Biegler. Simultaneous dynamic optimization strategies: Recent advances and challenges. *Computers and Chemical Engineering*, 30:1560–1575, 2006.

[KB08] S. Kameswaran and L.T. Biegler. Advantages of nonlinear-programming-based methodologies for inequality path-constrained optimal control problems - a numerical study. *SIAM Journal on Scientific Computing*, 30:957–981, 2008.

[KBS93] J. Kallrath, H.G. Bock, and J.P. Schlöder. Least squares parameter estimation in chaotic differential equations. *Celestial Mechanics and Dynamical Astronomy*, 56:353–371, 1993.

[KBSS11] C. Kirches, H.G. Bock, J.P. Schlöder, and S. Sager. Block structured quadratic programming for the direct multiple shooting method for optimal control. *Optimization Methods and Software*, 26(2):239–257, April 2011.

[Ked80] G. Kedem. Automatic differentiation of computer programs. *ACM Trans. on Math. Soft.*, 6:150–165, 1980.

[Kir06] C. Kirches. A numerical method for nonlinear robust optimal control with implicit discontinuities and an application to powertrain oscillations. Diploma thesis, Ruprecht–Karls–Universität Heidelberg, October 2006.

[KKBS04] S. Körkel, E. Kostina, H.G. Bock, and J.P. Schlöder. Numerical methods for optimal control problems in design of robust optimal experiments for nonlinear dynamic processes. *Optimization Methods and Software*, 19:327–338, 2004.

[Kör02] S. Körkel. *Numerische Methoden für Optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen*. PhD thesis, Universität Heidelberg, Heidelberg, 2002.

[Kra85] D. Kraft. On converting optimal control problems into nonlinear programming problems. In K. Schittkowski, editor, *Computational Mathematical Programming*, volume F15 of *NATO ASI*, pages 261–280. Springer, 1985.

[Kro66] F.T. Krogh. Predictor-corrector methods of high order with improved stability characteristics. *J. ACM*, 13(3):374–385, 1966.

[KS84] F.T. Krogh and K. Stewart. Asymptotic ($h \to \infty$) absolute stability for BDFs applied to stiff differential equations. *ACM Trans. Math. Softw.*, 10(1):45–57, 1984.

[KSBS10] C. Kirches, S. Sager, H.G. Bock, and J.P. Schlöder. Time-optimal control of automobile test drives with gear shifts. *Optimal Control Applications and Methods*, 31(2):137–153, March/April 2010.

[KT51] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, 1951. University of California Press.

[Lai06] K.-L. Lai. *Generalizations of the complex-step derivative approximation*. PhD thesis, State University of New York at Buffalo, 2006.

[LB92] J.S. Logsdon and L.T. Biegler. Decomposition strategies for large-scale dynamic optimization problems. *Chemical Engineering Science*, 47(4):851–864, 1992.

[LBBS03] D.B. Leineweber, I. Bauer, H.G. Bock, and J.P. Schlöder. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: Theoretical aspects. *Computers and Chemical Engineering*, 27:157–166, 2003.

[Lei95] D.B. Leineweber. Analyse und Restrukturierung eines Verfahrens zur direkten Lösung von Optimal-Steuerungsproblemen. Diploma thesis, Ruprecht–Karls–Universität Heidelberg, 1995.

[Lei99] D.B. Leineweber. *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, volume 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. VDI Verlag, Düsseldorf, 1999.

[Lin76] S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT*, 16:146–160, 1976.

[LM67] J.N. Lyness and C.B. Moler. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis*, 4:202–210, 1967.

[LP86] P. Lötstedt and L. Petzold. Numerical solution of nonlinear differential equations with algebraic constraints I: Convergence results for backward differentiation formulas. *Mathematics of Computation*, 46(174):491–516, 1986.

[LSBS03] D.B. Leineweber, A.A.S. Schäfer, H.G. Bock, and J.P. Schlöder. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part II: Software aspects and applications. *Computers and Chemical Engineering*, 27:167–174, 2003.

[LSF08] D. Lebiedz, D. Skanda, and M. Fein. Automatic complexity analysis and model reduction of nonlinear biochemical systems. In M. Heiner and A.M. Uhrmacher, editors, *Computational Methods in Systems Biology*, Lecture Notes in Bioinformatics, Volume 5307, pages 123–140. Springer, 2008.

[LV07] J. Lang and J.G. Verwer. On global error estimation and control for initial value problems. *SIAM J. Sci. Comput.*, 29:1460–1475, 2007.

[Lyn67] J.N. Lyness. Numerical algorithms based on the theory of complex variable. In *Proceedings of the 1967 22nd national conference*, ACM Annual Conference/Annual Meeting, pages 125 – 133, 1967.

[Map09] Maplesoft. *Maple 13*. Maplesoft, Inc., 2009.

[Mar78] N. Maratos. *Exact penalty function algorithms for finite-dimensional and control optimization problems*. PhD thesis, Imperial College, London, 1978.

[MI08] F. Mazzia and F. Iavernaro. Test set for initial value problem solver. Department of mathematics, University of Bari, August 2008. Available at `http://www.dm.uniba.it/~testset`.

[Mil26] W.E. Milne. Numerical integration of ordinary differential equations. *The American Mathematical Monthly*, 33(9):455–460, 1926.

[MO04] H. Maurer and N.P. Osmolovskii. Second order sufficient conditions for time-optimal bang-bang control. *SIAM Journal on Control and Optimization*, 42:2239–2263, 2004.

[Mom01] K.D. Mombaur. *Stability Optimization of Open-loop Controlled Walking Robots*. PhD thesis, Universität Heidelberg, 2001.

[Moo66] R.E. Moore. *Interval analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Mou26] F.R. Moulton. *New Methods in Exterior Ballistics.* Chicago University Press, Chicago, 1926.

[MP82] D.Q. Mayne and E. Polak. A superlinearly convergent algorithm for constrained optimization problems. *Mathematical Programming*, 16:45–61, 1982.

[MS86] K.R. Morison and R.W.H. Sargent. Optimization of multistage processes described by differential-algebraic equations. In J.P.Hennart, editor, *Numerical Analysis*, Lecture Notes in Mathematics 1230. Springer, Berlin, 1986.

[MSA03] J.R.R.A. Martins, P. Sturdza, and J.J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3):245–262, September 2003.

[MSTZ03] K.-S. Moon, A. Szepessy, R. Tempone, and G. Zouraris. A variational principle for adaptive approximation of ordinary differential equations. *Numerische Mathematik*, 96:131–152, 2003.

[Nau99] U. Naumann. *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs.* PhD thesis, Technical University of Dresden, 1999.

[Nau02] U. Naumann. Elimination techniques for cheap Jacobians. In G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors, *Automatic Differentiation of Algorithms: From Simulation to Optimization.* Springer, 2002.

[Nau08] U. Naumann. Optimal Jacobian accumulation is NP-complete. *Mathematical Programming*, 112(2):427 – 441, 2008.

[NAW98] J.C. Newman, W.K. Anderson, and D.L. Whitfield. Multidisciplinary sensitivity derivatives using complex variables. Technical Report MSSU-COE-ERC-98-08, Mississippi State University, July 1998.

[Nei92] R.D. Neidinger. An efficient method for the numerical evaluation of partial derivatives of arbitrary order. *ACM Trans. Math. Softw.*, 18(2):159–173, 1992.

[Nor62] A. Nordsieck. On numerical integration of ordinary differential equations. *Math. Comput.*, 16:22–49, 1962.

[NR83] G.N. Newsam and J.D. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM J. Alg. Disc. Meth.*, 4(3):404–417, 1983.

[NW99] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer Verlag, Berlin Heidelberg New York, 1st edition, 1999. ISBN 0-387-98793-2 (hardcover).

[OB05] D.B. Özyurt and P.I. Barton. Cheap second order directional derivatives of stiff ODE embedded functionals. *SIAM Journal on Scientific Computing*, 26(5):1725–1743, 2005.

[Osb69] M.R. Osborne. On shooting methods for boundary value problems. *Journal of Mathematical Analysis and Applications*, 27:417–433, 1969.

[PBE+08] A. Potschka, H.G. Bock, S. Engell, A. Küpper, and J.P. Schlöder. A Newton-Picard inexact SQP method for optimization of SMB processes. Preprint SPP1253-01-01, University of Heidelberg, 2008.

[PBGM62] L.S. Pontryagin, V.G. Boltyanski, R.V. Gamkrelidze, and E.F. Miscenko. *The Mathematical Theory of Optimal Processes*. Wiley, Chichester, 1962.

[PBS09] A. Potschka, H.G. Bock, and J.P. Schlöder. A minima tracking variant of semi-infinite programming for the treatment of path constraints within direct solution of optimal control problems. *Optimization Methods and Software*, 24(2):237–252, 2009.

[PD81] P.J. Prince and J.R. Dormand. High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67 – 75, 1981.

[Pes94] H.J. Pesch. A practical guide to the solution of real-life optimal control problems. *Control and Cybernetics*, 23:7–60, 1994.

[Pet82] L.R. Petzold. A Description of DASSL: A Differential-Algebraic System Solver. In *Proc. 10th IMACS World Congress*, Montreal, 1982.

[PL86] L. Petzold and P. Lötstedt. Numerical solution of nonlinear differential equations with algebraic constraints II: Practical implications. *SIAM Journal on Scientific and Statistical Computing*, 7(3):720–733, 1986.

[Pli81] K.J. Plitt. Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschränkter optimaler Steuerungen. Diploma thesis, Rheinische Friedrich–Wilhelms–Universität zu Bonn, 1981.

[PM76] U.M. Garcia Palomares and O.L. Mangasarian. Superlinearly convergent quasi-newton algorithms for nonlinearly constrained optimization problems. *Mathematical Programming*, 11:1–13, 1976.

[Pow78a] M.J.D. Powell. Algorithms for nonlinear constraints that use Lagrangian functions. *Mathematical Programming*, 14(3):224–248, 1978.

[Pow78b] M.J.D. Powell. The convergence of variable metric methods for nonlinearly constrained optimization calculations. In O.L. Mangasarian, R.R. Meyer, and S.M. Robinson, editors, *Nonlinear Programming 3*. Academic Press, 1978.

[Pow78c] M.J.D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G.A. Watson, editor, *Numerical Analysis, Dundee 1977*, volume 630 of *Lecture Notes in Mathematics*, Berlin, 1978. Springer.

[PRG+97] L. Petzold, J.B. Rosen, P.E. Gill, L.O. Jay, and K. Park. Numerical optimal control of parabolic PDEs using DASOPT. In Biegler, Coleman, Conn, and Santosa, editors, *Large Scale Optimization with Applications, Part II*, volume 93 of *IMA Volumes in Mathematics and its Applications*, pages 271–300. Springer, 1997.

[PSV94] C.C. Pantelides, R.W.H. Sargent, and V.S. Vassiliadis. Optimal control of multistage systems described by high-index differential-algebraic equations. In R. Bulirsch and D. Kraft, editors, *Computational Optimal Control*. Birkhäuser, Basel, 1994.

[Ral81] L.B. Rall. *Automatic Differentiation: Techniques and Applications*. Springer, Berlin, 1981.

[Res08] Wolfram Research. *Mathematica Edition: Version 7.0*. Wolfram Research, Inc., 2008.

[Rhe84] W.C. Rheinboldt. Differential-algebraic systems as differential equations on manifolds. *Math. of Comput.*, 43:473–482, 1984.

[Rid09] M.S. Ridout. Statistical applications of the complex-step method of numerical differentiation. *The American Statistician*, 63(1):66–74, 2009.

[Rie01] P. Riede. Entwicklung eines parallelen PRSQP-Verfahrens zur Lösung von Multiple-Setpoint-Optimierungsproblemen. Diploma thesis, Universität Heidelberg, 2001. Download at: http://www.rzuser.uni-heidelberg.de/~jx7/diplom.ps.

[Rie06] P. Riede. *Optimierung von dynamischen Multiple-Setpoint-Problemen mit Anwendung bei Fahrzeugmodellen*. PhD thesis, Universität Heidelberg, 2006.

[RMM94] J.B. Rawlings, E.S. Meadows, and K.R. Muske. Nonlinear model predictive control: A tutorial and survey. In *Proc. Int. Symp. Adv. Control of Chemical Processes, ADCHEM*, Kyoto, Japan, 1994.

[Rob74] S.M. Robinson. Perturbed kuhn-tucker points and rates of convergence for a class of nonlinear programming algorithms. *Mathematical Programming*, 7:1–16, 1974.

[Rüc99] G. Rücker. Automatisches Differenzieren mit Anwendung in der Optimierung bei chemischen Reaktionssystemen. Diploma thesis, Universität Heidelberg, 1999.

[Sag05] S. Sager. *Numerical methods for mixed–integer optimal control problems*. Der andere Verlag, Tönning, Lübeck, Marburg, 2005. ISBN 3-89959-416-9. Available at http://sager1.de/sebastian/downloads/Sager2005.pdf.

[Sag06] S. Sager. *Numerical methods for mixed–integer optimal control problems*. PhD thesis, Universität Heidelberg, 2006.

[SB92] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 1992.

[SBPD+07] S. Sager, U. Brandt-Pollmann, M. Diehl, D. Lebiedz, and H.G. Bock. Exploiting system homogeneities in large scale optimal control problems for speedup of multiple shooting based SQP methods. *Computers and Chemical Engineering*, 31:1181–1186, 2007.

[SBS98] V.H. Schulz, H.G. Bock, and M.C. Steinbach. Exploiting invariants in the numerical solution of multipoint boundary value problems for DAEs. *SIAM Journal on Scientific Computing*, 19:440–467, 1998.

[Sch75] J. Schäfer. A new approach to explain the 'high irradiance responses' of photomorphogenesis on the basis of phytochrome. *Journal of Mathematical Biology*, 2:41–56, 1975.

[Sch88] J.P. Schlöder. *Numerische Methoden zur Behandlung hochdimensionaler Aufgaben der Parameteridentifizierung*, volume 187 of *Bonner Mathematische Schriften*. Universität Bonn, Bonn, 1988.

[Sch96] V.H. Schulz. *Reduced SQP methods for large-scale optimal control problems in DAE with application to path planning problems for satellite mounted robots*. PhD thesis, Universität Heidelberg, 1996.

[Sch05] A.A.S. Schäfer. *Efficient reduced Newton-type methods for solution of large-scale structured optimization problems with application to biological and chemical processes*. PhD thesis, Universität Heidelberg, 2005.

[Sch08] A. Schmidt. Automatisches Differenzieren bei Differentialgleichungsmodellen der Systembiologie. Diploma thesis, Universität Heidelberg, 2008.

[SH99] S. Støren and T. Hertzberg. Obtaining sensitivity information in dynamic optimization problems solved by the sequential approach. *Computers & Chemical Engineering*, 23(6):807 – 819, 1999.

[Ske86] R.D. Skeel. Thirteen ways to estimate global error. *Numerische Mathematik*, 48:1–20, 1986.

[Spe80] B. Speelpenning. *Compiling fast partial derivatives of functions given by algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1980.

[SS78] R.W.H. Sargent and G.R. Sullivan. The development of an efficient optimal control package. In J. Stoer, editor, *Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2*, Heidelberg, 1978. Springer.

[ST98] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 40:110–112, 1998.

[Ste68] H.J. Stetter. Improved absolute stability of predictor-corrector schemes. *Computing*, 3:286–296, 1968.

[Ste95] M.C. Steinbach. *Fast recursive SQP methods for large-scale optimal control problems*. PhD thesis, Ruprecht–Karls–Universität Heidelberg, 1995.

[Ste02] M.C. Steinbach. Tree-sparse convex programs. *Math. Methods Oper. Res.*, 56(3):347–376, 2002.

[Sto98] W.J.H. Stortelder. *Parameter estimation in nonlinear dynamical systems*. PhD thesis, University of Amsterdam, 1998.

[Str93] O. von Stryk. Numerical solution of optimal control problems by direct collocation. In *Optimal Control: Calculus of Variations, Optimal Control Theory and Numerical Methods*, volume 111, pages 129–143. Bulirsch et al., 1993.

[Stu80] F. Stummel. Rounding error analysis of elementary numerical algorithms. *Computing*, Suppl. 2:169–195, 1980.

[SW94] R. von Schwerin and M. Winckler. A guide to the integrator library MBSSIM version 1.00. Technical Report 94-75, IWR, Universität Heidelberg, 1994.

[SW95] K. Strehmel and R. Weiner. *Numerik gewöhnlicher Differentialgleichungen*. Teubner, Stuttgart, 1995.

[SW96] R. von Schwerin and M. Winckler. Some aspects of sensitivity analysis in vehicle system dynamics. *Zeitschrift für Angewandte Mathematik und Mechanik*, 76(S3):155–158, 1996.

[TB95] P. Tanartkit and L.T. Biegler. Stable decomposition for dynamic optimization. *Industrial and Engineering Chemistry Research*, 34:1253–1266, 1995.

[TB96] P. Tanartkit and L.T. Biegler. A nested, simultaneous approach for dynamic optimization problems – I. *Computers and Chemical Engineering*, 20:735–741, 1996.

[TB09] L. Tran and M. Berzins. Defect sampling in global error estimation for ODEs using adjoint methods. *SIAM Review*, (submitted), 2009.

[TBK04] S. Terwen, M. Back, and V. Krebs. Predictive powertrain control for heavy duty trucks. In *Proceedings of IFAC Symposium in Advances in Automotive Control*, pages 451–457, Salerno, Italy, 2004.

[TED+07] A. Toumi, S. Engell, M. Diehl, H.G. Bock, and J.P. Schlöder. Efficient optimization of Simulated Moving Bed Processes. *Chemical Engineering and Processing*, 46 (11):1067–1084, 2007. (invited article).

[THE75] T.H. Tsang, D.M. Himmelblau, and T.F. Edgar. Optimal control via collocation and non-linear programming. *International Journal on Control*, 21:763–768, 1975.

[vH06] G.M. von Hippel. Taylur, an arbitrary-order automatic differentiation package for fortran 95. *Comput.Phys.Commun.*, 174:569–576, 2006.

[vS92] M. von Schwerin. Ein Trust-Region-Verfahren zur Lösung von Steuerungsproblemen mittels Multiple-Shooting und Sukzessiver Quadratischer Programmierung. Diploma thesis, University of Heidelberg, 1992.

[vS97] R. von Schwerin. *Numerical methods, algorithms, and software for higher index nonlinear differential-algebraic equations in multibody system simulation.* PhD thesis, Universität Heidelberg, 1997.

[vSB95] R. von Schwerin and H.G. Bock. A Runge-Kutta-starter for a multistep-method for differential-algebraic systems with discontinuous effects. *Applied Numerical Mathematics*, 18:337–350, 1995.

[WAK+08] L. Wirsching, J. Albersmeyer, P. Kuehl, M. Diehl, and H.G. Bock. An adjoint-based numerical method for fast nonlinear model predictive control. In *Proceedings of the 17th IFAC World Congress, Seoul 2008*. IFAC-PapersOnLine, 2008.

[Wal93] W. Walter. *Gewöhnliche Differentialgleichungen.* Springer, Heidelberg, 1993. ISBN: 038756294X.

[Wal00] A. Walther. *Program Reversal Schedules for Single- and Multi-processor Machines.* PhD thesis, TU Dresden, 2000.

[Wal08] A. Walther. A first-order convergence analysis of Trust-Region methods with inexact Jacobians. *SIAM Journal on Optimization*, 19(1):307–325, 2008.

[Wan69] G. Wanner. *Integration gewöhnlicher Differentialgleichungen, Lie Reihen, Runge-Kutta-Methoden*, volume vol.XI,B.I-Hochschulskripten. Bibliographisches Institut, Mannheim-Zuerich, Germany, 1969.

[WB02] A. Wächter and L.T. Biegler. Global and local convergence of line search filter methods for nonlinear programming. Technical report, Department of Chemical Engineering, Carnegie Mellon University, 2002.

[WB06] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[Wen64] R.E. Wengert. A simple automatic derivative evaluation program. *Commun. ACM*, 7(8):463–464, 1964.

[Wil63] R.B. Wilson. *A simplicial algorithm for concave programming.* PhD thesis, Harvard University, 1963.

[Wir06] L. Wirsching. An SQP algorithm with inexact derivatives for a direct multiple shooting method for optimal control problems. Diploma thesis, Universität Heidelberg, 2006.

[WPD01] R.C. Whaley, A. Petitet, and J.J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[WSW10] M. Wagner, B.-J. Schäfer, and A. Walther. On the efficient computation of high-order derivatives for implicitly defined functions. *Computer Physics Communications*, 181:756–764, 2010.