Diss. ETH No. 21366

Methods and Tools for Embedded Optimization and Control

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Alexander Domahidi

Dipl.-Ing., RWTH Aachen University, Germany
born 12.07.1982
citizen of Germany

accepted on the recommendation of

Prof. Dr. Manfred Morari (examiner)
Prof. Dr. Stephen Boyd (co-examiner)

2013

ISBN 978-3-906031-38-5

For

Kati, Sanyi, and Emese

Acknowledgments

My deepest gratitude goes to Manfred Morari for giving me the opportunity to pursue the PhD degree in one of the best control labs in the world. His continuous support and guidance, the freedom to investigate research topics according to my liking combined with his generous financial support for equipment and traveling have made my doctorate studies an invaluable and truly enjoyable experience. I am grateful for the countless interesting and smart people he and John Lygeros bring to Zurich, both visitors and co-workers, creating a unique research environment in which I have learned incredibly lot over the past years.

I am indebted to Stephen Boyd for always welcoming me with an open door in Stanford, for his continued and candid sharing of ideas, manuscripts and software (not only with me, but with all of us), and for his insistence on simplicity. Working with him is unbelievably enriching in many ways, and I am extremely grateful that he devoted his time to being my co-examiner. It is fair to say that his joy at work is highly contagious, and that he has successfully infected me, too.

I greatly appreciate the supervision by Colin Jones, who consistently fired tons of ideas into my direction: It was in the numerous meetings with him and Melanie Zeilinger that most of the ideas presented in this thesis were conceived. Thanks to him I learned about machine learning, and for the first time realized that we better pose *everything* as an optimization problem, which I believe will prove fundamental for my future career. I also thank him for passing on the race car project, which was a lot of fun to work on.

A gazillion thanks go to Melanie Zeilinger, who patiently mastered endless meetings to create a rigorous mathematical representation of vague ideas. I thank her also for teaching me how to teach Control Systems (for a reference, when I became her successor as head TA, the job had to be split among two people). Her meticulous corrections of papers, presentations and this thesis have significantly improved their quality, and her help and support has been – and is – invaluable throughout.

The effort of Moritz Diehl for pulling the strings creating the EMBOCON project, which has brought together numerous outstanding people and has created a great environment to exchange and discuss ideas on embedded optimization, is very much appreciated. His view of software artifacts as an additional, valuable outcome of research shaped also my attitude. The EMBOCON family would not have formed without Sebastian Sager, to which

I am sincerely grateful that he has made this effort. I thank all EMBOCON participants for the great time together – what happened in Bucharest stays in Bucharest.

If it were not for Sébastien Mariéthoz, my way would most probably never have crossed the Automatic Control Laboratory. Thanks to him for the supervision during my master's, and for teaching me about power electronics. I am also indebted to Thomas Baumgartner, who is unofficially known to be responsible for my application at IfA.

Special thanks go to Paul Goulart, whose excellent advice – depressingly, he tends to be right with probability one – is much more appreciated than he is aware. It is amazing that he never appears to be short in time despite supervising tons of people and projects.

Urban Mäder's unbiased opinion and his help for getting me settled at IfA four years ago were extremely enlightening and helpful. His achievement during the past years is a truly motivating example, and I am grateful that he shares his experience generously. Thanks to Stefan Richter for being a great companion in the world of embedded optimization, for patiently teaching me about first order methods and for correcting this and other manuscripts. The feedback from Marc Guadayol and Milan Vukov was crucial for the improvement of the early versions of FORCES, and great appreciation to Eric Chu for substantially increasing the quality and usability of ECOS by providing a test suite as well as the Python interface. Special thanks go to Sandro Merkli and Lorenz Meier for openly sharing their expertise in hardware design, ensuring the quality of the ORCA embedded boards.

I award Christian Conte the best office mate I could think of in four consecutive years, and I thank him as well as Stephan Huck, Christoph Zechner and Mathias Kühn for the creative and fruitful distractions, patiently keeping me attached to the life outside research – this is more appreciated than I like to admit. Special thanks go to Andreas Hempel, who has been an indispensable help in organizing the ECC lab tours. Thanks also to Aldo "T-800" Zgraggen for his effort in the early days of FORCES, and to Georg Schildbach for always helping out when needed. Kudos goes to Nikolaos Kariotoglou – for just being himself.

Last, but most importantly, I am forever grateful to my parents for the opportunity to grow up where and how I did, and to my sister for always being there for me in the last 31 years.

Alexander Domahidi
Zürich, October 2013

Contents

Abstract

Embedded optimal decision making based on mathematical optimization is a promising, systematic tool to achieve higher performance, tighter operating constraints, increased energy efficiency and lower operating cost of complex systems, which are key goals across all industries. One successful example of optimal decision making is model predictive control (MPC), which has been established as the standard control paradigm in the process industry in the past decades. The extension of MPC to systems with fast dynamics and limited computational power is however still considered a major challenge. The main contribution of this thesis is to enable the implementation of sophisticated MPC techniques on resource constrained embedded platforms at high sampling rates.

In general, there are two fundamentally different paradigms for the implementation of MPC controllers: solving the optimization problem offline for all possible problem parameters, thereby synthesizing an explicit state-to-input map that is inexpensive to evaluate online, and solving an optimization problem online at each sampling instance. Available approaches of explicit MPC are limited to low-dimensional systems, while implicit formulations based on convex problems can handle problems of all practically relevant dimensions, but are comparably expensive to compute.

In this thesis, novel methods and tools for embedded optimization and control are proposed, covering both domains of explicit MPC and online optimization in an embedded context. In the first part of the thesis, a new synthesis method for low-complexity suboptimal explicit MPC controllers for continuous inputs is introduced, which is based on function approximation from randomly chosen point-wise sample values. We provide sufficient conditions that can be imposed in standard machine learning algorithms in the form of tractable convex constraints that guarantee input and state constraint satisfaction, recursive feasibility and stability of the closed loop system. The resulting control law can be fully parallelized, which renders the approach particularly suitable for highly concurrent embedded platforms such as field-programmable gate arrays (FPGAs). A numerical example shows the effectiveness of the proposed method.

The idea of learning control laws from simulation data is then extended to binary decision rules for energy efficient building control. While rule based control (RBC) is current practice in most building automation systems that issue discrete control signals, recent simulation

studies suggest that hybrid model predictive control (HMPC) can potentially outperform RBC in terms of energy efficiency and occupancy comfort. We suggest an automated RBC synthesis procedure for binary decisions that extracts prevalent information from simulation data with HMPC controllers. The result is a set of simple decision rules that captures much of the control performance of HMPC. The proposed technique is based on standard machine learning algorithms, particularly support vector machines (SVMs) and adaptive boosting (AdaBoost). It is shown how an importance ranking and selection of measurements used for a decision can be established and utilized for a complexity reduction by pruning unnecessary sensors. The methods are applied to the problem of optimal building control and evaluated in simulation for six different case studies, where they are shown to maintain the performance of HMPC despite a tremendous reduction in complexity.

The second part of the thesis focuses on the solution of optimization problems on embedded systems. First, we present efficient interior point methods tailored to convex multistage problems, a problem class which most relevant MPC problems with linear dynamics can be cast in, and specify important algorithmic details required for a high speed implementation with superior numerical stability. We present extensive numerical studies for the proposed methods and compare our solver to three well-known solver packages, outperforming the fastest of these by a factor 2-5 in speed and 3-70 in code size. Moreover, our solver is shown to be very efficient for large problem sizes and for quadratically constrained QPs (QCQPs), extending the set of systems amenable to advanced MPC formulations on low-cost embedded hardware. A publicly accessible code generation system allows for generating tailored solver code that implements the proposed method, exploiting all problem structure for high speed computation. The generated solvers are library free and applicable on embedded platforms for which a C compiler is available.

The scope of optimization problems is further extended to second-order cone programming (SOCP) by a solver specifically designed for embedded applications, which not only provides high-speed computation but also the detection of infeasibilities. Based on a primal-dual Mehrotra interior point method with Nesterov-Todd scalings and self-dual embedding, the search directions are computed via a symmetric indefinite KKT system. The novelty is a sparse expansion of diagonal plus rank two blocks chosen to allow for a stable factorization with any pivoting order. As a result, the elimination ordering can be computed entirely symbolically and is fixed during all iterations. Using regularization and iterative refinement, the solver is numerically robust for accuracies typically required in embedded applications, and it is comprised of only about 800 lines of code including all linear algebra functionality. The resulting solver is faster than freely available SOCP solvers and it is still competitive with commercial solvers for problems with up to tens of thousands of variables.

Zusammenfassung

Eingebettete optimale Entscheidungsfindung auf der Grundlage der Mathematischen Optimierung ist ein vielversprechendes, systematisches Werkzeug, um eine höhere Leistung, engere Betriebsgrenzen, eine Steigerung der Energieeffizienz und niedrigere Betriebskosten von komplexen Systemen zu erreichen, welche über Branchen hinweg die wichtigsten Ziele sind. Ein erfolgreiches Beispiel für optimale Entscheidungsfindung ist die prädiktive Regelungstechnik (engl.: model predictive control, MPC), die sich in den vergangenen Jahrzehnten in der Prozessindustrie als Standard etabliert hat. Die Anwendung von MPC auf Systeme mit schneller Dynamik und begrenzter Rechenleistung ist jedoch immer noch eine grosse Herausforderung. Der wichtigste Beitrag dieser Arbeit ist es, die Umsetzung von anspruchsvollen MPC-Techniken auf ressourcenbeschränkte, eingebettete Plattformen bei hohen Abtastraten zu ermöglichen.

Grundsätzlich gibt es zwei fundamental verschiedene Ansätze für die Implementierung von MPC-Reglern. Zum einen kann die Lösung des parametrischen Optimierungsproblems für alle zulässigen Parameter vorberechnet werden, wodurch eine explizite Abbildung vom Zustandsraum auf den Reglereingang synthetisiert wird, die dann in der Implementierung mit wenigen Rechenschritten evaluiert werden kann. Zum anderen kann das Optimierungsproblem wähnend des Betriebes an jedem Abtastzeitpunkt für den aktuellen Parameterwert neu gelöst werden. Explizites MPC ist auf Systeme mit wenigen Zuständen beschränkt, während die implizite Methode basierend auf konvexer Optimierung für alle Systeme angewendet werden kann, aber vergleichsweise teuer in der sog. Online-Berechnung ist.

In dieser Arbeit werden neue Methoden und Werkzeuge für eingebettete Optimierung und Regelung vorgeschlagen, die sowohl explizites MPC als auch Online-Optimierung in einem eingebetteten Kontext betrachten. Im ersten Teil der Arbeit wird eine neue Methode zur Synthese von suboptimalen expliziten MPC Reglern von geringer Komplexität für kontinuierliche Eingänge eingeführt, die auf einer Näherung der optimalen Reglerfunktion an zufällig ausgewählten Punkten basiert. Wir leiten hinreichende Bedingungen her, die zu Standardalgorithmen des maschinellen Lernens in Form von konvexen Nebenbedingungen hinzugefügt werden können, so dass systemtheoretische Garantien auf Stabilität und Zulässigkeit des geschlossenen Regelkreises erreicht werden. Das resultierende Regelgesetz kann vollständig parallelisiert werden, was den Ansatz insbesondere für parallele eingebettete Plattformen wie

zum Beispiel Field-Programmable Gate Arrays (FPGAs) attraktiv macht. Ein numerisches Beispiel zeigt die Wirksamkeit des vorgeschlagenen Verfahrens auf.

Die Idee des maschinellen Lernens von Regelgesetzen an Hand von Simulationsdaten wird anschliessend auf binäre Entscheidungen für energieeffiziente Gebäuderegelung erweitert. Während der Stand der Technik in der Gebäudeautomation auf einfachen Entscheidungsregeln basiert (engl.: rule based control, RBC), haben jüngste Studien gezeigt, dass hybrides MPC, also MPC mit diskreten Entscheidungsvariablen, RBC in Bezug auf Energieeffizienz und Wohnkomfort übertreffen kann. Wir schlagen daher ein automatisiertes Verfahren zur Extraktion von binären Entscheidungsregeln aus Simulationsdaten mit HMPC Reglern vor, die einen vollständigen RBC Regler synthetisiert. Das Ergebnis ist eine Reihe von einfachen Entscheidungsregeln, die nahe an die Regelgüte des HMPC heranreichen. Die vorgeschlagene Methode basiert auf Standardalgorithmen des maschinellen Lernens, insbesondere Support Vector Machines (SVM) und Adaptive Boosting (AdaBoost). Es wird aufgezeigt, wie eine Rangfolge von Messungen bezüglich ihrer Wichtigkeit für eine Entscheidung erstellt und für eine Reduzierung der Komplexität durch das Weglassen von unnötigen Sensoren benutzt werden kann. Die Verfahren werden für das Problem der optimalen Gebäuderegelung in einer Simulationsstudie auf sechs verschiedene Gebäude angewendet, wobei sich durchweg eine vergleichbare Regelgüte zum HMPC trotz der enormen Komplexitätsreduktion zeigt.

Der zweite Teil der Arbeit konzentriert sich auf die Lösung von Optimierungsproblemen auf eingebetteten Systemen. Zunächst präsentieren wir effiziente Innere-Punkt-Methoden, die auf konvexe mehrstufige Probleme zugeschnitten sind, eine Problemklasse, die die meisten relevanten MPC Probleme mit linearer Dynamik abdeckt. Wir diskutieren wichtige algorithmische Details, die für eine schnelle Implementierung mit numerischer Stabilität erforderlich sind. Wir präsentieren umfangreiche numerische Untersuchungen für die vorgeschlagene Methode, wobei unserer den schnellsten der uns bekannten Löser um einen Faktor von 2-5 in der Rechengeschwindigkeit übertrifft, bei einer gleichzeitigen Reduktion der Code-Grösse um einen Faktor von 3-70. Darüber hinaus ist unser Löser sehr effizient für grosse Probleme, und geeignet für quadratische Probleme mit quadratischen Nebendebingungen (QCQPs). Dies erweitert die Klasse von Systemen bzw. MPC Formulierungen, die auf kostengünstigen eingebetteten Plattformen effizient gelöst werden können. Eine öffentlich zugängliche Code-Generierung ermöglicht die automatische Erzeugung massgeschneiderter Löser, die die vorgeschlagene Methode implementieren und die gesamte Problemstruktur für grösstmögliche Rechengeschwindigkeit ausnutzen. Die erzeugten Löser sind unabhänging von externen Bibliotheken und können somit auf eingebetteten Plattformen, für die ein C-Compiler zur Verfügung steht, unmittelbar eingesetzt werden.

Weiterhin wird der Umfang der auf eingebetteten Systemen lösbaren Optimierungsprobleme auf Probleme mit Nebenbedingungen aus der elliptischen Kegelmenge (second-order

cone programming, SOCP) erweitert. Wir präsentieren einen Löser, der neben hohen Rechengeschwindigkeiten auch die Detektion von unzulässigen Problemen unterstützt. Basierend auf einer Mehrotra Inneren-Punkt-Methode mit Nesterov-Todd Suchrichtungen und selbst-dualer Formulierung, werden die Suchrichtungen über ein symmetrisches, indefinites KKT-System berechnet. Die Neuerung ist eine speziell gewählte, dünnbesetzte Repräsentation des KKT Systems, die eine numerisch stabile Faktorisierung für jede Eliminationsreihenfolge ermöglicht. Dadurch kann diese vollständig symbolisch berechnet werden und ist während aller Iterationen die selbe. Mittels Regularisierung und der Technik der iterativen Verbesserung ist der Löser numerisch stabil für Genauigkeiten, die typischerweise in eingebettten Systemen erforderlich sind. Der Löser umfasst lediglich etwa 800 Zeilen Quellcode, einschliesslich der gesamten linearen Algebra Funktionalität, und ist heute der schnellste frei verfügbare SOCP Löser, während er mit kommerziellen Lösern für Probleme mit bis zu Zehntausenden von Variablen kompetititv ist.

1 Introduction

In recent years, inexpensive sensing and communication hardware has made information on the state and the environment of a system available in real-time at an unprecedented level of detail. For example, wide area measurement systems provide information of the power grid over large distances several times a second, and low-cost inertial measurement units enable the stabilization of autonomous flying machines that were prohibitively expensive only a decade ago, but are now affordable as toys. In addition to sensing and communication hardware, embedded computing platforms pervade our daily life and have become increasingly powerful. Modern cars, for instance, sense, control and react to the environment with the help of more than one hundred embedded processors, ensuring safety and comfort. Each of these processors is orders of magnitude more powerful than the processors of the early era of personal computing.

The main driver behind the trend of cheap embedded computing and the ubiquitous availability of information is the strive for higher performance, increased energy efficiency, lower cost of operation and operating regimes that are closer to the physical boundaries, which are the key goals across all industries. In order to achieve these targets while coping with the ever increasing complexity of systems, advanced autonomous decision making is required on embedded platforms.

The framework of mathematical optimization offers a significant potential in this context, as it provides a systematic methodology for selecting the best decision under a large (possibly infinite) number of possibilities while satisfying system constraints and maximizing performance or minimizing cost. One popular example is constrained optimal control, also known as *model predictive control* (MPC) or *receding horizon control* (RHC), where the optimal control input is obtained by the solution of a constrained optimization problem at each sampling instance. MPC has been applied with great success in the process industries, where today it is established as the standard control paradigm. With sampling times in the order of minutes, the arising optimization problems are generally solved on powerful computing platforms. In order to make these techniques available for applications with fast dynamics at a large scale, the solution of optimization problems in short sampling times in the range of milliseconds or below must be achieved on inexpensive embedded platforms.

This thesis develops methods and tools for embedded optimization and control, with the

goal to reduce the gap between successful synthesis methods for optimal decision making and resource constrained embedded platforms. Two fundamentally different approaches are presented: in the first part of the thesis, we propose to derive an explicit approximation of the optimal control law by collecting data samples in simulation and using high-dimensional function approximation methods from the field of machine learning. The resulting control laws are cheap to evaluate online, while maintaining much of the control performance of the optimal controller. In the second part of the thesis, efficient iterative solvers for high-speed online optimization on embedded platforms are developed, extending the class of problems that can be solved efficiently on embedded systems to quadratically constrained quadratic programs (QCQPs) and second-order cone programs (SOCPs). An implementation of the proposed methods is publicly available through source code and a code generation system which automatically tailors interior point solvers to the given problem.

A variety of other methods and solvers have been proposed for the efficient implementation of linear MPC on embedded platforms: For certain problem formulations and small dimensions, multi-parametric programming can be used to calculate the optimal state-to-input map offline, enabling model predictive control of extremely high-speed systems [2,14]. However, computing this so-called *explicit* solution is possible only for low-dimensional systems. *Suboptimal* explicit methods compute approximate control laws, see e.g. [14,57,105], aiming at extending the class of systems to which MPC can be applied. For the suboptimal explicit methods proposed in this thesis, the main idea is to take advantage of well established and successful machine learning methods, which have been proven to work remarkably well for practical high-dimensional function approximation problems [3, 133]. We address two key limitations when using standard learning methods for controller synthesis: Directly applying standard learning methods to approximate the continuous control law does not provide any guarantees on closed-loop constraint satisfaction or stability. We present tractable convex constraints that can be combined with any optimization based learning technique to ensure these crucial controller properties. In the case of learning binary decision rules, we propose a method to synthesise rule based controllers that are meaningful to human operators and which can be tuned post synthesis, which is in general not possible with other approximation schemes.

Online optimization can handle problems of any practically relevant dimension, and various solution methods and implementations have been recently developed in the literature. First order methods [108, 113] are for certain problem classes the preferred choice, since tight bounds on the computation time can be provided [113]. However, the performance of first order methods strongly depends on the problem conditioning and on the type of the feasible set, for which efficient projection operators are required. Fast active set methods that exploit information from the solution of one MPC problem to the next have been suc-

cessfully applied in practice at high sampling rates [44,45]. However, the runtimes can vary over a wide range and active set methods are limited to linear and quadratic programs (LPs & QPs). In this thesis, we therefore consider primal-dual interior point methods (IPMs), which allow for solving e.g. LPs, QPs, QCQPs and SOCPs independently of the problem conditioning and of the particular feasible sets within a nearly constant number of iterations with practically no tuning effort [19]. The focus in this thesis are structure exploitation techniques in the search direction computation of IPMs, which are the key to an efficient solution on embedded systems. In particular, we improve on the results from [112, 136] for linear MPC regulation problems and extend the structure exploitation to the more general class of multistage problems with convex quadratic constraints and to sparse SOCPs, which together cover a variety of sophisticated MPC formulations.

Examples requiring the use of quadratic constraints include formulations providing stability guarantees via ellipsoidal terminal sets [91], real-time stability guarantees in face of early termination of the solver [143] or robustness against modeling errors and noise in a tube MPC setting [90]. Moreover, some systems have inherent quadratic system constraints for which polytopic sets provide only a coarse approximation, such as power electronics [11]. Second-order cone constraints are used in robust MPC via affine disturbance feedback policies [55], for example, or in problems that naturally give rise to these type of nonlinear constraints such as antenna array weight design [81]. With the solvers presented in this thesis, these problems can efficiently be solved on resource constrained embedded platforms.

Automatic code generation is a recently proposed technique for the implementation of numerical optimization software on embedded systems, which exploits the fact that an optimization problem with fixed dimensions and structure is solved repeatedly for varying instances of the problem data. The key advantage of code generation is that a significant amount of time can be spent at code generation time to analyze the problem and to create problem-specific code that exploits all problem structure. For small problem sizes, C code generation of primal-dual IPMs with micro- to millisecond solution times (CVXGEN, [87]) has been demonstrated, and the ACADO toolkit [66] offers code generation for nonlinear MPC problems based on active set methods. Both systems are however limited to solving (a sequence of) QPs. There are currently two implementations of C code generators based on first order methods: μAO-MPC [145, 146] for linear MPC problems formulated as QPs, and FiOrdOs [128] for general convex problems, which also supports SOCPs. In this thesis, we present the first code generator that creates tailored primal-dual interior point solvers supporting quadratic constraints and without limitations on the problem size.

1.1 Outline and Contribution

Part I introduces the background material that is relevant for the main parts of this thesis. In Chapter 2, basic mathematical definitions and concepts from the field of convex optimization are reviewed, and a brief overview of second-order methods for solving convex problems is given, in particular of primal barrier and primal-dual interior point methods. In Chapter 3, the linear systems that arise in interior point methods are presented, and the corresponding direct solution methods from numerical linear algebra are briefly introduced. These methods form the basis for the computational techniques of the interior point methods in Chapter 8 and Chapter 9. The framework of model predictive control is introduced in Chapter 4, and a brief introduction to the employed machine learning methods is presented in Chapter 5.

Part II focuses on the first contribution of this thesis, the use of modern machine learning methods to approximate optimal controllers from simulation data. The proposed method allows for significantly reducing the computational burden for online evaluation of the control law while achieving a near-optimal performance and constraint satisfaction. This part consists of two chapters: In Chapter 6, learning of feedback policies for continuous inputs is considered, where the goal is to synthesize learned controllers that ensure stability and constrained satisfaction in closed loop. To the best of our knowledge, these are the first results on learning under system theoretic constraints that yield a tractable convex program. Chapter 7 focuses on learning binary decision rules with application to building control. It is shown how to extract meaningful if-then-else rules, which are easily implemented on low cost platforms, from simulation data with a complex hybrid MPC controller. In the presented simulation study, this approach yields a significant complexity reduction while maintaining near-optimal performance.

Part III presents two innovative approaches for solving convex problems on embedded systems. In Chapter 8, we focus on the class of so-called multistage problems, which are convex problems with a specific stage-wise structure capturing a large class of applications, including MPC and moving horizon estimation (MHE) problems with linear time varying dynamics, portfolio optimization problems and spline optimization problems. A tailored interior point method for multistage problems is derived, and significant speedups both in theory and practice are demonstrated. Results for parallelization of the proposed method targeting multi-core architectures and field-programmable gate arrays (FPGAs) are presented. One of the main contributions of this part is to make the solution of multistage problems on embedded platforms both *feasible* (in terms of computing time), and *publicly available* through a code generation system that generates tailored solvers for use on embedded platforms (cf. Section 1.3). To the best of our knowledge, the generated interior point solvers are the first to support quadratic constraints on embedded platforms, reducing the gap between

theoretically desired MPC formulations and practically solvable problems.

In Chapter 9, a new approach for solving convex conic programs with second-order cone constraints is presented, which allows for a small and simple implementation desirable for embedded platforms. Despite its small code size of about 800 lines of ANSI C code, the presented solver achieves computation times that are competitive even to commercial solvers. Numerical stability is obtained by a novel sparse expansion of the KKT system, which can be provably factored in a numerically stable way for accuracies beyond what is typically needed in embedded applications.

1.2 Publications

The work presented in this thesis was done in collaboration with colleagues and is largely based on previous publications, which are listed in the following.

Chapter 6 is entirely based on the following publication:

> *Learning a Feasible and Stabilizing Explicit Model Predictive Control Law by Robust Optimization.* A. Domahidi, M. N. Zeilinger, M. Morari, C. N. Jones, Proceedings of the 50th IEEE Conference on Decision and Control, Orlando, USA, pp. 513–519, Dec. 2011. [38]

Chapter 7 is an extension of the paper

> *Learning Near-optimal Decision Rules for Energy Efficient Building Control.* A. Domahidi, F. Ullmann, M. Morari, C. N. Jones, Proceedings of the 50th IEEE Conference on Decision and Control, Maui, USA, pp. 7571–7576, Dec. 2012. [37]

Parts of Chapter 8 are based on the results presented in the paper

> *Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control.* A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, C. N. Jones, Proceedings of the 50th IEEE Conference on Decision and Control, Maui, USA, pp. 668–674, Dec. 2012. [39]

The main ideas of Chapter 9 have been presented in the publication

> *ECOS: An SOCP Solver for Embedded Systems.* A. Domahidi, E. Chu, S. Boyd, Proceedings of the 12th biannual European Control Conference, Zurich, Switzerland, Jul. 2013 (to appear). [36]

Published work related to the topics of this thesis but not discussed or only cited is:

> *High-Bandwidth Explicit Model Predictive Control of Electrical Drives.* S. Mariéthoz, A. Domahidi, M. Morari, IEEE Transactions on Industry Applications, vol. 48, no. 6, pp. 1980–1992, Dec. 2012. [86]

Moving Horizon Estimation for Induction Motors. D. Frick, A. Domahidi, M. Vukov, S. Mariéthoz, M. Diehl, M. Morari, Proceedings of the IEEE International Symposium on Sensorless Controls for Electrical Drives, pp. 1–6, Sept. 2012. [47]

1.3 Software

The interior point solvers presented in Part III of this thesis are publicly available under the GNU General Public License version 3:

- The code generation system for multistage problems is available at `forces.ethz.ch`. Users can generate tailored ANSI C code that implements the methods presented in Chapter 8 and that runs on embedded platforms.
- An implementation of the second-order cone solver as presented in Chapter 9 is available in source code from `github.com/ifa-ethz/ecos`.

Part I

Preliminaries

Notation

We follow the notation of [19]: $\boldsymbol{R}$ denotes the set of real numbers, $\boldsymbol{R}_+$ the set of nonnegative real numbers, and $\boldsymbol{R}_{++}$ the set of positive real numbers. $\boldsymbol{R}^n$ denotes the set of column vectors of length n, and similarly, $\boldsymbol{R}^{m\times n}$ is the set of real-valued matrices with m rows and n columns. $\boldsymbol{S}^n_{++}$ and $\boldsymbol{S}^n_+$ denote the set of $n\times n$ positive definite and positive semi-definite matrices, respectively. If a matrix $P\in\boldsymbol{S}^n_{++}$, we use the notation $P\succ 0$, and if $P\in\boldsymbol{S}^n_+$ we use $P\succeq 0$. I_n is the $n\times n$ identity matrix. A composition of vectors is denoted by

$$(a,b,c)\triangleq\begin{bmatrix}a^T & b^T & c^T\end{bmatrix}^T. \tag{1.1}$$

$\mathrm{vec}\,(A)$ is the vector obtained by stacking the columns of matrix A. Similar to Matlab notation, if $x\in\boldsymbol{R}^n$ is a vector, $M=\mathrm{diag}(x)\in\boldsymbol{R}^{n\times n}$ denotes the matrix constructed from the elements of x such that $M_{ii}=x_i$ for all $i=1,\ldots,n$. The vector $\mathbf{0}_n$ denotes a column vector of n zeros, and the vector $\mathbf{1}_n$ is a column vector of n ones. The absolute value applied to a vector x, $|x|$, is understood to apply to each element of x and $|x|$ is therefore a vector of same dimension as x. Similarly, $|A|$ is a matrix of element wise absolute values of the matrix A. The expression x_i refers either to an iterate in a sequence $\{x_i\}$ or to the ith element of x; the distinction is clear from the context. A_i denotes the ith row of A. Consequently, $|A|_i$ is the ith row of $|A|$, for example. Given two sets $S_1,S_2\subseteq\boldsymbol{R}^n$, the Minkowski sum is defined as $S_1\oplus S_2\triangleq\{s_1+s_2|s_1\in S_1,s_2\in S_2\}$ and the Pontryagin difference as $S_1\ominus S_2\triangleq\{s|s+s_2\in S_1,s_2\in S_2\}=\{s|s\oplus S_2\subseteq S_1\}$. The operator $\otimes$ denotes the standard Kronecker product. For a twice differentiable function $f:\boldsymbol{R}^n\to\boldsymbol{R}$, $\nabla f\in\boldsymbol{R}^n$ denotes the gradient of f and $\nabla^2 f\in\boldsymbol{R}^{n\times n}$ the Hessian (matrix) of f. The vector $\nabla f(x)$ is the gradient of f evaluated at the point $x\in\boldsymbol{R}^n$, and similarly, the matrix $\nabla^2 f(x)$ is the Hessian of f evaluated at x. A *polyhedron* is an intersection of a finite number of closed half-spaces in $\boldsymbol{R}^n$, and a *polytope* is a bounded polyhedron. $\mathrm{extr}(P)$ denotes the set of vertices of the polytope P. All inequalities are component-wise if applied to vectors. Variables with an asterisk * denote their optimal values (for optimization problems clear from the context).

2 Convex Optimization

The following material summarizes the main results from convex programming relevant for this thesis and defines formulations of optimization problems used throughout the thesis. See [19, Chap. 2-5] for a comprehensive overview of the basics of convex functions, sets and optimization problems.

2.1 Basics on Convex Optimization

Throughout this thesis, we consider variations of convex programs of the form

$$\begin{array}{rll} f^\star \triangleq & \text{minimize} & f(x) \\ & \text{subject to} & Ax = b \\ & & g(x) \leq 0\,, \end{array} \tag{2.1}$$

where $A \in \boldsymbol{R}^{p\times m}$ and $b \in \boldsymbol{R}^p$, $x \in \boldsymbol{R}^n$ are the primal variables and the functions $f : \boldsymbol{R}^n \to \boldsymbol{R}$ and $g : \boldsymbol{R}^n \to \boldsymbol{R}^m$ are convex. The matrix A is assumed to have full row rank. The feasible set of (2.1) is defined as

$$\mathbb{Q} \triangleq \{x \in \boldsymbol{R}^n \mid Ax = b, g_i(x) \leq 0, i = 1, \ldots, m\}. \tag{2.2}$$

If $\mathbb{Q} = \emptyset$, problem (2.1) is said to be (primal) infeasible. If there exists some $x \in \mathbb{Q}$ the problem is said to be feasible, and if $g_i(x) < 0$ for some $1 \leq i \leq m$, the problem is *strictly* feasible. If the optimal value $f^\star$ is $-\infty$, the problem is unbounded. A point $x^\star \in \mathbb{Q}$ is called a minimizer if $f^\star \equiv f(x^\star)$.

Lagrange Duality

The function

$$d(y, z) \triangleq \inf_x f(x) + y^T(Ax - b) + z^T g(x) \tag{2.3}$$

is called the *Lagrange dual function* associated with (2.1) and has the domain $\boldsymbol{R}^p \times \boldsymbol{R}^m$. The vector $y \in \boldsymbol{R}^p$ is called *Lagrange multiplier* for the equality constraints $Ax = b$; similarly, the vector $z \in \boldsymbol{R}^m_+$ is the Lagrange multiplier associated with the inequality constraints.

If (2.1) is bounded below, (2.3) can be used as a lower bound on f by restricting z to the positive orthant: $d(y,z) \leq f(x)$ for all $x \in \mathbb{Q}$ and $(y,z) \in \boldsymbol{R}^p \times \boldsymbol{R}^m_+$. This is called weak duality. The best lower bound is

$$\begin{aligned} d^\star \triangleq \max\ & d(y,z) \\ \text{s.t. } & z \geq 0 \end{aligned} \tag{2.4}$$

Problem (2.4) is called the dual problem to (2.1). The *duality gap* is given by $f^\star - d^\star \geq 0$. Under certain conditions on the inequality constraints in (2.1) (constraint qualifications), the optimal values of the primal (2.1) and dual (2.4) problems coincide: $f^\star \equiv d^\star$. This is called strong duality. One such qualification is Slater's condition: if (2.1) is strictly feasible, strong duality holds [19, §5.2.3].

Optimality Conditions

The Karush-Kuhn-Tucker (KKT) optimality conditions for (2.1) are necessary and sufficient conditions for optimality for the primal-dual pair (2.1) and (2.4):

$$\nabla f(x) + A^T y + G(x)^T z = 0 \tag{2.5a}$$
$$Ax = b \tag{2.5b}$$
$$g(x) + s = h \tag{2.5c}$$
$$s_i z_i = 0, \ \forall i = 1, \ldots, m \tag{2.5d}$$
$$(s,z) \geq 0 \tag{2.5e}$$

The vector $s \in \boldsymbol{R}^m_+$ denotes nonnegative slack variables introduced for the inequality constraints $g(x) \leq 0$, and the matrix $G(x) \in \boldsymbol{R}^{m \times n}$ denotes the Jacobian of g evaluated at x. If vectors $(x^\star, y^\star, s^\star, z^\star)$ are found such that (2.5) holds, $x^\star$ is a minimizer for (2.1), and $(y^\star, z^\star)$ are maximizers for (2.4).

In the following, we discuss important instances of (2.1).

2.1.1 Convex Problems with Linear Inequalities

2.1.1.1 Linear Programming

A linear program (LP) is an instance of (2.1) where a linear function is minimized over a polyhedral feasible set:

Primal LP:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & Gx \leq h \end{array} \qquad \text{(LP)}$$

Dual LP:

$$\begin{array}{ll} \text{maximize} & -y^T b - z^T h \\ \text{subject to} & A^T y + G^T z + c = 0 \\ & z \geq 0 \end{array} \qquad \text{(LP-D)}$$

with $c \in \boldsymbol{R}^n$, $G \in \boldsymbol{R}^{m \times n}$ with full row rank, $h \in \boldsymbol{R}^m$ and A and b defined as above.

2.1.1.2 Quadratic Programming

A quadratic program (QP) is an instance of (2.1) where a quadratic function is minimized over a polyhedral feasible set. The primal problem is given by

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^T H x + c^T x \\ \text{subject to} & Ax = b \\ & Gx \leq h \end{array} \qquad \text{(QP)}$$

where $H \in \boldsymbol{R}^{n \times n}$ is positive semi-definite, and c, G, h, A and b are defined as above. For $H \succ 0$, the dual problem to (QP) is

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2} \begin{bmatrix} y \\ z \end{bmatrix}^T \begin{bmatrix} AH^{-1}A^T & AH^{-1}G^T \\ GH^{-1}A^T & GH^{-1}G^T \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} AH^{-1}c + b \\ GH^{-1}c + h \end{bmatrix}^T \begin{bmatrix} y \\ z \end{bmatrix} \\ \text{subject to} & z \geq 0 \end{array} \qquad \text{(QP-D)}$$

and the primal solution can be recovered from the optimal values $(y^\star, z^\star)$ of (QP-D) using

$$x^\star = -H^{-1} \left(c + A^T y^\star + G^T z^\star \right). \tag{2.6}$$

2.1.2 Conic Programming

All of Section 2.1 can be generalized to the setting of minimizing a linear function over convex pointed cones. Such a cone program can be written syntactically very similar to (LP):

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & Gx \preceq_{\boldsymbol{K}} h \end{array} \qquad \text{(CP)}$$

where the inequality is a generalized inequality with respect to the pointed convex cone $\boldsymbol{K}$ with nonempty interior, i.e. the cone is *proper*:

$$Gx \preceq_{\boldsymbol{K}} h \quad \Leftrightarrow \quad h - Gx \in \boldsymbol{K} \quad \Leftrightarrow \quad Gx + s = h,\ s \in \boldsymbol{K}. \tag{2.7}$$

The cone $\boldsymbol{K}$ is a direct product of N proper cones $\boldsymbol{K}_i$:

$$\boldsymbol{K} \triangleq \boldsymbol{K}_1 \times \boldsymbol{K}_2 \times \cdots \times \boldsymbol{K}_N. \tag{2.8}$$

Examples for $\boldsymbol{K}_i$ of interest are the positive orthant $\boldsymbol{R}_+$ (i.e. if $\boldsymbol{K} \equiv \boldsymbol{R}^n_+$, the standard linear program (LP) is recovered), the set of symmetric positive definite matrices $\boldsymbol{S}^n_{++}$ and the second-order cone of size n, which we define as

$$\boldsymbol{Q}^n \triangleq \left\{(u_0, u_1) \in \boldsymbol{R} \times \boldsymbol{R}^{n-1} \mid u_0 \geq \|u_1\|_2\right\} \tag{SOC}$$

for $n > 1$ and $\boldsymbol{Q}^1 \triangleq \boldsymbol{R}_+$. The dual problem associated with (CP) is

$$\begin{array}{ll} \text{maximize} & -b^T y - h^T z \\ \text{subject to} & G^T z + A^T y + c = 0 \\ & z \succeq_{\boldsymbol{K}^*} 0\,, \end{array} \tag{CP-D}$$

where $\boldsymbol{K}^*$ denotes the dual cone to $\boldsymbol{K}$:

$$\boldsymbol{K}^* \triangleq \{u \mid \langle u, x\rangle \geq 0 \ \forall x \in \boldsymbol{K}\}\,, \tag{2.9}$$

with $\langle \cdot\,, \cdot \rangle$ being an appropriately defined inner product. See [19, §2.6] for details on dual cones. For self-dual cones, $\boldsymbol{K} = \boldsymbol{K}^*$, i.e. the cone and its dual coincide; the three aforementioned examples, the positive orthant $\boldsymbol{R}_+$, the second-order cone $\boldsymbol{Q}^n$ and the set of symmetric positive definite matrices $\boldsymbol{S}^n_{++}$ are self-dual cones. The KKT conditions for (CP) are very similar to those of (LP) – the only difference is that slacks s have to belong to the cone $\boldsymbol{K}$ and the multipliers z to the dual cone $\boldsymbol{K}^*$. Since we consider only self-dual cones, the optimality conditions for (CP) are

$$c + A^T y + G^T z = 0 \tag{2.10a}$$
$$Ax - b = 0 \tag{2.10b}$$
$$Gz + s - h = 0 \tag{2.10c}$$
$$s^T z = 0 \tag{2.10d}$$
$$s, z \in \boldsymbol{K}\,. \tag{2.10e}$$

2.1.2.1 Equivalence to General Conic Programming Format

Another common format of (CP) used in the literature and e.g. in the conic solver SeDuMi [121] is

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \boldsymbol{R}^f \times \boldsymbol{K} \end{array} \tag{2.11}$$

	(CP)	(2.12)	(CP-D)	(2.11)
Variables	x	y	y	x_f
	s	z	z	$x_{\boldsymbol{K}}$
Data	c	b	b	c_f
	G	$A_{\boldsymbol{K}}^T$	A^T	A_f
	h	$c_{\boldsymbol{K}}$	G^T	$A_{\boldsymbol{K}}$
	A	A_f^T	h	$c_{\boldsymbol{K}}$
	b	c_f	$-c$	b

Table 2.1: Correspondence of the variables in the two standard forms for conic programming (CP) and (2.11) and the associated dual problems (2.12) and (CP-D).

where $\boldsymbol{K}$ is again the cone in (2.8). The first f entries of x are *free* variables; for notational convenience, let us denote this part of x by x_f and the remainder by $x_{\boldsymbol{K}}$, i.e. $x \triangleq (x_f, x_{\boldsymbol{K}})$. The dual to (2.11) is

$$\begin{array}{ll} \text{maximize} & -b^T y \\ \text{subject to} & A_f^T y = c_f \\ & A_{\boldsymbol{K}}^T y + z = c_{\boldsymbol{K}} \\ & z \in \boldsymbol{K} \end{array} \tag{2.12}$$

where c_f, $c_{\boldsymbol{K}}$ and A_f, $A_{\boldsymbol{K}}$ are the parts of c and A that correspond to x_f and $x_{\boldsymbol{K}}$ respectively, that is $c \triangleq (c_f, c_{\boldsymbol{K}})$ and $A \triangleq [A_f\, A_{\boldsymbol{K}}]$ respectively. The dual problem (2.12) has the form of our standard primal problem (CP), and the primal problem (2.11) has the form of our standard dual problem (CP-D), with the correspondences listed in Table 2.1. Usually, conic solvers solve both primal and dual problems, and therefore one can easily map a problem and its solution between the two formulations.

2.1.3 Symmetric Cones

Symmetric cones (another term is *self-scaled* cones, coined by Nesterov and Todd in [100]) have special properties that allows one to construct very efficient primal-dual interior point algorithms, and to describe them in a concise way making use of Euclidean Jordan algebra. The cones of interest in this thesis, the positive orthant $\boldsymbol{R}_+$ and the second-order cone $\boldsymbol{Q}^n$, are symmetric. We summarize the most important results as needed for the description of interior point methods from [43, 58].

Definition 2.1 (Euclidean Jordan Algebra [58, Definition 2.1])**.** *Let $\mathcal{J}$ be a finite dimensional real algebra along with a bilinear map $\circ : \mathcal{J} \times \mathcal{J} \to \mathcal{J}$. Then $(\mathcal{J}, \circ)$ is called a Euclidean Jordan algebra if for all $x, y \in \mathcal{J}$ the following holds:*

(i) $x \circ y = y \circ x$ *(Commutativity)*
(ii) $x \circ (x^2 \circ y) = x^2 \circ (x \circ y)$, *where* $x^2 \triangleq x \circ x$ *(Jordan's Axiom)*
(iii) $x^T(y \circ z) = (x \circ y)^T z$

Definition 2.2 (Symmetric Cone [43, § III.2-5])**.** *A cone is symmetric if and only if it is the cone of squares*

$$\{x^2 \mid x \in \boldsymbol{R}^n\} \tag{2.13}$$

of some Euclidean Jordan algebra.

The second-order cone $\boldsymbol{Q}^n$ (and therefore the positive orthant since $\boldsymbol{Q}^1 = \boldsymbol{R}_+$) and the set of symmetric positive definite matrices $\boldsymbol{S}^n_{++}$ belong to the five[1] existing symmetric cones. Direct products of these primitive cones are also symmetric.

Definition 2.3 (Rank of Symmetric Cones [43])**.** *The* rank M *of a symmetric cone* $\boldsymbol{K}$ *is defined as*

$$R \triangleq \sum_{i=1}^{N} R_i\,, \quad \textit{where} \quad R_i = \begin{cases} 1 & \textit{if } \boldsymbol{K}_i = \boldsymbol{Q}^1 = \boldsymbol{R}_+ \\ 2 & \textit{if } \boldsymbol{K}_i = \boldsymbol{Q}^{n_i},\, n_i > 1 \end{cases}. \tag{2.14}$$

Vector Product for Second-Order Cone

For the second-order cone $\boldsymbol{Q}^n$, we use the vector product

$$x \circ y \triangleq (x^T y,\ x_0 y_1 + y_0 x_1)\,, \tag{2.15}$$

where $(x, y) \in \boldsymbol{Q}^n \times \boldsymbol{Q}^n$. Note that (2.15) collapses to the usual product for the positive orthant, i.e. $x \circ y = xy$ for $(x, y) \in \boldsymbol{Q}^1 \times \boldsymbol{Q}^1$. If $(x, y) \in \boldsymbol{K}$ with $\boldsymbol{K}$ being a direct product of N second-order cones, the vector product $\circ$ is understood to be applied to each cone separately and then stacked to yield the final vector product (consequently, this yields for example $x \circ y = \operatorname{diag}(x)y$ for $(x, y) \in \boldsymbol{R}^n_+ \times \boldsymbol{R}^n_+$). The identity element $\boldsymbol{e}$ for the vector product $\circ$ is defined as

$$\boldsymbol{e} \circ x = x \circ \boldsymbol{e} = x\,, \quad \text{where} \quad \boldsymbol{e} \triangleq (1, \boldsymbol{0}_{n-1}) \tag{2.16}$$

for $x \in \boldsymbol{Q}^n$. Again, $\boldsymbol{e}$ is the usual identity element 1 if $x \in \boldsymbol{Q}^1$, and if $x \in \boldsymbol{K}$, the identity element is a stacked vector of identity elements for the individual cones, as illustrated by the following example.

[1] The other three symmetric cones are: the set of positive definite matrices with complex and quaternion entries and the set of 3×3 positive semidefinite matrices with octonion entries.

Example 2.1 (Composition of Identity Elements)**.** *Let* $\boldsymbol{K} = \boldsymbol{Q}^1 \times \boldsymbol{Q}^1 \times \boldsymbol{Q}^1 \times \boldsymbol{Q}^3$*. The corresponding identity element for* $\boldsymbol{K}$ *is*

$$\boldsymbol{e} = ((1), (1), (1), (1, 0, 0)) = (1, 1, 1, 1, 0, 0). \tag{2.17}$$

Definition 2.4 (Degree of Symmetric Cones [129, §2])**.** *We define the* degree D *of a symmetric cone* $\boldsymbol{K}$ *as*

$$D \triangleq \boldsymbol{e}^T \boldsymbol{e}. \tag{2.18}$$

Remark 2.1. *As a direct consequence of Definition 2.4 and our definition of the identity element* $\boldsymbol{e}$ *in* (2.16)*, the degree of a symmetric cone* $\boldsymbol{K}$ *as defined in* (2.8) *is*

$$D = \boldsymbol{e}^T \boldsymbol{e} = N, \tag{2.19}$$

i.e. it is simply equal to the number of cones constituting the product cone.

Remark 2.2. *The vector product* $\circ$ *and the identity element* $\boldsymbol{e}$ *can similarly be defined for the set of symmetric positive definite matrices, and consequently the subsequent results also hold for the cone* $\boldsymbol{S}^n_{++}$*. See e.g. [129] for the corresponding definitions.*

The advantage of representing symmetric cones in terms of Euclidean Jordan algebra is that properties can be studied using elementary concepts from algebra. Most importantly, one can define an eigenvalue decomposition not only for matrices, but also for vectors from the second-order cone $\boldsymbol{Q}^n$.

Spectral Decomposition

It can be shown that for each symmetric cone $\boldsymbol{K}$ and the associated vector product $\circ$, there exists a decomposition of any element $x \in \boldsymbol{K}$ such that it can be written as the linear combination of R orthogonal "eigenvectors" q_i [43]:

$$x = \sum_{i=1}^{R} \lambda_i q_i, \tag{2.20}$$

where the coefficients of the linear combination are the "eigenvalues" λ_i. Thus (2.20) can be interpreted as a spectral decomposition of $x \in \boldsymbol{K}$, where the set of M eigenvectors satisfies

$$q_i \circ q_i = q_i\,, i = 1, \ldots, R \qquad q_i \circ q_j = 0\,, i \neq j\,, \qquad \sum_{i=1}^{R} q_i = \boldsymbol{e}, \tag{2.21}$$

which is called a Jordan frame [58]. The spectral decomposition for $x \in \boldsymbol{Q}^1$ is trivial, with $q_1 = 1$ being the only eigenvector and $\lambda_1 = x$ being the eigenvalue. For $x \in \boldsymbol{Q}^n$ with $n > 1$, the spectral decomposition is

$$\lambda_i = x_0 \pm \|x_1\|_2\,, \quad q_i = \frac{1}{2}\left(1,\ \pm x_1/\|x_1\|_2\right)\,, \quad i = 1, 2\,. \tag{2.22}$$

With this spectral decomposition in place, we can express some useful relations which are given in (2.23).

Property	Definition	Cone $\boldsymbol{R}_+$	Cone $\boldsymbol{Q}^n$, $n > 1$	
$x \in \boldsymbol{Q}^n$	$\Leftrightarrow \lambda_i \geq 0\,,\ \forall i$	$x \geq 0$	$x_0 \pm \|x_1\|_2 \geq 0$	(2.23a)
$x \in \operatorname{int} \boldsymbol{Q}^n$	$\Leftrightarrow \lambda_i > 0\,,\ \forall i$	$x > 0$	$x_0 \pm \|x_1\|_2 > 0$	(2.23b)
Inverse x^{-1} s.t. $x \circ x^{-1} = \boldsymbol{e}$	$x^{-1} \triangleq \sum_{i=1}^{R} \lambda_i^{-1} q_i$	$x^{-1} = 1/x$	$x^{-1} = (x_0, -x_1)/\det(x)$	(2.23c)
Determinant: $\det(x)$	$\det(x) \triangleq \prod_{i=1}^{R} \lambda_i$	$\det(x) = x$	$\det(x) = x_0^2 - x_1^T x_1$	(2.23d)

The inverse and determinant are important for interior point algorithms for optimization over symmetric cones, since they allow to construct logarithmic barriers and to define the central path in a symmetric way. We detail on this in Section 2.2.2. For more details on properties of symmetric cones see [43].

2.1.4 Self-Dual Embeddings

2.1.4.1 Homogeneous Self-Dual Embedding

In the following, we discuss the embedding of (CP) and (CP-D) into a lifted problem (cf. [5, 7, 141])

$$\begin{aligned} \min\ & 0 \\ \text{s.t. } & \tau c + A^T y + G^T z = 0 \\ & Ax - \tau b = 0 \\ & Gx + s - \tau h = 0 \\ & c^T x + b^T y + h^T z + \kappa = 0 \\ & (s, z) \succeq_{\boldsymbol{K}} 0\,, \quad (\tau, \kappa) \geq 0 \end{aligned} \tag{HSD}$$

with two additional scalar variables τ and κ. Problem (HSD) is self-dual, i.e. the dual of (HSD) is structurally the same problem (see Lemma A.1 in the Appendix for a straightforward proof), there always exists a feasible solution, as the point $(x, y, z, s, \tau, \kappa) = 0$ trivially satisfies the constraints. The main advantage of problem (HSD) is that it allows for detecting infeasibility of the primal cone program (CP) or its dual (CP-D).

Lemma 2.1 (Certificates of optimality or infeasibility)**.** *Let a feasible point* $(x, y, z, s, \tau, \kappa)$ *for* (HSD) *be given. Then the following certificates can be provided:*

1. $\tau > 0$, $\kappa = 0$: *Certificate of optimality.*
2. $\tau = 0$, $\kappa > 0$: *Certificate of primal infeasibility if* $h^T z + b^T y < 0$, *or certificate of dual infeasibility if* $c^T x < 0$.
3. $\tau = 0$, $\kappa = 0$: *None.*

Proof. We prove the first two statements:

1. $\tau > 0$, $\kappa = 0$: The scaled point
$$(x^\star, y^\star, z^\star, s^\star) = (x, y, s, z)/\tau. \tag{2.24}$$
satisfies the optimality conditions (2.10) and is therefore optimal, i.e. $(x^\star, s^\star)$ is optimal for (CP) and $(y^\star, z^\star)$ is optimal for (CP-D).
2. $\tau = 0$, $\kappa > 0$: From $\kappa > 0 \Rightarrow c^T x + h^T z + b^T y < 0$, i.e. either $h^T z + b^T y < 0$ or $c^T x < 0$ (or both).
   (i) $h^T z + b^T y < 0$: From $\tau = 0$ we have $G^T z + A^T y = 0$. By the theorem of alternatives (cf. Lemma A.2 in the Appendix) the primal constraints $Gx + s = h, Ax = b, s \succeq_K 0$ cannot have a feasible solution. Hence (CP) is infeasible.
   (ii) $c^T x < 0$: From $\tau = 0 \Rightarrow Gx + s = 0$, $Ax = 0$, $s \succeq_K 0$ and then by the theorem of alternatives the dual constraint $G^T z + A^T y + c = 0$ cannot have a feasible solution.

□

However, interior point methods cannot be directly applied to formulation (HSD) as stated in the following lemma.

Lemma 2.2. *Problem* (HSD) *does not have* strictly *feasible points.*

Proof. Rewriting the equality constraints of (HSD) as
$$\begin{bmatrix} 0 \\ 0 \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \tau \end{bmatrix} \tag{2.25}$$
and multiplying both sides by $(x, y, z, \tau)^T$ yields $z^T s + \tau\kappa = 0$ due to the skew-symmetric coefficient matrix. Hence both z and s as well as τ and κ must satisfy a complementary condition: $z_i s_i = 0 \ \forall i$ and $\tau\kappa = 0$. As a result, all feasible points lie on the boundary of the feasible set. □

The homogeneous embedding can be extended to a problem that has strictly feasible points and is therefore amenable to interior-point methods, as will be shown next.

2.1.4.2 Extended Self-Dual Embedding

We follow [129] to construct an extended self-dual embedding that does have strictly feasible points by introducing an additional variable θ in problem (HSD):

$$\begin{aligned} &\min\ (D+1)\theta \\ &\text{s.t.}\ \begin{bmatrix} 0 \\ 0 \\ s \\ \kappa \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & A^T & G^T & c & q_x \\ -A & 0 & 0 & b & q_y \\ -G & 0 & 0 & h & q_z \\ -c^T & -b^T & -h^T & 0 & q_\tau \\ -q_x^T & -q_y^T & -q_z^T & -q_\tau & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \tau \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ D+1 \end{bmatrix} \\ &\quad (s,z) \succeq_{\boldsymbol{K}} 0\ ,\quad (\kappa,\tau) \geq 0 \end{aligned} \tag{ESD}$$

where D is the degree of $\boldsymbol{K}$ from Definition 2.4, and we define

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ q_\tau \end{bmatrix} = \frac{D+1}{s_0^T z_0 + 1} \left(\begin{bmatrix} 0 \\ 0 \\ s_0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \right) \tag{2.26}$$

for arbitrary initial points x_0, y_0, $s_0 \succ_{\boldsymbol{K}} 0$ and $z_0 \succ_{\boldsymbol{K}} 0$. The vectors q_x, q_y, q_z, q_τ are taken as the initial residuals of the homogeneous problem (HSD) for $\kappa_0 = \tau_0 = 1$ scaled by the inverse of an averaged value of the complementarity condition. As a result of this definition, taking the inner product on both sides of the equality constraint in (ESD) with (x, y, z, τ, θ) yields

$$\theta = \frac{s^T z + \kappa\tau}{D+1} \geq 0\,, \tag{2.27}$$

which is equal to the average value of the complementarity gap as defined in interior point methods. Problem (ESD) has strictly feasible points with $s^T z + \kappa\tau > 0$, and

$$(x, y, z, s, \kappa, \tau, \theta) = \left(x_0, y_0, z_0, s_0, 1, 1, \frac{s_0^T z_0 + 1}{D+1} \right) \tag{2.28}$$

is a strictly feasible point for (ESD) if $(s_0, z_0) \succ_{\boldsymbol{K}} 0$. Due to the minimization of θ (cf. (2.27)),

$$s^T z + \kappa\tau = 0 = \theta^\star \tag{2.29}$$

is recovered at the optimal point of (ESD), which then equals the homogeneous embedding (HSD). As a result, the certificates of optimality or infeasibility in Lemma 2.1 are valid in case $\kappa + \tau > 0$ [129]. The extended embedding therefore offers the advantage of detecting infeasibility, while it can be solved by interior-point methods and will therefore be employed as the basis of the path following primal-dual interior point method used in Chapter 9.

2.2 Second-Order Methods for Convex Programming

In this section, we present some numerical methods for finding a *minimizer* $x^\star$ to the optimization problem (2.1). Except in special cases, analytically finding a minimizer $x^\star$ is not possible. Hence, one has to resort to numerical methods that compute an approximate minimizer that is "good enough" (a precise definition of this term is given in (2.31)). Numerical methods proceed in an *iterative* manner, i.e. starting from an initial guess x_0, a sequence $\{x_i\}_{i=0}^{i=i_{\max}}$ is computed, where

$$x_{i+1} = \Psi\left(x_i, f, \mathbb{Q}\right), \tag{2.30}$$

with Ψ being an update rule that depends on the particular optimization method. A method should terminate after a finite number of $i_{\max}$ iterations and return an approximate minimizer $x_{i_{\max}}$ that satisfies

$$|f\left(x_{i_{\max}}\right) - f\left(x^\star\right)| \leq \epsilon \quad \text{and} \quad \operatorname{dist}(x_{i_{\max}}, \mathbb{Q}) \leq \delta, \tag{2.31}$$

where

$$\operatorname{dist}(x, \mathbb{Q}) \triangleq \min_{y \in \mathbb{Q}} \|y - x\| \tag{2.32}$$

is the shortest distance between a point and a set in $\boldsymbol{R}^n$ measured by the norm $\|\cdot\|$. The parameters ϵ, δ define the required accuracy of an approximate minimizer with respect to optimality (parameter ϵ) and feasibility (parameter δ).

There exist two general classes of numerical optimization methods for solving (2.1): first-order methods that make use of first-order information (subgradients or gradients) and second-order methods that additionally make use of second-order information (Hessian matrix). In this thesis, we focus on second-order methods, which are suitable for a wide range of optimization problems and therefore restrict the discussion in the following to these type of methods. See [113] for a detailed treatment of first-order methods in the context of embedded optimization and control.

In the following, we briefly review Newton's method as a second-order method for solving unconstrained optimization problems. Newton's method is the basis for interior point methods for constrained optimization, for which we present the background in Section 2.2.2 and that are subject of the thesis in Chapters 8 and 9.

2.2.1 Newton's Method for Unconstrained Optimization

In this section, consider the problem

$$\min_x \ f(x), \tag{2.33}$$

where $f : \boldsymbol{R}^n \to \boldsymbol{R}$ is convex and *twice continuously differentiable*. We assume that a minimizer $x^\star \in \boldsymbol{R}^n$ of (2.33) exists and denote the optimal value as $f^\star$.

Descent Methods

A popular class of methods for unconstrained optimization are *descent methods*, which obtain the next iterate x_{i+1} from the current iterate x_i by taking a step of size $h_i > 0$ along a *descent direction* Δx_i:

$$x_{i+1} = x_i + h_i \Delta x_i \,. \tag{2.34}$$

A descent direction Δx_i satisfies

$$\nabla f(x_i)^T \Delta x_i < 0 \;\forall x_i \neq x^\star \,. \tag{2.35}$$

Available methods differ in the way Δx_i and h_i are computed. The step sizes h_i have to be chosen carefully in order to ensure a sufficiently large decrease in the function value, which is crucial for convergence of descent methods. Since solving for the optimal step size,

$$h_i^\star \in \arg\min_{h \geq 0} f(x_i + h\Delta x_i), \tag{2.36}$$

is generally too expensive, *inexact line search* methods are used instead. With these methods, a step size h_i is inexpensive to compute, while achieving a reasonably good decrement in the function value and ensuring convergence of the algorithm. A more detailed discussion of line search methods is provided in Section 2.2.2.5.

Newton's Method

In the following, we assume f to be μ-strongly convex, as defined in Definition 2.5 and Theorem 2.1 below.

Definition 2.5 (μ-Strong Convexity [65, §B, Thm. 4.1.1])**.** *Let $f : \boldsymbol{R}^n \to \boldsymbol{R}$ be once continuously differentiable on $\boldsymbol{R}^n$. Then f is strongly convex on $\boldsymbol{R}^n$ with convexity parameter $\mu > 0$ if and only if for all pairs $(x, y) \in \boldsymbol{R}^n \times \boldsymbol{R}^n$*

$$f(x) \geq f(y) + \nabla f(y)^T (x - y) + \frac{\mu}{2} \|x - y\|^2 \,.$$

Theorem 2.1 (Second-Order Characterization of Strong Convexity [65, §B, Thm. 4.3.1])**.** *Let $f : \boldsymbol{R}^n \to \boldsymbol{R}$ be twice continuously differentiable on $\boldsymbol{R}^n$. Then f is strongly convex on $\boldsymbol{R}^n$ with convexity parameter $\mu > 0$ if and only if*

$$\nabla^2 f(x) \succeq \mu I_n \,, \; \forall x \in \boldsymbol{R}^n \,.$$

As a trivial but important consequence of strong convexity, the Hessian $\nabla^2 f$ is positive definite on $\boldsymbol{R}^n$. We assume further that the Hessian $\nabla^2 f$ is Lipschitz continuous on $\boldsymbol{R}^n$ with Lipschitz constant M, i.e.

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq M \|x - y\| \;\forall x, y \in \boldsymbol{R}^n \tag{2.37}$$

Algorithm 2.1 Newton's method for (2.33) (strongly convex case)

Require: Initial iterate $x_0 \in \boldsymbol{R}^n$
1: **loop**
2: Newton direction: $\Delta x_i = -\left[\nabla^2 f(x_i)\right]^{-1} \nabla f(x_i)$
3: Line search to select step size h_i (cf. Section 2.2.2.5)
4: $x_{i+1} = x_i + h_i \Delta x_i$
5: **end loop**

Intuitively, the constant M measures how well f can be approximated by a quadratic function ($M = 0$ for a quadratic function f).

The main idea of Newton's method is to approximate the function f by a local quadratic model q based on a second-order Taylor expansion,

$$f(x_i + \Delta x_i) \approx f(x_i) + \nabla f(x_i)^T \Delta x_i + \frac{1}{2} \Delta x_i^T \nabla^2 f(x_i) \Delta x_i \triangleq q(x_i, \Delta x_i), \tag{2.38}$$

where Δx_i is a search direction which is chosen to minimize the quadratic model q. From the optimality condition $\nabla_{\Delta x_i} q(x_i, \Delta x_i) = 0$, the so-called *Newton direction* or *Newton step*

$$\Delta x_i \triangleq -\left[\nabla^2 f(x_i)\right]^{-1} \nabla f(x_i) \tag{2.39}$$

is obtained. By strong convexity, the inverse of the Hessian $[\nabla^2 f(x)]^{-1}$ exists for all $x \in \boldsymbol{R}^n$ and is positive definite. As a result, the Newton direction is a descent direction, satisfying (2.35):

$$\nabla f(x_i)^T \Delta x_i = -\nabla f(x_i)^T [\nabla^2 f(x_i)]^{-1} \nabla f(x_i) < 0 \tag{2.40}$$

for all $x_i \neq x^\star$. If the Lipschitz constant of the Hessian, M, is small, this method works very well, since the quadratic model q provides a good approximation even if a full Newton step is taken. However, in general, the quadratic model $q(x_i, \Delta x)$ is in general neither an upper nor a lower bound on f, and a line search (cf. Section 2.2.2.5) must be used to ensure a decrease in the function value, which results in global convergence of the algorithm. Newton's method is summarized in Algorithm 2.1. It can be shown to converge globally in two phases (for technical details see [19, §9.5] and [98, §1.2.4]):

1. *Damped Newton phase:* When the iterates are "far away" from the optimal point, the function value is decreased *sublinearly*, i.e. it can be shown that there exist constants η, γ (with $0 < \eta < \mu^2/M$ and $\gamma > 0$) such that $f(x_{i+1}) - f(x_i) \leq -\gamma$ for all x_i for which $\|\nabla f(x_i)\|_2 \geq \eta$. The term *damped* is used in this context since during this phase, the line search returns step sizes $h_i < 1$, i.e. no full Newton steps are taken.
2. *Quadratic convergence phase:* If an iterate is "sufficiently close" to the optimum, i.e. $\|\nabla f(x_i)\|_2 < \eta$, a full Newton step is taken, i.e. $h_i = 1$. The set $\{x \mid \|\nabla f(x)\|_2 <$

$\eta\}$ is called the *quadratic convergence zone*, and once an iterate x_i is in this zone, all subsequent iterates remain in it. The function value $f(x_i)$ converges *quadratically* (i.e. with $i_{\max} \sim \mathcal{O}(\ln\ln\epsilon)$) to $f^\star$.

The quadratic convergence property makes Newton's method to one of the most powerful algorithms for unconstrained optimization of twice continuously differentiable convex functions. As will be shown in the following, it is also employed in interior point methods for constrained optimization to solve the (unconstrained or equality constrained) subproblems in barrier methods or to generate primal-dual search directions. We discuss these in detail in Section 2.2.2.

A disadvantage of Newton's method is the need to form the Hessian $\nabla^2 f(x_i)$, which can be numerically ill-conditioned, and to solve the linear system

$$\nabla^2 f(x_i)\Delta x_N = -\nabla f(x_i) \tag{2.41}$$

for the Newton direction Δx_N, which in general costs $\mathcal{O}(n^3)$ floating point operations. More efficient methods can be obtained by exploiting sparsity or the specific problem structure; we discuss efficient solution methods for such linear systems in Chapters 8 and 9.

Preconditioning and Affine Invariance

Unlike first-order methods, Newton's method is *affine invariant*, i.e. a linear (or affine) transformation of variables,

$$x = Py \tag{2.42}$$

with P invertible, will lead to the same iterates subject to transformation, i.e.

$$x_i = Py_i \quad \forall i = 0, \ldots, i_{\max}. \tag{2.43}$$

Therefore, the number of steps to reach a desired accuracy of the solution does not change with preconditioning.

Equality Constraints

Equality constraints are naturally handled in Newton's method as follows. Consider problem (2.33) with equality constraints,

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b\,, \end{aligned} \tag{2.44}$$

with $A \in \boldsymbol{R}^{p\times n}$ (assuming full row rank of p) and $b \in \boldsymbol{R}^p$. The Newton direction is again given by the minimization of the quadratic model $q(\Delta x_i)$ at the current feasible iterate x_i

(i.e. $Ax_i = b$) with an additional equality constraint:

$$\begin{aligned} \Delta x_i \triangleq \arg\min_{\Delta x} \; & \frac{1}{2}\Delta x_i^T \nabla^2 f(x_i)\Delta x_i + \nabla f(x_i)^T \Delta x_i + f(x_i) \\ \text{s.t. } & A\Delta x_i = 0 \end{aligned} \tag{2.45}$$

Since Δx_i is restricted to lie in the nullspace of A, we ensure that any iterate generated by Newton's method will satisfy $Ax_{i+1} = A(x_i + h_i \Delta x_i) = b$. From the optimality conditions of (2.45), it follows that Δx_i is obtained as the solution to the linear system of dimension $n + p$:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_i \\ y \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}, \tag{2.46}$$

where $y \in \boldsymbol{R}^p$ are the Lagrange multipliers associated with the equality constraint $A\Delta x = 0$.

2.2.2 Interior Point Methods

Interior point methods are one of the most widely used numerical methods for solving convex optimization problems. This is the result of 25 years of intense research that has been initiated by Karmarkar's seminal paper in 1984 on solving LPs [73]. In 1994, Nesterov and Nemirovskii generalized interior point methods to nonlinear convex problems, including second-order cone and semi-definite programming [99]. Issues of the early days such as numerical ill-conditioning, efficient and stable solution of linear systems etc. are today well understood, and numerous both free and commercial codes are available that can solve linear and quadratic programs with high reliability.

We discuss barrier methods in Section 2.2.2.1, which were the first polynomial-time algorithms for linear programming with practical relevance. Modern primal-dual methods are presented in Section 2.2.2.2. This powerful class of interior point methods forms the basis of almost all current implementations today. In particular, all methods presented in Chapters 8 and 9 are based on a variant of Mehrotra's primal-dual method, which has proven highly efficient in practice.

Throughout this section, we consider problem (2.1), which we assume to be strictly primal and dual feasible.

2.2.2.1 Primal Barrier Methods

The main idea of the barrier method is to convert the constrained optimization problem (2.1) into an unconstrained problem (with respect to the inequality constraints) by means

of a *barrier function* $\Phi_g : \boldsymbol{R}^n \to \boldsymbol{R}$:

$$x^\star(\mu) \in \arg\min \quad f(x) + \mu\Phi_g(x) \quad \text{s.t.} \quad Ax = b \tag{$P(\mu)$}$$

where $\mu > 0$ is called *barrier parameter*. The purpose of the barrier function Φ_g is to "trap" an optimal solution of $P(\mu)$, which we denote by $x^\star(\mu)$, in the set $\mathbb{Q}$ (consequently, $\Phi_g(x)$ must take on the value $+\infty$ whenever $g_i(x) > 0$ for some i, and a finite value otherwise). As a result, $x^\star(\mu)$ is feasible with respect to $\mathbb{Q}$, but it differs from $x^\star$ since we have perturbed the objective function by the barrier term.

With the above idea in mind, we are now ready to outline the (primal) barrier method summarized in Algorithm 2.2. Starting from (a possibly large) μ_0, a solution $x^\star(\mu_0)$ to problem $P(\mu_0)$ is computed by Newton's method (for equality constrained problems) as presented in Section 2.2.1; hence Φ_g should be twice continuously differentiable. After $x^\star(\mu_0)$ has been computed, the barrier parameter is decreased by a constant factor, $\mu_1 = \mu_0/\kappa$ (with $\kappa > 1$), and $x^\star(\mu_1)$ is computed starting from $x^\star(\mu_0)$. This procedure is repeated until μ has been sufficiently decreased. It can be shown under mild conditions that $x^\star(\mu) \to x^\star$ from the interior of $\mathbb{Q}$ as $\mu \to 0$ [19, §11.3]. The points $x^\star(\mu)$ form the so-called *central path*, which is a continuously differentiable curve in $\mathbb{Q}$ that ends in the solution set (cf. Example 2.2). Solving a subproblem $P(\mu)$ is therefore called *centering*.

If the decrease factor κ is not too large, $x^\star(\mu_i)$ will be a good starting point for solving $P(\mu_{i+1})$, and therefore Newton's method is applied to $P(\mu_{i+1})$ to exploit the local quadratic convergence rate.

Logarithmic Barrier

So far, we have not specified which function to use as a barrier. For example, the indicator function

$$I_g(x) \triangleq \begin{cases} 0 & \text{if } g(x) \leq 0 \\ +\infty & \text{otherwise} \end{cases}, \tag{2.47}$$

trivially achieves the purpose of a barrier, but it is not practically meaningful since methods for smooth convex optimization as discussed in Section 2.2.1 require the barrier function to be convex and continuously differentiable. A twice continuously differentiable convex barrier function that is commonly used and approximates I_g well is the *logarithmic barrier function*

$$\Phi_g(x) \triangleq -\sum_{i=1}^{m} \ln\left(-g_i(x)\right), \tag{2.48}$$

Algorithm 2.2 Barrier interior point method for (2.1)

Require: Strictly feasible initial iterate x_0 with $g(x) < 0$, μ_0, $\kappa > 1$, tolerance $\epsilon > 0$
1: **loop**
2: **Centering step:**
Compute $x^\star(\mu_i)$ by solving $P(\mu_i)$ starting from the previous solution x_{i-1}
3: Update: $x_i = x^\star(\mu_i)$
4: Stopping criterion: Stop if $m\mu_i < \epsilon$
5: Decrease barrier parameter: $\mu_{i+1} = \mu_i/\kappa$
6: **end loop**

with domain $\{x \in \boldsymbol{R}^n \mid g_i(x) < 0 \ \forall \ i = 1, \ldots, m\}$ (i.e. the logarithmic barrier confines x to the *interior* of the set $\mathbb{Q}$), continuous gradient

$$\nabla \Phi_g(x) = \sum_{i=1}^{m} \frac{1}{-g_i(x)} \nabla g_i(x) \tag{2.49}$$

and continuous Hessian

$$\nabla^2 \Phi_g(x) = \sum_{i=1}^{m} \frac{1}{g_i(x)^2} \nabla g_i(x) \nabla g_i(x)^T + \frac{1}{-g_i(x)} \nabla^2 g_i(x), \tag{2.50}$$

which makes it possible to use Newton's method for solving the barrier subproblems. The logarithmic barrier function is used in almost all implementations of barrier methods. In fact, the existence of polynomial-time algorithms for convex optimization is closely related to the existence of barrier functions for the underlying feasible sets [99].

Example 2.2 (Central Path of an LP)**.** *The solutions of $P(\mu)$ for a linear programming problem in $\boldsymbol{R}^2$ for different values of μ are depicted in Fig. 2.1. For large μ, the level sets (dashed contour lines) are almost identical to the level sets of $\Phi_g(x)$, i.e. $x^\star(1000)$ is very close to the* analytic center *of the feasible set given by $x^a \triangleq \arg\min \Phi_g(x)$ of the feasible set. As μ decreases, $x^\star(\mu)$ approaches $x^\star$, which is located at the lower vertex of the polytope, by following the central path.*

Example 2.3 (Barrier method for QPs)**.** *In the following, we exemplify the barrier method for quadratic programs of the form*

$$\begin{array}{ll} \textit{minimize} & \frac{1}{2} x^T H x + f^T x \\ \textit{subject to} & Ax = b \\ & Gx \leq h \end{array} \tag{2.51}$$

with $H \succ 0$, $A \in \boldsymbol{R}^{p \times n}$, $b \in \boldsymbol{R}^p$, $G \in \boldsymbol{R}^{m \times n}$ and $h \in \boldsymbol{R}^m$. The logarithmic barrier function for linear inequalities takes the form

$$\Phi_g(x) \triangleq -\sum_{i=1}^{m} \ln(h_i - g_i^T x)\,, \tag{2.52a}$$

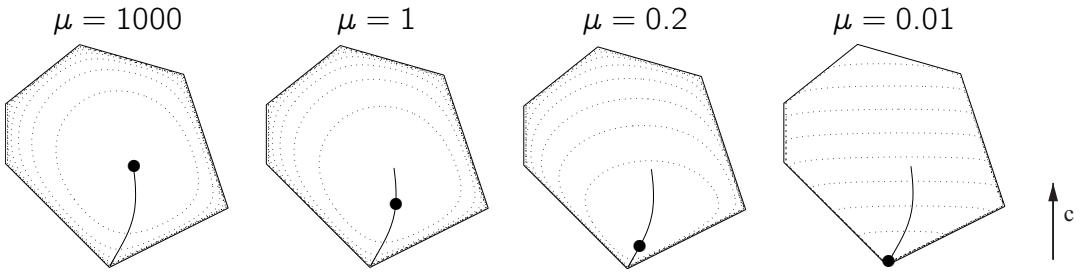


Figure 2.1: Illustration of the central path for an LP. The feasible set is a polytope, and the gradient c of the objective function $c^T x$ is pointing upwards. Each large dot corresponds to $x^\star(\mu)$ for different μ. The dashed lines indicate level sets of the augmented objective function $c^T x + \mu\Phi_g(x)$ using the logarithmic barrier (2.48).

with gradient

$$\nabla\Phi_g(x) = \sum_{i=1}^{m} \frac{1}{h_i - g_i^T x} g_i \,, \tag{2.52b}$$

and Hessian

$$\nabla^2\Phi_g(x) = \sum_{i=1}^{m} \frac{1}{(h_i - g_i^T x)^2} g_i g_i^T \,, \tag{2.52c}$$

where g_i^T denotes the ith row of G and h_i the ith element of h. Hence $P(\mu)$ takes the form

$$\begin{aligned} \min\ & \frac{1}{2} x^T H x + f^T x - \mu \sum_{i=1}^{m} \ln(h_i - g_i^T x) \\ \text{s.t. } & Ax = b \end{aligned} \tag{2.53}$$

which is solved at each iteration by applying Newton's method. Using the formula for the Newton direction Δx_i with equality constraints (2.46)*, we arrive at the following linear system of size $n + p$ for the Newton step at the current iterate x:*

$$\begin{bmatrix} H + \mu \sum_{i=1}^{m} \frac{1}{(h_i - g_i^T x)^2} g_i g_i^T & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_i \\ y \end{bmatrix} = - \begin{bmatrix} Hx + f + \sum_{j=1}^{m} \frac{1}{h_i - g_i x} g_i \\ 0 \end{bmatrix}. \tag{2.54}$$

The resulting barrier method for solving (2.51) *is summarized in Algorithm 2.3. Lines $3-8$ form the inner Newton iterations of the algorithm, computing $x^\star(\mu)$. Note that a primal feasible line search must be employed to ensure that the iterates remain in the interior of $\mathbb{Q}$.*

Algorithm 2.3 Barrier interior point method for QPs (2.51)

Require: Strictly feasible initial iterate x_0 with $Gx < h$, μ_0, $\kappa > 1$, tolerance $\epsilon > 0$
Ensure: Approximate solution with duality gap $p^\star - d^\star < \epsilon$ [19, §11.3.3]
Set $i \leftarrow 1$
loop
  Initialize Newton's method with $x_i^0 = x_{i-1}$. Set $k = 0$.
  loop
    Newton direction: solve (2.54) for $x \equiv x_i^k$ to obtain Δx_i^k.
    Feasible line search to obtain step size h_i^k (see Section 2.2.2.5).
    Update: $x_i^{k+1} = x_i^k + h_i^k \Delta x_i^k$
    End inner loop: **break** if $k = k_{\max}$, otherwise set $k \leftarrow k + 1$ and continue.
  end loop
  Update: $x_i = x_i^{k_{\max}}$
  Stopping criterion: Stop if $m\mu_i < \epsilon$
  Decrease barrier parameter: $\mu_{i+1} = \mu_i / \kappa$
  $i \leftarrow i + 1$
end loop

Barrier Function for Symmetric Cones

A general definition of the logarithmic barrier (2.48) for symmetric cones is given in the following.

Definition 2.6 (Log-det Barrier for Symmetric Cones)**.** *Let $\boldsymbol{K}$ be a product cone of rank M (cf. Definition 2.3) obtained by the direct product of N symmetric cones as in* (2.8)*, where the K_i are symmetric. For $x \in \boldsymbol{K}$, the function*

$$\Phi(x) \triangleq -\sum_{i=1}^{N} \ln \det x_i, \tag{2.55}$$

is a M-logarithmically homogeneous, self-scaled and self-concordant barrier for $\boldsymbol{K}$.

Note that (2.55) corresponds to (2.48) for the positive definite cone $\boldsymbol{R}_+$ when using the definition of the determinant $\det(x)$ for elements in symmetric cones given in (2.23d).

Conclusion

We conclude this section with a few remarks on the barrier method. The main computational burden is the computation of the Newton directions by solving the linear systems of type (2.54). This is usually done using direct methods (matrix factorizations), but iterative linear solvers can be used as well. In most applications the matrices involved have a special sparsity pattern, which can be exploited to significantly speed up the computation of the Newton direction. We detail on solving linear systems of type (2.54) in Section 3.

If the barrier parameter μ is decreased too quickly, a lot of inner iterations are needed, but only a few outer iterations. The situation is reversed if μ is decreased only slowly, which leads to a quick convergence of the inner problems (usually in one or two Newton steps), but more outer iterations are needed. It is however not difficult to tune the parameters κ and μ_0 in practice, and there exist commonly used values for which the total number of Newton steps is in the range of $20 - 40$, pretty much independent of the conditioning of the problem. The barrier method can be shown to converge linearly; more details can be found in [19, § 11.3].

Finally, we would like to point out that the barrier method as presented in this section is a *primal feasible* method, i.e. it requires a strictly feasible (with respect to the inequality constraints) initial iterate, and all subsequent iterates remain strictly feasible. This complicates the implementation in practice, as a line search must be employed that maintains feasibility, and a strictly feasible initial iterate x_0 has to be supplied by the user or computed first by what is called a *Phase I* method. Infeasibility with respect to the equality constraints can be handled in barrier methods by a so-called *infeasible start Newton method*. See [19, §11.4] for more details.

Computational experience has shown that modern primal-dual methods are significantly more effective with only little additional computational cost when compared to the barrier method. As a result, primal-dual methods are now predominant in commercial codes. They also allow infeasible iterates, that is, the equality and inequality constraints on the primal variables are satisfied only at convergence, which alleviates the necessity for finding a feasible initial point. We discuss this class of methods in the next section.

2.2.2.2 Primal-dual methods

The general idea of primal-dual interior point methods is to solve the Karush-Kuhn-Tucker (KKT) conditions by a modified version of Newton's method. The nonlinear KKT equations (2.5) represent necessary and sufficient conditions for optimality for convex problems. The modifications to a pure Newton method are necessary to obtain a practically useful method for constrained optimization. The name *primal-dual* indicates that the algorithm operates in both the primal and dual space.

There are many variants of primal-dual interior point methods. For a thorough treatment of the theory we refer the reader to the book by Wright [138], which gives a comprehensive overview of the different approaches. In this section, after introducing the general framework of primal-dual methods, we restrict ourselves to the presentation of a variant of Mehrotra's predictor-corrector method [92], because it forms the basis of most existing implementations of primal-dual interior point methods and has proven particularly effective in practice. The

method we present here allows for infeasible starting points and is the basic algorithm underlying the methods presented in Chapters 8 and 9.

Central Path

We start by outlining the basic idea first. Most interior point methods are path-following methods, which track the central path to a solution. In the primal-dual space, the central path is defined as the set of points (x, y, z, s), for which the following *relaxed* KKT conditions hold:

$$\nabla f(x) + A^T y + G(x)^T z = 0 \tag{2.56a}$$
$$Ax - b = 0 \tag{2.56b}$$
$$g(x) + s = 0 \tag{2.56c}$$
$$s_i z_i = \tau, \ \forall i = 1, \ldots, m \tag{2.56d}$$
$$(s, z) > 0 \tag{2.56e}$$

In (2.56) as compared to (2.5), the complementarity condition (2.5d) is relaxed by a scalar $\tau > 0$, the path parameter, and slacks s and multipliers z are required to be positive instead of nonnegative, ensuring interiority of the central path. The main idea of primal-dual methods is to solve (2.56) for successively decreasing values of τ, and thereby to generate iterates (x_i, y_i, z_i, s_i) that approach $(x^\star, y^\star, z^\star, s^\star)$ as $\tau \to 0$. The overall concept is therefore similar to a barrier method, but with a different Newton step, as discussed below.

Measure of Progress

The subproblem (2.56) is solved only approximately in primal-dual methods by applying only one Newton step. Consequently, there is no distinction between inner and outer iterations as in barrier methods, and the generated iterates are not exactly on the central path, i.e. $s_i z_i \neq \tau$ for some element i of s and z. Hence a more useful measure than the path parameter τ for the progress of the algorithm is the *average* value of the complementarity condition (2.56d),

$$\mu \triangleq (s^T z)/m\,. \tag{2.57}$$

Note that $\mu \to 0$ implies $\tau \to 0$. The scalar μ is called the *duality measure*, as it corresponds to the duality gap scaled by $1/m$ if all other equalities in (2.56) are satisfied. To be able to reduce μ substantially (at least by a constant factor) at each iteration of the method, the pure Newton step is enhanced with a centering component, which will be discussed in the following.

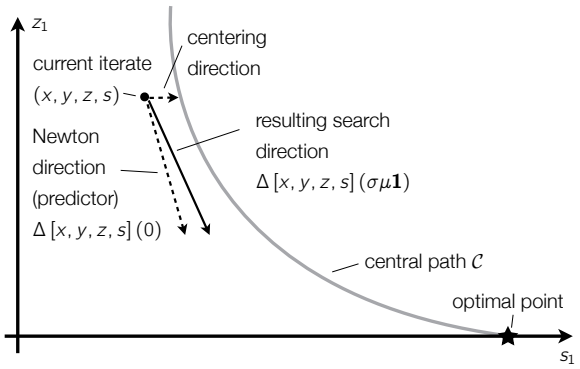


Figure 2.2: Affince-scaling (pure Newton) and centering search directions of classic primal-dual methods. The Newton direction makes larger progress towards the solution when it is computed from a point close to the central path, hence centering must be added to the pure Newton step to ensure that the iterates stay sufficiently close to the central path.

Remark 2.3. *When optimising over symmetric cones $\boldsymbol{K}$ (cf. Section 2.1.3), the number of linear constraints, m, in* (2.57) *is replaced by the degree D of $\boldsymbol{K}$ from Definition 2.4. Note that for the positive orthant $\boldsymbol{R}_+$, $m = D$.*

Newton Directions and Centering

A search direction is obtained by linearizing (2.56) at the current iterate (x, y, z, s) and solving the resulting linear system

$$\begin{bmatrix} H(x,z) & A^T & G(x)^T & 0 \\ A & 0 & 0 & 0 \\ G(x) & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + A^T y + G(x)^T z \\ Ax - b \\ g(x) + s \\ Sz - \nu \end{bmatrix}, \tag{2.58}$$

where $S \triangleq \operatorname{diag}(s_1, \ldots, s_m)$, $Z \triangleq \operatorname{diag}(z_1, \ldots, z_m)$, and

$$H(x,z) \triangleq \nabla^2 f(x) + \sum_{i=1}^{m} z_i \nabla^2 g_i(x)\,. \tag{2.59}$$

The vector $\nu \in \boldsymbol{R}^m$ allows for a modification of the right-hand side, thereby generating different search directions that depend on the particular choice of ν. For brevity, we denote the solution to (2.58) by $\Delta[x, y, z, s](\nu)$.

The vector ν can take values between the following two extremes:

Algorithm 2.4 Primal-dual interior point methods for (2.1)

Require: Initial iterates x_0, y_0, $z_0 > 0$, $s_0 > 0$, Centering parameter $\sigma \in (0, 1)$
1: **loop**
2: Duality measure: $\mu_i = s_i^T z_i / m$
3: Search direction: compute $\Delta[x_i, y_i, z_i, s_i](\sigma\mu_i 1)$ by solving (2.58) at the current iterate
4: Choose step length h_i such that $s_{i+1} > 0$ and $z_{i+1} > 0$
5: Update: $(x_{i+1}, y_{i+1}, z_{i+1}, s_{i+1}) = (x_i, y_i, z_i, s_i) + h_i \Delta[x_i, y_i, z_i, s_i](\sigma_i \mu_i 1)$
6: **end loop**

- If $\nu = 0$, the right hand side in (2.58) corresponds to the residual of the KKT conditions (2.5), i.e. $\Delta[x, y, z, s](0)$ is the pure Newton direction that aims at satisfying the KKT conditions (2.5) in one step based on a linearization. This direction is called *affine-scaling* direction. However, pure Newton steps usually decrease some products $s_i z_i$ prematurely towards zero, i.e. the generated iterate is close to the boundary of the feasible set (if $s_i z_i \approx 0$, either s_i or z_i must be small). This is disadvantageous since from such points only very small steps can be taken, and convergence might be prohibitively slow.
- If $\nu = \mu\mathbf{1}$, the complementarity condition (2.56d) is modified such that the resulting search direction aims at making the individual products $s_i z_i$ equal to their average value μ. This is called *centering*. Except for this modification, the right hand side in (2.58) corresponds to the residuals of the KKT conditions, as in the affine step. A centering step does not decrease the average value of μ, so no progress towards the solution is made, although the generated search direction might reduce the residuals of the first three equations.

A tradeoff between the two goals of progressing towards the solution and centering is achieved by computing search directions $\Delta[x, y, z, s](\nu)$ in (2.58) for

$$\nu = \sigma\mu\mathbf{1} \tag{2.60}$$

where $\sigma \in (0, 1)$ is called the *centering parameter*. This is the key difference to pure Newton steps, and it is the main ingredient in making Newton's method work efficiently for solving the nonlinear KKT system (2.5). Figure 2.2 illustrates this combination of search directions schematically, and a principle framework for primal-dual methods is given in Algorithm (2.4). We will instantiate it with Mehrotra's method in the next section.

In the discussion above, we have excluded almost all theoretical aspects concerned with convergence of primal-dual methods. See e.g. [138] for a thorough theoretical treatment.

2.2.2.3 Mehrotra-Type Predictor-Corrector Methods

In 1992, Mehrotra [92] suggested modifications to the basic primal-dual method from Algorithm 2.4, which proved very effective in practice and which are therefore applied in most existing interior point codes. The main idea is to use the affine-scaling direction, denoted here by $\Delta[x^{\text{aff}}, y^{\text{aff}}, z^{\text{aff}}, s^{\text{aff}}](0)$, as a *predictor* to estimate the linearization error for the nonlinear complementarity conditions (2.56d) if a pure Newton step were taken,

$$e \triangleq (S + \Delta S^{\text{aff}})(z + \Delta z^{\text{aff}}) = \underbrace{Sz + \Delta S^{\text{aff}} z + S\Delta z^{\text{aff}}}_{=0 \text{ cf. } (2.58)} + \Delta S^{\text{aff}} \Delta z^{\text{aff}} = m\mu^{\text{aff}}\,, \qquad (2.61)$$

where capital letters denote diagonal matrices constructed from the corresponding vectors. In order to compensate for this error, a *corrector* direction can be computed by setting the right hand side of (2.58) to $(0, 0, 0, -e)$, and added to the usual search direction $\Delta[x, y, z, s](\sigma\mu\mathbf{1})$. This corresponds to directly computing $\Delta[x, y, z, s](\sigma\mu\mathbf{1} - e)$, i.e. there is no need to compute the corrector direction separately. The predictor-corrector concept is illustrated in Figure 2.3. The compensation is imperfect since the predictor direction assumes no centering ($\sigma = 0$), while in general some centering is used (i.e. $\sigma > 0$). Mehrotra's heuristic for choosing the centering parameter σ based on the information from the predictor step is

$$\sigma \triangleq (\mu^{\text{aff}}/\mu)^3\,, \qquad (2.62)$$

which increases the amount of centering if the progress would be poor, that is if the predicted value of the duality measure, μ^{aff}, is not significantly smaller than the current value μ. If good progress can be made, then $\sigma \ll 1$, and the search direction $\Delta[x, y, z, s](\sigma\mu\mathbf{1} - e)$ is close to the pure Newton direction, except for the correction term e.

Having described the important aspects of Mehrotra's predictor-corrector method, we give a recipe for one variant in Algorithm 2.5 that works well in practice. The parameter γ is a safeguard against numerical errors, keeping the iterates away from the boundary. Due to the predictor-corrector scheme, two linear systems have to be solved in each iteration. Since the coefficient matrix is the same for both solves, direct methods are preferred to solve the linear systems in Steps 3 and 7 of the algorithm, as the computational bottleneck of the method is in the factorization of the coefficient matrix. Once it has been factored in Step 3, the computational cost of the additional solve in Step 7 is negligible, and is by far compensated by the savings in the number of iterations.

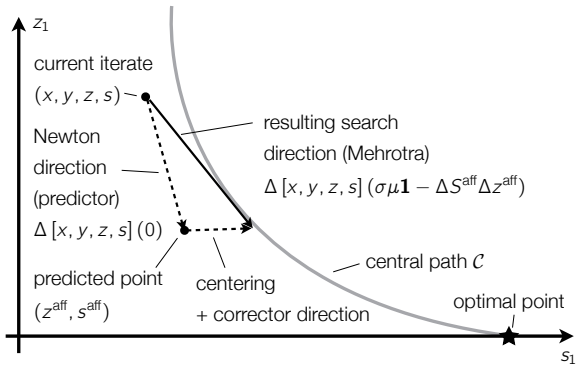


Figure 2.3: Search direction generation in predictor-corrector methods. The affine search direction can be used to correct for linearization errors, which results in better performance in practice due to closer tracking of the central path.

Algorithm 2.5 Mehrotra primal-dual interior point method for (2.1)

Require: Initial iterates x_0, y_0, $z_0 > 0$, $s_0 > 0$, $\gamma < 1$ (typically $\gamma = 0.99$)

1: **loop**
2: Duality measure: $\mu_i = s_i^T z_i / m$
3: Newton direction (predictor): compute $\Delta[x_i^{\text{aff}}, y_i^{\text{aff}}, z_i^{\text{aff}}, s_i^{\text{aff}}](0)$ by solving (2.58) at the current iterate
4: Line search: $h_i^{\text{aff}} \triangleq \max\{h \in [0,1] \mid s_i + h\Delta s_i^{\text{aff}} \geq 0,\ z_i + h\Delta z_i^{\text{aff}} \geq 0\}$
5: Predicted (affine) duality measure: $\mu_i^{\text{aff}} \triangleq (s_i + h_i^{\text{aff}}\Delta s_i^{\text{aff}})^T (z_i + h_i^{\text{aff}}\Delta z_i^{\text{aff}})/m$
6: Centering parameter: $\sigma_i = (\mu_i^{\text{aff}}/\mu_i)^3$
7: Final search direction: compute $\Delta[x_i, y_i, z_i, s_i](\sigma_i\mu_i 1 - \Delta S_i^{\text{aff}}\Delta z_i^{\text{aff}})$ by solving (2.58) at the current iterate
8: Line search: $h_i \triangleq \max\{h \in [0,1] \mid s_i + h\Delta s_i \geq 0,\ z_i + h\Delta z_i \geq 0\}$
9: Update: $(x_{i+1}, y_{i+1}, z_{i+1}, s_{i+1}) = (x_i, y_i, z_i, s_i) + \gamma h_i \Delta[x_i, y_i, z_i, s_i](\sigma_i\mu_i 1 - \Delta S_i^{\text{aff}}\Delta z_i^{\text{aff}})$
10: **end loop**

2.2.2.4 Relation of Primal-Dual to Primal Barrier Methods

An insightful relation between primal barrier and primal-dual methods can be established as follows. Writing the optimality conditions for $(P(\mu))$ gives

$$\nabla f(x) + \mu \nabla \Phi_g(x) + A^T y = 0 \tag{2.63a}$$
$$Ax = b \tag{2.63b}$$
$$g(x) \leq 0 \tag{2.63c}$$

where $g(x) \leq 0$ has been added for the barrier function Φ_g to be well defined. If one now defines

$$z \triangleq \mu \operatorname{diag}(-g(x))^{-1} \mathbf{1}\,, \tag{2.64}$$

the conditions (2.63) become

$$\nabla f(x) + \sum_{i=1}^{m} z_i \nabla g_i(x) + A^T y = 0 \tag{2.65a}$$
$$Ax = b \tag{2.65b}$$
$$g(x) + s = 0 \tag{2.65c}$$
$$Sz = \mu \mathbf{1} \tag{2.65d}$$
$$(s, z) \geq 0 \tag{2.65e}$$

where we have introduced slack variables s and used the expression for the gradient of the barrier function (2.49) and the relations $s = -g(x)$, $S = \operatorname{diag}(s)$. Using

$$\sum_{i=1}^{m} z_i \nabla g_i(x) \equiv G(x)^T z\,, \tag{2.66}$$

where $G(x)$ is the Jacobian of g evaluated at the point x, yields the same relaxed optimality conditions as (2.56).

Hence if subproblems $P(\mu)$ and (2.56) were solved exactly assuming (2.64), the primal iterates of the primal-barrier method and the primal-dual method would coincide (provided that the path parameters coincide, i.e. $\tau = \mu$). However, primal-dual methods are in practice more efficient than primal barrier methods, since they generate different search directions using also information from the dual space, and the iterates generated by Algorithms 2.2 and 2.4 do not coincide in general.

2.2.2.5 Line Search

A line search determines the step length h_i that is taken from the current iterate $x_i \in \boldsymbol{R}^n$ along a search direction $\Delta x_i \in \boldsymbol{R}^n$ to the next iterate,

$$x_{i+1} = x_i + h_i \Delta x_i \,. \tag{2.67}$$

In the most general setting, a merit function $\varphi : \boldsymbol{R}^n \to \boldsymbol{R}$ is employed to measure the quality of the step taken to determine a value of the step size h_i that ensures convergence of the optimization algorithm. Merit functions can for example be the objective function f, or the duality measure μ (2.57) in primal-dual interior point methods.

The general line search problem can be written as a one dimensional optimization problem over the step size h,

$$h^\star \in \arg\min_{h \geq 0} \varphi(x_i + h\Delta x_i) \,, \tag{2.68}$$

minimizing the merit function along the ray $\{x_i + h\Delta x_i\}$ for $h \geq 0$. It is often too expensive to solve this problem exactly, and therefore usually *inexact* line search methods are employed in practice that return a step size h_i that approximates the optimal step size $h^\star$ from (2.68) sufficiently well. In the following, we give a brief overview on practical inexact line search methods; see [104] for a thorough treatment of the subject.

Wolfe Conditions

In order to ensure convergence of descent methods for unconstrained optimization, for example Newton's method presented in Section 2.2.1, certain conditions on the selected step size h_i must be fulfilled. Popular conditions for inexact line searches are the so-called *Wolfe conditions* [101]

$$f(x_i + h_i \Delta x_i) \leq f(x_i) + c_1 h_i \nabla f(x_i)^T \Delta x_i \tag{2.69a}$$

$$\nabla f(x_i + h_i \Delta x_i)^T \Delta x_i \geq c_2 \nabla f(x_i)^T \Delta x_i \tag{2.69b}$$

with $0 < c_1 < c_2 < 1$. The first condition (2.69a) is called *Armijo's condition* [9], and ensures that a sufficient decrease in the objective is achieved. The second condition, the *curvature condition* (2.69b), ensures that the step length is not too small [101].

One of the most popular inexact line search methods that fulfils the Wolfe conditions is the backtracking line search as discussed in the following.

Backtracking Line Search for Newton's Method

The backtracking line search for Newton's method is given in Algorithm 2.6. It starts with a unit step size, which is iteratively reduced by a factor $t \in (0, 1)$ until Armijo's

Algorithm 2.6 Backtracking Line Search for Newton's Method (Algorithm 2.1) [101, Algorithm 3.1]

Require: Current iterate x_i, descent direction Δx_i, constants $c \in (0,1)$ and $t \in (0,1)$
Ensure: Step size h_i ensuring Wolfe conditions (2.69)
1: Set $h_i \leftarrow 1$
2: **while** $f(x_i + h_i \Delta x_i) \leq f(x_i) + c h_i \nabla f(x_i)^T \Delta x_i$ **do**
3: Set $h_i \leftarrow t h_i$
4: **end while**

Algorithm 2.7 Backtracking Line Search for Primal-Dual Interior Point Methods

Require: Current iterate s, z, search directions $\Delta s, \Delta z$, constant $t \in (0,1)$
Ensure: Step size h ensuring $s + h\Delta s > 0$ and $z + h\Delta z > 0$
1: Set $h \leftarrow 1$
2: **while** $s + h\Delta s \leq 0$ or $z + h\Delta z \leq 0$ **do**
3: Set $h \leftarrow th$
4: **end while**

condition (2.69a) is satisfied. It can be shown that it is not necessary to check the curvature condition because it is implied by the backtracking rule [101, Algorithm 3.1].

In order to obtain a line search method that ensures in addition feasibility of the next iterate x_{i+1}, the backtracking procedure is preceeded by finding $0 < \bar{h} \leq 1$ such that $x_i + \bar{h}\Delta x_i$ is feasible [19, pp. 465]. The backtracking then starts with $h_i = \bar{h}$ in line 1 of Algorithm 2.6.

Backtracking Line Search in Primal-Dual Methods

For primal-dual interior point methods, it can be shown that the factor by which a reduction in the duality measure μ occurs from one iteration to the next is proportional to $(1 - h_i)$, where h_i is the step size [139]. Hence the goal is to find the largest step size h_i that preserves feasibility of the slack variables s and the associated Lagrange multipliers z,

$$h_i = \max\{h \in (0,1] \mid (s,z) + h(\Delta s, \Delta z) > 0\}\,. \tag{2.70}$$

A simple algorithm that approximately solves (2.70) based on the idea of backtracking is given in Algorithm 2.7. It starts with $h = 1$, and iteratively reduces the step size by a factor of $t \in (0,1)$ until feasibility is recovered.

3 Direct Methods for Solving KKT Systems

As discussed in Section 2.2.2.2, the computational bottleneck in primal-dual interior point methods is the solution of linear systems of the form (cf. (2.58))

$$\begin{bmatrix} H(x,z) & A^T & G(x)^T & 0 \\ A & 0 & 0 & 0 \\ G(x) & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = \begin{bmatrix} r_S \\ r_E \\ r_I \\ r_C \end{bmatrix} \tag{3.1}$$

for different right-hand sides (r_S, r_E, r_I, r_C). In this chapter, we present some background material from numerical linear algebra related to solving (3.1). In general, (3.1) is reduced to a smaller system that has more structure, to the symmetric indefinite forms in Section 3.1 or to the positive definite (normal equations) form in Section 3.2. Along with these forms, the associated direct methods for solving the linear systems are discussed in this chapter.

Although iterative methods are a viable alternative, particularly for large, sparse problems with hundreds of thousands of variables, we exclude them from the discussion due to two reasons. First, the methods and tools presented in this thesis (in Chapters 8 and 9) aim at solving small- to medium sized problems, and second, in Mehrotra-type predictor-corrector interior point methods (cf. Section 2.2.2.3), where two or more linear systems have to be solved with the same coefficient matrix, direct methods are the preferred choice since the extra cost of solving for multiple right-hand sides is negligible once a factorization of the coefficient matrix has been computed. For a thorough treatment of iterative linear solvers see e.g. [114].

In addition to the presentation of standard factorization methods, various related aspects are discussed in this chapter. In Section 3.2.2, inexpensive methods for incorporating low rank modifications of a matrix into already computed factors are discussed. These methods avoid re-computing the factorization if the a matrix has not been significantly changed, which saves unnecessary floating point operations. In Section 3.3, the *iterative refinement* technique is summarized, which allows for improving upon an approximately computed solution by means of inexpensive iterations, of which usually one or two suffice to obtain

a highly accurate solution. Iterative refinement plays a central role in the approach taken in Chapter 9, along with *quasi-definite* matrices, a special class of indefinite matrices that can be factored stably for *any* permutation. This topic is subject of Section 3.1.2.

3.1 Symmetric Indefinite Forms

The coefficient matrix in (3.1) is not symmetric, hence it is standard to eliminate Δs. From the last equation,

$$\Delta s = Z^{-1}\left(r_C - S\Delta z\right) , \tag{3.2a}$$

where $Z \triangleq \operatorname{diag}(z)$, and hence (3.2a) is well defined since $z > 0$ in interior point methods (cf. (2.56e)). We obtain the symmetric indefinite linear system

$$\begin{bmatrix} H(x,z) & A^T & G(x)^T \\ A & 0 & 0 \\ G(x) & 0 & -Z^{-1}S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} r_S \\ r_E \\ r_I - Z^{-1}r_C \end{bmatrix} . \tag{3.2b}$$

More often than not, this system is further reduced by eliminating Δz,

$$\Delta z = -S^{-1}Z\left(r_I - Z^{-1}r_C - G(x)\Delta x\right) , \tag{3.3a}$$

where $S \triangleq \operatorname{diag}(s)$, which is again well defined since $s > 0$ in interior point methods (cf. (2.56e)). This gives rise to the most compact symmetric indefinite linear system

$$\begin{bmatrix} H(x,z) + G(x)^T S^{-1} Z G(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_S \\ r'_E \end{bmatrix} \tag{3.3b}$$

with

$$r'_E \triangleq r_E + G(x)^T S^{-1} Z r_I - G(x)^T S^{-1} r_C . \tag{3.3c}$$

3.1.1 LDL Factorization

The classical direct method for solving linear systems (3.2b) or (3.3b) with symmetric indefinite coefficient matrices was established by Bunch and Parlett in 1971 [22]. First, a factorization of the coefficient matrix is computed,

$$PKP^T = LDL^T , \tag{3.4}$$

where K denotes the coefficient matrix in (3.2b) or (3.3b), P is a permutation matrix, L is a unit lower triangular matrix and D is a block diagonal matrix with blocks of size 1×1 or 2×2. This procedure corresponds to standard Gaussian elimination with pivoting for

numerical stability, but it exploits the fact that K is a symmetric matrix. Both 1×1 and 2×2 pivots are necessary for numerical stability, see [22] for details.

Once a factorization has been computed, the solution to the linear system $Kx = b$ can be computed at comparably low cost by solving a sequence of equations,

$$Lu = Pb\,, \tag{3.5a}$$
$$Dv = u\,, \tag{3.5b}$$
$$L^T w = v\,, \tag{3.5c}$$
$$x = P^T w\,, \tag{3.5d}$$

with intermediate vectors u, v, w. Solving (3.5a) is the standard forward substitution for unit lower triangular matrices and (3.5c) is the standard backward substitution. Solving (3.5b) is done blockwise, using the fact that for a 2×2 block in D the inverse is easily obtained explicitly. In order to compute the permutations in (3.5a) and (3.5d), no floating point operations are necessary. The cost of the solve procedure (3.5) is most of the time negligible with respect to the cost of computing the factorization (3.4).

One drawback of this classical scheme is that the pivoting sequence P is data-dependent, as it is computed during the numerical factorization, and that row and column interchanges are necessary during this process. This greatly reduces the performance of numerical codes, since memory access patterns of these interchanges are non-contiguous and therefore in general quite expensive. In Chapter 9, we will employ other techniques that allow using only 1×1 pivots, i.e. the matrix D is diagonal, for *any* permutation P of the coefficient matrix K. In particular, we will determine P *before* the factorization based solely on sparsity arguments. One class of matrices (not KKT matrices), for which this is always possible are discussed next.

3.1.2 Quasi-Definite Matrices

Vanderbei introduced in 1991 the class of quasi-definite matrices and showed that a stable LDL factorization exists for any permutation [130], which allows one to avoid the complicated Bunch-Parlett factorization as discussed in Section 3.1. We repeat the main results from [130] in the following.

Definition 3.1 (Quasi-Definite Matrix)**.** *Let $H \in \boldsymbol{S}^n_{++}$ and $E \in \boldsymbol{S}^m_{++}$ be positive definite matrices, and $F \in \boldsymbol{R}^{m \times n}$ be a matrix of full row rank. Then the composite matrix*

$$M = \begin{bmatrix} H & F^T \\ F & -E \end{bmatrix} \tag{3.6}$$

is said to be quasi-definite*.*

Definition 3.2 (Strongly Factorizable Matrix [130])**.** *A symmetric nonsingular matrix is said to be* strongly factorizable *if, for every permutation P, there exists a diagonal matrix D and a unit lower triangular matrix L such that $PKP^T = LDL^T$.*

Theorem 3.1 (Strong Factorizability of Quasi-Definite Matrices [130, Theorem 2.1])**.** *Symmetric quasidefinite matrices are strongly factorizable.*

Unfortunately, the KKT systems (3.2b) and (3.3b) are not quasi-definite, but can be made so by small perturbations of the diagonals. We detail on this approach in Chapter 9.

3.2 Positive Definite Form

Assuming that the (1,1) block in the coefficient matrix of (3.3b),

$$\Phi \triangleq H(x,z) + G(x)^T S^{-1} Z G(x)\,, \tag{3.7}$$

is positive definite, which holds in all practical problems without free variables, we can eliminate Δx,

$$\Delta x = \Phi^{-1}\left(r_s - A^T \Delta y\right)\,, \tag{3.8a}$$

to obtain a linear system with positive definite coefficient matrix,

$$A\Phi^{-1}A\Delta y = -\left(r'_E - \Phi^{-1} r_S\right)\,. \tag{3.8b}$$

This form is also called *normal equations form*. For convenience, we introduce $Y \in \boldsymbol{R}^{p\times p}$ as the positive definite coefficient matrix,

$$Y \triangleq A\Phi^{-1}A^T\,, \tag{3.8c}$$

and $\beta \in \boldsymbol{R}^p$ for the right-hand side of (3.8b),

$$\beta \triangleq -\left(r_E + G(x)^T S^{-1} Z r_I - G(x)^T S^{-1} r_C - \Phi^{-1} r_S\right)\,. \tag{3.8d}$$

3.2.1 Cholesky Factorization

Direct methods for linear systems with positive definite coefficient matrix Y use a *Cholesky factorization* that computes a lower triangular matrix L

$$Y = LL^T\,, \tag{3.9}$$

or, equivalently, a unit lower triangular matrix $\bar{L}$ and a diagonal matrix D with all positive entries,

$$Y = \bar{L}D\bar{L}^T\,, \tag{3.10}$$

Algorithm 3.1 Cholesky Decomposition

Require: Positive definite matrix $A \in \boldsymbol{S}^n_{++}$
Ensure: Lower triangular matrix L such that $A = LL^T$
1: **for** $j = 1, \ldots, n$ **do**
2: $\quad L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}$
3: $\quad$ **for** $i = j+1, \ldots, n$ **do**
4: $\quad\quad L_{i,j} = \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}\right) / L_{j,j}$
5: $\quad$ **end for**
6: **end for**

where the equivalence between the two representations is established by $L = \bar{L} D^{1/2}$. It can be shown that a Cholesky factorization is numerically stable for any permutation of the matrix Y [34]. Hence, for dense matrices, one might just not perform any pivoting. This is an advantage when compared to LDL methods for symmetric indefinite matrices as presented above. As a result, a basic version of the Cholesky decomposition can be easily implemented in only a few lines of code, cf. Algorithm 3.1, which computes L column wise. Permutations of Y that favor sparsity in the factor L are important to solve large sparse systems efficiently; see e.g. [32] for modern sparse Cholesky methods.

The disadvantage of solving the positive definite KKT system is that the coefficient matrix Y (3.8c) can be full, even if the problem data is sparse. These cases occur due to the outer product with A and A^T in (3.8c) and due to the second term in (3.7), which might have low rank. One possibility to deal with such "sparse plus low rank" structure is discussed in the next section.

Remark 3.1. *In the following, we use the term* Cholesky factor *interchangeably for both L in* (3.9) *and $\bar{L}$, D in* (3.10)*. It will be clear from the context which form is used.*

3.2.2 Low Rank Modifications of Matrix Factors

Low rank modifications of matrices arise in interior point methods for example when solving the positive definite form (3.8). In particular, it might be beneficial to compute the Cholesky factorization of (3.7) by first factoring $H(x, z)$ and then modifying the computed factors to incorporate the outer product $G(x)^T S^{-1} Z G(x)$, if it has low rank. We start by discussing one such update in Section 3.2.2.1; multiple updates result in a so-called product-form Cholesky factorization, which is subject of Section 3.2.2.2.

Algorithm 3.2 Modification of Matrix Factor after Rank One Update [50, Algorithm C1, pp. 515]

Require: Positive definite matrix $A \in \boldsymbol{S}^n_{++}$, factors L and D s.t. $A = LDL^T$, scalar α and vector z such that $A + \alpha zz^T \in \boldsymbol{S}^n_{++}$
Ensure: Lower triangular matrix $\tilde{L}$ and diagonal matrix $\tilde{D}$ such that $A + \alpha zz^T = L\tilde{L}\tilde{D}\tilde{L}^T L^T$
1: Solve $Lq = z$ for vector q by forward substitution
2: Set $\alpha_1 \leftarrow \alpha$
3: **for** $j = 1, \ldots, n$ **do**
4: $\quad \tilde{D}_{j,j} = D_{j,j} + \alpha_j q_j^2$
5: $\quad \beta_j = \alpha_j q_j / \tilde{D}_{j,j}$
6: $\quad$ **for** $i = j+1, \ldots, n$ **do**
7: $\quad\quad \tilde{L}_{i,j} = \beta_j q_i$
8: $\quad$ **end for**
9: $\quad$ **if** $j < n$ **then**
10: $\quad\quad \alpha_{j+1} = \alpha_j D_{j,j} / \tilde{D}_{j,j}$
11: $\quad$ **end if**
12: **end for**

3.2.2.1 Rank One Modifications

An efficient and numerically stable method for updating the LDL^T factors of a symmetric positive definite matrix $A \in \boldsymbol{S}^n_{++}$ after a rank one modification $A + \alpha zz^T$, where α is a scalar and $z \in \boldsymbol{R}^n$ is a column vector, is given in Algorithm 3.2. It computes a unit lower triangular matrix $\tilde{L}$ and a diagonal matrix $\tilde{D}$ such that

$$A + \alpha zz^T = L\tilde{L}\tilde{D}\tilde{L}^T L^T . \tag{3.11}$$

This is usually cheaper than recomputing the factorization of the modified matrix when solving linear systems, since the resulting $\tilde{L}$ has a special structure:

$$\tilde{L}(\beta, p) \triangleq \begin{bmatrix} 1 & & & & \\ \beta_1 q_1 & 1 & & & \\ \beta_1 q_2 & \beta_2 q_2 & 1 & & \\ \beta_1 q_3 & \beta_2 q_3 & q_3 q_3 & 1 & \\ \vdots & \vdots & \ddots & \ddots & \\ \beta_1 q_{n_i-1} & \cdots & \cdots & \beta_{n_i-1} q_{n_i-1} & 1 \end{bmatrix}, \tag{3.12}$$

i.e. to store $\tilde{L}$, only the two vectors $\beta \in \boldsymbol{R}^{n-1}$ and $q \in \boldsymbol{R}^{n-1}$ need to be stored, and solving linear equations of the form $\tilde{L}(\beta, q)x = b$ for x requires only n multiplications.

Solving Linear Systems after a Rank One Modification

In order to solve the equation

$$(A + \alpha zz^T)x = b \tag{3.13}$$

using the relation (3.11), the sequence of equations

$$Lr = b \tag{3.14a}$$
$$\tilde{L}u = r \tag{3.14b}$$
$$\tilde{D}v = u \tag{3.14c}$$
$$\tilde{L}w = v \tag{3.14d}$$
$$Lx = w \tag{3.14e}$$

can be solved for the intermediate vectors r, u, v, w and finally for x. Due to the special structure of $\tilde{L}$ (3.12), solving (3.14b) and (3.14d) incurs only n multiplications, while (3.14a) and (3.14e) are standard forward and backward substitutions, respectively.

It is possible, and most of the time useful in practice, to directly compute the resulting Cholesky factor $\bar{L} \triangleq L\tilde{L}$ such that $A + \alpha zz^T = \bar{L}\tilde{D}\bar{L}^T$. This computation can be easily incorporated into Algorithm 3.2, see [50, Algorithm C1, pp. 515] for details. We omitted this here since we consider only cases where either $L = I_n$ and hence $\bar{L} = \tilde{L}$, or we perform multiple rank one modifications to obtain a product form Cholesky factorization as presented in the next section.

3.2.2.2 Product Form Factorization

The rank one update approach from Section 3.2.2.1 can be generalized to rank k modifications as follows. Let the LDL^T factorization of matrix $A \in \boldsymbol{S}^n_{++}$ be given such that $A = LDL^T$, as above. Consider the matrix $A + VSV^T$ with $V \in \boldsymbol{R}^{n\times k}$ and $S = \operatorname{diag}(s_1, \ldots, s_k) \in \boldsymbol{S}^k_{++}$ diagonal. The product form Cholesky factorization in Algorithm 3.3 computes matrices $\tilde{D}$ and $\tilde{L}_i$, $i = 1, \ldots, k$ such that

$$A + VSV^T = L\tilde{L}_1\tilde{L}_2 \ldots \tilde{L}_k\tilde{D}\tilde{L}_k^T \ldots \tilde{L}_2^T\tilde{L}_1^T L^T . \tag{3.15}$$

Each matrix $\tilde{L}_i$ can be computed by applying the recurrence relations for rank one updates (lines 3-12 in Algorithm 3.2) in $\mathcal{O}(n^2)$ flops and is stored by two vectors $\beta_i \in \boldsymbol{R}^{n-1}$ and $q_i \in \boldsymbol{R}^{n-1}$. Hence the total cost of Algorithm 3.3 is $\mathcal{O}(kn^2)$ instead of $\mathcal{O}(n^3)$, which is the order of the number of floating point operations for the standard Cholesky factorization from Algorithm 3.1. Under mild conditions, which hold for the purposes in this thesis, Algorithm 3.3 is numerically stable; further details on product form factorizations can be found for example in [53, 54].

3.2.2.3 Efficient Forward Substitutions

In the interior point method presented in Chapter 8, equations of the form

$$X\tilde{L}(\beta, q)^T = B \tag{3.16}$$

Algorithm 3.3 Product form Cholesky Factorization using Low Rank Updates

Require: Factors L, D of $A \in \boldsymbol{S}^n_{++}$ s.t. $A = LDL^T$ with D diagonal, L lower unit triangular, $V \in \boldsymbol{R}^{n\times k}$, $S = \mathrm{diag}(s_1, \ldots, s_k) \in \boldsymbol{S}^k_{++}$
Ensure: $A + VSV^T = L\tilde{L}_1(\beta_1, q_1)\ldots\tilde{L}(\beta_k, q_k)\tilde{D}\tilde{L}_k(\beta_k, q_k)^T\ldots\tilde{L}_1(\beta_1, q_1)^T L^T$
1: $\tilde{D}_0 \leftarrow D$
2: **for** $i = 1, \ldots, k$ **do**
3: Solve $Lp_i = v_i$ for p_i
4: **for** $j = 1, \ldots, i-1$ **do**
5: Solve $\tilde{L}_j(\beta_j, q_j)p_{j+1} = p_j$ for p_{j+1}
6: **end for**
7: Compute $\beta_i, q_i, \tilde{D}_i$ using lines 3-12 of Algorithm 3.2 with $\alpha_1 \equiv s_i$, $D \equiv \tilde{D}_{i-1}$ and $q = p_i$
8: **end for**
9: $\tilde{D} \leftarrow \tilde{D}_k$

will be solved for the matrix $X \in \boldsymbol{R}^{n\times m}$ for a given right hand side matrix $B \in \boldsymbol{R}^{m\times n}$. We can exploit the special structure of $\tilde{L}(\beta, q)$ in (3.12) to obtain the solution in only $\mathcal{O}(nm)$ floating point operations instead of the standard $\mathcal{O}(mn^2)$ as follows. The standard matrix forward substitution formula defines element i, j of the matrix X as

$$X_{i,j} = B_{i,j} - \sum_{k=1}^{j-1} \tilde{L}_{j,k} X_{i,k}\,. \tag{3.17}$$

Using the relation $\tilde{L}_{j,k} = \beta_j q_k$ gives

$$X_{i,j} = B_{i,j} - \beta_j \underbrace{\sum_{k=1}^{j-1} q_k X_{i,k}}_{\triangleq \gamma_i}\,, \tag{3.18}$$

where β_j can be moved in front of the sum since it is independent of k due to the special structure of $\tilde{L}(\beta, q)$ (cf. (3.12)). The sum in (3.18) can be stored in the vector $\gamma \in \boldsymbol{R}^{n-1}$ and be updated for each new column. This effectively eliminates the innermost loop of the backward substitution at the cost of storing the vector γ. The resulting matrix forward substitution method is of complexity $\mathcal{O}(nm)$ and is given in Algorithm 3.4.

3.3 Iterative Refinement

Iterative refinement (IR) is a technique to improve a numerically computed approximate solution $\hat{x}$ to a linear system $Ax = b$. It is implemented in all major linear algebra packages such as BLAS/LAPACK, for example. We will make use of iterative refinement in Chapter 9 to obtain accurate solutions to a slightly perturbed linear system.

Algorithm 3.4 Efficient Matrix Forward Substitution for Solving $XL(\beta, q)^T = B$ for X

Require: Vectors $\beta \in \boldsymbol{R}^{n-1}$ and $q \in \boldsymbol{R}^{n-1}$ for product form Cholesky factor $L(\beta, q)$ (3.12), right hand side matrix $B \in \boldsymbol{R}^{m \times n}$
Ensure: Matrix $X \in \boldsymbol{R}^{m \times n}$ such that $XL(\beta, q)^T = B$
1: Initialize $\gamma \leftarrow 0_m$
2: **for** $j = 1, \ldots, n$ **do**
3: **for** $i = 1, \ldots, m$ **do**
4: $X_{i,j} = B_{i,j} - \beta_j \gamma_i$
5: Set $\gamma_i \leftarrow \gamma_i + q_j X_{i,j}$
6: **end for**
7: **end for**

Algorithm 3.5 Iterative Refinement for Solving Linear Systems $Ax = b$ [8, Theorem 2]

Require: Initial guess $\hat{x}$ such that $A\hat{x} \approx b$ (computed by one of the standard direct methods), factors of A, machine precision ϵ and the number of non-zeros in A (denoted by nnz)
1: Compute residual $r \triangleq A\hat{x} - b$
2: **while** $e = \max_i |r_i|/(|A||\hat{x}| + |b|)_i > (nnz + 1)\epsilon$ **do**
3: Solve $Ad = r$ for d (using the saved factors of A)
4: Update: $\hat{x} \leftarrow \hat{x} - d$
5: Compute residual $r = A\hat{x} - b$
6: **end while**

It can be shown that, as long as A is not "too ill-conditioned" and under other rather complicated conditions, iterative refinement as given in Algorithm 3.5 converges and produces a backward error [8, Theorem 1],

$$e = \max_i \frac{|A\hat{x} - b|_i}{(|A||\hat{x}| + |b|)_i}, \tag{3.19}$$

which is the smallest componentwise relative perturbation of A and b that makes $\hat{x}$ an exact solution to the perturbed system $(A + \Delta A)\hat{x} = b + \Delta b$, i.e.

$$|\delta a_{ij}|/|a_{ij}| \leq e \quad \text{and} \quad |b_i|/|\delta b_i| \leq e \tag{3.20}$$

for all i, j, where δa_{ij} is the element of ΔA in the ith row and jth column, and δb_i is the ith element of Δb. The backward error (3.19) holds in particular for perturbations ΔA that have the same sparsity pattern as A; see [8] for further details. The cost of Algorithm 3.5 is usually negligible to that of obtaining the factorization of A, since it requires only one solve per iteration. In practice, one to two steps of iterative refinement suffice to obtain convergence of Algorithm 3.5.

4 Model Predictive Control

4.1 Linear MPC

We consider dynamical systems with discrete-time, time-varying affine dynamics

$$x_{i+1} = A_i x_i + B_i u_i + k_i, \tag{4.1}$$

where $x_i \in \boldsymbol{R}^{n_x}$ is the state at the current time step i, $u_i \in \boldsymbol{R}^{n_u}$ is the current control input and x_{i+1} is the successor state. In most applications, matrices $A_i \in \boldsymbol{R}^{n_x \times n_x}$, $B_i \in \boldsymbol{R}^{n_x \times n_u}$ and the vector $k_i \in \boldsymbol{R}^{n_x}$ are derived from first principles modeling, by system identification or, in the case of nonlinear dynamics, result from a linearization at the current time step i. As a special case of (4.1), we consider time-invariant linear systems with dynamics

$$x^+ = Ax + Bu, \tag{4.2}$$

where we use a simplified notation, denoting the successor state short by x^+. Throughout the thesis, the following assumption is made.

Assumption 4.1. *The pair (A, B) is stabilizable.*

4.1.1 Lyapunov Stability

If (4.2) is controlled by the control law $u = \kappa(x)$, the closed loop system is given by

$$x^+ = Ax + B\kappa(x)\,. \tag{4.3}$$

Definition 4.1 (Positively invariant (PI) set)**.** *A set $\mathcal{X} \subseteq \boldsymbol{R}^{n_x}$ is a Positively Invariant (PI) set of system* (4.3)*, if $Ax + B\kappa(x) \in \mathcal{X}\ \forall x \in \mathcal{X}$.*

Definition 4.2 (K-class function)**.** *A real-valued function $\alpha : \boldsymbol{R}_+ \to \boldsymbol{R}_+$ belongs to class K if it is continuous, strictly increasing and $\alpha(0) = 0$.*

Definition 4.3 (Lyapunov function)**.** *Let $\mathcal{X}$ be a PI set for system* (4.3) *containing a neighborhood $\mathcal{N}$ of the origin in its interior and let $\underline{\alpha}(\cdot)$, $\bar{\alpha}(\cdot)$ and $\beta(\cdot)$ be K-class functions.*

A non-negative function $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$ *with* $V(0) = 0$ *is called a Lyapunov function in* $\mathcal{X}$ *if:*

$$V(x) \geq \underline{\alpha}(\|x\|) \qquad \forall x \in \mathcal{X}\,, \tag{4.4}$$

$$V(x) \leq \bar{\alpha}(\|x\|) \qquad \forall x \in \mathcal{N}\,, \tag{4.5}$$

$$V(x^+) - V(x) \leq -\beta(\|x\|) \qquad \forall x \in \mathcal{X}\,. \tag{4.6}$$

Theorem 4.1 (Asymptotic stability [132])**.** *If system* (4.3) *admits a Lyapunov function in* $\mathcal{X}$*, then the equilibrium point at the origin is asymptotically stable with region of attraction* $\mathcal{X}$*.*

4.1.2 Nominal MPC

In linear MPC, the goal is to regulate the system state to the origin, while satisfying a set of system constraints. This is expressed in form of the following optimal control problem, minimizing a cost function reflecting a particular control tradeoff over a finite time horizon N with respect to the control inputs $\boldsymbol{u} \triangleq [u_0, \ldots, u_{N-1}]$ and the state trajectory subject to linear or affine dynamics:

$$V_N^\star(x) \triangleq \min_{\boldsymbol{u}} V_N(x, \boldsymbol{u}) = V_f(x_N) + \sum_{i=0}^{N-1} l(x_i, u_i) \tag{4.7a}$$

$$\text{s.t. } x_0 = x, \tag{4.7b}$$

$$(4.1) \text{ or } (4.2) \qquad i = 0, \ldots, N-1, \tag{4.7c}$$

$$(x_i, u_i) \in \mathbb{X} \times \mathbb{U} \qquad i = 0, \ldots, N-1, \tag{4.7d}$$

$$x_N \in \mathbb{X}_f. \tag{4.7e}$$

The sets $\mathbb{X}$ and $\mathbb{U}$ are assumed to be closed and convex sets, containing the origin in their interior. We define

$$\mathbb{X}_N \triangleq \{x \in \boldsymbol{R}^{n_x} \mid \exists \boldsymbol{u} \text{ s.t. } (4.7b) - (4.7e) \text{ is satisfied}\} \tag{4.8}$$

and call $\mathbb{X}_N$ the feasible set for problem (4.7). The stage costs are taken as definite functions $l : \boldsymbol{R}^{n_x} \times \boldsymbol{R}^{n_u} \to \boldsymbol{R}_{++}$. The terminal penalty function is given by a positive definite function $V_f : \boldsymbol{R}^{n_x} \to \boldsymbol{R}_{++}$. The terminal target set $\mathbb{X}_f \subseteq \mathbb{X}$ is a compact convex set containing the origin in its interior.

Example 4.1 (Constrained Linear Quadratic Regulator)**.** *A recurring example in this thesis is the constrained finite horizon linear quadratic regulator problem*

$$\min\ x_N^T P x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \tag{4.9a}$$

$$s.t.\ x_0 = x\,, \tag{4.9b}$$
$$x_{i+1} = Ax_i + Bu_i\,, i = 0, \ldots, N-1\,, \tag{4.9c}$$
$$(x_i, u_i) \in \mathbb{X} \times \mathbb{U}\,, i = 0, \ldots, N-1\,, \tag{4.9d}$$
$$x_N \in \mathbb{X}_f\,, \tag{4.9e}$$

where $Q, P \in \boldsymbol{S}^{n_x}_{++}$ *and* $R \in \boldsymbol{S}^{n_u}_{++}$ *with the initial state* x *as a parameter.*

Assumption 4.2. *It is assumed that* $\mathcal{X}_f$ *is a PI set under the control law* $\kappa_f(x)$ *and that* $V_f(\cdot)$ *is a Lyapunov function in* $\mathcal{X}_f$*:*

A1: $Ax + B\kappa_f(x) \in \mathcal{X}_f$ *and* $\kappa_f(x) \in \mathbb{U}\ \forall\, x \in \mathcal{X}_f$ *,*

A2: $V_f\left(Ax + B\kappa_f(x)\right) - V_f(x) \leq -l(x, \kappa_f(x))\ \forall\, x \in \mathcal{X}_f$ *.*

Theorem 4.2 (Lyapunov Function for RHC [91])**.** *Let* $\boldsymbol{u}^\star(x)$ *denote the minimizer of* (4.7) *when solved for the linear time invariant system* (4.2)*. The resulting state feedback control law is the first element of the optimal input sequence,* $\kappa(x) = u_0^\star(x)$*, such that the closed loop system dynamics under receding horizon control are given by*

$$x^+ = Ax + Bu_0^\star(x)\,. \tag{4.10}$$

Under Assumption 4.2, $V_N^\star(\cdot)$ *is a Lyapunov function for the closed loop system* (4.10)*.*

4.1.3 Suboptimal MPC

In the context of approximate explicit methods for MPC, suboptimal control laws are applied to the system instead of the optimal control law $u_0^\star(x)$. We use the following definition of a suboptimal control sequence and the associated suboptimal cost in Chapter 6.

Definition 4.4 (Suboptimal input sequence and cost)**.** *Define*

$$\tilde{U} \triangleq \{(x, u_0)\,|\exists u_1, \ldots, u_{N-1}\ s.t.\ \text{(4.7b)-(4.7e)}\ \textit{is satisfied}\} \tag{4.11}$$

and let some suboptimal state feedback control law $\tilde{u}(x)$ *be given with* $\tilde{u} : \boldsymbol{R}^{n_x} \to \boldsymbol{R}^{n_u}$ *such that* $(x, \tilde{u}(x)) \in \tilde{U}$ *for all* x *in some convex set* $\tilde{X}$*. Then the suboptimal control sequence* $\tilde{\boldsymbol{u}}^\star \triangleq [\tilde{u}_0, \ldots, \tilde{u}_{N-1}]$ *is defined in the following as*

$$\tilde{\boldsymbol{u}}^\star \triangleq \arg\min_{\tilde{\boldsymbol{u}}} V(x, \tilde{\boldsymbol{u}}) = V_f(\tilde{x}_N) + \sum_{i=0}^{N-1} l(\tilde{x}_i, \tilde{u}_i) \tag{4.12a}$$
$$s.t.\quad \text{(4.7b)} - \text{(4.7e)}\,, \tag{4.12b}$$
$$u_0 = \tilde{u}(x)\,, \tag{4.12c}$$

where $\tilde{V}(x) \triangleq V\left(x, \tilde{\boldsymbol{u}}^\star\right)$ *is the suboptimal cost.*

The following theorem gives a sufficient condition on stability of the closed loop system under some approximate control law $\tilde{u}(x)$.

Theorem 4.3 ([116] and [72, Theorem 1]). *Let $\tilde{u}(x)$ be the suboptimal control law of (4.7) as defined in Definition 4.4 and $\tilde{V}(x)$ the associated suboptimal cost for a suboptimal input sequence. The optimal value function $V_N^\star(x)$ is a Lyapunov function for the closed loop system $x^+ = Ax + B\tilde{u}(x)$, if $\tilde{V}(x) - V_N^\star(x) \leq \mu l(x, \tilde{u}(x))$ for all $x \in \tilde{\mathcal{X}}$ and some $\mu \in [0, 1)$, where $l(\cdot, \cdot)$ is the stage cost in (4.7a), and the closed-loop system $x^+ = Ax + B\tilde{u}(x)$ is asymptotically stable with region of attraction $\tilde{\mathcal{X}}$.*

4.2 Hybrid Models and MPC

Physical processes that interact with or are part of a system that involves some form of logic can be modeled in the general framework of *mixed logical dynamical* (MLD) systems [13]. Examples are power grids with continuous dynamics and switches in transmission lines or the drivetrain of cars coupled with a gear box. Similarly, any system controlled by decision rules (i.e. some form of computational logic) and systems with discrete inputs or piece-wise affine dynamics can be modeled in the MLD framework. By appropriate transformations, a discrete-time MLD system with linear or piece-wise affine dynamics can be represented in the form [127]

$$\begin{bmatrix} x_c \\ x_d \end{bmatrix}^+ = \begin{bmatrix} A_c & A_{cd} \\ A_{dc} & A_{dd} \end{bmatrix} \begin{bmatrix} x_c \\ x_d \end{bmatrix} + \begin{bmatrix} B_c & B_{cd} \\ B_{dc} & B_{dd} \end{bmatrix} \begin{bmatrix} u_c \\ u_d \end{bmatrix} + \begin{bmatrix} B_{2c} \\ B_{2d} \end{bmatrix} d + \begin{bmatrix} B_{3c} \\ B_{3d} \end{bmatrix} z \tag{4.13a}$$

$$\begin{bmatrix} y_c \\ y_d \end{bmatrix} = \begin{bmatrix} C_c & C_{cd} \\ C_{dc} & C_{dd} \end{bmatrix} \begin{bmatrix} x_c \\ x_d \end{bmatrix} + \begin{bmatrix} D_c & D_{cd} \\ D_{dc} & D_{dd} \end{bmatrix} \begin{bmatrix} u_c \\ u_d \end{bmatrix} + \begin{bmatrix} D_{2c} \\ D_{2d} \end{bmatrix} d + \begin{bmatrix} D_{3c} \\ D_{3d} \end{bmatrix} z \tag{4.13b}$$

$$0 \leq E_1 \begin{bmatrix} x_c \\ x_b \end{bmatrix} + E_2 \begin{bmatrix} u_c \\ u_b \end{bmatrix} + E_3 d + E_4 z + E_5 \tag{4.13c}$$

where $x_c \in \boldsymbol{R}^{n_{xc}}$, $u_c \in \boldsymbol{R}^{n_{uc}}$ are the continuous states and inputs, respectively, and $x_b \in \{0, 1\}^{n_{xb}}$, $u_b \in \{0, 1\}^{n_{ub}}$ are the binary states and inputs of the system. (These models are therefore also called *hybrid*.) The auxiliary variables $z \in \boldsymbol{R}^{n_z}$ and $d \in \{0, 1\}^{n_d}$ define the interplay between continuous and discrete variables via (4.13c), and their effect on the state evolution of the system (4.13a) and the outputs (4.13b). Software is available to transform a high-level description of MLD systems into the canonical form (4.13), for example the Hybrid Systems Description Language (HYSDEL) compiler [126].

When using MLD models in a finite horizon optimal control framework (4.7), the resulting optimization problem is a mixed-integer (oftentimes linear or quadratic) program due to the

integer variables x_b, u_b and d. Efficient heuristics exist to compute globally optimal solutions to such problems, for example by branch-and-bound techniques [17].

5 Machine Learning

5.1 Classification

One of the basic problems of supervised learning is to infer from M given tuples of samples

$$(\chi_i, z_i) \in \mathcal{P} \subseteq \boldsymbol{R}^n \times \{-1, 1\}\,,\ i = 1, \ldots, M\,, \tag{5.1}$$

a decision function $g : \mathcal{P} \to \{-1, 1\}$, which, with high probability, makes correct decisions about whether an *unseen* sample $\chi \in \mathcal{P}$ has label $z(\chi) = -1$ or $z(\chi) = +1$. This is generally a difficult problem as the optimization operates in an infinite dimensional *function* space, which makes the problem fundamentally more difficult than finding a vector in Euclidian space, for example. In order to obtain a tractable algorithm, one usually restricts the function space of g to the space of functions that can be parametrized by a finite number of parameters [30]. In the following, we discuss two effective parametrizations and the corresponding training algorithms in order to classify unseen data.

5.1.1 Support Vector Machines

The basic formulation of the support vector machine for classification is based on linear discriminant functions $g : \boldsymbol{R}^n \to \boldsymbol{R}$, i.e.

$$g(\chi) = w^T \chi + b\,, \tag{5.2}$$

where w is the normal vector and b the offset. The sign of $g(x)$ is then used to construct the binary decision, i.e.

$$z(\chi) = \operatorname{sign}\left(g(\chi)\right)\,. \tag{5.3}$$

The learning problem is to find w and b given the observations (5.1) in order to classify unseen data. Soft margin SVMs compute the *maximum margin hyperplane* with soft constraints [23, 29] by solving

$$\begin{aligned} \min_{w,b,\xi}\ & \frac{1}{2}\|w\|^2 + cf(\xi) \\ \text{s.t. } & z_i(w^T\chi_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, M \\ & \xi_i \geq 0 \qquad\qquad\qquad\ \ \forall i = 1, \ldots, M \end{aligned} \tag{5.4}$$

Since a linear function may be unable to strictly to separate the two classes fully, soft constraints are employed in (5.4), relaxing the absolute separation requirement using slack variables $\xi \geq 0$. The convex function $f : \boldsymbol{R}^M \rightarrow \boldsymbol{R}_+$ penalizes the violation of the separation constraints, while the weight c in the cost trades off the two goals of minimum margin constraint violation vs. margin maximization, where the latter corresponds to good generalization performance [29].

There are three important points to note about SVM classification. First, (5.4) is a convex problem and thus efficiently solvable, even for large data sets (each data point introduces constraints). Second, (5.4) is not solved directly, but first dualized, which allows for solving the problem without ever explicitly calculating w. The discriminant function g can be expressed by a weighted sum of inner products of χ with the data points χ_i,

$$g(\chi) = \sum_{i=1}^{M} \alpha_i z_i \langle \chi_i, \chi \rangle + b \,, \tag{5.5}$$

where α_i are the optimal Lagrange multipliers associated with the inequality constraints in (5.4). This representation of g makes it possible to replace the inner product $\langle \chi_i, \chi \rangle$ by *any* inner product in a lifted space, expressed by a (generally nonlinear) *kernel* function:

$$g(\chi) = \sum_{i=1}^{M} \alpha_i z_i k(\chi_i, \chi) + b \,, \tag{5.6}$$

where

$$k(\chi_i, \chi) \triangleq \langle \Phi(\chi_i), \Phi(\chi) \rangle \tag{5.7}$$

with $\Phi : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$, $m \gg n$, being a (possibly unknown) *feature map*. This so-called *kernel trick* enables the efficient search for non-linear discriminant functions (5.6) and makes SVMs a very powerful tool in high dimensional classification problems with little training data [115]. Note that although nonlinear kernels are used, the optimization problem to be solved is still convex (the dual of (5.4)). Third, the resulting solution is usually *sparse*, i.e. only few α_i are nonzero, and therefore only few data samples have to be stored and evaluated online in order to evaluate $g(\chi)$. These samples are called *support vectors* due to their nature of "supporting the hyperplane". For further details on SVMs the interested reader is referred to the tutorials on SVM classification [23] and regression [118].

5.1.2 (Adaptive) Boosting

Adaptive Boosting, or short *AdaBoost*, is a classification algorithm that was introduced at about the same time as SVMs with the seminal paper by Freund and Schapire, [46]. The

underlying idea is that the learning problem might be too complex for a single learning machine. Instead, one can *boost* the performance of many simple learning machines, so-called *weak learners*, by combining them in an appropriate way to a powerful learning machine. Each weak learner has to do only slightly better than random for convergence of the algorithm, and it can be shown that the algorithm minimizes an exponential loss function on wrong decisions [46].

The weak learners are (simple) discriminant functions $h_j : \mathcal{P} \to \{-1, 1\}$, $j = 1, \ldots, L$, which can be evaluated very quickly but usually have mediocre performance. The final discriminant function is a linear combination of L weak learners,

$$g(\chi) = \sum_{j=1}^{L} a_j h_j(\chi) \,, \tag{5.8}$$

which yields a strong classifier. As for SVMs, the binary decision is obtained by taking the sign of the discriminant function, see (5.3). The advantage of AdaBoost is that it works with basically all types of discriminant functions, for example *classification and regression trees* (CARTs) [21], which are difficult to cast in a closed form.

Training AdaBoost Classifiers

Training an AdaBoost classifier is an iterative process. The following assumption has to be met by the weak learners.

Assumption 5.1. *The training procedure of the j^{th} weak learner returns a function h_j and a training error rate ϵ_j, i.e. the fraction of misclassified points, given the training samples (χ_i, z_i) and* importance weights *$w_{i,j}$, $i = 1, \ldots, M$, that are used to assign a relative relevance to the sample points.*

At the beginning of the algorithm, all points are equally important, i.e. $w_{i,1} = 1 \; \forall i = 1, \ldots, M$. After training the jth weak learner that produces an error ϵ_j on the given (weighted) data, its coefficient a_j in (5.8) is computed according to

$$a_j(\epsilon_j) = \frac{1}{2} \log \left(\frac{1 - \epsilon_j}{\epsilon_j} \right) \,. \tag{5.9}$$

Intuitively, the weighting rule rates weak learners that perform well higher.

Next, the importance weights w_i are adjusted in order to emphasize wrongly classified data while attenuating correctly classified samples. The data weight factors for the next weak learner $j + 1$ are given by

$$w_{i,j+1}(a_j, \chi_i, z_j) = \exp\left(-a_j z_i h_j(\chi_i)\right) \,. \tag{5.10}$$

Then, the next weak learner $h_{j+1}(\chi)$ is trained using the reweighted data, and the process of weak learner weight computation (5.9), data reweighting (5.10) and weak learner training is repeated until L weak learners have been trained. See [46] for more details.

5.2 Kernel Regression

In classical nonlinear regression, the aim is to approximate a function $f : \Omega \subseteq \boldsymbol{R}^n \to \boldsymbol{R}$ in some function space $\mathcal{F}$, by a function $\tilde{f} : \Omega \to \boldsymbol{R}$ in some space $\tilde{\mathcal{F}} \subseteq \mathcal{F}$, such that the approximation error $\|f - \tilde{f}\|_{\mathcal{F}}$, where $\|\cdot\|_{\mathcal{F}}$ is a norm defined on $\mathcal{F}$, is minimized:

$$\min_{\tilde{f} \in \mathcal{F}} \|f - \tilde{f}\|_{\mathcal{F}} .$$

Since the minimization is carried out over the space of functions $\tilde{\mathcal{F}}$, it is in general infinite dimensional and computationally intractable. A common approach is to parametrize $\tilde{\mathcal{F}}$ by a weighted sum of L nonlinear basis functions $k_i : \Omega \to \boldsymbol{R}$ and to minimize the *empirical error* instead. This involves drawing M samples χ_i according to a distribution $\mathcal{P} : \Omega \to \boldsymbol{R}_+$. At the samples, $f(\cdot)$ is evaluated to yield $f_i \triangleq f(\chi_i)$. With

$$k(x) \triangleq [k_1(x), \ldots, k_L(x)]^T , \tag{5.11}$$

the finite dimensional problem of *learning* f is then given by

$$\theta^\star \in \arg\min_{\theta} \sum_{i=1}^{M} \mathcal{L}(r_i) \tag{5.12a}$$

$$\text{s.t. } r_i = f(\chi_i) - \tilde{f}(\chi_i) \quad i = 1, \ldots, M \tag{5.12b}$$

$$\tilde{f}(\chi_i) = \theta^T k(\chi_i) \qquad i = 1, \ldots, M \tag{5.12c}$$

where $\theta^\star \in \boldsymbol{R}^L$ is in the set of weights that minimize the empirical error under loss function $\mathcal{L} : \boldsymbol{R} \to \boldsymbol{R}_+$. For some common choices of basis functions and the relation to kernel methods see e.g. [118] and [48].

Part II

Approximation of MPC Control Laws from Data

6 Certified Learning of MPC Control Laws

6.1 Introduction

It is well known that for certain problem structures, the quadratic or linear programs arising from linear MPC (4.7) can be solved offline using multiparametric programming [14]. The solution is a piece-wise affine (PWA) state feedback control law defined over polyhedral regions which form partitioning of the state space. Thus, online evaluation of the control law reduces to a point location problem, which can be solved efficiently using a binary search tree [70, 111]. This so-called *explicit* MPC enables finite time optimal control of fast systems such as power electronics, see e.g. [85] and the references therein.

Explicit MPC is however limited mainly in two ways. First, the computation of the parametric solution may be computationally intractable, even for low dimensional systems. This stems from the fact that the solution to multiparametric programs can grow exponentially in the problem size. Second, if a PWA state feedback map has been obtained, generation of the associated search tree may fail or lead to lookup tables of sizes prohibitive for embedded control hardware.

In this chapter, we propose a new synthesis method for suboptimal explicit MPC based on function approximation that yields low-complexity control laws with a low degree of suboptimality, enhancing applicability of MPC on resource constrained platforms. Our method is based on approximating the optimal PWA control law from point-wise sample values by a weighted sum of nonlinear basis functions. As the samples are chosen randomly, we call this process *learning*, and the learning problem consists of finding a suitable set of weights by means of convex optimization. Our method permits the use of any differentiable function as basis function, which gives freedom to the control designer to trade off approximation accuracy vs. memory and computing power limitations. As the basis functions can be evaluated fully in parallel, the resulting state feedback controller is particularly suited for highly concurrent embedded platforms such as FPGAs or VLIW processors. Moreover, our algorithm relies solely on point-wise values of the optimal solution and is hence potentially suited for MPC problems of high complexity, for which the explicit solution cannot be computed. We

approximate only the first control move of the input trajectory, which is sufficient for closed loop control in a receding horizon implementation.

Main Contributions

While standard methods from machine learning, e.g. soft margin support vector regression [118] or more general kernel methods (cf. Section 5.2 and [30, 115]), may work well in practice, no system theoretic guarantees are provided when applying a standard learning algorithm to the problem of learning MPC control laws from sampled data. The main contribution of this work is to present a learning problem that provides guarantees for input and state constraint satisfaction, recursive feasibility and stability of the closed loop system with only mild requirements on the class of basis functions. This is accomplished by deriving suitable tractable convex constraints which can be added to any of the standard learning setups relying on convex optimization; in particular, any combination with learning methods that favor sparse solutions – l_1-regularized least squares, for instance – can be used to approximate the control law. This allows to trade off approximation performance vs. memory requirements and online evaluation time, while ensuring a certain minimal performance of the learned controller that guarantees stability of the closed-loop system.

Related Work

A vast number of other approximate explicit MPC schemes have been proposed in the literature [2]. While geometric methods (e.g. [12,14,57,71,105]) are based on the polyhedral partitioning of the state space, our approach belongs to the class of function approximation methods. An early work in this context is [106], employing neural networks and a gradient-based search to find the associated weights through non-convex optimization. Function approximation by PWA basis functions is considered in [15], where the offline problem is a convex quadratic or linear program. However, the optimal feedback control law is needed to compute the approximation, and closed-loop stability is verified a-posteriori by constructing a piece-wise linear (PWL) Lyapunov function, which is generally difficult to obtain. Both the exact solution and a PWL Lyapunov function are also prerequisites for stability guarantees given in [77], which is based on a global polynomial approximation and is restricted to $1, \infty$-norm cost functions. Barycentric interpolations on a hierarchical grid are computed in [122] by a series of convex optimization problems.

Outline

The outline of the chapter is as follows: In Section 6.2, we define the MPC formulation under consideration. In Section 6.3, a semi-infinite learning problem is presented, which, if feasible, provides the desired closed loop guarantees. Subsequently, finite tractable convex inner approximations of the semi-infinite problem are derived in Section 6.4. Section 6.5 concludes this chapter with a numerical example, showing some of the properties of the proposed method.

6.2 Problem Setup

Throughout this chapter, we consider a linear discrete time dynamical system (4.2), and the standard finite horizon constrained linear quadratic regulator MPC setup (4.9) with polytopic constraint sets

$$\mathbb{X} \triangleq \{x \in \boldsymbol{R}^{n_x} \mid H_X x \leq h_X\} \ , \tag{6.1a}$$

$$\mathbb{U} \triangleq \{u \in \boldsymbol{R}^{n_u} \mid H_u u \leq h_u\} \ . \tag{6.1b}$$

The terminal target set $\mathbb{X}_f \subseteq \mathbb{X}$ is a compact convex polytope containing the origin in its interior.

The goal is to approximate the receding horizon feedback control law $\kappa(x) = u_0^\star(x)$ by a suboptimal control law $\tilde{u}(x)$ by learning from data samples $(\chi_i, u_0^\star(\chi_i))$, where the χ_i are generated by randomly sampling the state space. We use the general kernel regression formulation (5.12) from Section 5.2, i.e. the approximate control law is parameterized as a linear combination of L nonlinear basis functions $k_j : \Omega \to \boldsymbol{R}$, $j = 1, \ldots, L$. To generate the data χ_i, we define $\Omega = \mathbb{X}_N$ as the domain of the basis functions $k_i(\cdot)$, which ensures that the samples χ_i, $i = 1, \ldots, M$ are feasible for the original MPC problem (4.7).

6.3 The Semi-Infinite Learning Problem

The learning problem in (5.12) has to be adjusted in order to fit our purpose of learning an MPC control law. In particular, we extend (5.12) to learning vector valued functions and define the suboptimal control law as follows.

Definition 6.1 (Suboptimal control law)**.** *The suboptimal state feedback control law employs a dual-mode strategy [91, 95] and is given by*

$$\tilde{u}(x) \triangleq \begin{cases} \Theta k(x) & x \notin \mathbb{X}_f \\ \kappa_f(x) & x \in \mathbb{X}_f \end{cases} , \tag{6.2}$$

where $\Theta \in \boldsymbol{R}^{n_u \times L}$ *is a set of weights and* $k(x)$ *is a column vector of basis functions evaluated at the state* x *as defined in* (5.11)*. The suboptimal closed loop dynamics are given by*

$$x^+ = Ax + B\tilde{u}(x)\,. \tag{6.3}$$

We now state a rather general result for this class of suboptimal controllers. By incorporating input and state constraint satisfaction, recursive feasibility and stability constraints into the learning problem (5.12), we impose system theoretic constraints on (6.2) and (6.3). Thus the following theorem is key to our method.

Theorem 6.1 (Primary learning problem)**.** *Let* $\tilde{\mathbb{X}} \subseteq \mathbb{X}_N$ *be a polytopic set containing the origin in its interior. Furthermore, let* $\chi_i \in \mathbb{X}_N$*,* $i = 1, \ldots, M$ *denote randomly drawn samples. If the semi-infinite optimization problem over the weights* Θ *of* (6.2)*,*

$$\min_{\Theta} \sum_{i=1}^{M} \mathcal{L}(r_i) \tag{6.4a}$$

$$\text{s.t. } r_i = u_0^*(\chi_i) - \tilde{u}(\chi_i), \qquad i = 1, \ldots, M\,, \tag{6.4b}$$

$$\tilde{u}(x) \in \mathbb{U} \qquad \forall x \in \tilde{\mathbb{X}}\,, \tag{6.4c}$$

$$Ax + B\tilde{u}(x) \in \tilde{\mathbb{X}} \qquad \forall x \in \tilde{\mathbb{X}}\,, \tag{6.4d}$$

$$V_N^*(x^+) - V_N^*(x) \leq -\varepsilon l(x, 0) \qquad \forall x \in \tilde{\mathbb{X}}\,, \tag{6.4e}$$

with x^+ *as given in* (6.3)*,* $r_i \in \boldsymbol{R}^{n_u}$*,* $\mathcal{L} : \boldsymbol{R}^{n_u} \to \boldsymbol{R}_+$ *is feasible for some* $\varepsilon \in \boldsymbol{R}_{++}$*, then* $\tilde{\mathbb{X}}$ *is a PI set for the closed loop system* (6.3)*. Furthermore,* (6.3) *is asymptotically stable with region of attraction* $\tilde{\mathbb{X}}$*.*

Proof. The origin is an equilibrium point using (6.2). Input constraints are satisfied due to (6.4c). By (6.4d), $\tilde{\mathbb{X}}$ is a PI set for (6.3), and by $\tilde{\mathbb{X}} \subseteq \mathbb{X}_N \subseteq \mathbb{X}$, state constraints are satisfied for any trajectory of (6.3) starting in $\tilde{\mathbb{X}}$. Feasibility of (6.4e) yields the claim on asymptotic stability. □

Remark 6.1. *As the suboptimal control law employs a dual-mode strategy, it suffices to impose* (6.4c)*,* (6.4d) *and* (6.4e) *for all* x *in* $\tilde{\mathbb{X}} \backslash \mathbb{X}_f$*, as the terminal controller* $\kappa_f(x)$ *satisfies these conditions by Assumption 4.2.*

The primary learning problem (6.4) is merely theoretical, as the optimization problem has an infinite number of constraints. In the next section, we derive tractable convex inner approximations of the constraints (6.4c), (6.4d) and (6.4e) with a finite number of constraints.

6.4 Derivation of a Tractable Learning Problem

We will first derive a tractable formulation of the input constraints (6.4c), by making use of a robust optimization reformulation technique, see e.g. [16]. A similar reformulation is also employed in the subsequent derivation of convex conditions sufficient for (6.4d). Finally, we give sufficient finite convex constraints for (6.4e).

Some mild assumptions are required for the basis functions and the involved sets, which are met by all standard basis functions used in learning [63].

Assumption 6.1. *The basis functions $k_i(\cdot)$ are continuously differentiable on $\tilde{\mathbb{X}}$.*

Assumption 6.2. *Let the set $\tilde{\mathbb{X}} \subseteq \mathbb{X}$ be compact and convex with non-empty interior and implicit representation, i.e. $\tilde{\mathbb{X}} \triangleq \{x \in \boldsymbol{R}^{n_x} \mid \exists \mathbf{u} : \mathcal{G}x + \mathcal{H}\mathbf{u} \leq b, \}$ with $\mathcal{G} \in \boldsymbol{R}^{m \times n_x}$, $\mathcal{H} \in \boldsymbol{R}^{m \times N n_u}$, $b \in \boldsymbol{R}^m$, where $\mathbf{u}$ is the input vector. For details on how to derive such a representation see e.g. [18].*

Lemma 6.1 (First order Taylor approximation)**.** *Let a region*

$$\mathcal{R} \triangleq \left\{x \in \tilde{\mathbb{X}} |\ H_r x \leq h_r\right\} \subseteq \tilde{\mathbb{X}} \tag{6.5}$$

be defined by a finite number of linear inequalities with $h_r \in \boldsymbol{R}^{N_r}$. The first order approximation of $k(x)$ around a point $x_c \in \mathcal{R}$ allows for a representation

$$k\left(x\right) \in J_k\left(x_c\right)\left(x - x_c\right) + k\left(x_c\right) \oplus \mathbb{W}\,,\ x \in \mathcal{R}\,, \tag{6.6}$$

where $J_k(x_c) \triangleq \nabla_x k(x)|_{x=x_c} \in \boldsymbol{R}^{L \times n_x}$ is the Jacobian of $k(\cdot)$ evaluated at x_c and

$$\mathbb{W} \triangleq \{w \in \boldsymbol{R}^L \mid H_w w \leq h_w\} \tag{6.7}$$

is a compact convex polytope, defined by $H_w \in \boldsymbol{R}^{N_w \times L}$ and $h_w \in \boldsymbol{R}^{N_w}$, overbounding the remainder of the Taylor expansion.

Proof. Since $k(\cdot)$ is differentiable on $\mathbb{X}$ by Assumption 6.1, the Jacobian exists. Since $\tilde{\mathbb{X}} \subseteq \mathbb{X}$ is bounded, the remainder $w(x) \triangleq k(x) - J_k(x_c)(x - x_c) - k(x_c)$ must be bounded for all $x \in \mathcal{R}$. □

Remark 6.2. *In general, the map of $\mathcal{R}$ under $w(\cdot)$ as defined in the proof of Lemma 6.1 is nonlinear, but can be overbounded by interval arithmetic methods [75], yielding the polytope $\mathbb{W}$.*

Assumption 6.3. *The set $\mathbb{W}$ has a non-empty interior.*

6.4.1 Satisfying Input Constraints

The following theorem states the tractable counterpart of invariance condition (6.4c).

Theorem 6.2 (Tractable input constraints)**.** *Define $\mathcal{R}$ and $\mathbb{W}$ as in Lemma 6.1 and fix $x_c \in \mathcal{R}$. Furthermore, define*

$$\mathcal{A} \triangleq \begin{bmatrix} \mathcal{G} & 0 & \mathcal{H} \\ 0 & H_w & 0 \\ H_r & 0 & 0 \end{bmatrix}, \quad \mathcal{B} \triangleq \begin{bmatrix} b \\ h_w \\ h_r \end{bmatrix},$$
$$\mathcal{C}(\Theta, x_c) \triangleq \begin{bmatrix} H_u \Theta J_k(x_c) & H_u \Theta & 0 \end{bmatrix},$$
$$\mathcal{D}(x_c) \triangleq h_u + H_u \Theta \left(J_k(x_c) x_c - k(x_c) \right),$$

and let N_u be the number of rows of H_u. Then, the existence of $\Theta \in \boldsymbol{R}^{n_u \times L}$ and $Y \in \boldsymbol{R}^{N_u(m+N_w+N_r)}$ such that

$$(I_{N_u} \otimes \mathcal{B})^T Y \leq \mathcal{D}(x_c), \tag{6.8a}$$
$$(I_{N_u} \otimes \mathcal{A})^T Y = vec\left(\mathcal{C}(\Theta, x_c)^T \right), \tag{6.8b}$$
$$Y \geq 0, \tag{6.8c}$$

is a sufficient condition for (6.4c) *on $\mathcal{R}$, i.e.* (6.8) *implies $\tilde{u}(x) \in \mathbb{U}\ \forall x \in \mathcal{R}$.*

Proof. Using Lemma 6.1, a sufficient condition for $\tilde{u}(x) \in \mathbb{U}\ \forall x \in \mathcal{R}$ is given by

$$\begin{aligned} & H_u \Theta \left(J_k(x_c)(x - x_c) + k(x_c) + w \right) \leq h_u \\ & \qquad \forall x \in \mathcal{R},\ \forall w \in \mathbb{W} \qquad (6.9a) \\ \Leftrightarrow\ & H_u \Theta J_k(x_c) x + H_u \Theta w \leq h_u + H_u \Theta \left(J_k(x_c) x_c - k(x_c) \right) \\ & \qquad \forall x \in \mathcal{R},\ \forall w \in \mathbb{W}. \qquad (6.9b) \end{aligned}$$

Using the matrices introduced in the theorem, we rewrite the infinite condition as constraints on the row-wise maxima of the LHS of (6.9b) as follows:

$$\max_{x \in \mathcal{R}, w \in \mathbb{W}} \mathcal{C}_i(\Theta, x_c) \begin{bmatrix} x \\ w \\ 0 \end{bmatrix} \leq \mathcal{D}_i(x_c), \tag{6.10a}$$

$$\Leftrightarrow \qquad \max_{x, w, \mathbf{u}} \mathcal{C}_i(\Theta, x_c) \begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \leq \mathcal{D}_i(x_c), \tag{6.10b}$$

$$\text{s.t. } \mathcal{A}\begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \leq \mathcal{B} \tag{6.10c}$$

for $i = 1, \ldots, N_u$, where $\mathcal{C}_i(\Theta, x_c)$, $\mathcal{D}_i(x_c)$ denote the ith row of the corresponding matrices. Since both $\tilde{\mathbb{X}}$ and $\mathbb{W}$ are compact convex sets with nonempty interior (Assumption 6.2 and Assumption 6.3), strong duality for LPs holds and we then have that

$$\begin{aligned} \max_{x,w,\mathrm{u}} \mathcal{C}_i(\Theta, x_c)\begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \quad &= \quad \min_{y_i} \mathcal{B}^T y_i \leq \mathcal{D}_i(x_c) \\ \text{s.t. } \mathcal{A}\begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \leq \mathcal{B} \quad &\qquad \text{s.t. } \mathcal{A}^T y_i = \mathcal{C}_i(\Theta, x_c)^T, \\ &\qquad\quad y_i \geq 0 \end{aligned} \tag{6.11}$$

for $i = 1, \ldots, N_u$. Since the existence of y_i satisfying (6.11) is sufficient, the minimization in (6.11) can be dropped. Repeating this construction for every row of $\mathcal{C}(\Theta, x_c)$ and defining $Y \triangleq (y_1, \ldots, y_{N_u})$ yields the claim. □

Remark 6.3 (Input feasibility for union of regions)**.** *Let R regions $\mathcal{R}^i$, $i = 1, \ldots, R$ correspond to a covering of $\tilde{\mathbb{X}}$, i.e. $\cup_{i=1}^{R} \mathcal{R}^i = \tilde{\mathbb{X}}$ and let Theorem 6.2 be satisfied for all R regions with separate Y^i and one Θ. Then Theorem 6.2 holds on $\tilde{\mathbb{X}}$.*

Remark 6.4 (Tractability of (6.8))**.** *The number of constraints in Theorem 6.2 depends polynomially on the dimensions involved, with $N_u(1 + m + N_w + N_r)$ linear inequality and $N_u(n_x + L + Nn_u)$ linear equality constraints in $N_u(m + N_w + N_r)$ variables.*

6.4.2 Recursive Feasibility

The following theorem states the tractable counterpart of invariance condition (6.4d).

Theorem 6.3 (Tractable invariance constraint)**.** *Let $\tilde{\mathbb{X}}$ be given by an explicit polytopic representation, i.e. $\tilde{\mathbb{X}} = \{x \in \boldsymbol{R}^{n_x} \mid H_x x \leq h_x\}$ with $h_x \in \boldsymbol{R}^{N_x}$. Let furthermore $x_c \in \mathcal{R} \subseteq \tilde{\mathbb{X}}$ be fixed and $\mathcal{R}$, $\mathcal{A}$, $\mathcal{B}$ and $\mathbb{W}$ be defined as in Theorem 6.2. Define furthermore*

$$\begin{aligned} \mathcal{E}(\Theta, x_c) &\triangleq \begin{bmatrix} H_x(A + B\Theta J_k(x_c)) & H_x B\Theta & 0 \end{bmatrix}, \\ \mathcal{F}(\Theta, x_c) &\triangleq h_x + H_x B\Theta(J_k(x_c)x_c - k(x_c)). \end{aligned}$$

Then a sufficient condition for the semi-infinite constraint

$$x \in \mathcal{R} \Rightarrow Ax + B\tilde{u}(x) \in \tilde{\mathbb{X}}$$

is the existence of weights $\Theta \in \boldsymbol{R}^{n_u \times L}$ *and a column vector* $\mathcal{Y} \in \boldsymbol{R}^{N_x(m+N_w+N_r)}$ *such that*

$$(I_{N_x} \otimes \mathcal{B})^T \mathcal{Y} \leq \mathcal{F}(\Theta, x_c) , \tag{6.12a}$$

$$(I_{N_x} \otimes \mathcal{A})^T \mathcal{Y} = vec\left(\mathcal{E}(\Theta, x_c)^T\right) , \tag{6.12b}$$

$$\mathcal{Y} \geq 0 . \tag{6.12c}$$

Proof. By linearizing $\tilde{u}(x)$ according to Lemma 6.1, a sufficient condition for (6.12a) is

$$Ax + B\Theta\left(J_k(x_c)(x - x_c) + k(x_c) + w\right) \in \tilde{\mathbb{X}}$$

$\forall x \in \mathcal{R}, \forall w \in \mathbb{W}$, which, by using the polytopic representation of $\tilde{\mathbb{X}}$ and reordering the constraints, yields

$$\begin{aligned} &H_x\left(Ax + B\Theta\left(J_k(x_c)(x - x_c) + k(x_c) + w\right)\right) \leq h_x \\ &\quad \forall x \in \mathcal{R}, \forall w \in \mathbb{W} \end{aligned} \tag{6.13a}$$

$$\begin{aligned} &H_x\left(A + B\Theta J_k(x_c)\right)x + H_x B\Theta w \\ &\quad \leq h_x + H_x B\Theta\left(J_k(x_c)\, x_c - k(x_c)\right) \\ &\quad \forall x \in \mathcal{R}, \forall w \in \mathbb{W} . \end{aligned} \tag{6.13b}$$

With $\mathcal{E}_i(\Theta, x_c)$ and $\mathcal{F}_i(\Theta, x_c)$, $i = 1, \ldots, N_x$ denoting the ith row of the corresponding matrices, we obtain the equivalent condition: $\exists \Theta \in \boldsymbol{R}^{n_u \times L}$ such that

$$\max_{x \in \mathcal{R}, w \in \mathbb{W}} \mathcal{E}_i(\Theta, x_c) \begin{bmatrix} x \\ w \\ 0 \end{bmatrix} \leq \mathcal{F}_i(\Theta, x_c) , \tag{6.14a}$$

$$\Leftrightarrow \quad \max_{x, w, \mathrm{u}} \mathcal{E}_i(\Theta, x_c) \begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \leq \mathcal{F}_i(\Theta, x_c) , \tag{6.14b}$$

$$\text{s.t. } \mathcal{A} \begin{bmatrix} x \\ w \\ \mathbf{u} \end{bmatrix} \leq \mathcal{B} \tag{6.14c}$$

for $i = 1, \ldots, N_x$. The maximization in (6.14) is structurally equivalent to that in (6.10), hence the construction from this point on is along the lines of the proof of Theorem 6.2. Dualizing, dropping the minimum and stacking the dual variables obtained from these row-wise constructed dual problems yields the claim. □

Remark 6.5. *Any convex inner approximation of $\tilde{\mathbb{X}}$, defined as*

$$\bar{\mathbb{X}} \triangleq \{x \in \boldsymbol{R}^{n_x} \mid H_x x \leq h_x\} \subset \tilde{\mathbb{X}}\,, \tag{6.15}$$

can be used in Theorem 6.3 to impose $x \in \mathcal{R} \Rightarrow Ax + B\Theta k(x) \in \bar{\mathbb{X}}$, yielding a sufficient condition for invariance. This is important in practice, where $\tilde{\mathbb{X}}$ is usually defined only implicitly.

Remark 6.6 (Invariance for union of regions)**.** *Let R regions $\mathcal{R}^i$, $i = 1, \ldots, R$ correspond to a covering of $\tilde{\mathbb{X}}$ as in Remark 6.6 and let Theorem 6.3 be satisfied for all R regions with separate $\mathcal{Y}^i$ and one Θ. Then Theorem 6.3 holds on $\tilde{\mathbb{X}}$, and subsequently $\tilde{\mathbb{X}}$ is an invariant set for* (6.3)*.*

Remark 6.7 (Tractability of (6.12))**.** *The number of constraints in Theorem 6.3 depends polynomially on the dimensions involved, with $N_x\,(1 + m + N_w + N_r)$ linear inequality and $N_x\,(n_x + L + Nn_u)$ linear equality constraints in $N_x\,(m + N_w + N_r)$ variables.*

6.4.3 Stability

We will now derive sufficient conditions for (6.4e). We state convex constraints which can be added to the learning problem to impose that the optimal value function $V_N^*(\cdot)$, which, by Theorem 4.2, is a Lyapunov function for (4.10), is also a Lyapunov function for (6.3). The following results will be useful for the derivation of this result in Theorem 6.4.

Lemma 6.2. *Let $x_i^+(x)$ denote the ith element of the successor state of x satisfying* (6.3)*. Let furthermore a_i and b_i denote the ith row of A and B, respectively. Then there exist $\underline{y}_i \in \boldsymbol{R}_+^{N_w}$ and $\bar{y}_i \in \boldsymbol{R}_+^{N_w}$ such that*

$$\underline{x}_i^+\left(x, \Theta, \underline{y}_i\right) \leq x_i^+(x) \leq \bar{x}_i^+\left(x, \Theta, \bar{y}_i\right)\ \forall x \in \tilde{\mathbb{X}}\,, \tag{6.16a}$$

$$-H_w^T \underline{y}_i = \Theta^T b_i^T\,, \qquad H_w^T \bar{y}_i = \Theta^T b_i^T\,, \tag{6.16b}$$

where

$$\begin{aligned}
\underline{x}_i^+(x, \Theta, \underline{y}_i) &\triangleq x^{+,L}(x, \Theta, x_c) - h_w^T \underline{y}_i\,,\\
\bar{x}_i^+(x, \Theta, \bar{y}_i) &\triangleq x^{+,L}(x, \Theta, x_c) + h_w^T \bar{y}_i\,,\\
x_i^{+,L}(x, \Theta, x_c) &\triangleq a_i x + b_i \Theta\left(J_k(x - x_c) + k(x_c)\right)\,.
\end{aligned}$$

Proof. We first linearize the control law according to Lemma 6.1. Writing an underbound on the ith component of the successor state, we have

$$\underline{x}_i^+(x, \Theta) = \min_{w \in \mathbb{W}} a_i x + b_i \Theta\left(J_k(x - x_c) + k(x_c) + w\right)$$

$$= \underbrace{a_i x + b_i \Theta \left(J_k(x - x_c) + k(x_c) \right)}_{x_i^{+,L}(x,\Theta,x_c)} + \min_{w \in \mathbb{W}} b_i \Theta w \, .$$

Since $\mathbb{W}$ is a nonempty compact convex set, the minimum exists and is attained. Hence, strong duality of LPs yields

$$\begin{aligned} \min_{w} \; & b_i \Theta w && = \max_{\underline{y}_i} \; - h_w^T \underline{y}_i \\ \text{s.t. } & H_w w \leq h_w && \text{s.t. } - H_w^T \underline{y}_i = \Theta^T b_i^T \, , \\ & && \quad \underline{y}_i \geq 0 \, . \end{aligned} \tag{6.17}$$

Since *some* lower bound is sufficient, we can drop the maximization in (6.17), yielding the left inequality in (6.16a). By a similar construction, the overbound on x_i^+ is obtained by maximizing $x_i^+(x)$ w.r.t. to w, i.e.

$$\bar{x}_i^+(x, \Theta) = x_i^{+,L}(x, \Theta, x_c) + \max_{w \in \mathbb{W}} b_i \Theta w \, , \tag{6.18}$$

where again the maximization can be dualized as follows:

$$\begin{aligned} \max_{w} \; & b_i \Theta w && = \min_{\bar{y}_i} \; h_w^T \bar{y}_i \\ \text{s.t. } & H_w w \leq h_w && \text{s.t. } H_w^T \bar{y}_i = \Theta^T b_i^T \, , \\ & && \quad \bar{y}_i \geq 0 \, . \end{aligned} \tag{6.19}$$

Some overbound is sufficient, so dropping the minimum is valid and yields the right inequality in (6.16a). □

Corollary 6.1 (Set of successor states)**.** *The set of successor states obtained by overbounding the linearization of the suboptimal control law,*

$$X^+(x, \Theta) \triangleq \left\{ x^+ \in \boldsymbol{R}^{n_x} \mid \underline{x}_j^+ \leq x_j^+ \leq \bar{x}_j^+ \right\} \, ,$$

with $\underline{x}_j^+ \equiv \underline{x}_j^+ \left(x, \Theta, \underline{y}_j \right)$ *and* $\bar{x}_j^+ \equiv \bar{x}_j^+ \left(x, \Theta, \bar{y}_j \right)$ *satisfying Lemma 6.2, is a* convex outer bound *of the true successor state, i.e. we always have that*

$$x^+(x, \Theta) \triangleq Ax + B\Theta k(x) \in X^+(x, \Theta) \; \forall x \in \tilde{\mathbb{X}}, \forall \Theta \, .$$

Proof. Follows from the definition of $X^+(x, \Theta)$. □

We define $V_N^*(x^+)$ in (6.4e) as the best cost-to-go under the N-step MPC (4.7), with the initial state being the successor state of x under the *linearized* suboptimal state feedback law in Lemma 6.1:

Definition 6.2 (Cost of successor state)**.** *For $x \in \tilde{\mathbb{X}}$ and $w \in \mathbb{W}$, define*

$$V_N^*(x^+(x,\Theta,w)) \triangleq \min_{\bar{\mathbf{u}}} \sum_{i=0}^{N-1} l(x_i,u_i) + V_f(x_N) \tag{6.20a}$$

$$\text{s.t.} \quad (4.7\text{b}) - (4.7\text{e}),$$

$$x_0^+ = Ax + B\Theta\left(J_k\left(x_c^i\right)\left(x_0 - x_c^i\right) + k\left(x_c^i\right) + w\right) \tag{6.20b}$$

as the optimal cost of the successor state $x^+(x,w)$ of x under the linearized control law according to Lemma 6.1.

Theorem 6.4 (Finite convex stability constraints)**.** *Let $\tilde{\mathbb{X}}$ be given as in Assumption 6.2, $x_c \in \mathcal{R} \subseteq \tilde{\mathbb{X}}$ be fixed and $\mathcal{R}$ be defined as in Theorem 6.2. A sufficient condition for* (6.4e) *is the existence of weights $\Theta \in \boldsymbol{R}^{n_u \times L}$ such that*

$$V_N^*\left(x^+(x,\Theta)\right) - V_N^{*,L}(x,x_c) \leq -\varepsilon l(x,0) \tag{6.21a}$$

$$\forall x \in \mathit{extr}(\mathcal{R}) \ , \ x^+(x,\Theta) \in \mathit{extr}\left(X^+(x,\Theta)\right), \tag{6.21b}$$

where

$$V_N^{*,L}(x,x_c) \triangleq \nabla_x V_N^*(x)|_{x=x_c}(x-x_c) + V_N^*(x_c)$$

is the optimal value function linearized in x_c.

Proof. By linearizing $V_N^*(x)$ around x_c, we obtain a sufficient condition for (6.4e)

$$V_N^*(x^+(x,\Theta,w)) - V_N^{*,L}(x,x_c) + \varepsilon l(x,0) \leq 0 \tag{6.22}$$
$$\forall x \in \mathcal{R}, \ \forall w \in \mathbb{W}.$$

The LHS of (6.22) is convex in x and w, as $x^+(x,\Theta,w)$ is affine in x and w when applying the linearized control law. Maximizing the LHS of (6.22) over all $x \in \tilde{\mathbb{X}}$ and $w \in \mathbb{W}$ yields the worst Lyapunov decrease which should be nonpositive. Note that $X^+(x,\Theta)$ is a hyper rectangle, hence a compact convex polytope. The maximum of a convex function over a convex polytope is attained at one of the vertices of the polytope, hence it suffices to check (6.22) at the vertices of $\mathcal{R}$ and the corresponding vertices of $X^+(x,\Theta)$. □

Tractable Loss Functions

In order to yield a convex problem, convex loss functions $\mathcal{L}(\cdot)$ should be used. A common loss function is the squared 2-norm of the residuals r_i, i.e. $\mathcal{L}(r_i) \triangleq \|r_i\|_2^2$, yielding a constrained least-squares problem. Many other convex loss functions exist [118]. In particular, regularization terms of type $\lambda\|\mathrm{vec}(\Theta)\|_1$, $\lambda \in \mathbb{R}_+$, are often added to facilitate sparse solutions. Here λ allows to trade off sparsity vs. approximation accuracy.

Assumption 6.4 (Convex loss function)**.** *The loss function* $\mathcal{L}(\cdot)$ *in* (6.4a) *is chosen such that it is convex on* $\mathbb{U}$*.*

6.4.4 Main Result

We are now ready to state a tractable formulation of (6.4).

Theorem 6.5 (Tractable learning problem)**.** *Let* $\tilde{\mathbb{X}} \subseteq \mathbb{X}_N$ *be a polytopic set containing the origin in its interior. Furthermore, let* $\chi_i \in \mathbb{X}_N$*,* $i = 1, \ldots, M$ *denote randomly drawn samples. If, for some fixed* $\varepsilon > 0$*, the convex optimization problem*

$$\min_{\Theta, Y, \mathcal{Y}, \underline{y}, \bar{y}} \sum_{i=1}^{M} \mathcal{L}(r_i) \tag{6.23a}$$

$$\text{s.t.} \quad (6.4b), (6.8), (6.12), (6.21), \tag{6.23b}$$

is feasible, returning an optimal set of weights Θ^**, then the closed loop system* (6.3) *under the proposed suboptimal control law in* (6.2) *with weights* Θ^* *is asymptotically stable with region of attraction* $\tilde{\mathbb{X}}$*.*

Proof. This follows directly from the fact that (6.8), (6.12) and (6.21) are sufficient conditions for (6.4c), (6.4d) and (6.4e), respectively. Convexity of the optimization problem stems from the convex loss function according to Assumption 6.4 and the fact that constraints (6.4b), (6.8), (6.12) and (6.21) are convex functions in the optimization variables $\Theta, Y, \mathcal{Y}, \underline{y}_i$ and $\bar{y}_i$. □

Remark 6.8 (A posteriori verification)**.** *Conditions* (6.8), (6.12) *and* (6.21) *can also be used for* verification *of the corresponding system theoretic guarantees for* (6.3) *when using the standard learning approach in* (5.12)*.*

Remark 6.9 (Stability implies invariance)**.** *Feasibility of* (6.22) *for* $x_0 = x^+(x, \Theta)$ *for fixed* Θ *and* $\forall x \in \tilde{\mathbb{X}}$ *implies that* $\tilde{\mathbb{X}}$ *is an invariant set for* (6.3)*. Hence,* (6.12) *may also be dropped from the tractable learning problem.*

Remark 6.10. *The tractable learning problem* (6.23) *is a quadratically constrained quadratic program (QCQP). In case the MPC problem is formulated with a linear (1 or* ∞*) norm as the stage and terminal cost, the resulting learning problem is a QP.*

We have derived a tractable convex inner approximation of the semi-infinite learning problem (6.4). The overall learning algorithm is summarized in Algorithm 6.1. If (6.23) is feasible, the resulting controller is equipped with the desired system theoretic properties. In

Algorithm 6.1 Tractable Learning of MPC Control Laws

Require: Differentiable basis functions (Assumption 6.1) $k(\cdot)$.
Ensure: Stabilizing control law $\tilde{u}(x)$ with constraint satisfaction and region of attraction $\tilde{\mathbb{X}}$ for closed-loop system (6.3) (if $k(\cdot)$ is "expressive" enough).
1: Define $\tilde{\mathbb{X}} \subseteq \mathbb{X}_N$ such that it can be an invariant set e.g. by scaling $\mathbb{X}_N$.
2: Sample $\mathbb{X}_N$ to obtain feasible initial states χ_i and solve MPC problem (4.9) to obtain $u_0^\star(\chi_i)$.
3: Compute a tessellation of $\tilde{\mathbb{X}}$ (regions $\mathcal{R}_i$) and the representations of the regions by linear inequalities, e.g. by using a Voronoi tessellation or a tessellation into hypercubes.
4: Compute linearization error sets $\mathbb{W}_i$ and generate constraints (6.12) and (6.21) for each region $\mathcal{R}_i$.
5: Solve convex program (6.23) with the constraints generated in 4.
6: **if** (6.23) is infeasible **then**
7: Resample (go back to 2).
8: **end if**

case of infeasibility of (6.23), either the chosen complexity in terms of type and number of basis functions is insufficient to provide an invariant and stabilizing control law, or (6.23) is a too conservative approximation of (6.4). To exclude the latter, the regions $\mathcal{R}^i$ can be refined according to the size of the linearization error sets $\mathbb{W}^i$. Once a feasible solution to (6.23) has been obtained, more samples can be added to increase approximation accuracy.

6.5 Numerical Example

In this section, we present the results for learning the optimal finite horizon controller of a double integrator system. We employ the LQR state feedback law as terminal controller, i.e. $\kappa_f(x) = K_{\text{LQR}}x$. The terminal set $\mathbb{X}_f$ was chosen as the maximum invariant set under LQR control along with the terminal weight $V_f(x) = x^T P x$. The problem data is given by

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix},$$
$$Q = Q_{\text{LQR}} = I_2, \quad R = R_{\text{LQR}} = 1, \quad N = 5,$$
$$P = \begin{bmatrix} 1.8085 & 0.231 \\ 0.231 & 2.6489 \end{bmatrix}, \quad \|x\|_\infty \leq 5, \quad |u| \leq 1.$$

We use the following basis function for the learning:

$$k_i(x) = \left(1 - \frac{\|x - \chi_i\|_2}{\sigma}\right)^3, \tag{6.24}$$

with $\sigma = 20$. This function has been investigated on a 1-dimensional domain e.g. in [30, 48]. We draw $M = 200$ samples by sampling from a uniform distribution, yielding

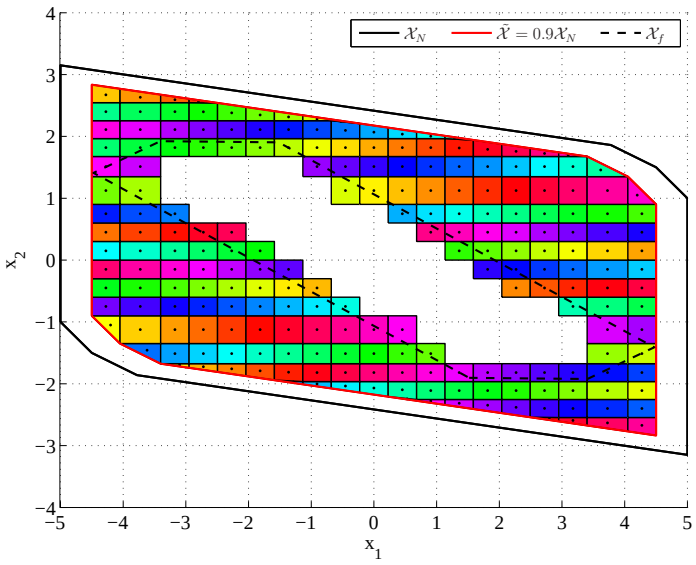


Figure 6.1: Tessellation of region of interest, $\tilde{\mathbb{X}} = 0.9\mathbb{X}_N$, into 274 regions, 60 of which are strictly inside the terminal set $\mathbb{X}_f$ and are not shown here. The black dots depict the linearization points x_c.

$L = M = 200$ basis functions centered at the sample points. We have chosen $\tilde{\mathbb{X}} \triangleq 0.9\mathbb{X}_N$ and its tessellation into 274 rectangular regions, out of which 60 lie strictly inside $\mathbb{X}_f$ (Figure 6.1). In the learning problem (6.23), we use a quadratic loss function without regularization, $\mathcal{L}(r) = \|r\|_2^2$, and impose input constraints (6.8) and invariance constraints (6.12). The problem has been formulated with YALMIP [82] and the resulting QP is solved by CPLEX in 43s on an Intel Xeon 2.53 GHz machine with 8 cores. Stability has been verified a-posteriori using (6.21).

Figure 6.2 shows the feasible set $\mathbb{X}_N$ of the nominal MPC controller with the random sample points used for learning. We have added the vertices of $\mathbb{X}_N$ to the sample set. The samples not encircled were assigned an almost zero weight (less than 10^{-6} in magnitude), which shows that the constrained learning is sparse even without regularization. In this example, only 42 out of the 200 samples suffice to evaluate the nonlinear control law. This result suggests that oversampling does not necessarily increase the complexity of the resulting control law.

The learned controller is a good approximation of the optimal PWA control law, as can be seen in Figure 6.3. Due to the use of nonlinear basis functions, the global behavior of the PWA law is captured well, although local approximation errors can be high. This is particularly the case near the hinges of the PWA law. The largest error on a grid of 7480 points is 0.339 at $x = [2.5 \ -0.315]^T$ with $u_0^*(x) = -1$ and $\tilde{u}(x) = -0.661$.

To investigate the closed loop performance under the approximate control law, trajectories

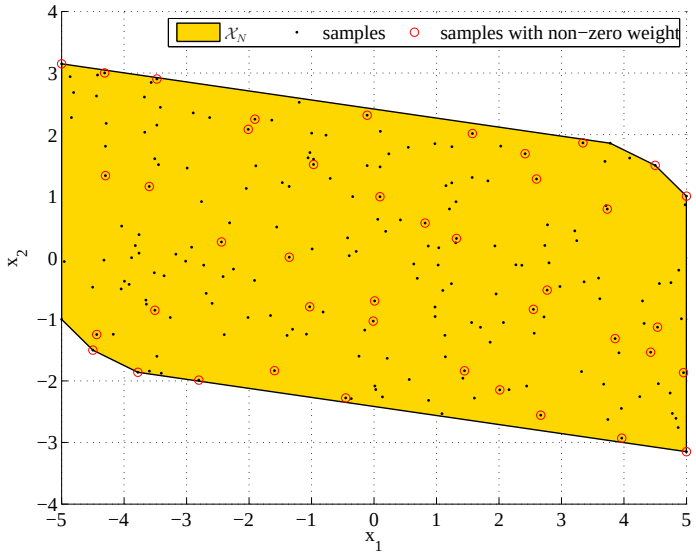


Figure 6.2: Samples used for learning. Only the encircled 42 samples out of 200 have been assigned weights with a magnitude bigger than 10^{-6}, i.e. the resulting weight vector is sparse, which significantly speeds up online evaluation of the control law.

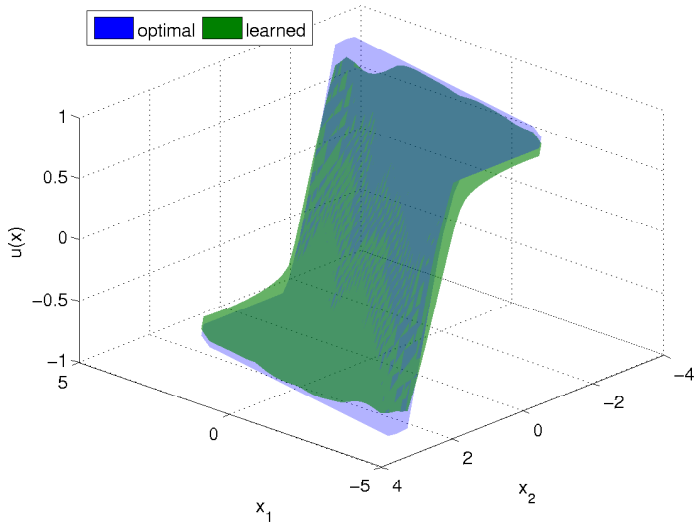


Figure 6.3: Optimal (in blue) vs. learned (in green) controller.

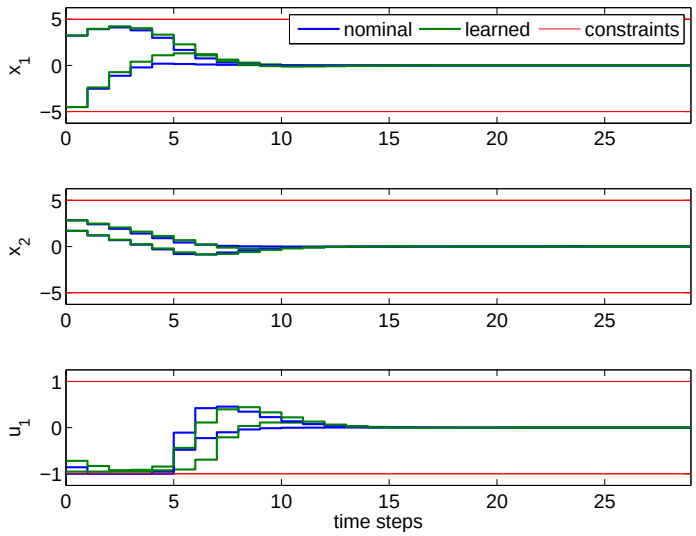


Figure 6.4: State and input trajectories for two different initial states.

have been simulated from 7480 initial conditions (the same points as used above to evaluate the approximation error). Figure 6.4 depicts two selected trajectories from different initial conditions. The closed loop system is stable, which has been verified a-posteriori with (6.21). The same result would have been obtained by including the constraints into the learning problem. No state constraint violations occurred for the trajectories. The number of steps needed to reach the terminal region increases for about 10% of the trajectories by one step as compared to the nominal controller.

7 Learning Decision Rules for Energy Efficient Building Control

7.1 Introduction

About 20-40% of total primary energy usage is spent for the heating, ventilation and cooling (HVAC) of commercial buildings and private housing [109]. While currently most buildings are equipped, if at all, with simple automatic control systems, many studies have indicated the merits of advanced building automation over current state of the art control practice in terms of energy usage and comfort regulation, see e.g. [61,62,83,84,110] and the references therein. Particularly the field of model predictive control (MPC) offers a systematic framework for optimal operation of buildings. Incorporating a model of the building, these controllers can combine measurements available through on-site sensors with weather forecast data to control the building such that energy consumption is minimized while respecting specifications (constraints) on occupancy comfort [76, 103]. These advanced controllers generally outperform current state of the art solutions and thus have a huge potential in reducing green house gas emissions when widely applied.

However, the field only slowly adapts to these advanced optimization based control strategies mainly due to two practical reasons. First, the resulting optimization problem, which needs to be re-solved after new measurements are available, requires substantial IT infrastructure both in terms of hardware and software. Second, commission engineers are not trained to set up complex control systems based on numerical optimization, tune them and respond adequately to eventual malfunctions or error codes. Contrary to the petrochemical industry, where optimization based control of large scale dynamic systems emerged in the early 1970s, a building is usually operated without on-site engineers carefully monitoring and supervising the correct functioning of the building control system. It therefore remains a major challenge to derive control schemes which allow both an *energy efficient* operation of the building and, at the same time, a *simple implementation* of the control algorithm.

Main Contributions

We suggest a framework that allows automatic extraction of decision rules from simulation data generated with advanced control schemes such as hybrid MPC (HMPC). In contrast to Chapter 6, which deals with learning of continuous control inputs, we focus on learning *binary decisions* in this chapter. The result is a significant complexity reduction, while our simulations suggest that much of the HMPC control performance can be maintained by the extracted set of rules. These can be automatically transformed into computer code for various embedded control platforms. We show that for a certain choice of synthesis methodology, the resulting rules are simple enough to be readable by humans. This enables building operators to adjust the controller on-site if the behavior is unsatisfactory due to model-plant mismatch, for example.

The ranking of measurements in terms of importance for approximation performance is also discussed. This *feature selection* allows pruning sensors or data streams which contain information irrelevant to a decision and further reduces the complexity of the resulting controller. When used with accurate building models, feature selection potentially saves investment costs in unnecessary sensor infrastructure. In our simulations, near-optimal control is achieved with only a small number of sensors and weather predictions.

Once the discrete decisions have been set by the learned controllers, the resulting MPC optimization problem is continuous. Computation of local solutions to these problems on embedded control platforms has been subject of intensive research, and software tools are available for this task. Widely used methods are multiple shooting [66] and solving a sequence of linear MPC problems arising from a linearization of the dynamics using efficient convex programming solvers such as in [137] or the solvers presented in presented Chapter 8 and Chapter 9, depending on the particular problem.

Related Work

Controller approximation from simulation data has been considered by means of linear interpolation [27] or nonlinear sparse approximation [123] of MPC control laws for continuous inputs. Rule extraction for binary inputs in a building control context has been investigated in [89] with a logistic regression model. The methods used in this chapter are based on standard machine learning algorithms (cf. Chapter 5). In particular, we consider *support vector machines* (SVMs) for classification [23] and *AdaBoost* [46], two well researched and widely applied algorithms. Examples of successful applications of these tools are gene expression profiling [3] for cancer detection, robust real-time face detection [133] and handwritten digit and character recognition [29].

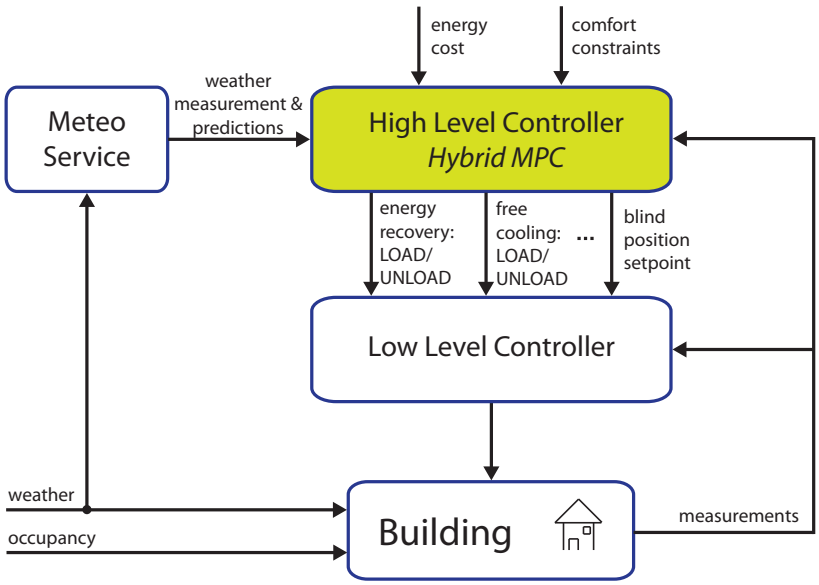


Figure 7.1: Hierarchical control system employing model predictive control at the highest level to enhance energy efficiency. The setpoints calculated by the HLC are issued to standard low-level controllers which ensure good tracking despite disturbances.

Outline

The chapter is organized as follows. In Section 7.2, we outline the considered control scheme along with a possible hybrid MPC formulation. In Section 7.3, we briefly describe SVM and AdaBoost in the setting of approximating controllers and discuss their features, in particular feature selection. The main contribution of the chapter is Section 7.4, where we present simulation studies with approximate controllers synthesized by learning algorithms. Section 7.5 concludes the chapter.

7.2 Hybrid Model Predictive Building Control

7.2.1 Hierarchical Control Structure

Consider the hierarchical control scheme depicted in Figure 7.1. The high level controller (HLC) computes an optimal plan for a certain prediction horizon subject to parameters such as energy price and comfort constraints, as well as measurements of the current state of the building and weather forecast data. In order to predict the building's behavior, the HLC employs a mathematical model of the building. The resulting setpoint commands are then issued to a low level controller (LLC), which ensures tracking of the given setpoints despite

disturbances such as varying occupancy of the building and imperfect weather forecast. The LLC is in practice usually a proportional-integral-derivative (PID) controller or a rule based controller (RBC) [60]. The LLC is however not the focus of this chapter, and good tracking of the setpoints provided by the HLC is assumed in the following. At the next sampling instance of the HLC, the optimization is repeated based on the current state of the system, and new setpoint trajectories are computed. This scheme is repeated in each sampling time, which coined the term *receding horizon control* (RHC) widely used for MPC in literature.

There are two fundamentally different types of control inputs computed by the HLC: discrete ones, which are referred to as *binary decisions* in the rest of the chapter, and continuous ones. Examples for binary decisions used throughout the chapter are *energy recovery* and *free cooling* systems. Energy recovery controls mechanical ventilation and heat exchangers that extract energy from exhaust air, while free cooling controls the amount of available chilled water generated with a wet cooling tower. Both systems have two modes of operation, `LOAD` and `UNLOAD`, for increasing and decreasing the thermal energy stored in the building. The HLC decides upon these modes and issues the commands to the LLC. An example for continuous inputs is the blind position.

7.2.2 Hybrid MPC Formulation

In this section, we describe the main parts of our hybrid model predictive control formulation for the high-level building controller.

7.2.2.1 Dynamic building model

A common approach to obtain a mathematical description of the building dynamics is to model the building as one zone using a resistance-capacitance network model [61, 76, 84]. The resulting discrete-time model is *bilinear* (cf. [61, §5]):

$$\begin{aligned} x^+ &= Ax + B_u u + B_v v + \sum_{j=1}^{n_u} \left(B_{vu_j} v + B_{xu_j} x \right) u_j \\ y &= Cx + D_u u + D_v v + \sum_{j=1}^{n_u} D_{vu_j} v \, u_j \end{aligned} \tag{7.1}$$

where $x \in \boldsymbol{R}^{n_x}$ is the current state of the system, $x^+ \in \boldsymbol{R}^{n_x}$ the successor state at the next sampling time, $u = [u_c^T \; d^T]^T \in \boldsymbol{R}^{n_{u,c}} \times \{0,1\}^{n_d}$ with $n_u = n_{u,c} + n_d$ denotes the continuous input variables u_c and binary input variables d, respectively, and $v \in \boldsymbol{R}^{n_v}$ is the disturbance (weather and occupancy) acting on the system. The matrices A, B_u, B_v, B_{vu_j}, B_{xu_j}, C, D_u, D_v, D_{vu_j} of corresponding size are obtained in the modeling process.

We denote the input trajectories as $u_c \triangleq [u_{c,0}, \ldots, u_{c,N-1}]$ and $d \triangleq [d_0, \ldots, d_{N-1}]$, the disturbance trajectory as $v \triangleq [v_0, \ldots, v_{N-1}]$, the state trajectory as $x \triangleq [x_0, \ldots, x_N]$ and the output trajectory as $y \triangleq [y_0, \ldots, y_{N-1}]$. Linearization of (7.1) around the current operating point and an *assumed* disturbance trajectory $\hat{v}$, e.g. a weather forecast profile and occupancy predictions, yields the affine prediction model

$$\begin{aligned} x_{k+1} &= \tilde{A}_k x_k + \tilde{B}_{u,k} u_k + \tilde{B}_{a,k} \\ y_k &= \tilde{C}_k x_k + \tilde{D}_{u,k} u_k + \tilde{D}_{a,k} \end{aligned} \tag{7.2}$$

for $k = 0, \ldots, N-1$, where N is the prediction horizon, and $x_0 \equiv x$ the current state.

7.2.2.2 Constraints

MPC inherently supports system constraints. We employ input and output constraints of the form

$$H_u u_{c,k} \leq h_{u,k}, \qquad k = 0, \ldots, N-1\ , \tag{7.3a}$$
$$d_k \in \{0,1\}^{n_d}\ , \qquad k = 0, \ldots, N-1\ , \tag{7.3b}$$
$$H_y y_k \leq h_{y,k} + s_k, \quad s_k \geq 0, \qquad k = 0, \ldots, N-1\ , \tag{7.3c}$$

where (7.3a) models actuator limitations such as minimum and maximum blind positions, and (7.3c) models occupant comfort specifications, for example minimal room temperature or maximum admissible CO_2 level. In order to ensure feasibility of the problem at all times, (7.3c) are *soft constraints* with slack variables $s_k \geq 0$ that measure the comfort violations.

7.2.2.3 Cost function

The cost function reflects a trade-off among various control objectives. In building control, the cost function is usually an economic performance index such as the cost of non-renewable primary energy (NRPE) usage. Ecological costs – green house gas emissions for instance – could be added as well. Let us consider the following quadratic performance index to be minimized:

$$J(x, v, u_c, d, s) \triangleq \sum_{k=0}^{N-1} c_k(x, v, d)^T u_{c,k} + s_k^T S s_k\ , \tag{7.4}$$

with c_k being the cost weights for the continuous inputs depending on e.g. the discrete inputs. The quadratic penalty term on s with $S \succ 0$ allows one to trade off comfort constraint violations vs. NRPE usage.

7.2.2.4 Hybrid MPC – Summary

We now summarize the finite time constrained optimal control problem, which the HLC has to solve at every time instant. The input trajectories u_c^* and d^* are determined by minimizing (7.4) subject to the assumed disturbance trajectory $\hat{v}$, the dynamics, and the constraints:

$$\begin{aligned} J^\star(x, \hat{v}) \triangleq \min_{u_c, d, x, y, s} \; & J(x, \hat{v}, u_c, d, s) \\ \text{s.t.} \quad & (7.2),\ (7.3),\ x_0 = x \end{aligned} \tag{7.5}$$

Due to the integrality constraint (7.3b), problem (7.5) is in the class of *hybrid MPC* problems.

Remark 7.1. *Problem* (7.5) *neglects stochasticity in the system and is by no means the only way to formulate a sensible optimization problem for the control task at hand, cf. [55, 102] for stochastic formulations for instance. However, the proposed approach can be applied to any other MPC formulation involving binary decision variables.*

In the following, we describe two methods which synthesize simple decision rules representing an approximate controller for binary decisions in (7.5).

7.3 Rule Extraction from Simulation Data

To be able to extract a controller from simulation data, we first first define the quantities needed in the general learning formulation for classification from Section 5.1. In particular, we assume that M data points

$$\chi_i \triangleq (x_{0,i}, y_{0,i}, \hat{v}_i) \in \mathcal{P} \subseteq \boldsymbol{R}^n \,, \tag{7.6}$$

have been obtained from simulation, as well as the corresponding optimal binary solutions $d^\star_{0,i}$ to the MPC problem (7.5). We consider the approximation of the first control move only, since the other computed inputs are actually never applied to the system in a receding horizon framework. Instead of representing these decisions as zeros and ones, we map a zero to minus one, i.e.

$$z_i = 2d^\star_{0,i} - 1 \in \{-1, 1\}^{n_d} \,, \tag{7.7}$$

to match common learning algorithm formulations (cf. Section 5.1). The problem of learning a controller is then as follows: We would like to infer n_d functions (one for each binary decision to be taken) $g : \mathcal{P} \to \{-1, 1\}$ from the data with minimal error for any *unseen* sample χ.

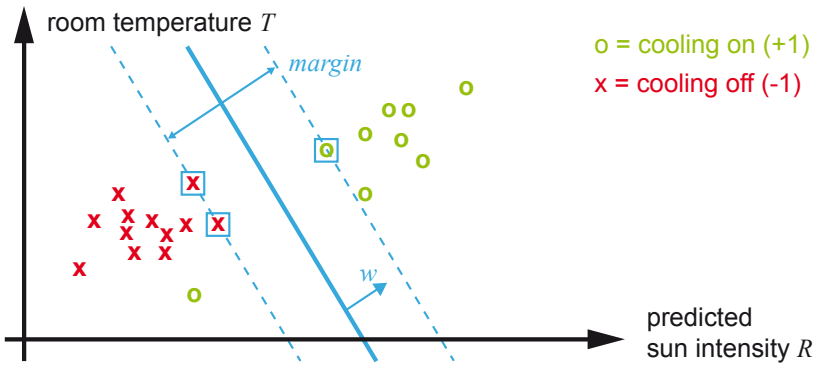


Figure 7.2: Maximum margin classification example. The classifier decides on two *features*, the current room temperature T and the predicted sun radiation intensity R, whether to switch on or off the cooling system. The given data points represent optimal decisions by a hybrid MPC formulation, and the goal is to learn simple functions which approximate the optimal controller well.

Throughout this section, we will make use of an illustrative example of "learning" the decisions of an optimal controller for a cooling system that is switched on or off according to measurements of room temperature T and predictions of sun radiation intensity R, see Figure 7.2. For illustration, let the data points (crosses and circles) represent the solution of the hybrid MPC problem (7.5) for 20 selected (sampled) combinations of T and R.

Consider now for this simple classification task a *discriminant* function, the blue line in Figure 7.2, which partitions the feature space formed by T and R into two regions: a new measurement on the right (left) hand side of the separating line (generally a hyperplane in $\boldsymbol{R}^n$) would cause the controller to switch on (off) the cooling system. The task of *tractable* learning is to find the (finite dimensional) parameters of the discriminant function of a chosen type (e.g. a linear function) that separates the two classes. We consider for this tasks the two algorithms described in Section 5.1 and give more details in the following.

7.3.1 Support Vector Classification

The maximum margin SVM classifier, when using a *linear* discriminant function, is depicted in Figure 7.2. The resulting support vectors (cf. Section 5.1) are marked with a rectangle. Note however that the discriminant function is not linear in general when nonlinear kernels are used for the SVM.

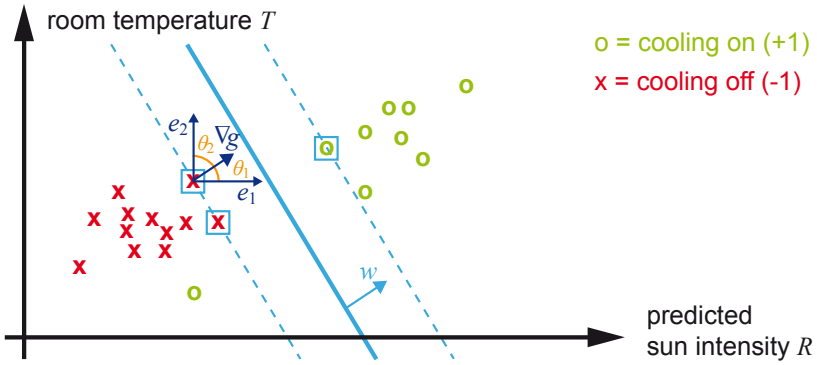


Figure 7.3: Feature selection for support vector machines according to [64]. From $|\frac{\pi}{2} - \theta_1| > |\frac{\pi}{2} - \theta_2|$ follows that $\sigma_1 > \sigma_2$, i.e. the hyperplane is more sensitive to feature 1.

7.3.1.1 Resulting Controller Structure

After solving problem (5.4), each binary input can be obtained by the evaluation of a linear combination of nonlinear basis functions (5.6). This allows for example an FPGA implementation, where each kernel function (or groups of kernel functions) are evaluated in parallel, followed by an adder tree and a sign function to obtain the final decision. However, the resulting controller is in general virtually impossible to tune such a controller once it has been synthesized.

7.3.1.2 Feature Selection with SVMs

It is desirable for any controller approximation method to provide information on the importance of measurements with respect to the final binary decision. For example, the weather prediction 24 hours ahead may be less important than the one 6 hours ahead, and hence the decision function for heating on/off will be less sensitive to long-term weather predictions. Having such information available, the control designer can choose to prune less important measurements or data streams from the system. In the machine learning terminology, this process of selecting certain input variables in favor of others according to their relative importance for the final decision is called *feature selection*.

We follow [64] and give an illustration in Figure 7.3. Using the discriminant function of SVMs, (5.6), sensitivities of features can be computed by perturbations of support vectors along the unit vectors

$$e_k \triangleq [0, \ldots, 0, \underbrace{1}_{k^{\text{th}} \text{ entry}}, 0, \ldots, 0]^T, \quad k = 1, \ldots, n. \tag{7.8}$$

Feature k is unimportant for sample χ_i if an infinitesimal perturbation of χ_i along the unit

vector e_k does not change the separating hyperplane. This is equivalent to $\theta_{i,k} \approx \frac{\pi}{2}$, where

$$\theta_{i,k} \triangleq \angle\left(\nabla g(\chi_i), e_k\right) = \arccos\left(\frac{\nabla g(\chi_i)^T e_k}{\|\nabla g(\chi_i)\|}\right) \in [0, \pi] \tag{7.9}$$

is the angle between the discriminant function's gradient $\nabla g(\chi_i)$ and the unit vector e_k. Thus, the score for feature k at sample χ_i is defined as

$$\sigma_{i,k} \triangleq \frac{2}{\pi}\left|\frac{\pi}{2} - \theta_{i,k}\right| \in [0, 1]\,. \tag{7.10}$$

Averaging over all samples that are ϵ-close to the margin then yields the score for feature k:

$$\sigma_k \triangleq \frac{1}{|I_\epsilon|}\sum_{i \in I_\epsilon} \sigma_{i,k}\,, \quad I_\epsilon \triangleq \{i : |g(\chi_i) - 1| \leq \epsilon\}\,. \tag{7.11}$$

Sorting according to scores yields the ranking of features.

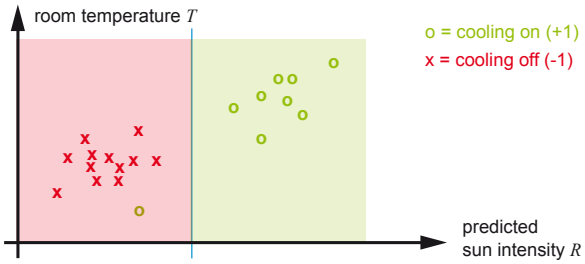
7.3.2 AdaBoost

7.3.2.1 Illustration of AdaBoost Training

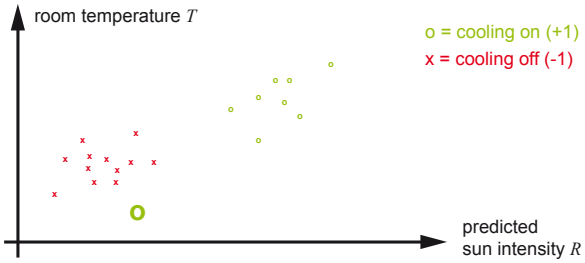
The principle of training a controller for binary decisions with AdaBoost as described in (5.1.2) is illustrated in Figure 7.4 for our toy example from above. The initial weak learner is depicted in Figure 7.4(a), where only the sample in the lower left corner has been wrongly classified. The error rate is $\epsilon_1 = 0.05$, and thus the decision of $h_1(\chi)$ would be assigned a coefficient of 1.47 according to (5.9). By the weighting rule (5.10), $w_{i,2} = 0.23$ for correctly classified points for which $z_j h_1(\chi_j) = 1$, and $w_{i,2} = 4.36$ for the misclassified sample in the lower left corner of Figure 7.4(b). Figure 7.4(c) shows the resulting second weak learner, which has been trained on the reweighed data and which now correctly classifies the previously misclassified sample in Figure 7.4(b). The resulting decision (5.8) is the linear combination of the weak learner's individual decisions, depicted at the bottom of Figure 7.4, which suggests that indeed complex discriminant functions g can be obtained by the combination of many simple decision functions.

7.3.2.2 Choice of weak learners

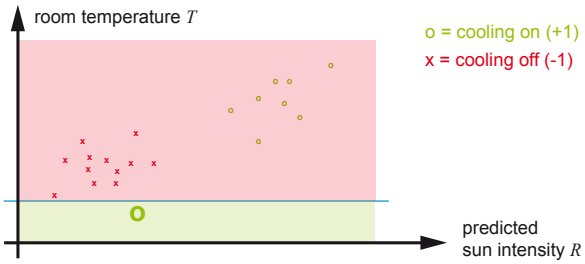
Once the parameters of the weak learners h_j and the corresponding coefficients a_j have been found, the evaluation of the control law boils down to the evaluation of the weak learners (just as in SVMs it boils down to the evaluation of the kernels). It is therefore desirable to have functions h_j which can be evaluated quickly. The simplest weak learner that works very well with AdaBoost is a so-called *decision stump*.



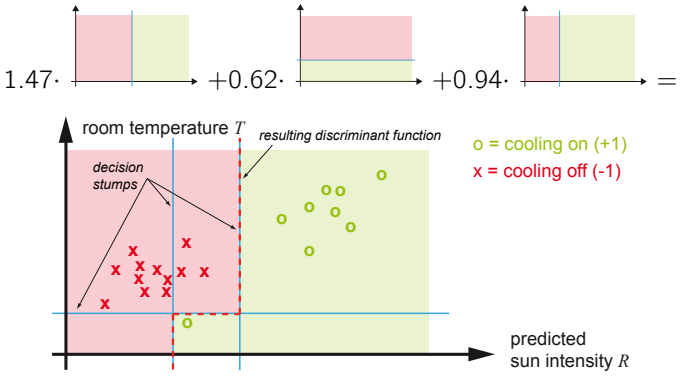
(a) Initial weak learner $h_1(\chi)$, which has an error rate of $\epsilon_1 = 0.05$. The weight of this decision is therefore according to (5.9) $a_1 = 1.47$.



(b) Reweighting of data points using (5.10) emphasizes points which have been wrongly classified by the first weak learner.



(c) The second weak learner $h_2(\chi)$ is trained on the reweighted data, and the previously misclassified point is now correctly assigned. $a_2 = 0.62$.



(d) The final decision is a weighted sum of the individual decisions.

Figure 7.4: The principle of AdaBoost: each weak learner performs poorly, but the overall combined decision yields a correct classification.

Definition 7.1 (Decision stumps)**.** *A decision stump is defined by*

$$h(\chi) = \mathit{sign}\left(e_k^T \chi - d\right) , \tag{7.12}$$

where e_k is the kth unit vector. Its parameters in terms of learning are the feature index k *and the offset d.*

It turns out that, for our purposes, decision stumps are a very good fit because of the inherent similarity of the resulting controller to current building control practice and because decisions stumps allow a straightforward feature selection. We detail on both aspects in the following.

7.3.2.3 Resulting controller structure

It is noteworthy that each decision stump decides on one feature only, which can be viewed as a simple if-then-else rule:

if $\chi[k] > d$ **then**
  $h(\chi) = +1$
else
  $h(\chi) = -1$
end if

One rule could for example read "if the room temperature is greater than $24.32^\circ\,\mathrm{C}$ then switch on the cooling system". Recall that each weak learner j represents such a rule with corresponding feature index k_j and offset d_j, so combining them with the associated coefficients a_j of (5.8) results in a very simple majority voting system for the overall decision:

```
vote = 0;
for j = 1 to L do
  if χ[k_j] > d_j then
    vote += a_j;
  else
    vote -= a_j;
  end if
end for
```

The overall decision results from the sign of the accumulated votes:

```
if vote > 0 then
  return SWITCH_ON_COOLING
else
```

```
    return SWITCH_OFF_COOLING
  end if
```

This majority voting structure is very similar to state of the art *rule based controllers* (RBCs) [60], where a set of rules is combined to yield a final decision. These rules are usually defined and tuned by experienced building control engineers, often in a long trial and error procedure. In contrast, the proposed rule based controller (5.8) can be synthesized automatically as a chain of if-then-else rules. We give an example in Section 7.4.

One advantage of using AdaBoost with decision stumps as weak learners is is that the resulting decision rules are readable and adjustable by humans. This allows intuitive insight and manual tuning, should the controller behave unsatisfactorily on-site. As illustrated in Section 7.3.1.1, SVMs generally synthesize decision functions (5.6) that are a weighted sum of vector products of nonlinear maps, making intuitive insight generally impossible.

7.3.2.4 Feature selection with AdaBoost

Feature selection using sensitivity analysis as in SVMs is not applicable to AdaBoost if weak learners are employed that are not differentiable with respect to their parameters. This is for example the case for decision stumps. However, decision stumps perform an inherent feature selection by the choice of the feature index k_j. The modulus of the coefficient assigned to the jth weak learner, a_j, is therefore a direct indication of how important that very decision on feature k_j is, and therefore weak learners with "small" coefficients (relative to the others) can be discarded from the overall decision, effectively performing a feature selection using the absolute value of the coefficients obtained by AdaBoost training.

7.4 Simulation Results

In this section, we present the main results of this chapter based on a simulation study. Section 7.4.1 explains the setup, whereas the later sections present the results.

7.4.1 Setup

The case study is carried out using *BACLab*, a MATLAB-based modeling and simulation environment for building climate control developed within the OptiControl project [28]. We consider six test cases with different building types and integrated room automation (IRA) equipment at two different locations in Europe, see Table 7.1. The test scenario is one year of building operation with real weather data from the year 2007 (measurements and COSMO-7 predictions) obtained from MeteoSwiss, the Swiss Federal Office of Meteorology

Table 7.1: Case studies used in numerical experiments, from [61].

	Case study no.					
	1	2	3	4	5	6
Automated Subsystems						
Blinds	√	√	√	√	√	√
Electric Lighting	√	√	√	√	√	√
Mech. ventilation: heating/cooling	√	√	√	√	-	-
Mech. ventilation: energy recovery	√	√	√	√	-	-
Natural ventilation at night-time	-	-	-	√	-	-
Cooled ceiling (capillary tubes)	√	√	-	-	√	√
Wet cooling tower (free cooling)	√	√	-	-	√	√
Radiator heating	√	√	-	-	√	√
Floor heating	-	-	-	√	-	-
Location						
Zurich, Switzerland	√	-	√	√	√	√
Marseille, France	-	√	-	-	-	-
Building Type						
Passive / high window area fraction	√	-	-	√	√	-
Swiss average / low window area frac.	-	√	√	-	-	√

and Climatology.

7.4.1.1 Data Generation with Hybrid MPC

The training data for the learning machines is generated by simulating each test case under the control loop as depicted in Figure 7.1 with a step size of one hour and a prediction horizon of 24 hours.

The HMPC high-level controller is formulated as a MLD system (4.13) (cf. Section 4.2) using HYSDEL [126] and YALMIP [82], taking into account the linearized dynamics (7.2) of the building and the logic of the low level controller. This results in the following MPC

problem formulated as a mixed-integer quadratic program (MIQP):

$$\begin{aligned}
\min_{u,\bar{w},x,\bar{y},s} \quad & \sum_{t=0}^{N-1} c_y^T \bar{y}_k + s_k^T S s_k \\
\text{s.t.} \quad & x_0 = x, \\
& x_{k+1} = \bar{A}_k x_k + \bar{B}_{u,k} u_k + \bar{B}_{w,k} \bar{w}_k + \bar{B}_{a,k}, \\
& \bar{y}_k = \bar{C}_k x_k + \bar{D}_{u,k} u_k + \bar{D}_{w,k} \bar{w}_k + \bar{D}_{a,k}, \\
& \bar{E}_{x,k} x_k + \bar{E}_{u,k} u_k + \bar{E}_{w,k} \bar{w}_k \leq \bar{E}_{a,k}, \\
& \bar{H}_y \bar{y}_k \leq h_{y,k} + s_k, \; s_k \geq 0, \quad k = 0 \dots N-1,
\end{aligned} \tag{7.13}$$

where $\bar{w}_k$ are auxiliary continuous and binary variables. Since the building model is bilinear, the optimal input is obtained iteratively by solving a sequence of problems (7.13). In each iteration, the linearization of the dynamics (7.2) is updated around the current iterate, (7.13) is re-built by HYSDEL and solved by CPLEX [67] (version 10). The LLC is implemented as a one step MPC controller that is adjusted according to the modes set by the HLC (cf. [61, §3.3]).

During simulation, we recorded the binary inputs generated by the HLC together with 126 variables that will serve as features for the learning, e.g. states, outputs, weather measurements, weather predictions for the next 24 hours, occupancy predictions, time in year.

7.4.1.2 Controller Approximation by Learning

We were interested in learning the decisions on *energy recovery* and *free cooling*. For this, the learning algorithms described in Section 5.1 were applied to the recorded simulation data using freely available software packages. For SVM learning, we used LIBSVM [25] with the parameters $C = 1$, $f(\xi) = \|\xi\|_1$, and radial basis function (RBF) kernels of type

$$k(x, y) = \exp\left(-\gamma \|x - y\|_2\right) \tag{7.14}$$

with $\gamma = 1$. As for AdaBoost, we applied the GML AdaBoost Matlab Toolbox [131] with decision stumps as weak learners and using $L = 200$ boosting steps (iterations of the algorithm). Section 7.4.2 discusses the obtained approximation accuracy, and Section 7.4.3 details on feature selection.

7.4.1.3 Closed-loop Simulation

In order to assess the impact of suboptimality of the approximate controllers, we simulated the test cases 1-6 under the control loop as depicted in Figure 7.5, where the learned

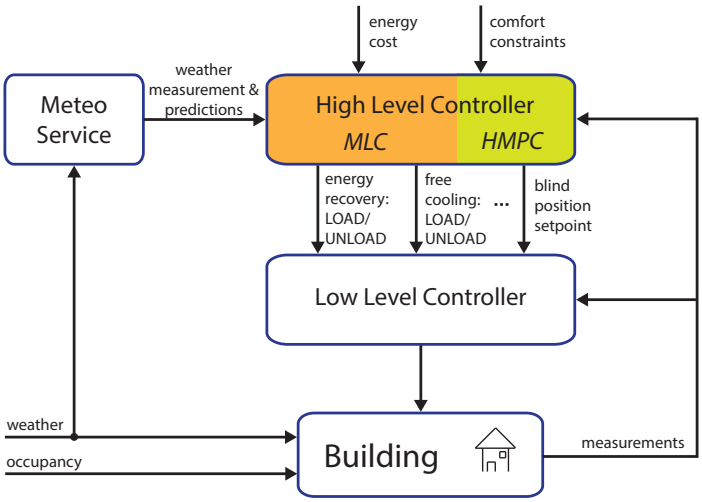


Figure 7.5: Simulation strategy used to evaluate the performance of the synthesized rule based controllers. The suboptimal controller is learned using simulation data, replacing the binary decisions on energy recovery and free cooling of the hybrid MPC controller.

controllers (MLC) override the decisions of the HMPC for the two modes *energy recovery* and *free cooling*. All other decisions are still obtained from the HMPC controller. The resulting closed-loop performance is presented in Section 7.4.4.

7.4.2 Approximation Accuracy

In order to assess the approximation accuracy of the learned controllers, we perform a 10-fold cross validation procedure, which is a widely used technique for estimating the generalization performance of learning machines. The method splits the training data into 10 sets of equal cardinality. Then, the controllers are trained using data from 9 out of the 10 sets, while the approximation error is validated on the remaining data set which has not been used for training. This process is repeated with the 10 possible combinations of training and validation data sets, and the empirical training error is averaged.

As can be seen from Figure 7.6, the learned controllers have a rather high error rate, up to 20% for the energy recovery mode in test case 2. Furthermore, the error rate of the learned controllers is in half of the cases not significantly better than for the trivial controller "always `UNLOAD`" (for free cooling in cases 1, 5, 6, and energy recovery in case 3). SVM and AdaBoost perform very similarly in all cases, with the SVM having a slightly lower error rate in the majority of the cases.

The poor learning performance in terms of high error rates stems partly from the fact that we compare the hybrid MPC's decisions with the learned ones directly. However, this

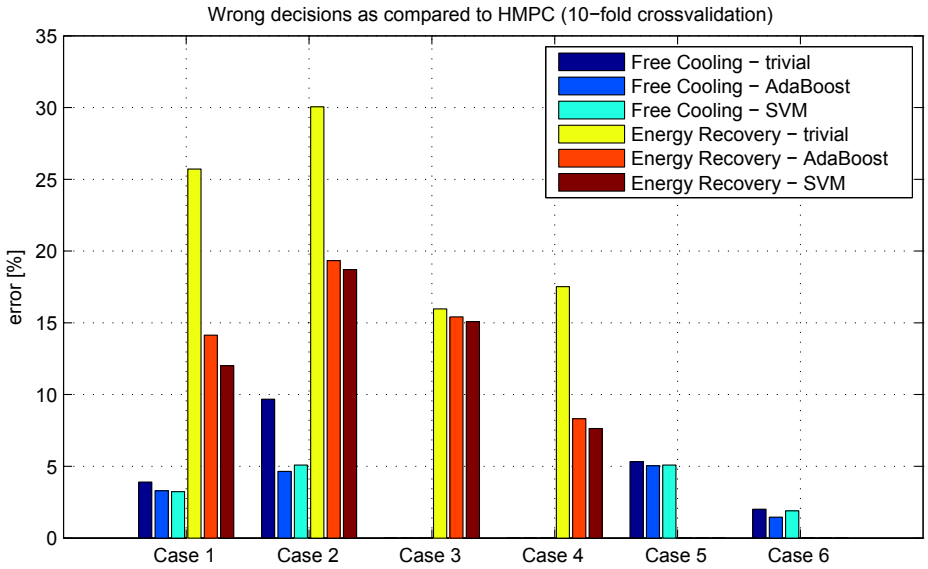


Figure 7.6: Averaged error rate of the controllers in % of the total number of tested decisions using 10-fold cross validation for the six different test cases and the two binary inputs *energy recovery* and *free cooling*. If any of the modes is missing, the associated equipment is not present on the particular building. The error rate of the "trivial" controller (setting the binary decision to *always* `UNLOAD`) is depicted for comparison.

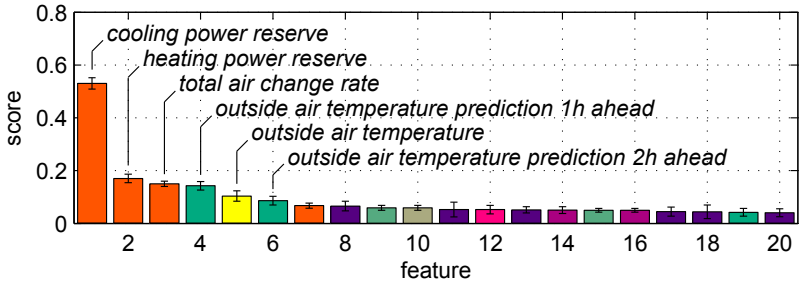
approximation accuracy assessment does not reveal the *closed-loop* performance, which will be the final performance measure.

7.4.3 Feature Selection

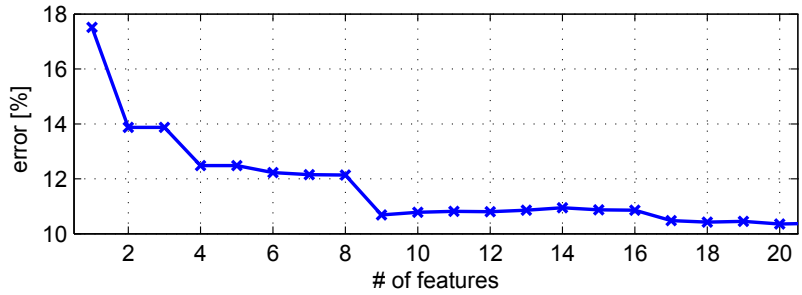
In order to limit the complexity of the resulting controllers by relying only on the most important information for the decision to be made, feature selection has been carried out according to the algorithms in Section 7.3.1.2 (SVMs) and Section 7.3.2.4 (AdaBoost). In the following, we present the results for the energy recovery mode of test case 4. The results for other test cases and other binary modes are qualitatively similar.

7.4.3.1 Feature selection for SVM-based controllers

Figure 7.7(a) depicts the importance ranking of the features according to the score (7.11) between 0 and 1. Note that the colors assign the individual features to a certain group, e.g. states or outputs (cf. legend in Figure 7.8). It can be seen that the feature *cooling power reserve* is of major importance for the decision on energy recovery. Figure 7.7(b) shows the approximation error evaluated by 10-fold cross validation when using only a certain number



(a) Ranking of features according to score function (7.11)



(b) Dependence of approximation error on number of features

Figure 7.7: Feature selection for SVM based controllers for energy recovery mode of test case 4. Only 9 out of 126 features are enough to recover most of the approximation performance when compared to Figure 7.6.

of the most important features. With already 9 features, the SVM based controller achieves an approximation error which is only 3 % worse than the full SVM with 126 features. This yields a significant complexity reduction with only a small performance loss.

7.4.3.2 Feature selection for AdaBoost-based controllers

Figure 7.8 shows the ranking of the features performed according to the accumulated weights of the weak learners, i.e. if two weak learners decide on the same feature, their absolute weights are added.

Interestingly, it turns out that the most important feature is again the cooling power reserve, which was also identified by the feature selection procedure for SVMs. Other features that were ranked high by both methods are the outside air temperature prediction one and two hours ahead, and the total air change rate.

However, we found also features that are highly ranked in SVM but not in AdaBoost-based controllers, for example the heating power reserve, which apparently plays an important role for SVM-based controllers but not so much for the weak learners of AdaBoost. This suggests that feature selection yields controller-dependent results, which are hard to generalize to

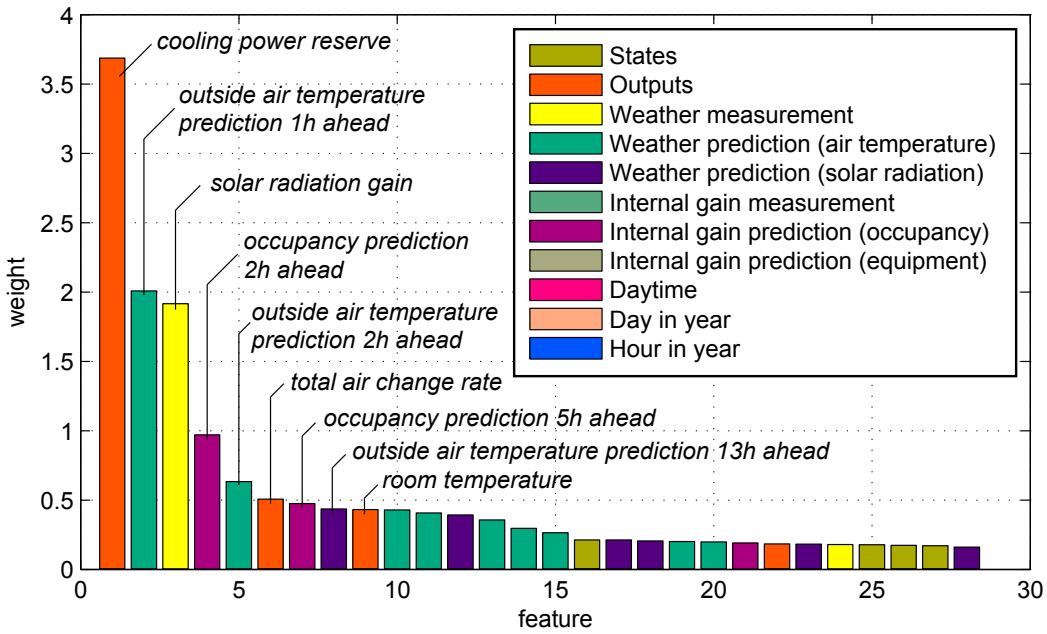


Figure 7.8: Feature ranking by accumulated weights for AdaBoost-based controller for energy recovery mode of test case 4.

other learned controllers. Nevertheless, a significant complexity reduction is possible and therefore feature selection algorithms should be considered whenever possible once the particular controller type has been decided on.

7.4.4 Closed-loop Performance

Due to the advantages of the resulting controller structure of AdaBoost over SVM as discussed in Section 7.3.2.3, and because AdaBoost performs similarly to SVM in the approximation performance assessment in Section 7.4.2, we found AdaBoost more suitable for the problem at hand. Therefore, we focus in this section on closed-loop results for AdaBoost-based controllers only.

The following seven approximate controllers with AdaBoost were employed in the control loop of Figure 7.5: one for each of the six test cases, denoted as MLC-1 to MLC-6, and MLC-A, a controller that was trained with all the data and should serve as a controller for all cases. In the training data for MLC-A, the case number was added as an additional feature, but it has not been ranked very important in the feature selection process, in which we generally kept the 50 most important out of the 200 weak learners. An excerpt of the resulting controller code for MLC-A is shown in Figure 7.10.

The closed-loop simulation results are shown in Figure 7.9. The horizontal axis counts

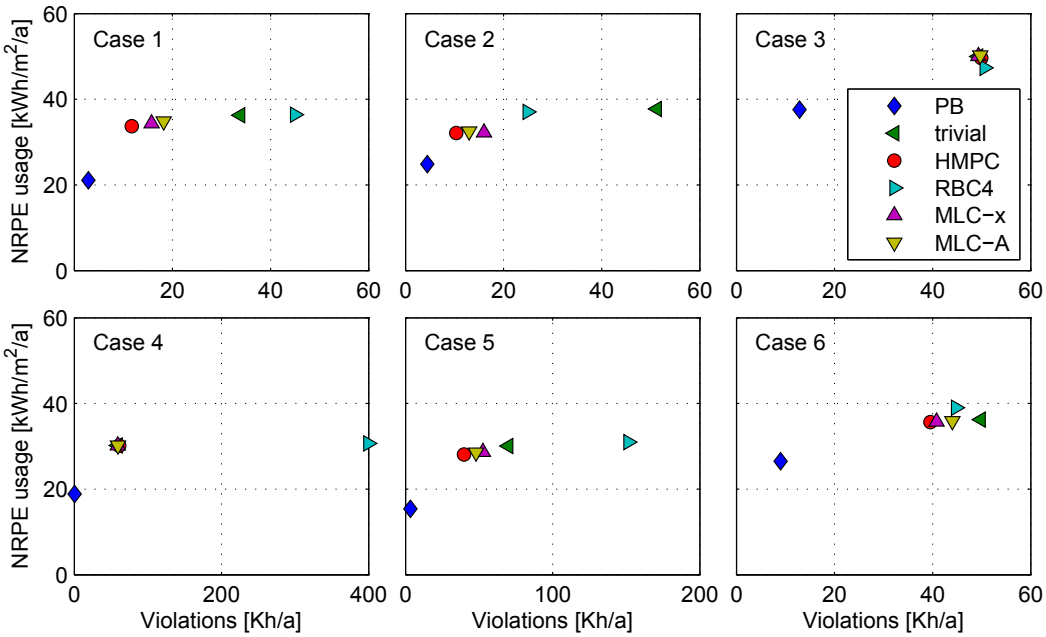


Figure 7.9: Closed-loop simulation results. MLC-x denotes the individual controllers MLC-1 to MLC-6, applied in the corresponding test case.

the comfort constraint violations in Kelvin-hours per year, while the vertical axis depicts the primary energy usage in kWh per year and square meter ground area of the building. If the weather, i.e. the disturbance acting on the system, was known perfectly in advance, the performance bound (PB) would be the best any controller could achieve [103]. The PB therefore serves as a reference point – the closer any controller to the PB point, the better its performance. The six compared high-level controllers are HMPC, the learned controllers MLC-x (where x corresponds to the case number) and MLC-A, the "trivial" controller (always `UNLOAD`) and a state-of-the-art rule based controller (RBC4 [61, §3.2.4]).

We see that, generally, the closed-loop performance of the learned controllers is very similar to the hybrid MPC controller, while the "trivial" controller performs significantly worse on most cases. That is, the closed-loop performance is very good despite the poor approximation performance, and the learning is indeed effectively capturing the behavior of the optimal controller. All learned controllers perform significantly better than RBC4, which relies on manually designed rules. Moreover, while both RBC4 and hybrid MPC are building-dependent, MLC-A performs equally well on *all* building types *with the same 50 decision rules*. This could significantly simplify the deployment of advanced controllers in practice and shows the effectiveness of the proposed approach.

```
float vote = 0;

if( INTERNAL_GAIN_OCCUPANCY > 0.5000 ) // feature 68
        vote += -0.8703;
else
        vote -= -0.8703;

if( T_ROOM_STATE > 22.2935 ) // feature 1
        vote += -0.3746;
else
        vote -= -0.3746;

if( WEATH_PRED_TA_01H > 21.9875 ) // feature 18
        vote += -0.3604;
else
        vote -= -0.3604;

if( T_ROOM_OUT > 21.3625 ) // feature 13
        vote += -0.3224;
else
        vote -= -0.3224;
```

Figure 7.10: Synthesized rules for MLC-A, the controller valid for all building types. Shown are the first 4 out of 50 rules.

7.5 Conclusion

We have proposed a novel way of synthesizing rule based controllers for energy efficient building control, focussing on binary decisions. The main idea is to apply well known methods from machine learning to simulation data with advanced control formulations such as hybrid model predictive control in order to learn the behavior of the controller that otherwise would be difficult to implement in practice.

We have focused on two widely known machine learning methods for classification: SVM and AdaBoost. In our simulation studies, we found that AdaBoost with decision stumps as weak learners has many advantages over SVMs: it directly maps to a rule based controller code with intuitive meaning and outperforms existing rule based controllers. Controller synthesis with AdaBoost is a simple iterative procedure with only a few parameters. The resulting `if-then-else` rules are readable by humans, and the controller code can be executed on existing building control platforms. Despite a significant complexity reduction, near-optimal performance is maintained. Feature selection, possible both for AdaBoost and SVM, is an important tool to further reduce the complexity of the resulting controller. Our results furthermore indicate that it is possible to learn a single controller for many different buildings with good performance, which eases deployment of controller code among different building types.

Part III

Fast Interior Point Methods for Embedded Optimization

8 Code Generation for Convex Multistage Problems

8.1 Introduction

Most optimization problems of the form (2.1) have some kind of underlying structure that arises from physical interconnections or dependencies of variables that evolve over time. This structure oftentimes has a certain regularity, which we exploit in this chapter to enable solving special instances of (2.1) efficiently on embedded platforms.

Such a regular structure is in particular present for optimization problems that arise in the context of model predictive control, where state predictions over a finite horizon are used to approximate the infinite horizon constrained optimal controller. Here, a clear temporal dependence is established by assuming memoryless (Markov) systems: the state at time k depends on the state and action taken at time $k-1$: $x_k = f(x_{k-1}, u_{k-1})$, where f denotes our model of the dynamic system. Similarly, in portfolio optimization, we seek an investment strategy for assets for which prices evolve over time, and the composition of a portfolio at time k depends on the portfolio and investment decisions at time $k-1$. Another example for optimization problems with temporal dependent variables is an energy management system for smart home appliances that decides how much of the locally produced energy to store locally or to sell to the grid. In moving horizon estimation (MHE), a sequence of temporally related measurements is used to estimate the current state of the system.

Besides these temporal dependencies of variables, there are also problems where spatial constraints introduce a coupling that can be modelled by *interdependencies between subsequent variables* (although the ordering of the variables might be less obvious for these problems). Examples are spline fitting, where each segment of the spline is constrained on the left and the right, or any kind of networked system that has an underlying graph representation through a spanning tree.

In these examples, variable $v_i \in \boldsymbol{R}^{n_i}$ depends on the variable $v_{i-1} \in \boldsymbol{R}^{n_{i-1}}$ through some equality constraint $L_i(v_{i-1}, v_i) = 0$, and vice versa. Because of this relation, we call v_i a *stage variable*, and consider *multistage* optimization problems with $N+1$ stages of the

form

$$\min_{\{v_i\}} \sum_{i=0}^{N} l_i(v_i) \tag{8.1a}$$

$$\text{s.t. } g_i(v_i) \leq 0\,, \qquad i = 0, \ldots, N\,, \tag{8.1b}$$

$$L_0\,(v_0) = 0 \tag{8.1c}$$

$$L_i\,(v_{i-1}, v_i) = 0\,, \quad i = 1, \ldots, N\,, \tag{8.1d}$$

where the sum of stage costs specified by convex functions $l_i : \boldsymbol{R}^{n_i} \to \boldsymbol{R}$ is minimized. All inequalities on the stage variables v_i are stage-wise (or act only locally), cf. (8.1b). We furthermore assume that $g_i : \boldsymbol{R}^{n_i} \to \boldsymbol{R}^{m_i}$ are component-wise convex and that the sets $\{v_i \mid g_i(v_i) \leq 0\}$, $i = 0, \ldots, N$ have nonempty interior. Finally, we assume that the functions $L_0 : \boldsymbol{R}^{n_0} \to \boldsymbol{R}^{p_0}$ and $L_i : \boldsymbol{R}^{n_{i-1}} \times \boldsymbol{R}^{n_i} \to \boldsymbol{R}^{p_i}$ for $i = 1, \ldots, N$ are affine:

$$L_0\,(v_0) \triangleq D_0 v_0 - c_0 \tag{8.2a}$$

$$L_i\,(v_{i-1}, v_i) \triangleq C_{i-1} v_{i-1} + D_i v_i - c_i\,, \quad i = 1, \ldots, N\,, \tag{8.2b}$$

with $c_i \in \mathbb{R}^{p_i}$ and $C_{i-1} \in \boldsymbol{R}^{p_i \times n_{i-1}}$, $D_i \in \boldsymbol{R}^{p_i \times n_i}$ such that $[C_{i-1}\ D_i]$ has full row rank. The equality constraint (8.2a) is used to capture the notion of an initial value of the variable v_0. The (primal) optimization variable $v \triangleq (v_0, \ldots, v_N)$ is of dimension $n = \sum_{i=0}^{N} n_i$, and problem (8.1) has $m = \sum_{i=0}^{N} m_i$ inequality constraints and $p = \sum_{i=0}^{N} p_i$ equality constraints in total. Problem (8.1) is convex (and potentially nonlinear) and will be solved by interior point methods in this thesis.

Most relevant MPC and MHE problems can be cast into the form of (8.1), including problems with 1- or ∞-norm cost terms [18]. The auxiliary variables for the LP reformulation become part of the stage variable v_i. Problems with equality constrained initial state are captured as well as formulations where the initial state is an optimization variable, such as in robust MPC problems [90]. Note that stage-wise inequalities (8.1b) can always be established by artificially introducing copies of (parts of) stage variables in each stage, and equating them via (8.1d), as is necessary for example to model input rate constraints for linear MPC.

Main Contributions

The key to obtain an efficient interior point method for solving (8.1) is to exploit the inherent problem structure. One of the main contributions of this part of the thesis is to make the solution of general multistage problems of type (8.1) on embedded platforms both *feasible* (in terms of computing time) by the efficient interior point method described

in Section 8.2, and *publicly available* through the code generation system described in Section 8.4 and available under `forces.ethz.ch`. We focus on interior point methods due to their robustness to problem conditioning and their applicability to a large class of problems, including problems with convex quadratic constraints. To the best of our knowledge, the FORCES solvers are the first interior point implementations targeting embedded systems that can handle these type of nonlinear constrains.

Outline

The remainder of this chapter is organised as follows:

- First, we detail on how to obtain efficient primal-dual interior point methods for multistage problems by exploiting their structure in Section 8.2. A thorough theoretical cost analysis reveals that our method of computing the search direction is faster than previously proposed methods for standard linear MPC. We show in which cases further structure exploitation, beyond the multistage structure, is possible and quantify possible speedups based on the theoretical cost analysis. An interesting result is that some MPC problems with quadratic constraints are in fact less expensive than problems with polytopic approximations of the latter, which are however usually used in practice due to the limited availability of solvers that can handle quadratic constraints directly.

  We present extensive numerical benchmarks of the well known interior point codes CPLEX [67] and CVXGEN [88] in comparison to our solver called FORCES. We include timings of the sparse factorization code MA57 [40] as called from OOQP [49] in order to show the merits of problem specific vs. general sparse factorization. While it is immediate that a customized solver, such as CVXGEN, is faster than general-purpose solvers, we show that FORCES is between 2-5 times faster than CVXGEN, which in turn is one to two orders of magnitude faster than CPLEX. This speedup is achieved by structure exploitation and the fact that the FORCES code is of near-constant size even for large problem dimensions.
- Section 8.3 focuses on further speedups by exploiting parallelism. In particular, we propose a shared memory multiprocessing computing model for the proposed solver in Section 8.3.1. Parallelization using FPGAs is discussed in Section 8.3.2.
- Bringing the findings of the previous sections together, Section 8.4 describes the FORCES code generator system for multistage problems that generates problem-specific, library-free ANSI C code. We describe implementation details and show benchmark results for the generated code, which matches the performance of the hand-written solver from Section 8.2, since it is automatically exploiting as much

structure as possible in the given problem.

- Last, we present two advanced examples in Section 8.5 showing that the FORCES code generation system creates high-performance solvers even for complex problems. One of the presented applications is automatic racing of RC race cars, which has been implemented on a real-world system and demonstrates that the generated solvers enable high-speed optimal control on a large number of embedded platforms.

8.2 Efficient Interior Point Solver for Multistage Problems

8.2.1 Problem Structure

We solve (8.1) using the primal-dual Mehrotra predictor-corrector interior point method in Algorithm 2.5 from Section 2.2.2.2. A primal-dual search direction is computed by solving the linear system (2.58) (we use v instead of x for the primal variables to avoid confusion with the state x of a dynamic system),

$$\begin{bmatrix} H(v,z) & A^T & G(v)^T & 0 \\ A & 0 & 0 & 0 \\ G(v) & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta v \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = - \begin{bmatrix} \nabla f(v) + A^T y + G(v)^T z \\ Av - b \\ g(v) + s \\ Sz - \nu \end{bmatrix} = \begin{bmatrix} r_S \\ r_E \\ r_I \\ r_C(\nu) \end{bmatrix} \tag{8.3}$$

for different values of ν. For multistage problems (8.1) with equality constraints (8.2),

$$v = (v_0, \ldots, v_N)\,,\ y = (y_0, \ldots, y_N)\,, \tag{8.4a}$$

$$s = (s_0, \ldots, s_N)\,,\ z = (z_0, \ldots, z_N)\,, \tag{8.4b}$$

$$\nabla f(v) = (\nabla l_0(v_0), \ldots, \nabla l_N(v_N)) \in \boldsymbol{R}^n\,, \tag{8.4c}$$

$$A = \begin{bmatrix} D_0 & 0 & \cdots & \cdots & 0 \\ C_0 & D_1 & 0 & \cdots & 0 \\ 0 & C_1 & D_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & C_{N-1} & D_N \end{bmatrix} \in \boldsymbol{R}^{p \times n}\,, \tag{8.4d}$$

$$G(v) = \text{blkdiag}\left(\nabla g_0(v_0)^T,\ \ldots,\ \nabla g_N(v_N)^T\right)\ \in \boldsymbol{R}^{m \times n}\,, \tag{8.4e}$$

$$b = (c_0, \ldots, c_N) \in \boldsymbol{R}^p\,, \tag{8.4f}$$

$$g(v) = (g_0(v_0), \ldots, g_N(v_N)) \in \boldsymbol{R}^m\,. \tag{8.4g}$$

The diagonal matrix $S \triangleq \text{diag}\,(s_0, \ldots, s_m)$ is constructed from the slack variables s_i associated with inequalities $g_i(v_i),\ i = 0, \ldots, N$, and similarly $Z \triangleq \text{diag}\,(z_0, \ldots, z_m)$ is the

diagonal matrix constructed from the associated Lagrange multipliers z_i. The (1,1) block of the coefficient matrix in (8.3) is

$$H(v,z) \triangleq \text{blkdiag}\left(H_0\left(v_0,z_0\right),\ldots,H_N\left(v_N,z_N\right)\right) \in \boldsymbol{R}^{n\times n} \tag{8.4h}$$

where

$$H_i(v_i,z_i) \triangleq \nabla^2 l_i(v_i) + \sum_{j=1}^{m_i} z_{ij}\nabla^2 g_{ij}(v_i)\,. \tag{8.4i}$$

Note that due to the multistage structure, $H(v,z)$ and $G(v)$ are block diagonal (because the objective and inequality constraints in (8.1) act only stage-wise), and the equality constraint matrix A has a banded structure. Therefore, when reducing the system (8.3) to the positive definite/normal equations form as described in Section 3.2, one obtains for (3.7)

$$\Phi \triangleq \text{blkdiag}\left(\Phi_0,\ldots,\Phi_N\right)\,, \tag{8.5}$$

where each individual block is defined as

$$\Phi_i \triangleq H_i\left(v_i,z_i\right) + \nabla g_i(v_i)^T S^{-1} Z \nabla g_i(v_i)\,. \tag{8.6}$$

The resulting linear system to be solved twice (predictor-corrector, cf. Section 2.2.2.3) per interior point iteration is (3.8b), $Y\Delta y = \beta$. The coefficient matrix $Y \in \boldsymbol{R}^{p\times p}$ is defined in (3.8c) as $Y \triangleq A\Phi^{-1}A^T$, which due to the multistage structure is block-banded:

$$\begin{aligned} Y &= \begin{bmatrix} D_0 & 0 & \cdots & 0 \\ C_0 & D_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & C_{N-1} & D_N \end{bmatrix} \begin{bmatrix} \Phi_0^{-1} & & & \\ & \Phi_1^{-1} & & \\ & & \ddots & \\ & & & \Phi_N^{-1} \end{bmatrix} \begin{bmatrix} D_0^T & C_0^T & \cdots & 0 \\ 0 & D_1^T & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & D_{N-1}^T & C_{N-1}^T \\ 0 & \cdots & 0 & D_N^T \end{bmatrix} \\ &= \begin{bmatrix} Y_{0,0} & Y_{0,1} & 0 & \cdots & 0 \\ Y_{0,1}^T & Y_{1,1} & Y_{1,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & Y_{N-1,N} \\ 0 & 0 & \cdots & Y_{N-1,N}^T & Y_{N,N} \end{bmatrix}, \end{aligned} \tag{8.7}$$

where

$$Y_{0,0} \triangleq D_0\Phi_0^{-1}D_0^T \qquad \in \boldsymbol{R}^{p_0\times p_0}\,, \tag{8.8a}$$

$$Y_{i,i} \triangleq C_{i-1}\Phi_{i-1}^{-1}C_{i-1}^T + D_i\Phi_i^{-1}D_i^T \qquad \in \boldsymbol{R}^{p_i\times p_i}\,, \qquad i=1,\ldots,N\,, \tag{8.8b}$$

$$Y_{i,i+1} \triangleq D_i\Phi_i^{-1}C_i^T \qquad \in \boldsymbol{R}^{p_i\times p_{i+1}}\,, \qquad i=0,\ldots,N-1\,. \tag{8.8c}$$

Solving $Y\Delta y = \beta$ is the major bottleneck in the interior point method. We discuss in the next section in detail how to solve this equation efficiently and analyze the computational cost. Note that once we have calculated Δy, the other search directions follow from (3.8a), (3.3a) and (3.2a):

$$\Delta v = \Phi^{-1}\left(r_s - A^T \Delta y\right) , \tag{8.9a}$$

$$\Delta z = -S^{-1}Z\left(r_l - Z^{-1}r_C - G(v)\Delta v\right) , \tag{8.9b}$$

$$\Delta s = Z^{-1}\left(r_C - S\Delta z\right) . \tag{8.9c}$$

When carrying out these operations, an efficient implementation of course exploits the structure of A, $G(v)$ and Φ^{-1}. In general, the cost of evaluating (8.9) is negligible compared to the cost of calculating Δy, particularly because the Cholesky factors of Φ_i are available from the solution of (3.8b) as we show in the next section.

8.2.2 Search Direction Computation in the General Case

In the following, we outline the block Cholesky procedure from [135] to solve (3.8b) adapted to multistage problems (8.1) and give a cheaper method to compute Y (8.8).

8.2.2.1 Efficient calculation of Y

The authors of [135] suggest to carry out the computations for Y in (8.8) as follows. First, obtain the Cholesky factors L_i such that $\Phi_i = L_i L_i^T$. Then, a matrix forward- and backward-substitution is carried out to form $\tilde{C}_{i-1} \triangleq C_{i-1}\Phi_{i-1}^{-1}$ and $\tilde{D}_i \triangleq D_i\Phi_i^{-1}$, respectively. To obtain (8.8b), $\tilde{C}_{i-1}$ and $\tilde{D}_i$ are multiplied from the right by C_{i-1}^T and D_i^T, respectively; to obtain (8.8c), $\tilde{D}_i$ is multiplied by C_i^T.

However, we can do better than that as follows. After obtaining the Cholesky factor L_i as above, we solve

$$V_i L_i^T = C_i , \tag{8.10a}$$

$$W_i L_i^T = D_i , \tag{8.10b}$$

for $V_i \in \boldsymbol{R}^{p_i \times n_i}$ for $i = 0, \ldots, N-1$ and $W_i \in \boldsymbol{R}^{p_i \times n_i}$ for $i = 0, \ldots, N$ by matrix forward substitution. We now have a *rectangular factorization* of the quantities needed for (8.8), i.e.

$$C_{i-1}\Phi_{i-1}^{-1}C_{i-1}^T = V_{i-1}V_{i-1}^T , \tag{8.11a}$$

$$D_i\Phi_i^{-1}D_i^T = W_i W_i^T , \tag{8.11b}$$

$$D_i\Phi_i^{-1}C_i^T = W_i V_i^T . \tag{8.11c}$$

Table 8.1: Cost of elementary matrix operations for dense matrices $A, B : m \times n$, $C : n \times p$, $D, L : n \times n$, L lower triangular.

Operation	Cost (flops)
Matrix matrix multiplication AC	$2mnp$
Symmetric matrix matrix multiplication AA^T	m^2n
LL^T decomposition s.t. $D = LL^T$	$1/3n^3$
Matrix forward substitution s.t. $AL^T = B$	mn^2
Matrix backward substitution s.t. $AL = B$	mn^2

Table 8.2: Cost for computing $Y_{i,i}$ and $Y_{i,i+1}$ in (8.8) with different methods (in flops).

Step	Operation	[135]	Operation	Proposed
1	Factor Φ_i $(= L_i L_i^T)$	$1/3n_i^3$	Factor Φ_i $(= L_i L_i^T)$	$1/3n_i^3$
2	$\tilde{C}_{i-1} \triangleq C_{i-1}\Phi_{i-1}^{-1}$	$2n_{i-1}^2 p_i$	Solve $V_i L_i^T = C_i$ for V_i (8.10a)	$n_{i-1}^2 p_i$
3	$\tilde{D}_i \triangleq D_i \Phi_i^{-1}$	$2n_i^2 p_i$	Solve $W_i L_i^T = D_i$ for W_i (8.10b) for W_i	$n_i^2 p_i$
4	$\tilde{C}_{i-1} C_{i-1}^T$	$n_{i-1} p_i^2$	$C_{i-1}\Phi_{i-1}^{-1} C_{i-1}^T = V_{i-1} V_{i-1}^T$ (8.11a)	$n_{i-1} p_i^2$
5	$\tilde{D}_i D_i^T$	$n_i p_i^2$	$D_i \Phi_i^{-1} D_i^T = W_i W_i^T$ (8.11b)	$n_i p_i^2$
6	$\tilde{D}_i C_i^T$	$2n_i p_i^2$	$D_i \Phi_i^{-1} C_i^T = W_i V_i^T$ (8.11c)	$2n_i p_i^2$

Based on elementary matrix operations in Table 8.1, we compare the total cost of the two methods in Table 8.2. The proposed method saves two matrix back-substitutions and ensures that $Y_{i,i}$ is symmetric even in case of rounding errors, which is not the case for [135] due to the asymmetric matrix products. Overall, our method saves $\left(n_{i-1}^2 + n_i^2\right) p_i$ floating point operations (flops[1]) and is numerically superior. Furthermore, it enables significant computational savings for special instances of (8.1) as described in Section 8.2.3.2. These simplifications are possible only to a limited extent when using [135].

8.2.2.2 Block-wise Cholesky factorization of Y

After obtaining Y, we compute its lower triangular Cholesky factor L_Y,

$$L_Y = \begin{bmatrix} L_{0,0} & 0 & 0 & \dots & 0 \\ L_{1,0} & L_{1,1} & 0 & \dots & 0 \\ 0 & L_{2,1} & L_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & L_{N,N-1} & L_{N,N} \end{bmatrix} \in \boldsymbol{R}^{p \times p},$$

by solving

$$Y_{0,0} = L_{0,0} L_{0,0}^T \quad \text{for } L_{0,0} \in \boldsymbol{R}^{p_0 \times p_0}, \tag{8.12a}$$

[1] addition, multiplication or division of two double precision numbers

Table 8.3: Costs for the block-wise Cholesky factorization of Y.

Operation	Cost (flops)	
(8.12a)	$2/3p_0^3$	
(8.12b)	$p_i^2 p_{i+1}$	$i = 0, \ldots, N-1$
(8.12c)	$p_{i-1}^2 p_i + 2/3p_i^3$	$i = 1, \ldots, N$

$$Y_{i,i+1} = L_{i,i} L_{i+1,i}^T \quad \text{for } L_{i+1,i}^T \in \boldsymbol{R}^{p_i \times p_{i+1}} \quad i = 0, \ldots, N-1\,, \text{ and} \tag{8.12b}$$

$$Y_{i,i} - L_{i,i-1} L_{i,i-1}^T = L_{i,i} L_{i,i}^T \quad \text{for } L_{i,i} \in \boldsymbol{R}^{p_i \times p_i}\,, \quad i = 1, \ldots, N\,, \tag{8.12c}$$

where $L_{i,i}$ are lower triangular and $L_{i+1,i}$ are generally dense matrices. Equation (8.12b) is solved by matrix forward substitution, while (8.12a) and (8.12c) are solved by Cholesky factorizations of the corresponding matrices on the left hand side. This is a standard procedure and follows immediately from the structure of Y [135]. The total cost of this step is given in Table 8.3.

Based on the cost analysis in Tables 8.2 and 8.3 and on measurements in practice, we observe that for typical problems the block-wise Cholesky factorization of Y amounts to roughly 20% of the total costs for computing the search direction, while its actual construction makes up about 80% of the computations. Moreover, the problem data of (8.1) does not have a major influence on the cost of the block-wise Cholesky factorization (8.12). This is not the case for the computation of Y in (8.8), the cost of which significantly depends on the specific problem formulation. It is therefore worth investigating under which circumstances the computational burden can be lowered.

8.2.3 Structure Exploitation for Common Problem Instances

In addition to the multistage structure presented in Section 8.2.1, most practical instances of (8.1) have a special structure in terms of constraints and objective, which we exploit in the following to save unnecessary computations. These savings can be significant: an extra speedup of five or more for typical problems can be achieved.

There are two sources for speedup: when "inverting" the matrix Φ_i defined in (8.6) we can exploit its structure, for example if it is diagonal or diagonal plus low rank. This accelerates not only the computation of the Cholesky factor of Φ_i, L_i, in Step 1 of Table 8.2, but also the subsequent matrix forward substitutions in Steps 2 and 3. We detail on structure exploitation related to Φ_i in Sections 8.2.3.1 to 8.2.3.4. Additional speedup can be achieved by exploiting the special structure of equality constraints (in particular, the structure of D_i in (8.2)), which we discuss in Section 8.2.3.5. This further speeds up Steps 3, 5 and 6 of Table 8.2, and combined with the results for structure exploitation of Φ_i extremely efficient

code is obtained.

The considerations in this section hold block-wise, i.e. at each stage, any of these structure exploitation techniques can be applied, depending on the particular structure of the cost and constraints for the stage considered.

8.2.3.1 Diagonal Costs and Bound Constraints

For general costs and inequalities, the matrix Φ_i is dense, and so is the (lower triangular) Cholesky factor L_i. However, it is obvious that Φ_i is diagonal if the Hessian of the objective, $\nabla^2 l_i(v_i)$, is a diagonal matrix and $g_i(v_i) \leq 0$ are simple bound constraints $\underline{v}_i \leq v_i \leq \bar{v}_i$. In this case, the cost of Steps 1-3 in Table 8.2 is negligible since the Cholesky factor of Φ_i, L_i, is a diagonal matrix, so each operation related to L_i costs in the order of p_i^2 flops, instead of p_i^3. In particular, the Cholesky factorization in Step 1 costs only p_i square roots, and the cost of the matrix forward substitutions in Steps 2 and 3 are reduced to p_i^2. Also, the Cholesky factor L_i can be stored as a vector of size p_i (instead of $(p_i(p_i+1)/2)$.

8.2.3.2 Diagonal Costs and One Diagonal Quadratic Constraint

Sometimes Φ_i (8.6) is dense but has diagonal plus rank one structure,

$$\Phi_i = \mathcal{D} + \alpha w w^T , \tag{8.13}$$

where $\alpha > 0$, $w \in \boldsymbol{R}^{n_i}$ and $\mathcal{D} \triangleq \operatorname{diag}(d_1, \ldots, d_{n_i}) \in \boldsymbol{R}^{n_i}$. This happens, for example, if the Hessian of the objective, $\nabla^2 l_i(v_i)$ is diagonal, one has only simple bound constraints $\underline{v}_i \leq v_i \leq \bar{v}_i$, and one additional convex quadratic constraint $v_i^T Q v_i + g^T v_i \leq r_i$ with Q diagonal. (This includes the case of one affine constraint $g^T v_i \leq r_i$ if $Q \equiv 0$). In this case, the computational cost for obtaining the Cholesky factor of Φ_i, L_i, in Step 1 of Table 8.2 can be made of order n_i^2 flops instead of order n_i^3 by a low rank modification, cf. Section 3.2.2. We use Algorithm 3.2 with the correspondence $L \equiv I_{n_i}$, $\tilde{L} \equiv L_i$, $D \equiv \mathcal{D}$ and $z \equiv w$ to obtain the recursion

$$L_{i,kk} = \sqrt{d_k + \alpha_k w_k^2} , \qquad k = 1, \ldots, n_i \tag{8.14a}$$

$$L_{i,kj} = \beta_j w_k , \qquad k = j+1, \ldots, n_i , \tag{8.14b}$$

where

$$\beta_j \triangleq \alpha_j w_j / L_{i,jj} , \qquad j = 1, \ldots, n_i , \tag{8.14c}$$

$$\alpha_{j+1} \triangleq d_j \alpha_j / L_{i,jj}^2 , \qquad j = 1, \ldots, n_i - 1 \leq j < n , \tag{8.14d}$$

$\alpha_1 = \alpha$ and $L_{i,kj}$ denotes the element of L_i in the kth row and jth column. This makes Step 1 of order n_i^2. Furthermore, note that the L_i computed above has the special structure (3.12). This allows for an efficient matrix forward substitution as given in Algorithm 3.4, which reduces the cost of Steps 2 and 3 to $n_i p_i$ instead of $n_i^2 p_i$. For more details see Section 3.2.2.3.

Remark 8.1 (Numerical Instability of Sherman-Morrison Formula)**.** *It is easy to show that with the formula by* Sherman and Morrison *[117], calculating V_i and W_i in* (8.10a) *and* (8.10b) *directly (without calculating the Cholesky factor L_i first) is possible at an overall cost of order $n_i p_i$ (the same as our method). However, it is well known that [117] is an unstable method [142], which we also observed in our experiments. Our method, instead, is numerically stable, as we effectively calculate the Cholesky factor L_i in a stable manner [50].*

8.2.3.3 Diagonal Costs, Bounds and Multiple Diagonal Quadratic Constraints

The approach of Section 8.2.3.2 can be generalized to k quadratic constraints by making use of the product form Cholesky factorization from Section 3.2.2.2. For this to be efficient, $H_i(z_i, v_i)$ in (8.6) must be a diagonal matrix; consequently, Φ_i has diagonal plus rank k structure, and the product form Cholesky factorization can be computed with Algorithm 3.3. Since $H_i(z_i, v_i)$ is assumed diagonal, $L = I_{n_i}$ in (3.15), and

$$\Phi_i = L_1(\beta_1, q_1) \dots L_k(\beta_k, q_k) D L_k(\beta_k, q_k)^T \dots L_1(\beta_1, q_1)^T , \tag{8.15}$$

where each $\beta_j \in \boldsymbol{R}^{n_i}$, $q_j \in \boldsymbol{R}^{n_i}$ for $j = 1, \dots, k$. This has the advantage that any forward/backward substitution can be split up into k inexpensive forward/backward substitutions with special Cholesky factors $L_j(\beta_j, q_j)$, cf. (3.12) for their structure. The matrix $L_j(\beta_j, q_j)$ is never explicitly formed, as it is completely defined by the two vectors β_j and q_j. This saves memory if k is small. Using the product form Cholesky factorization, the cost for Step 1 in Table Table 8.2 reduces to the order $k n_i^2$ instead of n_i^3. The matrix forward substitutions, Steps 2 and 3 in Table 8.2, reduce to the order of $k p_i n$ instead of $n_i^2 p_i$. The speedup is about n_i/k for these steps; hence this structure exploitation makes sense if $k \ll n_i$, which is often the case for quadratic constraints.

8.2.3.4 Dense Cost and Quadratic Constraint with Same Hessian

In some cases, the Hessian of the objective coincides with the Hessian of a quadratic constraint, $\nabla^2 l_i(v_i) = \nabla^2 g_i(v_i) \triangleq H_i$. This is the case for example in MPC with terminal state cost $x_N^T P x_N$ and terminal constraint $x_N^T P x_N \leq \alpha$. In these cases,

$$H(v_i, z_i) = H_i + \sum_{i=1}^{m_i} z_i H_i , \tag{8.16}$$

for which the Cholesky factor can be easily calculated in the order of n_i^2 operations:

$$L_{H(v_i,z_i)}(z_i) \triangleq L_{H_i}\left(I_{n_i} + \sum_{j=1}^{m_i} z_i I_{n_i}\right)^{1/2}. \tag{8.17}$$

The matrix L_{H_i} is the Cholesky factor of H_i that can be computed offline and stored. If now in addition $\nabla g(v_i)S^{-1}Z\nabla g_i(v_i)^T$ is low rank (for example as in Sections 8.2.3.2 and 8.2.3.3), then the Cholesky factor of Φ_i can be computed by a low rank modification of $L_{H(v_i,z_i)}$ at a cost of order n_i^2. This procedure makes the cost of Step 1 in Table 8.2 negligible. If the product form Cholesky factorization from (3.2.2.2) is used, further savings in the matrix forward substitutions in Steps 2 and 3 are possible.

8.2.3.5 Zero-Eye Structure of D_i

In most problems (such as MPC, for example), the matrix D_i is used merely to select parts of the stage variable, and has the structure

$$D_i = \begin{bmatrix} 0 & I_{p_i} \end{bmatrix}, \tag{8.18}$$

which does not have to be stored if the structure is known. We have two cases:

1. Dense Φ_i, i.e. dense L_i. In Step 3 of Table 8.2, unnecessary computations can be saved when solving (8.10b) for W_i, since the latter has the structure

   $$W_i = D_i/L_i^T = \begin{bmatrix} 0 & W_i^u \end{bmatrix} \tag{8.19}$$

   with an upper triangular $W_i^u \in \boldsymbol{R}^{p_i \times p_i}$ that can be computed by solving

   $$W_i^u \tilde{L}_i^T = I_{p_i} \tag{8.20}$$

   by backward substitution. In (8.20), we define $\tilde{L}_i$ to be the part of L_i consisting of the last p_i columns and rows. (In Matlab notation, $\tilde{L}_i = L_i(n_i-p_i+1 : n_i, n_i-p_i+1 : n_i)$). Using this knowledge of the structure of D_i, the cost of Step 3 in Table 8.2 can be reduced from $n_i^2 p_i$ to $\frac{1}{2}p_i^3$. This can be significant if $p_i \ll n_i$.
   After W_i has been computed in Step 3, computing $W_i W_i^T$ in Step 5 and $W_i V_i^T$ in Step 6 is less expensive since the first $n_i - p_i$ columns of W_i are zero, and the remainder is an upper triangular matrix. The cost for Step 5 reduces therefore to $\frac{1}{4}p_i^3$ instead of $n_i p_i^2$. Similarly, the cost of Step 6 reduces from $2n_i p_i^2$ to $\frac{1}{2}p_i^3$.
2. Diagonal Φ_i, i.e. diagonal L_i. In this case, W_i^u in (8.19) is diagonal: it is the inverse of L_i, which can be computed by p_i divisions. The computation of W_i in Step 3 is therefore negligible (and W_i can be stored as a vector), and so is the computation of

$W_i W_i^T$ in Step 5. The cost of Step 6 reduces from $2n_i p_i^2$ to p_i^2 and is also negligible. This is the least expensive case, with only Step 4 remaining of cubic complexity as given in Table 8.2.

8.2.3.6 Application to MPC Problems

Table 8.4 lists the approximate theoretical speedup factors for computing Y_i and $Y_{i,i+1}$ (8.8) when making use of the fine-grained structure exploitations from Sections 8.2.3.1 to 8.2.3.4 as compared to the base case presented in Table 8.2. We assume that the multistage problem is an MPC problem, which also enables the structure exploitation presented in Section 8.2.3.5. For example, a problem with diagonal quadratic cost matrices and bound constraints incurs about a factor 9.3x less floating point operations than the general problem. Since the considerations of this section hold block-wise, each stage can have a different structure of constraints and objective; the net speedup is then the average speedup computed for each stage individually according to Table 8.4.

Remark 8.2. *The numbers in Table 8.4 represent theoretical estimates obtained by considering only the highest order cost terms of the Steps in Table 8.2 adapted to the respective situation; in practice, the effective speedup is usually lower due to lower order terms and properties of the computing hardware such as cache and memory bandwidth.*

The results presented in this section indicate which MPC formulations can be solved more quickly than others when using the proposed interior point method. This information can be helpful in the control synthesis procedure whenever the structure of the constraints and objectives represent degrees of freedom for the control designer. For example, the most expensive case occurs when dense linear constraints are present, as often used in MPC formulations with polytopic sets. The least expensive problem occurs for bound constraints and diagonal costs, while problems with quadratic constraints are in between these two extremes. The common case of a quadratic terminal cost, $l_N(x_N) = x_N^T P x_N$, along with a level set of the latter as terminal set constraint, $x_N^T P x_N \leq \alpha$, is still quicker to compute than problems with polytopic sets. This is an important result, since the computation of polytopic terminal, invariant sets is prohibitively complex for high dimensions and quadratic terminal sets therefore represent a computationally beneficial alternative that can be applied to all problem dimensions.

Example 8.1. *Consider a typical MPC regulation problem of type* (4.9) *with horizon length of* $N = 10$, $n_x = n_u$ *states and inputs (thus* $n_i = 2p_i$*), bound constraints for the first* 9 *stages on state* x *and on input* u, *i.e.* $\mathbb{X} \triangleq \{x \in \boldsymbol{R}^{n_x} \mid \underline{x} \leq x \leq \bar{x}\}$ *and* $\mathbb{U} \triangleq \{u \in \boldsymbol{R}^{n_u} \mid \underline{u} \leq u \leq \bar{u}\}$, *respectively, and a terminal constraint* $x_N^T P x_N \leq 1$. *The stage*

Table 8.4: Theoretical (approximate) speedup factors when calculating the matrices Y_i and $Y_{i,i+1}$ (8.8) and exploiting the particular structure of objective $l_i(v_i)$ and constraints $g_i(v_i)$ as compared to the base case from Table 8.2. We assume an MPC multistage problem, matrices $F \in \boldsymbol{R}^{m \times n_i}$ with $m > n_i$, $Q, M \in \boldsymbol{S}^{n_i}_{++}$ and $n_i = 2p_i$.

		Objective $l_i(v_i)$		
		$c_i^T v_i$	$v_i^T Q_i v_i$, Q_i diag.	$v_i^T Q_i v_i$, Q_i dense
Constraints $g_i(v_i) \leq 0$	$\underline{v} \leq v_i \leq \bar{v}$	9.3x	9.3x	1.4x
	$Fv_i \leq f_i$	1.4x	1.4x	1.4x
	$v_i^T M_i v_i \leq r$, M_i diag.	6.7x	6.7x	1.4x
	$v_i^T M_i v_i \leq r$, M_i dense	1.4x	1.4x	1.4x (1.8x if $Q = M_i$)

costs are $l(x_i, u_i) = x_i^T Q x_i + u_i^T R u_i$ *with* $Q \in \boldsymbol{S}^{n_x}_{+}, R \in \boldsymbol{S}^{n_u}_{++}$ *diagonal, and a terminal penalty* $x_N^T P x_N$*. Thus when evaluating* (8.8)*,* 9 *stages allow for a speedup of* $9.3x$ *when compared to the base case according to Table 8.4 (the matrix* Φ_i *is diagonal in these stages, cf. Section 8.2.3.1). The computations for the last stage are more intense due to the terminal cost and constraint. The last stage is in the category of Section 8.2.3.4, and a speedup of about* $1.8x$ *can be expected by exploiting the structure. The total speedup for computing* (8.8) *is therefore about* $8.5x$*, and since* (8.8) *amounts to about* 80% *of the total computations in the interior point method, structure exploitation reduces the number of floating point operations by about* 70% *in this example when compared to a general multi-stage problem setup.*

8.2.3.7 Summary

To summarize, we have extended the tailored Newton step computation introduced in [135] to the more generic problem class (8.1) and provided a detailed cost analysis of the search direction computation. Based on this analysis, special cases of (8.1) which allow for significant computational savings were identified. Among these are problems with box constraints, diagonal costs and, most importantly, quadratic constraints which are preferable in MPC formulations with stability guarantees. These can be in fact cheaper than formulations with general polytopic constraints, which, despite their wide usage, belong to the most expensive category.

8.2.4 Computational Results

The results presented in this section have been obtained with a code written in strict ANSI-C in order to support a wide range of embedded platforms. For high performance, the operations for computing Φ, Y, L_Y and the forward substitutions needed for $\Delta\nu$ are carried out block-wise; for instance, instead of computing Y as a whole matrix, we directly factor Y_{ii} as soon as it is available and perform the forward substitution. This interleaving, or "zipping" of basic linear algebra operations greatly enhances both temporal and spatial locality of the code, reducing expensive cache misses. Note that the matrices involved in the computation of (8.8) and (8.12) are of small dimension (in case of linear MPC with quadratic costs: n_x and n_u) and thus typically fit into the first level cache. Therefore, slow data transfers to or from main memory are significantly reduced and the CPU or DSP can be kept as busy as possible.

Our solver builds on top of a small linear algebra library for operations on symmetric matrices, which we store in lower triangular format. We use static memory allocation, and store the working set of the code in simple C arrays. Computations were cached in memory if the result is needed at different places of the solver (e.g. the term $S^{-1}\Lambda$).

Despite the aforementioned measures for high performance, we would like to point out that we use standard, naive code with nested `for`-loops for our linear algebra system. Since the matrix dimensions are fixed a-priori, it is easy for the compiler to perform loop unrolling optimizations. As a result, our code is very small and library free[2]. In fact, we observed that our implementation of the block-wise Cholesky factorization outperforms the routine of the BLAS/LAPACK-package for factoring symmetric banded matrices, `DPBSV`. We suspect that this is due to the comparably small matrix dimensions, which BLAS/LAPACK is not optimized for, and additional overhead to copy the data into its banded storage scheme. A direct comparison to `MA57BD`, a widely used LDL^T sparse factorization code, is given in the following.

8.2.4.1 The oscillating masses benchmark problem

Benchmark problem 1 (BP1). In order to evaluate the performance of the solver for various problem sizes, the following MPC problem is formulated for a chain of masses

[2] A *square root* function is needed, which is available on all platforms

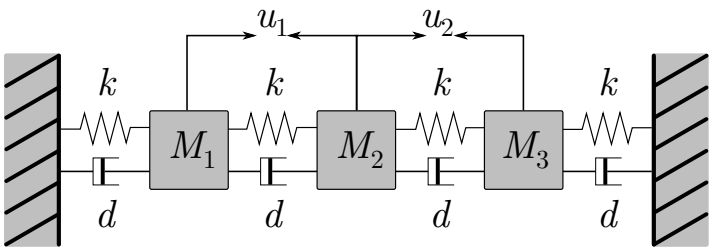


Figure 8.1: System of oscillating, interconnected masses.

interconnected with spring-dampers [135] (see Figure 8.1):

$$\begin{aligned}
&\min_{\mathbf{u}} \; V_N(x, \mathbf{u}) \triangleq V_f(x_N) + \sum_{n=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\
&\text{s.t. } x_0 = x(0)\,, \\
&\qquad x_{i+1} = A_i x_i + B_i u_i \qquad\quad i = 0, \ldots, N-1\,, \\
&\qquad -4\,\mathbf{1}_{n_x} \leq x_i \leq 4\,\mathbf{1}_{n_x} \qquad i = 0, \ldots, N\,, \\
&\qquad -0.5\,\mathbf{1}_{n_u} \leq u_i \leq 0.5\,\mathbf{1}_{n_u} \quad i = 0, \ldots, N-1\,,
\end{aligned} \tag{8.21}$$

with $R = I$, $Q = 3I$ and $V_f(x_N) \triangleq x_N^T Q x_N$. Note that this formulation does not provide stability guarantees since no terminal constraint is included. The problem size can be chosen by means of the number of masses M, with the relation $n_x = 2M$ and $n_u = M - 1$. The dynamic matrices are dense due to discretization (time step 0.5s). The problem was solved for various numbers of masses M and prediction horizons N on 3 different platforms for a test set of 100 randomly chosen feasible initial states. This benchmark problem has structure as discussed in Section 8.2.3.3 and has a total cost of $(N+1)(8/3n_x^3 + n_x^2 n_u) + N n_x^3$ flops per interior point iteration.

Benchmark problem 2 (BP2). To show the code performance for a more complex problem, we add a quadratic terminal cost $V_f(x_N) \triangleq x_N^T P x_N$ and a quadratic terminal constraint $x_N^T P x_N \leq \alpha$ to (8.21), where P solves the discrete-time Riccati equation related to LQR control and α determines the maximum level set of V_f such that no constraints are violated. In addition, we add a real-time constraint $V_N(x, \mathbf{u}) \leq \tau$ to ensure that the system is stable even in case of early termination of the solver, see [143] for details. The parameter τ is the cost of the previous solution minus a small multiple of the stage cost $x_0^T Q x_0$. To exploit the block structure in (8.1), we express $V_N(x, \mathbf{u}) \leq \tau$ by

$$x_n^T Q x_n + u_n^T R u_n \leq \gamma_n \qquad n = 0, \ldots, N-1\,, \tag{8.22a}$$

$$J_0 = \tau\,, \tag{8.22b}$$

$$J_{n+1} = J_n - \gamma_n \qquad n = 0, \ldots, N-1\,, \tag{8.22c}$$

$$J_N - x_N^T P x_N \geq 0\,. \tag{8.22d}$$

Note that this results in a QCQP with stage sizes $n_x + n_u + 2$. The rank-1 modification scheme of Section 8.2.3.2 can be used for the quadratic constraints. The problem belongs to complexity class presented in Section 8.2.3.2 for the first N blocks, the last block is of structure as discussed in Section 8.2.3.4.

8.2.4.2 Solvers and compilers

In the following, we compare the solve times of the FORCES code against those of CPLEX and CVXGEN. To assess the performance of the block-wise Cholesky factorization with respect to standard sparse LDL^T factorization, we measure the time needed for latter using the MA57 library, which is currently one of the best sparse factorization codes.

On the desktop PC and the Atom platform, we have used the *Intel C Compiler 12.0.3* with options `-O3` and, depending on the CPU architecture `-m64` for 64 bit or `-m32` for 32 bit, to compile all solvers except CPLEX 12.2, which comes in binary format. *Sourcery G++ Lite 4.5.2* was used for cross compilation for the ARM platform with flag `-O3`. We compiled MA57 (version 3.7.0) using the *Intel Fortran Compiler 12.1.0* and the *Intel Math Kernel Library* that is shipped with *Intel Composer XE 2011 SP1 7.256*. OOQP 0.99.22 was used to call MA57.

Remark 8.3. *Note that some software packages, such as* OOQP*, can be supplied with a custom linear algebra system. In these cases, the methods proposed in this chapter could be implemented in order to speed up computations.*

8.2.4.3 Code performance

The results of our numerical case study are summarized in Table 8.6 for BP1. The last column gives the computation times for BP2 when solved by our solver FORCES (denoted by RT for real-time formulation). BP2 cannot be solved by OOQP and CVXGEN due to the quadratic constraints.

For small problem sizes, the tailored solvers CVXGEN and FORCES outperform CPLEX by two orders, and MA57 by one order of magnitude in speed, although, for the latter only, we compare the factorization time vs. *full* interior point iterations. As the problem size increases, CPLEX scales very well, while code generation with CVXGEN fails for the last three problems. It can be seen that FORCES is the fastest solver out of the 4 candidates, solving the problem at least twice as quickly as CVXGEN. This gap grows with the size of the problem, as CVXGEN generates more and more code which has to be loaded into the instruction decoding unit of the CPU. A comparison of code sizes of the binary code

Table 8.5: Code size of compiled code on desktop PC.

MPC Problem				Code Size [kB]		
M	n_x	n_u	N	MA57 library	CVXGEN	FORCES
2	4	1	10	1102	318	104
2	4	1	30	1102	746	115
4	8	3	10	1102	848	134
6	12	5	10	1102	1885	156
6	12	5	30	1102	5631	155
8	16	7	20	1102	7079	185
11	22	10	10	1102	7654	127
15	30	14	10	1102	#	131
20	40	19	20	1102	#	168
30	60	29	30	1102	#	191

Unable to generate code

is given in Table 8.5, showing that FORCES is the smallest code. Consequently, for the 11-masses problem, the proposed solver is more than 5 times faster than CVXGEN, and it scales extremely well with the problem size. Furthermore, the problem formulation with superior theoretical properties (BP2) is solved more quickly by FORCES than BP1, which does not provide stability guarantees, by CVXGEN.

We have excluded qpOASES [44] from the comparison in Table 8.6 since it is not suited to solve QCQPs and warm-starting is crucial for its effectiveness, making a fair comparison between interior point and active set methods difficult in general. Using a dense formulation on the desktop PC and without warm starting, qpOASES needs on average 17.04 ms (max. 25.57 ms) per initial value to solve the 8-masses problem for $N = 20$ to optimality (qpOASES is generally faster when states are eliminated from the problem by substituting the dynamics). The average number of active set iterations is 86 (max. 125). In contrast, FORCES needs on average 11 (max. 21) interior point iterations, which take 2.87 ms (max. 10.88 ms) for the same level of optimality. This is about 2-6 times faster than a cold started qpOASES.

Figure 8.2 depicts the performance of our code, defined as the ratio of the theoretical number of floating point operations performed by the algorithm, which we determine according to the cost analysis in Section 8.2.2.1, to the execution time of the code, in percentage of the maximum machine performance (Intel Core i7: 6.4, Intel Atom Z530: 1.6 and ARM Cortex-A8: 1 Gflop/sec). Our implementation achieves with more than 35% a reasonable performance on the desktop PC, while the code is less efficient on the Intel embedded platform. Moreover, our code utilizes only a fraction of the potential performance on the ARM,

Table 8.6: Run times for 10 interior point iterations (for MA57: 10 factorizations of matrix Y (8.7)), averaged over 100 random initial states.

	MPC Problem				Optimization Problem			Runtimes [ms]				
Platform	M	n_x	n_u	N	n	p	m	CPLEX	MA57	CVXGEN	FORCES	FORCES RT
	2	4	1	10	54	44	108	10.73	1.60	0.18	0.09	0.25
	2	4	1	30	154	124	308	12.00	5.75	0.69	0.31	0.66
Desktop PC	4	8	3	10	118	88	236	10.80	5.11	0.71	0.27	0.67
	6	12	5	10	182	132	364	12.19	10.94	1.70	0.63	1.42
Intel Core i7	6	12	5	30	522	372	1044	16.20	31.45	5.90	1.95	4.15
3.2 GHz, 12 GB RAM	8	16	7	20	476	336	952	17.00	33.53	6.88	2.45	5.50
Ubuntu Linux 10.04	11	22	10	10	342	242	684	16.60	29.19	17.28	2.98	5.53
	15	30	14	10	470	330	940	23.20	59.87	#	6.37	11.63
	20	40	19	20	1220	840	2440	59.00	190.75	#	27.83	50.86
	30	60	29	30	2730	1860	5460	177.70	686.12	#	126.43	218.94
	2	4	1	10	54	44	108	*	*	1.62	0.77	2.06
	2	4	1	30	154	124	308	*	*	5.57	2.77	5.60
Embedded 1	4	8	3	10	118	88	236	*	*	5.77	2.33	5.77
	6	12	5	10	182	132	364	*	*	14.14	5.10	11.72
Intel Atom Z530	6	12	5	30	522	372	1044	*	*	48.07	15.71	35.46
1.6 GHz, 2GB RAM	8	16	7	20	476	336	952	*	*	†	19.25	46.75
Ubuntu Linux 11.04	11	22	10	10	342	242	684	*	*	†	22.83	43.03
	15	30	14	10	470	330	940	*	*	#	48.97	90.71
	20	40	19	20	1220	840	2440	*	*	#	215.08	383.67
	30	60	29	30	2730	1860	5460	*	*	#	984.96	1676.82
	2	4	1	10	54	44	108	*	*	62.09	24.61	59.73
	2	4	1	30	154	124	308	*	*	228.82	87.04	171.82
Embedded 2	4	8	3	10	118	88	236	*	*	245.65	93.58	174.68
	6	12	5	10	182	132	364	*	*	562.65	217.90	381.02
ARM Cortex A8 (TI OMAP 3530)	6	12	5	30	522	372	1044	*	*	†	662.23	1157.98
500 MHz, 256 MB RAM	8	16	7	20	476	336	952	*	*	†	853.94	2913.62
Ångström Linux 2.6.32	11	22	10	10	342	242	684	*	*	†	933.38	1461.68
	15	30	14	10	470	330	940	*	*	#	2072.76	3208.53
	20	40	19	20	1220	840	2440	*	*	#	8784.28	13728.71
	30	60	29	30	2730	1860	5460	*	*	#	39505.55	61348.73

* No running implementation † Unable to compile # Unable to generate code

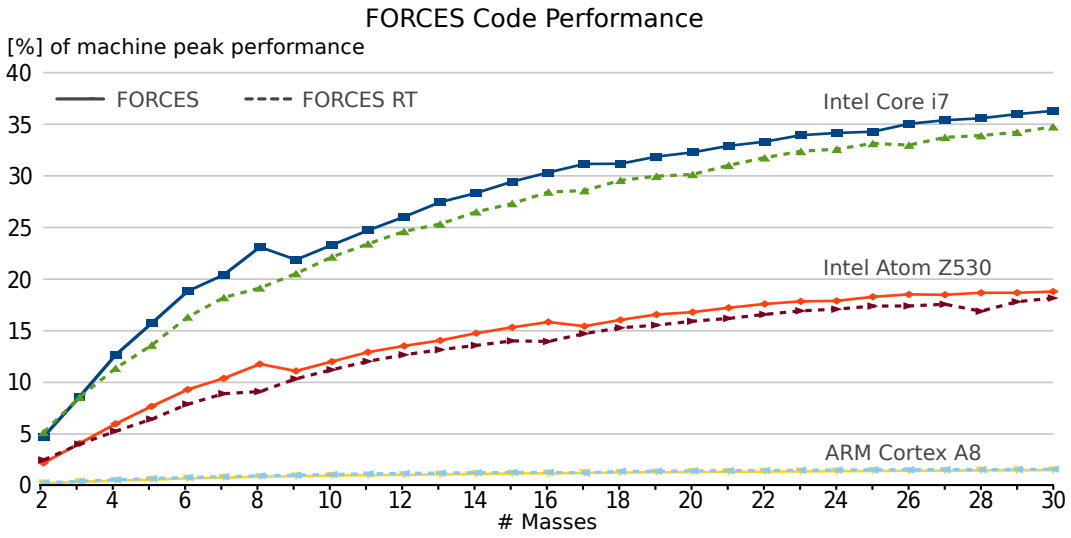


Figure 8.2: Performance plot indicating the machine efficiency of the FORCES solver.

which shows that there is a large potential for platform-dependent code optimizations.

8.3 Parallelization

It is generally understood that "the free lunch is over" [124] for accelerating existing software by making the underlying hardware faster. Existing software will no longer benefit from new generations of central processing units (CPUs) in an extent it did until a decade ago, since single-core performance is reaching a physical limit. Therefore, a fundamental change toward concurrency of algorithms is necessary. On today's high-end processors, multicore chips with four and more cores are standard, and essentially all roadmaps of the leading manufacturers of embedded computing units such as Freescale and Texas Instruments contain multicore designs (for example, today's TI C6678 digital signal processor (DSP) has eight cores). Another trend in embedded computing are field-programmable gate arrays (FPGAs) that offer an extremely high level of parallelism, but are also more difficult to program than standard CPUs. Nevertheless, many application specific designs for signal processing and embedded control have emerged and use FPGAs as their main platform for custom high speed computations.

In this section, we examine two possible approaches for accelerating the search direction computation in the multistage interior point method from Section 8.2.2. The first approach is targeted at shared memory multi-core architectures, while the second approach accelerates the matrix computations by outsourcing them to an FPGA.

8.3.1 Computational Model for Multi-Core Architectures

There are two main steps in the search direction computation in Section 8.2.2:

1. Compute Y (8.8) - 80% of total effort
2. Factor Y (8.12) - 20% of total effort

Step 2, the block Cholesky factorization (8.12), is inherently a serial operation and can therefore not be parallelized easily on a block level. On linear algebra level, parallelization is possible, but these are standard techniques and we do not discuss them here further.

In contrast, Step 1, which entails a large portion of the total computations, can be easily parallelized on a block level. The concept is illustrated by the example of a multistage problem with four stages. In this example, the matrix (8.7) that needs to be formed and factored has the structure

$$Y = \begin{bmatrix} Y_{0,0} & Y_{0,1} & & \\ Y_{0,1}^T & Y_{1,1} & Y_{1,2} & \\ & Y_{1,2}^T & Y_{2,2} & Y_{2,3} \\ & & Y_{2,3}^T & Y_{3,3} \end{bmatrix}. \tag{8.23}$$

In a serial implementation of (8.8), the following computations are carried out sequentially to compute Y (in practice, these steps would be appropriately interleaved to increase cache efficiency):

$$\begin{aligned} L_0 &= \mathrm{chol}(\Phi_0) \\ L_1 &= \mathrm{chol}(\Phi_1) \\ L_2 &= \mathrm{chol}(\Phi_2) \\ L_3 &= \mathrm{chol}(\Phi_3) \end{aligned} \tag{8.24a}$$

$$\begin{aligned} V_0 &= C_0/L_0^T \\ V_1 &= C_1/L_1^T \\ V_2 &= C_2/L_2^T \end{aligned} \tag{8.24b}$$

$$\begin{aligned} W_0 &= D_0/L_0^T \\ W_1 &= D_1/L_1^T \\ W_2 &= D_2/L_2^T \\ W_3 &= D_3/L_3^T \end{aligned} \tag{8.24c}$$

$$\begin{aligned} Y_{0,0} &= D_0\Phi_0 D_0 &&= W_0 W_0^T \\ Y_{1,1} &= C_0\Phi_0^{-1}C_0 + D_1\Phi_1 D_1 &&= V_0 V_0^T + W_1 W_1^T \\ Y_{2,2} &= C_1\Phi_1^{-1}C_1 + D_2\Phi_2 D_2 &&= V_1 V_1^T + W_2 W_2^T \\ Y_{3,3} &= C_2\Phi_2^{-1}C_2 + D_3\Phi_3 D_3 &&= V_2 V_2^T + W_3 W_3^T \end{aligned} \tag{8.24d}$$

$$\begin{aligned} Y_{0,1} &= D_0\Phi_0^{-1}C_0 &&= W_0V_0 \\ Y_{1,2} &= D_1\Phi_1^{-1}C_1 &&= W_1V_1 \\ Y_{2,3} &= D_2\Phi_2^{-1}C_2 &&= W_2V_2 \end{aligned} \tag{8.24e}$$

The first group of instructions (8.24a) are Cholesky factorizations, the second and third group (8.24b) and (8.24c) are matrix forward substitutions. Note that these computations are completely independent of each other. The fourth and fifth group, (8.24d) and (8.24e), are matrix-matrix products, which depend on the results of the first three groups, but are otherwise also completely independent of each other. It is therefore natural to parallelize the computations (8.24) by distributing (8.24a), (8.24b) and (8.24c) amongst the available cores (this is called forking), then join after all V_i and W_i are computed, then fork again to compute Y using (8.24d) and (8.24e). This concept is depicted for the simple example (8.24) in Figure 8.3.

One drawback of this approach is that a barrier is necessary in between the two parallel sections. This reduces the overall efficiency of parallelization, since synchronization among many threads is expensive. A more sophisticated parallelization approach would work with some form of a *mutex* (mutual exclusion) mechanism such as semaphores to synchronize the access to W_i and V_i.

We defer the benchmark results using multi-core processors within the multistage interior point solver to Section 8.4. Specifically, benchmarks for automatically generated solvers with built-in parallelization following the model from Figure 8.3 are presented in Section 8.4.3.

8.3.2 Linear Algebra Acceleration Using FPGAs

To date, most approaches in embedded optimization to use programmable hardware are concentrated on first-order optimization methods, see e.g. [69] for a high-speed implementation of the fast gradient method for MPC. This is a natural choice, since first order methods are very well suited for a hardware implementation, involving mostly matrix-vector products and additions that can be deeply pipelined and implemented in fixed-point arithmetic. However, there are problems for which first-order methods converge too slowly in practice, and a second-order method should be used instead.

Implementing an interior point method entirely in programmable logic is very complex, because apart from solving the linear system to compute a search direction, various helper functions such as a line search, evaluation of residuals and stopping criteria etc. are necessary. Recent progress in this area is [68], which reports on a complete FPGA implementation of a primal dual interior point method (without Mehrotra's heuristics) for solving MPC problems, with a speedup of about one order of magnitude when compared to a software implementation. The linear system that constitutes the search direction computation is solved

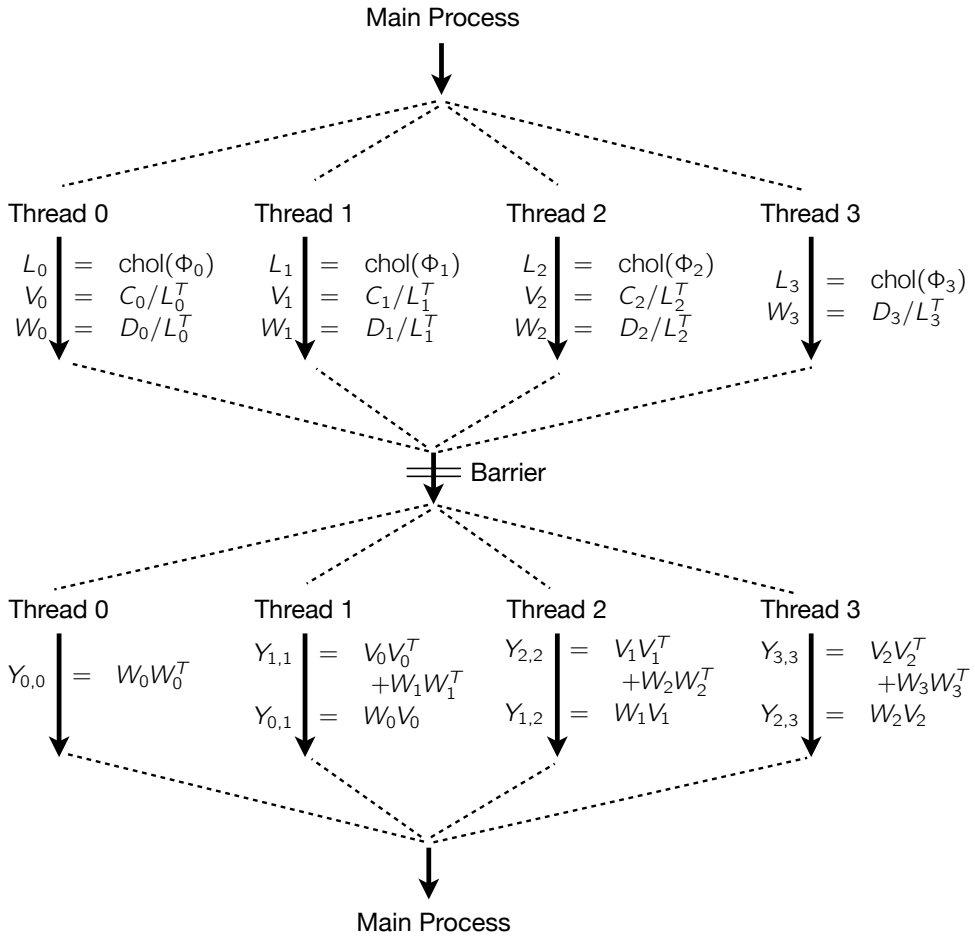


Figure 8.3: Simple concurrent computational model for shared memory multicore architectures. The barrier in between the two concurrent streams is necessary since computations in the lower part are dependent on the results of the upper part. In a more complicated model, this could be resolved by using semaphores on W_i and V_i.

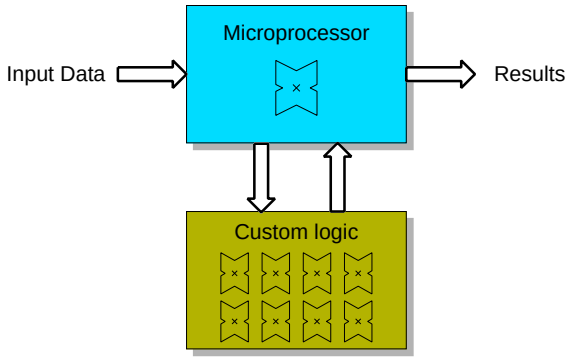


Figure 8.4: Concept of hybrid platforms with both a microcontroller unit (MCU) and programmable logic on one chip. Computationally intense tasks can be outsourced to the FPGA, while code with complex data structures, branching and irregular data access can be executed on the MCU. (From [94].)

using an iterative method, the preconditioned MINRES algorithm [114]. One drawback of iterative algorithms is that it is unclear a priori how much iterations the algorithm needs to compute an approximate solution to the linear system, and how accurate this solutions needs to be in order to ensure convergence of the interior point method. Theoretical bounds exist, but are problem dependent. Moreover, increased ill-conditioning of the Newton system for iterates close to optimality is difficult to handle for iterative solvers, which work best for a well conditioned problem. In a real-time environment with changing problem parameters, fixed-time interior point solvers are therefore difficult to obtain with iterative linear solvers, and direct methods with exactly predictable solution times are usually the preferred choice.

In this section, we put forward the idea to use emerging hybrid FPGA/microcontroller architectures to accelerate the search direction computation in Section 8.2.2 by outsourcing expensive, but regular operations such as matrix-matrix multiplications to programmable logic, while most of the complex, branching-intense operations are performed on a microcontroller unit (MCU). Figure 8.4 depicts this concept. The advantage of this approach is that, on one hand, complex multistage problems, for example with quadratic constraints on the states, can be solved using the very same steps as discussed in Section 8.2, including all structure exploitations. On the other hand, all computationally intense operations are accelerated using low-level parallelism in the FPGA. From a development point of view, hybrid FPGA/MCU architectures allow one to successively port functionality from software to hardware, while always maintaining functional code. This greatly simplifies the development process of complex architectures such as interior point solvers.

The fundamental source of speedup when using programmable logic for matrix operations is that a dot product of two vectors of size n can be computed fully in parallel with a latency of 1 cycle followed by an adder reduction tree, which has a latency of $\lceil \log_2 n \rceil$ cycles. By

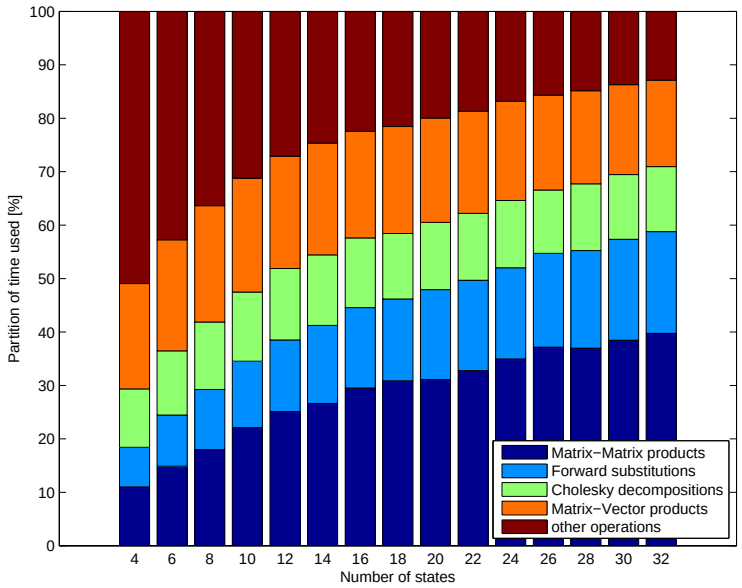


Figure 8.5: Profiling of the interior point solver from Section 8.2. Asymptotically, operations of cubic complexity such as matrix-matrix products, Cholesky factorizations, and forward substitutions amount to more than 70% of the total time. These operations can be accelerated by outsourcing them to an FPGA. (Figure from [94].)

keeping the dot product unit as busy as possible through pipelining, the complexity of matrix-matrix multiplications, Cholesky factorizations and matrix forward substitutions can be reduced to $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ that is present in software. By accelerating these types of operations, which amount to about 70 % of the total computations according to Figure 8.4, extremely energy efficient implementations of interior point solvers even for complex problems are made possible.

8.3.2.1 Implementation on Hybrid FPGA/MCU Chip

One of the most recent hybrid platforms is the Xilinx Zynq 7020 which hosts a dual-core ARM Cortex A9 core and a programmable logic on one System-on-a-Chip (SoC) design. Merkli implemented in [94] two out of the three most expensive operations, i.e. matrix-matrix multiplications and matrix forward substitutions, on a ZedBoard development board depicted in Figure 8.6. The CPU core runs at a clock rate of 666 MHz, while the synthesized logic is clocked with 100 MHz.

The computations on both the MCU and the FPGA were carried out in single floating point precision, which according to the author's experience is necessary to cover the dynamic range of the numbers arising during the interior point method. No convergence problems

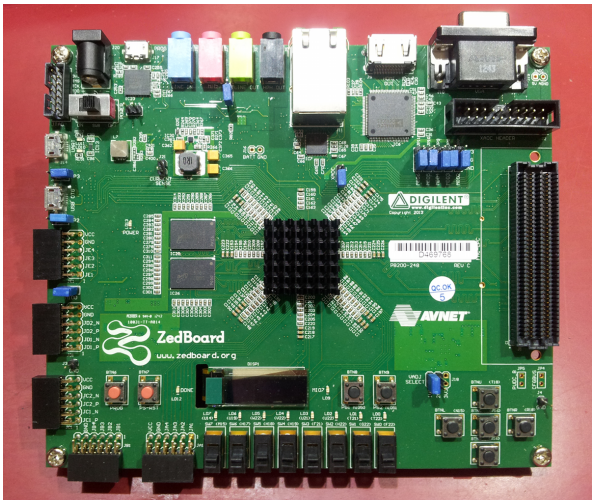


Figure 8.6: ZedBoard development board hosting the Xilinx Zynq 7020 System-on-a-Chip device with both a processor core and programmable logic on one chip. (From [94].)

were detected by switching from double to single precision number representation.

In the C code, calls to the linear algebra functions (for example a matrix-matrix multiplication) were replaced by a call to the logic. After the data transfer to the logic is accomplished, the control unit in the logic starts the computations and returns the result to the MCU as soon as it is available. Each outsourced operation was timed separately, and a theoretical cost model for both data transfer times between MCU and FPGA and the actual FPGA computation were established and calibrated with measured data.

8.3.2.2 Results

Figure 8.7 depicts the result for computing the operation $AA^T + BB^T$ for two square matrices $A, B \in \boldsymbol{R}^{n \times n}$. This operation occurs in Steps 4 and 5 of Table 8.2, and is one of the most expensive steps. It can be seen that already for small matrix sizes, computing this operation in hardware is much faster than doing so in software, with a speedup factor of ten for matrix sizes of 32×32. This matrix sizes do however not fit into the logic of the Zynq 7020 chip, so we had to resort to predictions with a cost model, extrapolating from measurements for matrix sizes up to 16×16. It can be seen that the gap between the software and hardware implementation widens as the matrix size increases, that is, much higher speedups can be expected for larger problems. This is an interesting observation, as this technique offers a means of accelerating large problems in particular (as soon as a larger chip is available). The plot for the matrix forward substitution looks qualitatively similar to Figure 8.7, although the total speedup for matrices of size 32×32 is only a factor

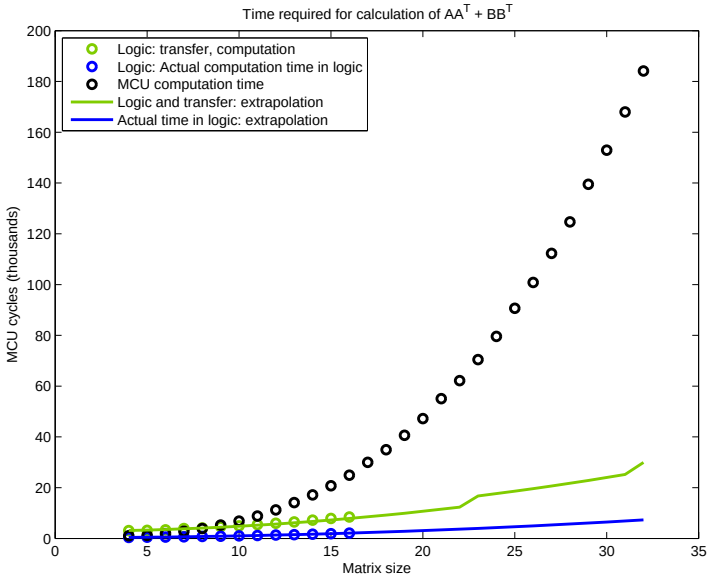


Figure 8.7: Measured and predicted computation times in MCU cycles for the operation $AA^T + BB^T$ for different matrix sizes (A and B are square matrices). For a matrix size of 32×32, the FPGA implementation is an order of magnitude faster than the software implementation. (From [94].)

of three. This stems from the fact that the forward substitution (and also the Cholesky factorization) are recursive algorithms, hence pipelining is not possible (the $n+1$st column can be processed only once the nth column has left the adder tree). However, note again that the computational times of these operations in hardware scale with $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$, so for larger problems the speedup will be even more significant.

Figure 8.8 shows predictions of the computation times for large MPC problems based on the theoretical cost model that has been calibrated with the measurements for matrix sizes up to 16×16. As the results show, outsourcing the two operations matrix-matrix multiplication and matrix forward substitution yields an estimated speedup of about five for large problems of size 64×64. As these results are based on predictions only, it will be interesting to see whether they materialize in a real-world implementation on larger FPGA chips.

8.4 Code Generation

Traditionally, numerical optimization software is designed to solve arbitrary problems from a certain problem class, for example the class of convex quadratic programs. Each time the user calls the solver, an analyze phase precedes the actual solve phase to determine the

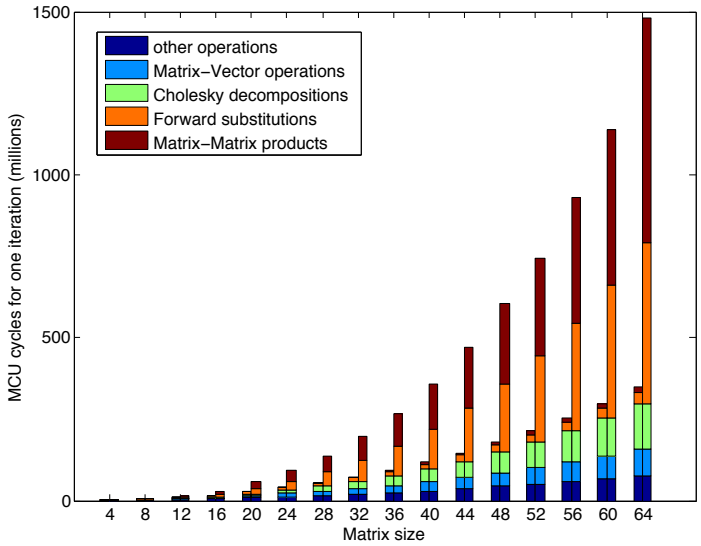


Figure 8.8: Predictions of computation times for large MPC problems based on a theoretical model that has been calibrated with measurements for matrix sizes up to 16×16. Left columns: using FPGA acceleration for matrix-matrix multiplications and for matrix forward substitution, right columns: microcontroller core only. Data transfers are excluded from the prediction model. (From [94].)

problem structure, elimination orderings etc. This traditional paradigm is necessary, because it is assumed that two consecutive problems can have a completely different structure (number of variables, number and type of constraints etc.). In this setting, many compiler performance optimizations such as loop unrolling cannot be carried out, as the problem dimensions and sparsity pattern are not known at compile time. Memory requirements are also unknown, and hence memory management systems are necessary that allow for allocating and freeing memory at runtime. Although the described situation is standard for most computer programs, it is not desirable for numerical solvers that run on embedded platforms with limited resources.

In most applications of embedded optimization such as model predictive control, each problem has the same underlying structure, and the dimensions are known at compile time. The only varying aspect from one optimization problem to the next is the data (in case of nominal MPC, for example, the initial state is the only varying parameter). This allows one to build tailored solvers that solve problems with a specific structure, but with varying data. Such automatic code generation systems for numerical optimization came into place with CVXGEN [87] in 2009, and are for example also built into the ACADO toolkit [66] since 2011. The structure analysis can be done at code generation time, and since all dimensions are known beforehand, extremely efficient compiler optimizations are possible which are

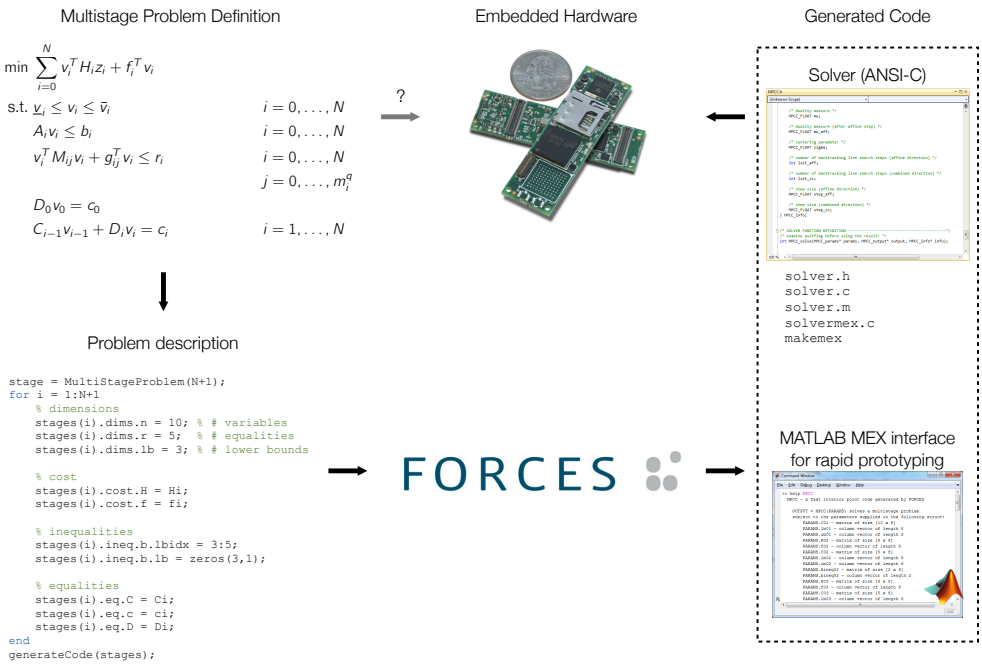


Figure 8.9: Code generation workflow with the FORCES code generation system.

automatically carried out if the code is compiled with the appropriate optimization flag such as -O3.

In this section, we present a new code generation system for multistage problems (8.1), FORCES[3], which enables one to automatically generate a primal-dual interior point solver that is specialized to the supplied problem instance [35]. Besides the general multistage structure discussed in Section 8.2.1, fine-grained structure as presented in Section 8.2.3 is detected and automatically exploited to achieve maximal code performance. The generated solvers do not rely on external libraries other than the square root function, which is available on all platforms.

The code generation workflow is depicted in Figure 8.9. The user can send a multistage problem to the FORCES server, which generates code and returns the source file of the solver in the ANSI C language. To be able to quickly test the generated solver, a Matlab MEX interface is also generated. The FORCES code generator, examples and documentation is available at `forces.ethz.ch`; some details are discussed in the following subsections.

[3]Fast Optimal Real-time Control on Embedded Systems

8.4.1 Supported Problem Class

The FORCES code generator [35] supports convex multistage quadratically constrained QPs of the form

$$\min \sum_{i=0}^{N} \frac{1}{2} v_i^T H_i v_i + f_i^T v_i \tag{8.25a}$$

$$\text{s.t. } \underline{v}_{ik(j)} \leq v_{ij}\,, \qquad j \in \mathcal{J}_i^l, i = 0, \ldots, N \tag{8.25b}$$

$$v_{ij} \leq \bar{v}_{ik(j)} \qquad j \in \mathcal{J}_i^u, i = 0, \ldots, N \tag{8.25c}$$

$$A_i v_i \leq b_i\,, \qquad i = 0, \ldots, N\,, \tag{8.25d}$$

$$v_i(\mathcal{J}_{ij}^q)^T M_{ij} v_i(\mathcal{J}_{ij}^q) + g_{ij}^T v_i(\mathcal{J}_{ij}^q) \leq r_{ij} \qquad j = 1, \ldots, m_i^q, \quad i = 0, \ldots, N \tag{8.25e}$$

$$D_0 v_0 = c_0\,, \tag{8.25f}$$

$$C_{i-1} v_{i-1} + D_i v_i = c_i\,, \qquad i = 1, \ldots, N\,, \tag{8.25g}$$

where $v_i \in \boldsymbol{R}^{n_i}$ are the (primal) stage variables and $H_i \in \boldsymbol{S}_+^{n_i}$, $f_i \in \boldsymbol{R}^{n_i}$ define the convex quadratic (or linear if $H_i = 0$) stage cost. The following constraints can be defined:

- **Upper/lower bounds**: Lower bounds (8.25b) can be defined on a subset of the elements of stage variables with indices from the index set $\mathcal{J}_i^l$, and similarly, upper bounds (8.25c) can be defined on the elements of variables with index set $\mathcal{J}_i^u$. The lower and upper bounds are given by vectors $\underline{v}_i \in \boldsymbol{R}^{|\mathcal{J}_i^l|}$ and $\bar{v}_i \in \boldsymbol{R}^{|\mathcal{J}_i^u|}$, respectively. The notation $k(j)$ denotes the index that maps element j from v_i to the index of its associated lower or upper bound. We assume that $\underline{v}_{ik(j)} < \bar{v}_{ik(j)}$ for all $j \in \mathcal{J}_i^l \cap \mathcal{J}_i^u$ (interiority assumption).
- **Affine inequalities** (8.25d) are defined by the matrix $A \in \boldsymbol{R}^{m_i^a \times n_i}$ and the vector $b \in \boldsymbol{R}^{m_i^a}$. We assume that A_i has full row rank and that the set $\{v_i \mid A_i v_i \leq b\}$ has a non-empty interior.
- **Convex quadratic inequalities** (8.25e) are defined by m_i^q matrices $M_{ij} \in \boldsymbol{S}_{++}^{|J_{ij}^q|}$, vectors $g_{ij} \in \boldsymbol{R}^{|\mathcal{J}_{ij}^q|}$ and scalars r_{ij}. The index set $\mathcal{J}_{ij}^q$ contains the indices of elements of v_i that participate in the quadratic constraint, which is denoted by the notation $v_i(\mathcal{J}_{ij}^q)$. It is assumed that the set $\{v_i \mid \text{(8.25e) holds}\}$ has a non-empty interior, which holds for example if $M_{ij} \succ 0$ and $r_{ij} > 0$.

Note that not all constraints have to be defined, as the next example shows.

Example 8.2 (Solver for Unconstrained Finite Horizon LQR)**.** *Consider the finite horizon linear quadratic regulator problem*

$$\min \sum_{i=0}^{N} x_i^T Q x_i + u_i^T R u_i + x_N^T P x_N \tag{8.26a}$$

$$s.t.\ x_0 = x \tag{8.26b}$$
$$x_{i+1} = Ax_i + Bu_i \tag{8.26c}$$

for the linear dynamic system (4.2) *and for* $Q \in \boldsymbol{S}^{n_x}_{++}$ *and* $R \in \boldsymbol{S}^{n_u}_{++}$. *To solve this problem with FORCES by converting it into the form* (8.25)*, define for example the stage variables as* $v_i = (u_i, x_i)$, $v_N = x_N$. *From this the cost terms follow:* $H_i = 2\,\mathrm{blkdiag}(R, Q)$, $f_i = 0_{n_x+n_u}$, $H_N = 2P$, $f_N = 0_{n_x}$ *and the equality constraints are defined by* $D_0 = (0_{n_u}, I_{n_x})$, $c_0 = x$, $C_i = -(A, B)$, $D_{i+1} = (0_{n_u}, I_{n_x})$ *and* $c_{i+1} = 0_{n_x}$ *where* $i = 0, \ldots, N-1$. *Note that since no inequality constraints are present, the generated solver reaches optimality in one step.*

8.4.2 Implementation Details

8.4.2.1 The FORCES Web Service

The FORCES code generation system is implemented as a web service that can be accessed over the internet through the SOAP (Simple Object Access Protocol) standard. Currently, the only available client is for Matlab. It allows one to send code generation requests directly from Matlab and to receive and to test the generated code. Other clients that can place SOAP requests are conceptually also possible, for example written in the Python programming language.

8.4.2.2 Templates

The code generator is based on specialized templates that cover all linear algebra functionality that is needed for the interior point solver, from evaluating a quadratic function to Cholesky factorization. Conceptually, the code generator implements Algorithm 2.5, with the search direction computation for multistage problems presented in Section 8.2.2 in an object oriented language. Like a stand-alone solver, it executes a predefined sequence of linear algebra functions. However, instead of carrying out actual computations, each call to a linear algebra routine generates the corresponding code by instantiating a particular ANSI-C template associated with it. By this mechanism, a linear algebra routine with fixed dimensions is created, and added to the list of already created linear algebra functions. If such a function (with the same dimensions) already exists, no additional source code is created. In this way a custom linear algebra library is built up that contains only functions needed for the specific problem. In addition to creating the linear algebra routines, the "virtual" function call is mapped to a corresponding line of code in the generated code. As a result, the main solver routine is simply a sequence of calls into the customized linear algebra library.

The different cases presented in Section 8.2.3 and their combinations are easily handled by exploiting polymorphism and operator overloading. Depending on the type of the argument, different linear algebra templates are used. For example, a matrix-matrix multiplication $C = AB$ can be generated by calling the function `C=MMM(A, B)`, which will then behave differently depending on the type of A and B. In this simple example, A and B could be diagonal, lower triangular and dense matrices, which gives 9 different cases to be covered by templates for multiplying matrices. The FORCES code generator system has currently 131 different linear algebra templates in place, all of which are variations of the general case, and are thus easy to maintain. Most templates are functions of only a few lines of C code.

Because only textual substitutions are performed during code generation (to fix the dimensions of the linear algebra building blocks), code generation time is below a second even for large problems. Also, the code generator is not limited by the problem size, as this does not change the size of the code that is emitted. In fact, most of the code generation time is spent transferring the data to the FORCES server and downloading the generated code.

8.4.2.3 Data Compression

Prior to code generation, the specific matrices that are fixed for all problem instances are stored, and all arrays needed for the interior point method are defined. During this "virtual" memory allocation process, it is checked whether two matrices coincide, in which case the numerical values are stored only once. This data compression can be significant for MPC problems with long horizons, as typically the matrices Q, R, P, A, B do not vary over the horizon and thus need to be stored only once. Standard solvers store and handle large matrices that contain N times more data than actually necessary to define a multistage problem with repeating data.

8.4.2.4 Parametric Problems

The FORCES code generator has strong support for parametric problems: Any of the problem-defining matrices or vectors $H_i, f_i, \underline{v}_i, \bar{v}_i, A_i, b_i, c_i, C_i, D_i, M_{ij}, g_{ij}, r_{ij}$ in (8.25) can be defined as a parameter. Parameters have to be supplied by the user when calling the solver. The header file, the generated MEX interface in Matlab and the corresponding help file contain the list of parameters for further reference. For more details, see [35].

8.4.2.5 Multi-Core Parallelization

The FORCES code generation system supports shared-memory multi-core architectures by automatically parallelizing the generated code according to Figure 8.3. In order to generate platform independent code, the parallelization is achieved through embedding OpenMP pragmas into the code. OpenMP is an open application programming interface (API) for shared memory multiprocessing. The standard is supported by a large number of platforms and operating systems, including embedded platforms such as the TI C6678 with eight DSP cores. If a compiler does not support OpenMP, the pragmas are ignored and the compiler creates a single-threaded binary instead.

8.4.3 Benchmark Results for Generated Code

In this section, we present benchmarks of solvers generated by the FORCES code generator for the MPC problem (8.21) from Section 8.2.4.1. The goal is to bring a chain of oscillating masses (cf. Figure 8.1) to halt, and is formulated as a simple MPC regulation problem with diagonal cost matrices and lower and upper bounds on both states and inputs. The horizon is $N = 9$, which results in a multistage problem with ten stages.

We measure the total solve times for 100 randomly sampled initial states for different levels of structure exploitation and parallelization and compare this to the IBM CPLEX solver version 12.2 (which uses multithreading). The benchmarks were executed on a PC with an Intel Core i5 CPU running at $3.1\,\text{GHz}$ with $8\,\text{GB}$ of RAM. The operating system was Microsoft Windows 7, and the Visual Studio 2010 C++ compiler was used to compile the generated FORCES code with compiler optimizations turned on (option `-O`).

Figure 8.10 shows the resulting computation times for solving the MPC problem (8.21) for 2 to 30 masses. For M masses, the system has $n_x = 2M$ states and $n_u = M - 1$ inputs, as in Section 8.2.4.1. The generated code matches the performance of the hand-written code from Section 8.2 when comparing the computation times to those in Table 8.6. Note that the latter are times for ten interior point iterations, while the figures in Figure 8.10 are average solve times to an accuracy of 10^{-6}. Using the general multistage search direction computation from Section 8.2.2, the single-threaded FORCES solver is faster than the multithreaded CPLEX solver for a problem size of up to 40 states. Using multithreading as described in Section 8.3.1 and Section 8.4.2.5, the general multistage solver is competitive with CPLEX even for large problems up to 60 states or more. When, exploiting the fine grained structure from Section 8.2.3 in addition to the multistage structure, even the single core FORCES solver is faster than CPLEX, by up to almost two orders of magnitude for the smallest, by about one order of magnitude for medium-sized and by a factor of two for large problems. Figure 8.11 depicts the actual speedups of the FORCES solvers with respect to

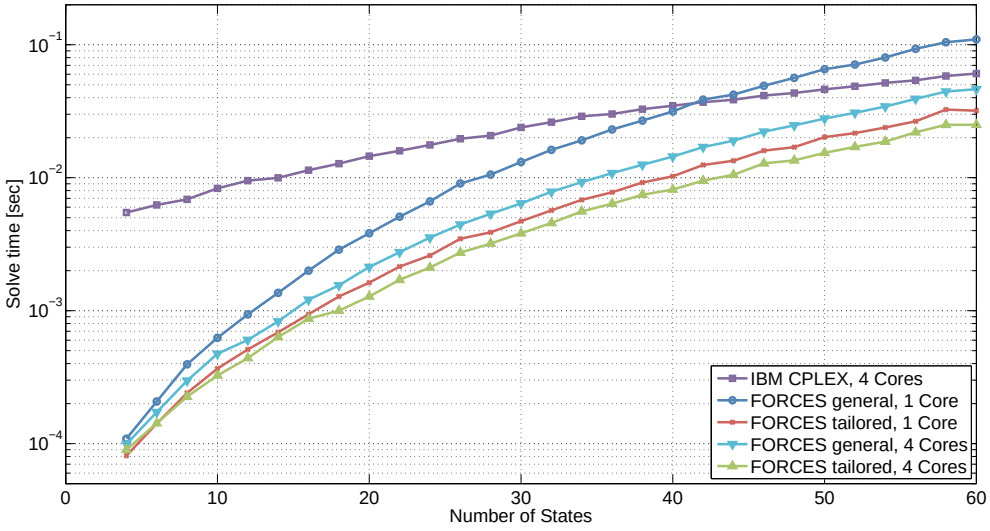


Figure 8.10: Timings for solving the MPC benchmark problem (8.21) from Section 8.2.4.1 with solvers generated by the FORCES code generator for different levels of structure exploitation and parallelism. "FORCES general" is the general multistage approach from Section 8.2.2, whereas "FORCES tailored" exploits also fine-grained structure in the multistage problem as described in Section 8.2.3. The computation times are averaged over 100 randomly sampled initial states. Measured on a PC with Intel Core i5 CPU at 3.1 GHz.

IBM CPLEX.

Interestingly, multicore parallelization does not have a large effect on the tailored solvers (i.e. when the fine-grained structure from Section 8.2.3 is exploited) for the problem sizes considered. Note however that benchmark problem (8.21) is the cheapest case in terms of computational cost, and that a theoretical speedup of about nine (when compared to the general multistage problem) can be achieved solely by structure exploitation, as concluded from the theoretical cost analysis in Section 8.2 and as shown in Table 8.4. Therefore, the less structure can be exploited by the techniques in Section 8.2.3, the more speedup is achieved by parallelizing the computations. The general multistage problem benefits most from parallelization, since the workload per core is substantial and outweighs the overhead of parallelization (starting and stopping threads, synchronization etc.). One can also clearly see that the gap between parallelized and unparallelized code widens as the problem size increases, which underlines the aforementioned observation. To conclude, parallelization is most effective for large problems that have little structure beyond the multistage one.

We conclude this section with a remark on the code size. The compiled MEX file as generated by FORCES has a size of 52 kB, and it is almost independent of the problem size.

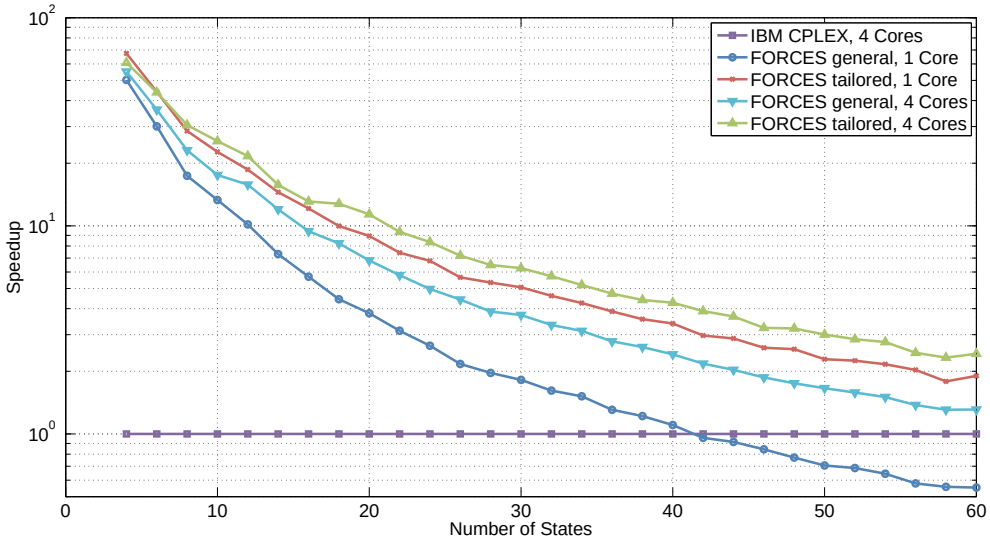


Figure 8.11: Speedup of the FORCES solvers compared to IBM CPLEX (calculated from the timing data shown in Figure 8.10).

The CPLEX binary is also independent of the problem size, but has a size of 11′700 kB. This shows that FORCES generates solvers of extremely small size that should fit on all embedded devices *as is*, while CPLEX requires in addition a runtime environment, a memory manager, support for dynamically linked libraries etc. This makes the FORCES code generation system a unique tool that generates fast, lightweight and library-free code.

8.5 Application Examples

In the following, we present application examples of the FORCES code generation system that show its flexibility on one hand, and the efficiency of the generated code on the other hand.

8.5.1 Robust Real-time MPC for Setpoint Tracking

The first example is an advanced MPC formulation by Zeilinger [144] that allows terminating the solver after *any* number of interior point iterations while guaranteeing stability of the closed loop system despite disturbances. This allows one to satisfy strict real-time constraints without sacrificing system-theoretic guarantees on the closed-loop behavior. The formulation combines ideas from tube-based robust MPC [90] with that of reference tracking of piecewise constant references [79] to guarantee ISS stability of the closed loop system.

To allow for arbitrary real-time constraints for the computation, a Lyapunov constraint is added to the MPC problem. With a suitable warm-starting technique, it can be shown that (recursive) feasibility can be established in a fixed amount of time, and the optimization then only improves on the solution in terms of performance, but closed-loop stability is guaranteed even without any optimization.

8.5.1.1 General Problem Formulation

Consider the linear dynamical system (4.2) subject to disturbances,

$$x^+ = Ax + Bu + w \tag{8.27}$$

where $w \in \mathbb{W}$ is a disturbance from a convex and compact set $\mathbb{W}$ containing the origin in its interior. The goal is to bring the system state and input (x, u) to a desired steady state setpoint (x_r, u_r) while satisfying state constraints $x \in \mathbb{X}$ and input constraints $u \in \mathbb{U}$ for all times and for all disturbances $w \in \mathbb{W}$. Define

$$V_N^{tr}(x, \mathbf{x}, \mathbf{u}, x_s, u_s, x_r, u_r) \triangleq V_f(x_N - x_s) + \sum_{i=0}^{N-1} l(x_i - x_s, u_i - u_s) + V_o(x_s - x_r, u_s - u_r)$$

for positive definite functions $l : \boldsymbol{R}^{n_x \times n_u} \to \boldsymbol{R}_+$, $V_f : \boldsymbol{R}^{n_x} \to \boldsymbol{R}_+$, $V_o : \boldsymbol{R}^{n_x \times n_u} \to \boldsymbol{R}_+$. The robust real-time tracking MPC problem is [144, Problem $\mathbb{P}_N^{\tau,tr}$ (8)]:

$$\begin{aligned}
\min\ & V_N^{tr}(x, \mathbf{x}, \mathbf{u}, x_s, u_s, x_r, u_r) + V_f(x - x_0) && \text{(8.28a)}\\
\text{s.t. } & x \in x_0 \oplus \mathcal{Z}_w && \text{(8.28b)}\\
& x_{i+1} = Ax_i + Bu_i \quad i = 0, \dots, N-1 && \text{(8.28c)}\\
& (x_i, u_i) \in \tilde{\mathbb{X}} \times \tilde{\mathbb{U}} \quad i = 0, \dots, N-1 && \text{(8.28d)}\\
& (x_s, u_s) \in \tilde{\mathbb{X}} \times \tilde{\mathbb{U}} && \text{(8.28e)}\\
& x_s = Ax_s + Bu_s && \text{(8.28f)}\\
& x_N \in \mathbb{X}_f^{tr}(x_s, u_s) && \text{(8.28g)}\\
& V_N^{tr}(x, \mathbf{x}, \mathbf{u}, x_s, u_s, x_r, u_r) + V_f(x_{nom} - x_0) \leq \Pi_{prev}^{tr} && \text{(8.28h)}
\end{aligned}$$

where x is the current state of the system (8.27), $\mathbf{x} \triangleq (x_0, \dots, x_N)$ and $\mathbf{u} \triangleq (u_0, \dots, u_{N-1})$ are the state and input trajectories, $\mathcal{Z}_w$ is a robust positively invariant (RPI) set for (8.27) (cf. [90]) and the set $\mathbb{X}_f^{tr}(x_s, u_s)$ is a PI set for tracking [79]. The scalar Π_{prev}^{tr} is defined as the value of the objective (8.28a) from the previous time step minus a small multiple of the stage cost [144, Definition 5.2]. The vector x_{nom} is the (simulated) state of the nominal system (4.2) without noise when applying the tube MPC control law

$$\kappa(x) = K(x - x_0^\star) + u_0^\star \tag{8.29}$$

(with x the *current* state). The control law (8.29) applied to system (8.27) in a receding horizon fashion,

$$x^+ = Ax + B\kappa(x) + w\,, \tag{8.30}$$

ensures convergence in an ISS sense to a region around the closest admissible steady state despite noise and early termination of the solver [144, Theorem 5.8], if a feasible optimization method is used and the solver is warm-started according to a particular scheme [144, Algorithm 2].

In what follows, we apply (8.29) to the masses benchmark problem from Section 8.2.4.1 by solving problem (8.28) with a solver generated by FORCES [35].

8.5.1.2 Particular Problem Instance

Disturbance Set

We use a norm-bounded disturbance set,

$$\mathbb{W} \triangleq \{w \in \boldsymbol{R}^{n_x} \mid \|w\|_2 \leq r\}\,. \tag{8.31}$$

Parametrization of steady states

In the following, we will use the steady-state parametrization from Appendix B.1:

$$x_s = M_x\theta \tag{8.32a}$$
$$u_s = M_u\theta \tag{8.32b}$$

for appropriately define matrices M_x and M_u, and where $\theta \in \mathbb{R}^{n_\theta}$ and typically $n_\theta \ll n_x$. This reduces the number of variables per stage significantly in most problems; as a result, the problem can be solved more efficiently.

Cost Functions

In our implementation example, we use the functions

$$l(x,u) \triangleq \frac{1}{2}x^TQx + \frac{1}{2}u^TRu \text{ with } Q \succ 0, R \succ 0, \tag{8.33a}$$
$$V_f(x) \triangleq \frac{1}{2}x^TPx \text{ with } P \succ 0 \text{ and} \tag{8.33b}$$
$$V_o(x_s,u_s) \equiv V_o(\theta) \triangleq \frac{1}{2}(\theta-\theta_r)T_\theta(\theta-\theta_r), \quad T_\theta \succ 0. \tag{8.33c}$$

Constraint Sets

For simplicity, we choose the state and input constraints sets as simple bound constraints. Note that for using the robust tube-based MPC from [90], the constraints have to be tightened:

$$\tilde{\mathbb{X}} \triangleq \{x \mid \underline{x} \leq x \leq \bar{x}\} \ominus \mathbb{W}\,, \tag{8.34}$$
$$\tilde{\mathbb{U}} \triangleq \{u \mid \underline{u} \leq u \leq \bar{u}\} \ominus K\mathbb{W}\,, \tag{8.35}$$

where $\underline{x}, \bar{x}$ and $\underline{u}, \bar{u}$ are the lower and upper bounds on the states and inputs, respectively. The initial tube set and the terminal target set are chosen to be ellipsoidal,

$$(8.28b) \Leftrightarrow (x - x_0)^T T (x - x_0) \leq 1 \text{ with } T \succ 0, \tag{8.36a}$$
$$(8.28g) \Leftrightarrow \begin{bmatrix} x_N \\ \theta \end{bmatrix}^T \underbrace{\begin{bmatrix} P_{E,x} & -P_{E,x}M_x \\ -M_x^T P_{E,x} & P_{E,\theta} + M_x^T P_{E,x} M_x \end{bmatrix}}_{P_E} \begin{bmatrix} x_N \\ \theta \end{bmatrix} \leq 1 \tag{8.36b}$$

for positive definite matrices $P_{E,x}, P_{E,\theta}$ that define the terminal set for tracking $\mathbb{X}_f^{tr}$. With these choices and the definition $\|x\|_P^2 \triangleq x^T P x$, the problem instance of (8.28) that we are interested in solving is

$$\min \frac{1}{2}\|x_N - M_x\theta\|_P^2 + \sum_{i=0}^{N-1} \frac{1}{2}\|x_i - M_x\theta\|_Q^2 + \frac{1}{2}\|u_i - M_u\theta\|_R^2 + \frac{1}{2}\|\theta - \theta_r\|_{T_\theta}^2 + \frac{1}{2}\|x - x_0\|_P^2 \tag{8.37a}$$
$$\text{s.t. } \|x - x_0\|_T^2 \leq 1 \tag{8.37b}$$
$$x_{i+1} = Ax_i + Bu_i, \quad i = 0, \ldots, N-1 \tag{8.37c}$$
$$(x_i, u_i) \in \tilde{\mathbb{X}} \times \tilde{\mathbb{U}} \quad i = 0, \ldots, N-1 \tag{8.37d}$$
$$(M_x\theta, M_u\theta) \in \tilde{\mathbb{X}} \times \tilde{\mathbb{U}} \tag{8.37e}$$
$$\|(x_N, \theta)\|_{P_E}^2 \leq 1 \tag{8.37f}$$
$$\frac{1}{2}\|x_N - M_x\theta\|_P^2 + \sum_{i=0}^{N-1} \frac{1}{2}\|x_i - M_x\theta\|_Q^2 + \frac{1}{2}\|u_i - M_u\theta\|_R^2 + \frac{1}{2}\|\theta - \theta_r\|_{T_\theta}^2 + \frac{1}{2}\|x - x_0\|_P^2 \leq \Pi \tag{8.37g, 8.37h}$$

with optimization variables $(x_0, \ldots, x_N, u_0, \ldots, u_{N-1}, \theta)$ and appropriate positive definite matrices $T, P, Q, R, T_\theta, P_E$.

8.5.1.3 Conversion to Multistage Problem

When converting (8.37) to a multistage problem of the form (8.25), it is necessary to introduce N copies of the variable θ since it enters at each stage into the cost function and the constraints. Moreover, the Lyapunov constraint (8.28h) couples all stages. To retain the multistage structure, we introduce 3 additional variables δ_i, γ_i and J_i per stage (as in Section 8.2.4.1). Problem (8.37) can then be rewritten as (symbols marked red are parameters of the problem)

$$\min \quad \gamma_N + \sum_{i=0}^{N-1} \gamma_i + \delta_i \tag{8.38a}$$

$$\text{s.t.} \quad J_0 = \Pi \tag{8.38b}$$

$$x_{i+1} = Ax_i + Bu_i, \quad i = 0, \dots, N-1 \tag{8.38c}$$

$$\theta_{i+1} = \theta_i \quad i = 0, \dots, N-1 \tag{8.38d}$$

$$J_{i+1} = J_i - \gamma_i - \delta_i, \quad i = 0, \dots, N-1 \tag{8.38e}$$

$$x_0^T T x_0 - 2x^T T x_0 \leq 1 - x^T T x \tag{8.38f}$$

$$\underline{x} \leq x_i \leq \bar{x}, \quad \underline{u} \leq u_i \leq \bar{u} \quad i = 0, \dots, N-1 \tag{8.38g}$$

$$(M_x, M_u, -M_x, -M_u)\theta_N \leq (\bar{x}, \bar{u}, -\underline{x}, -\underline{u}) \tag{8.38h}$$

$$\frac{1}{2}\begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix}^T \begin{bmatrix} Q+P & -QM_x \\ -M_x^T Q & M_x^T Q M_x + T_\theta \end{bmatrix} \begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix} + \begin{bmatrix} -x^T P \\ -\theta_r^T T_\theta \end{bmatrix}^T \begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix} - \gamma_0 \leq -\frac{1}{2}x^T P x - \frac{1}{2}\theta_r^T T_\theta \theta_r \tag{8.38i}$$

$$\frac{1}{2}\begin{bmatrix} x_i \\ \theta_i \end{bmatrix}^T \begin{bmatrix} Q & -QM_x \\ -M_x^T Q & M_x^T Q M_x \end{bmatrix} \begin{bmatrix} x_i \\ \theta_i \end{bmatrix} - \gamma_i \leq 0 \quad i = 1, \dots, N-1 \tag{8.38j}$$

$$\frac{1}{2}\begin{bmatrix} \theta_i \\ u_i \end{bmatrix}^T \begin{bmatrix} M_u^T R M_u & -M_u^T R \\ -R M_u & R \end{bmatrix} \begin{bmatrix} \theta_i \\ u_i \end{bmatrix} - \delta_i \leq 0 \quad i = 0, \dots, N-1 \tag{8.38k}$$

$$\frac{1}{2}\begin{bmatrix} x_N \\ \theta_N \end{bmatrix}^T \begin{bmatrix} P & -PM_x \\ -M_x^T P & M_x^T P M_x \end{bmatrix} \begin{bmatrix} x_N \\ \theta_N \end{bmatrix} - \gamma_N \leq 0 \tag{8.38l}$$

$$\begin{bmatrix} x_N \\ \theta_N \end{bmatrix}^T P_E \begin{bmatrix} x_N \\ \theta_N \end{bmatrix} \leq 1 \tag{8.38m}$$

$$\gamma_N - J_N \leq 0 \tag{8.38n}$$

Stage Variables

We use the following stage variables:

$$v_i \triangleq (x_i, \theta_i, J_i, u_i, \gamma_i, \delta_i) \in \boldsymbol{R}^{n_x+n_u+n_\theta+3}, \ \ i = 0, \ldots, N-1 \tag{8.39a}$$

$$v_N \triangleq (x_N, \theta_N, J_N, \gamma_N) \in \boldsymbol{R}^{n_x+n_\theta+2} \tag{8.39b}$$

The particular multistage matrices for the FORCES code generator follow now immediately from this definition and (8.38) and are given in Appendix B.2.

8.5.1.4 Numerical Results

In the following, we apply the real-time robust tracking approach to the system of three, six and nine oscillating masses from Section 8.2.4.1 to demonstrate the closed-loop behavior of the advanced MPC formulation (8.28), and to show that the FORCES code generator emits very efficient code even for complex multistage problems with many quadratic constraints.

First, consider the system of three masses from Figure 8.1, with $k = 0.3\,\text{Nm}$ and $d = 0.1\,\text{Ns/m}$. The inputs are constrained to lie in $\pm 1\,\text{N}$ and the displacement of the masses in $\pm 4\,\text{m}$. The horizon length is chosen as $N = 10$ and in closed-loop a worst-case disturbance with $\|w\|_2 = 0.02\,\text{N}$ is acting on the system. At the beginning of the simulation, the tracking reference is a displacement of the second mass of $-0.9\,\text{m}$ (and no displacement for the first and third mass). At time step $k = 50$, a step change to $+0.9\,\text{m}$ in the reference position of the second mass is applied. Figure 8.12 shows the simulated closed-loop trajectories of the position of the second mass, the corresponding steady-state, the second input and the value of the Lyapunov function for different starting strategies and number of optimization steps. We compare the closed-loop performance of applying the warm-start solution without any optimization and with two and five interior-point steps, respectively, to a cold-start solution with two iterations and the optimal solution. The simulation illustrates the role of the artificial reference. While directly using the new reference of $+0.9\,\text{m}$ starting from position $-0.9\,\text{m}$ would be infeasible, the artificial reference is chosen to ensure recursive feasibility and is then steered towards the real reference.

The results demonstrate that only applying the warm-start solution already provides convergence to the desired reference, even after the step change. This is enabled by the special warm-start procedure Algorithm [144, Algorithm 2], which includes a warm-start of the artificial steady-state, moving it towards the desired target. This is particularly evident after the step change, where the artificial reference is moved to $-0.1\,\text{m}$. The Lyapunov function decreases monotonically proving asymptotic convergence of all masses to the desired position. Note that the increase in the Lyapunov function at $k = 50$ is caused by the reference change, which is enabled by resetting the Lyapunov constraint. Since the

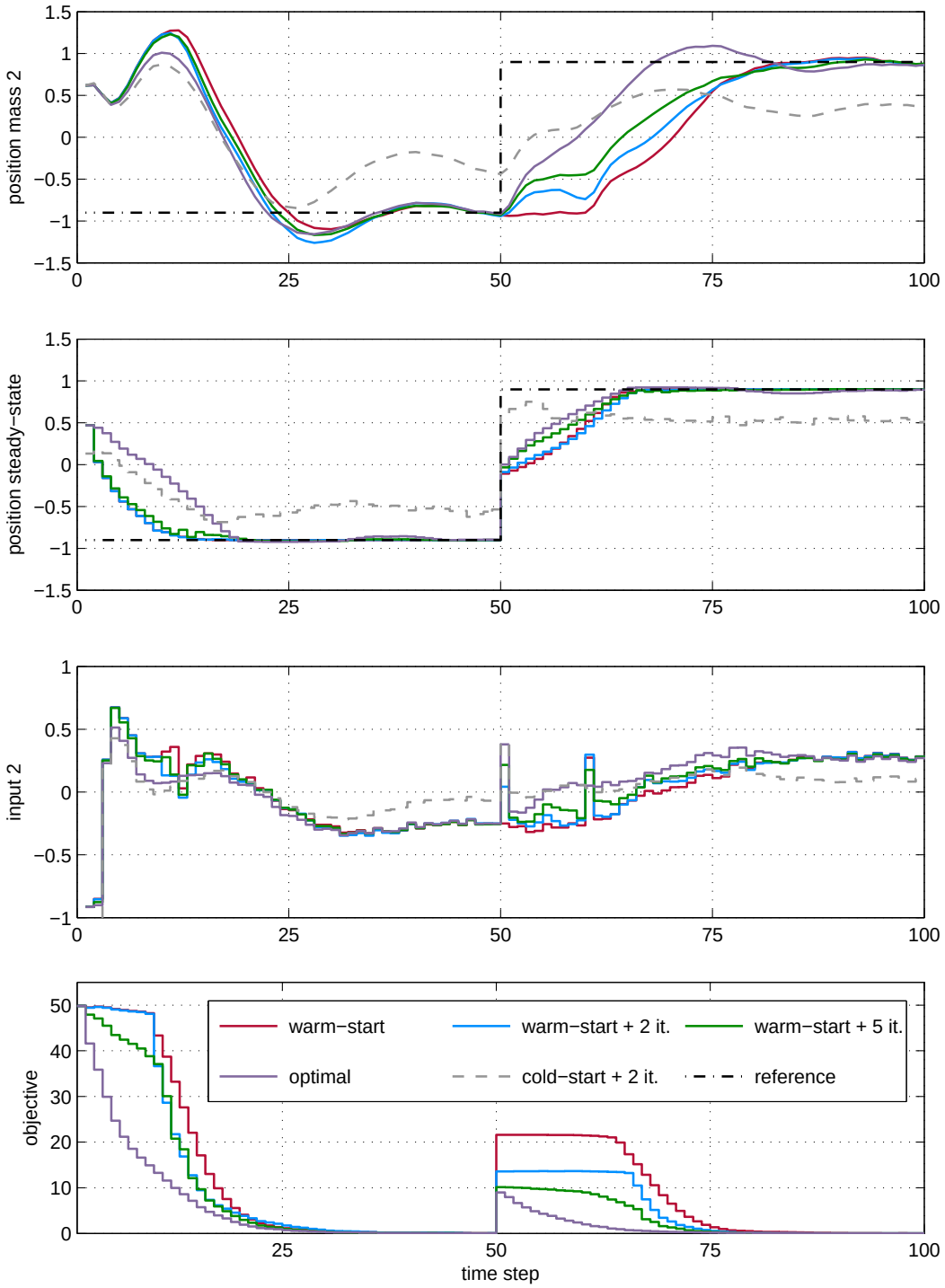


Figure 8.12: Comparison of the closed-loop performance using the warm-start solution, the suboptimal solution after 2 and 5 online iterations (with warm-start), the suboptimal solution after 2 iterations using cold-start from the origin, and the optimal solution.

#masses	n	m	min [μs]	mean [μs]	max [μs]
3	6	2	53.95	55.90	58.52
6	12	5	189.75	195.77	231.83
9	18	8	479.63	498.74	544.68

Table 8.7: Computation times for one interior-point iteration solving the tracking problem (8.37) for a horizon length of $N = 10$ on a MacBook Pro with Intel Core i7 CPU at 2.6 GHz.

warm-start is given by the shifted input sequence, a delay of $N - 1$ steps occurs after the artificial reference is moved at $k = 50$, until the control inputs react to the reference change. This behavior could be improved by small adaptations in the warm-start procedure, but the effect is automatically mitigated by performing a few optimization steps. This can be seen in Figure 8.12, where the control input for only two optimization steps already reacts at $k = 50$.

Comparing the performance of the warm-start without optimization with two and five online iterations, we can see that the performance gradually improves with the number of performed iterations. The closed-loop performance deterioration with respect to the optimal closed-loop trajectory applying the warm-start without optimization is 16.60%, for two iterations it is 15.58% and for five iterations 5.61%. This shows that only five online optimization steps already provide very good performance. The cold-start method with two online iterations, in contrast, results in an input constraint violation at time step $k = 1$ and is not able to track the desired reference. Note that the cost function cannot be compared in this case, since the resulting solutions are infeasible for the constraints of the optimization problem.

We investigate the computation times for the tracking method by recording the time to perform one iteration of the online optimization. The minimum, average and maximum times are given in Table 8.7, demonstrating that the real-time tracking method can be implemented for sampling times below milliseconds using code generated by FORCES. For example, for the three masses system, five iterations can be computed in less than 300 μs.

8.5.2 Automatic Racing of 1:43 Scale RC Race Cars

One real-world application of solvers generated by the FORCES code generation system is automatic racing of 1:43 scale RC race cars. The goal is to automatically steer the cars around a race track as fast as possible, while staying on the track and avoiding/overtaking other cars on the way. The Kyosho dnano race cars came into their own in 2008, and can move at very high speeds up to 40 km/h (25 miles per hour).

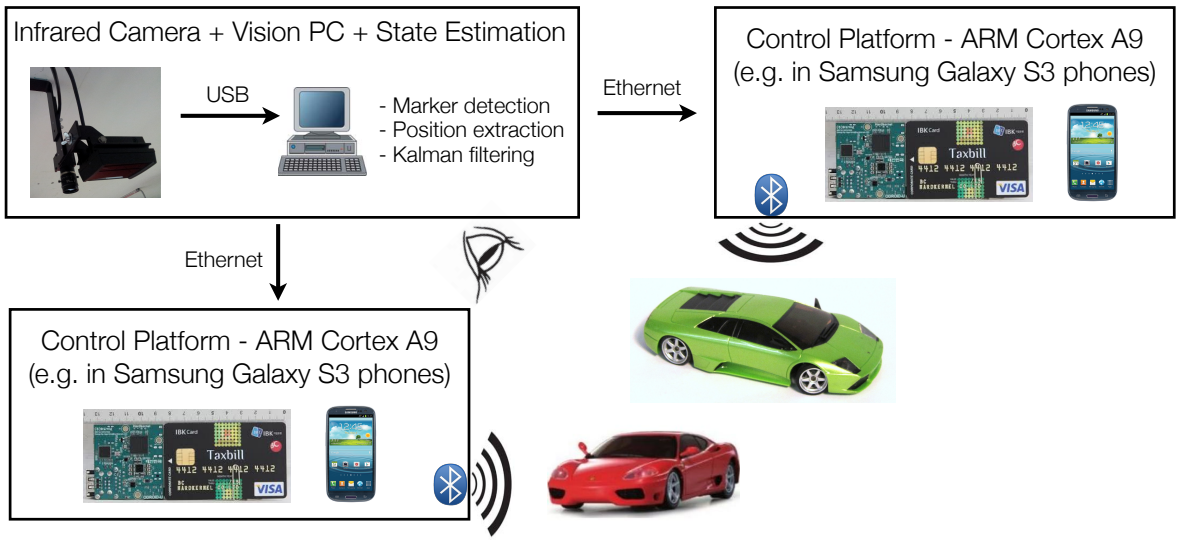


Figure 8.13: Overview of autonomous race car control loop.

8.5.2.1 Control Loop Overview

The overall control loop is depicted in Figure 8.13. A PointGrey One Flea camera equipped with an infrared lens and mounted above the race track captures infrared light reflected by markers on the cars at a rate of 100 Hz and a resolution of 1280×1024 pixels. Each car has a certain marker configuration, which encodes the orientation of the car and its unique car ID to distinguish different cars. A vision processing unit based on the OpenCV library [20] processes the image, and runs a Kalman filter for state estimation. The state estimates are sent into a network via Ethernet, where clients can attach and read out the state estimates for each car. The controller nodes then compute setpoints for the duty cycle of the DC traction motor and for the steering angle, which are then sent via Bluetooth to the corresponding car. An embedded platform on the car, which replaces Kyosho's proprietary hardware, executes low level controllers for tracking the communicated steering and velocity setpoints. A picture of the race track is shown in Figure 8.14.

8.5.2.2 Control Concept

There are many approaches for the given control task, for example model predictive contouring control [78] could be used to directly compute a trajectory and inputs at the same time. In this section, we focus on the hierarchical approach proposed by Liniger [80]. The controller is split into two parts: a trajectory planner, which generates a feasible trajectory that maximizes progress on the track, and a trajectory tracking MPC controller that is implemented using a solver generated by the FORCES code generation system. We explain

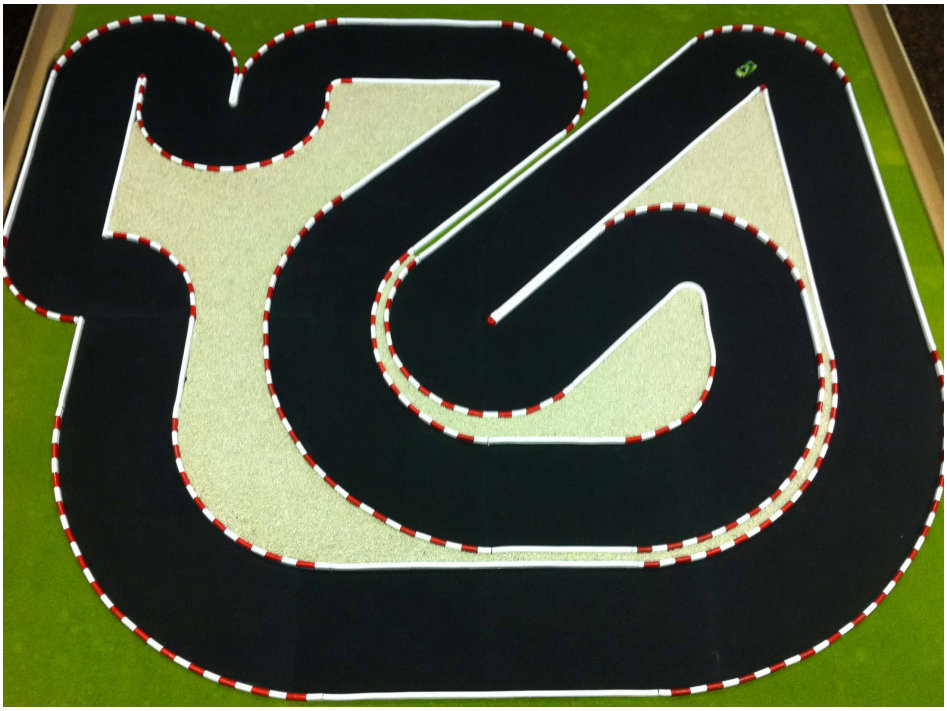


Figure 8.14: Custom built race track for 1:43 scale race cars.

them in the following.

8.5.2.3 Path Planning

The trajectory planner exploits stationary cornering conditions of the nonlinear car model to generate trajectories where all accelerations are zero. The employed nonlinear car model in [80] is a simplified bicycle model with lateral slip and nonlinear tire forces modelled by a simplified version of Pacejka's magic formula. Figure 8.15 illustrates the concept of the path planner; see [80] for all details. In particular, note that at each sampling time a new trajectory and new constraints for the MPC controller are generated.

8.5.2.4 MPC Tracking Controller

After selecting a trajectory to follow, the nonlinear model is linearized around the trajectory and a reference tracking MPC formulation is used to determine the optimal control inputs to the system:

$$\begin{aligned}\min \sum_{i=0}^{N-1} &(x_i - x_i^{\text{ref}})^T Q (x_i - x_i^{\text{ref}}) + (u_i - u_i^{\text{ref}})^T R (u_i - u_i^{\text{ref}}) \\ &+ (x_N - x_N^{\text{ref}})^T P (x_N - x_N^{\text{ref}}) \end{aligned} \tag{8.40a}$$

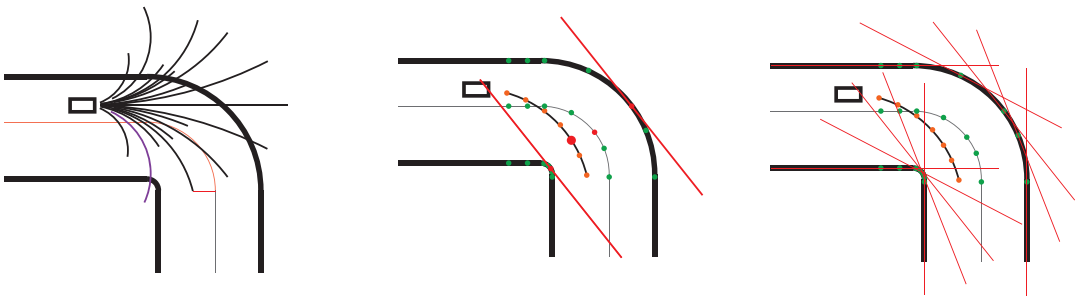


Figure 8.15: Overview of path planning mechanism proposed in [80]. Left: using a nonlinear model of the car, trajectories with zero acceleration are found by gridding the stationary cornering conditions. A feasible trajectory with the largest progress on the track is selected for tracking. Middle: for the selected trajectory, the dots represent the predicted positions at the sampling times. For each sampling time, two linear inequalities are generated to describe the constraint for staying on the track. Right: the direct product of all generated corridors gives the feasible set for the MPC tracking controller. (From [80].)

$$\text{s.t. } x_0 = x \tag{8.40b}$$
$$x_{i+1} = A_i x_i + B_i u_i + g_i \qquad i = 0, \ldots, N-1 \tag{8.40c}$$
$$G_i x_i \leq h_i \qquad i = 0, \ldots, N \tag{8.40d}$$
$$(x_i, u_i) \in \mathbb{X} \times \mathbb{U}, \qquad i = 0, \ldots, N \tag{8.40e}$$

where the nonlinear model is linearized around the selected trajectory. This works well as long as the tracking controller performs well in keeping the car close to this linearization point. Note that (8.40) is a parametric problem with parameters $x, A_i, B_i, g_i, G_i, h_i, x_i^{\text{ref}}$ and u_i^{ref}. Each of these parameters changes between two problem instances.

8.5.2.5 Control Performance

Problem (8.40) is formulated for a horizon length of $N = 14$, i.e. the corresponding multistage problem has 15 stages. With a sampling time of 25 milliseconds, this corresponds to a lookahead time of about 0.35 seconds. Each stage has 8 variables (6 states and 2 inputs), and all parameters are supplied to the solver for each stage. The problem has more than 100 parameters, where e.g. the matrix A_i is counted as one parameter. This demonstrates the strength of the FORCES code generation system in supporting a large number of parameters. The weight matrices Q and R were chosen to be diagonal matrices, and the sets $\mathbb{X}$ and $\mathbb{U}$ were simple upper and lower bounds on the respective variables.

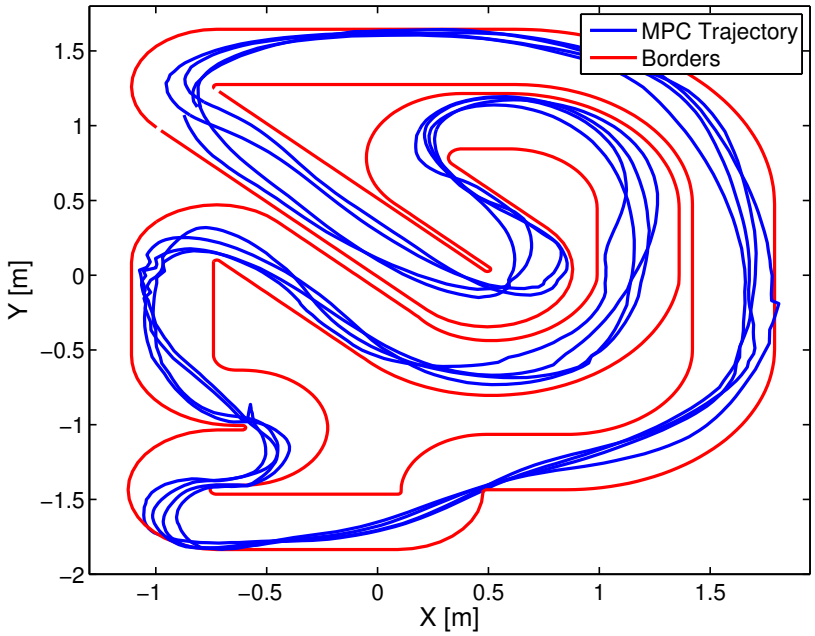


Figure 8.16: Closed loop control performance logged from experiments. The MPC problem (8.40) was solved using a solver generated by the FORCES code generation system. (From [80].)

Figure 8.16 shows the resulting closed loop trajectories with the MPC tracking controller (8.40), recorded in experiments. The resulting lap times are about 23 % faster (9.3 vs. 11.7 seconds) than previous controllers based on proportional-integral-derivative (PID) reference tracking. A video of Liniger's experiment is available at `http://www.youtube.com/watch?v=3XnlmHUJ9RE`.

8.5.2.6 Computational Performance of FORCES Solver

Table Table 8.8 shows the timings for different parts of the algorithm [80]. Solving the MPC problem with the single-threaded solver generated by FORCES is faster than computing the actual problem matrices, i.e. linearization of the nonlinear dynamics, discretization and generation of the inequalities for the constraints. It is also interesting to note that IBM's (multithreaded) CPLEX solver takes about 6 milliseconds for solving the QP while the *entire* control algorithm can be executed in only 6.15 milliseconds when using the FORCES solver. This effectively doubles the sampling rate, allowing for faster feedback and higher performance.

To investigate the performance of FORCES on different platforms, we have timed the solver generated for problem (8.40) on various embedded platforms. The results are given

	Time
Path planning:	4.13 ms
Problem generation:	1.19 ms
Solving QP with CPLEX:	6.00 ms
Solving QP with FORCES:	0.83 ms

Table 8.8: Average computation times for the different parts of the racing algorithm executed on Intel Core i7 CPU with 2.5 GHz. (From [80].)

in Table 8.9. On the B&R industrial automation platforms, FORCES is slower than for example on the ODROID U2 platform with an ARM Cortex A9 chip, which achieves almost desktop-like performance despite being an embedded low power processor that is built into many of today's smartphones and tablets, including the Samsung Galaxy S3 phone. This shows that FORCES generates platform independent code that runs reasonably fast on already existing embedded devices.

Architecture	**CPU Frequency**	**Operating System**	**Time per QP**
B&R CP-1484 Celeron	266 MHz	N/A – proprietary	207 ms
Raspberry Pi ARM6	700 MHz	Debian Wheezy	36.7 ms
Raspberry Pi ARM6	900 MHz	Debian Wheezy	28.4 ms
B&R CP-3586 Atom	1.6 Ghz	N/A – proprietary	22.1 ms
Raspberry Pi ARM6	700 MHz	Raspbian Wheezy	21.3 ms
Raspberry Pi ARM6	900 MHz	Raspbian Wheezy	15.6 ms
ODROID U2 ARM9	$4x1.7$ GHz	Ubuntu 11.04	3.3 ms
PC C2D P8600	$2x2.53$ GHz	Ubuntu 10.04	1.1 ms
PC i7 3770T	$4x2.5$ GHz	Debian Wheezy	0.8 ms

Table 8.9: Computational performance of the FORCES solver for the race car tracking MPC problem (8.40). The ODROID's ARM9 chip is also built into Samsung Galaxy S3 smartphones, for example.

9 A Second-order Cone Programming Solver for Embedded Systems

9.1 Introduction

In this chapter, we describe a new numerical solver designed to run on embedded platforms for computing solutions to second-order cone programs of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b\,, \\ & Gx \preceq_{\mathcal{K}} h\,. \end{array} \tag{SOCP}$$

Most commonly solved convex optimization problems can be transformed into this problem class, including LPs, QPs and quadratically constrained QPs (QCQPs), see e.g. [19].

Beyond these more traditional problem classes, the ability to solve convex programs with second-order cone constraints has enabled important applications and opened up new possibilities for sophisticated automated decision making. One of the main drivers, from an application point of view, for formulating a decision problem as an SOCP is *robustness* against uncertainty in the problem parameters. Tractable robust counterparts of LPs generally result in SOCPs when considering the parameters of affine inequalities to lie in an ellipsoidal uncertainty set. Robust formulations that yield SOCPs are used in fields as diverse as finance (robust portfolio selection problems [52]), communications (robust beamforming [42, 134]), signal processing (robust error correction [24]), medicine (robust radiation therapy planning [26]), machine learning (robust support vector machines [6]) and robust receding horizon control [55]. Other applications (not motivated by robustness) are group lasso [93] or multiple kernel learning [10, 120] in the domain of machine learning, optimal power flow problems [119], FIR filter design [140], minimum-fuel powered descent for landing spacecraft [1] and soft-constrained model predictive control with stability guarantees [143]. See also [81] or [19, Chap. 6-8] for more potential applications of SOCPs. In general, second-order cone programming is well understood, and several solver implementations have been developed and released, both in the public domain (e.g. SeDuMi [121] and SDPT3 [125]) and as commercial codes (e.g. Gurobi [59] and MOSEK [5]).

Existing codes are designed for desktop computers, while in many applications, the convex solver is to be run on a computing platform that is embedded in a physical device, or that forms a small part of a larger software system. In these applications, the same problem is solved repeatedly with different data and most of the time with a hard real-time constraint. In such cases, currently available solvers cannot be used due to code size, memory requirements, dependencies on external libraries, availability of source code, and other complexities.

The proposed *embedded conic solver* (ECOS) is written in single-threaded ANSI-C with no library dependence other than the square root function. It can thus be used on any platform for which a C compiler is available. The implemented algorithm is conceptually the same as in Vandenberghe's CVXOPT [6, 7], a standard primal-dual Mehrotra predictor-corrector interior-point method with Nesterov-Todd scaling and self-dual embedding (cf. Section 2.1.4), which allows for detection of infeasibility and unboundedness.

Main Contributions

The main difference to existing solvers is the search direction computation, for which we present a new method that is both efficient and easy to implement. While CVXOPT and most other available SOCP solvers compute the search directions via a Cholesky factorization of the reduced KKT system, we propose to use a sparse LDL solver on a particularly chosen, sparse symmetric indefinite KKT system, which permits a stable LDL factorization with diagonal D for any ordering. This is in contrast to standard factorization methods for indefinite matrices that employ dynamic row and column interchanges during the factorization, cf. Section 3.1.1. As a result, a fixed elimination ordering can be employed, chosen once and for all on the basis of the problem's sparsity structure. Our approach preserves sparsity in the problem data, and keeps the ECOS code short and simple, with only about 800 lines of code for the core solver, including all linear algebra functionality.

In order to ensure that the factorization exists for all orderings, and to enhance numerical stability, we use regularization and iterative refinement, as is done in CVXGEN [87]. Additionally, we present a new sparse expansion of diagonal plus rank one blocks, which appear in the linear system due to second-order cone constraints. This expansion is crucial when optimizing over large second-order cones (SOCs). It is chosen in such a way that the lifted KKT matrix can be factored in a numerically stable way, despite a fixed pivoting sequence. To our best knowledge, this stable factorization of sparse indefinite KKT systems arising in primal-dual IPMs for SOCPs is a new result and is therefore proven here using elementary linear algebra.

With these techniques, ECOS is a low footprint, high performance SOCP solver that is

numerically reliable for accuracies beyond what is typically needed in embedded applications. Despite its simplicity, ECOS solves small problems more quickly than most existing SOCP solvers, and is competitive for medium-sized problems up to about 20 k variables. Our current implementation uses a fill-in reducing ordering, based on the approximate minimum degree heuristic [4]. A native Matlab MEX interface and integration with CVX, a Matlab package for specifying and solving convex programs [56], is provided, as well as a Python interface[1]. ECOS is available from `github.com/ifa-ethz/ecos` under the GNU General Public License version 3 (GPL3).

Outline

In Section 9.2, we describe the interior-point method used in ECOS in detail. Section 9.3 describes the proposed search direction computation and the new sparse expansion of the symmetric indefinite KKT matrix, for which we prove important properties regarding numerical stability. Starting and stopping of ECOS are discussed in Section 9.4. Section 9.5 presents three numerical examples and run time comparisons to existing solvers. We close this chapter with some concluding remarks in Section 9.6.

9.2 Path-Following Infeasible Primal-Dual IPM

In this section, we provide all details of the particular interior point method implemented in ECOS. It is conceptually the same as in CVXOPT [7]: a path following infeasible primal-dual interior point method that solves the extended self-dual homogeneous embedding problem (ESD) by (loosely) tracking its central path. Consequently, the material presented in this section is mostly a review of [129], although we expand the presentation by certain aspects that we felt would ease the understanding of the method. Section 9.3 then gives details that are different from the implementation in CVXOPT.

9.2.1 Central Path

The self-dual embedding formulation allows one to detect infeasible problems (Lemma 2.1), while always providing a feasible starting point (2.28). See Section 2.1.4 for more details and properties of self-dual embeddings. We use the logarithmic barrier function from Definition 2.6 for symmetric cones (the standard "log-det" barrier (2.55)).

[1] Thanks to Eric Chu

The central path for (ESD) is defined by the set of points $(x, y, s, z, \kappa, \tau, \theta)$ satisfying

$$\begin{bmatrix} 0 \\ 0 \\ s \\ \kappa \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & A^T & G^T & c & q_x \\ -A & 0 & 0 & b & q_y \\ -G & 0 & 0 & h & q_z \\ -c^T & -b^T & -h^T & 0 & q_\tau \\ -q_x^T & -q_y^T & -q_z^T & -q_\tau & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \tau \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ D+1 \end{bmatrix}, \tag{9.1a}$$

$$s \circ z = \mu \boldsymbol{e}\,, \quad \kappa\tau = \mu\,, \tag{9.1b}$$

$$(s, z) \succ_{\boldsymbol{K}} 0\,, \quad (\kappa, \tau) > 0\,, \tag{9.1c}$$

where μ is the path parameter and all other quantities are defined in Section 2.1.4.2. Note that $e^T(s \circ z) = (e \circ s)^T z = s^T z$ by the third property of the vector product $\circ$ from Definition 2.1, and $\boldsymbol{e}^T\boldsymbol{e} = D$ from Definition 2.4, hence it follows from the complementarity condition (9.1b) that

$$\mu = \frac{s^T z + \kappa\tau}{D+1} = \theta \tag{9.2}$$

for points on the central path, where the second equality results from (2.27). Due to the latter, the variable θ can be eliminated. Furthermore, the last equality in (9.1a) is redundant since it is implied by the way (q_x, q_y, q_z, q_τ) are defined (take the inner product with (x, y, z, τ) in (2.26) to see this) and can be eliminated, which yields a simplified expression for the central path:

$$\begin{bmatrix} 0 \\ 0 \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \tau \end{bmatrix} + \mu \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_\tau \end{bmatrix}, \tag{9.3a}$$

$$\left(W^{-T}s\right) \circ (Wz) = \mu\mathbf{e}\,, \quad \kappa\tau = \mu\,, \tag{9.3b}$$

$$(s, z) \succ_{\boldsymbol{K}} 0\,, \quad (\kappa, \tau) > 0\,, \tag{9.3c}$$

where W is a *scaling matrix*. Primal-dual methods generate iterates close to the central path by linearizing (9.3) and taking a certain step size along the resulting search direction. By different choices of the scaling matrix W, different search directions are obtained. The effect of the scaling W and its definition are discussed in detail in the next section.

9.2.2 Scalings

Nesterov and Todd showed that the self-scaled property of the barrier function (2.55) can be exploited to construct interior-point methods that tend to make long steps along the

search direction that follows from a linearization of (9.3) [100]. A primal-dual scaling W is a linear transformation [7]

$$\tilde{s} = W^{-T}s, \qquad \tilde{z} = Wz \tag{9.4}$$

that leaves the cone and the central path unchanged, i.e.,

$$s \in \boldsymbol{K} \Leftrightarrow \tilde{s} \in \boldsymbol{K}\,, \tag{9.5a}$$
$$z \in \boldsymbol{K} \Leftrightarrow \tilde{z} \in \boldsymbol{K}\,, \tag{9.5b}$$
$$s \in \operatorname{int} \boldsymbol{K} \Leftrightarrow \tilde{s} \in \operatorname{int} \boldsymbol{K}\,, \tag{9.5c}$$
$$z \in \operatorname{int} \boldsymbol{K} \Leftrightarrow \tilde{z} \in \operatorname{int} \boldsymbol{K}\,, \tag{9.5d}$$
$$s \circ z = \mu\mathbf{e} \Leftrightarrow \tilde{s} \circ \tilde{z} = \mu\mathbf{e}\,. \tag{9.5e}$$

We use the symmetric *Nesterov-Todd (NT)* scaling, which is defined by the scaling point $w \in \operatorname{int} \boldsymbol{K}$ such that

$$\nabla^2\Phi(w)s = z,$$

i.e. w is the point where the Hessian of the barrier function (2.55) maps s onto z. It can be shown that such a w always exists and that it is unique [100]. The NT-scaling W is then obtained by a symmetric factorization of $\nabla^2\Phi(w)^{-1} = W^TW$ [7]. This transformation can be used to define a point $\lambda \in \operatorname{int} \boldsymbol{K}$ such that the scaling W maps both s and z on λ:

$$\lambda \triangleq W^{-T}s = Wz. \tag{9.6}$$

We give the NT-scalings [100] for the cones employed in this thesis, i.e. for the positive orthant and for the second order cone, in the following.

9.2.2.1 NT-scaling for Positive Orthant $\boldsymbol{Q}^1$

For $s, z \in \boldsymbol{Q}^1$,

$$W_+ = \sqrt{s/z}. \tag{9.7}$$

This is the standard scaling used in primal-dual interior point methods for LPs and QPs.

9.2.2.2 NT-scaling for Second-Order Cone $\boldsymbol{Q}^n$, $n > 1$

Consider the normalized points for $s, z \in \boldsymbol{Q}^n$ with $n > 1$,

$$\bar{z} = \frac{1}{(z_0^2 - z_1^Tz_1)^{1/2}}\, z\,, \qquad \bar{s} = \frac{1}{(s_0^2 - s_1^Ts_1)^{1/2}}\, s\,,$$
$$\gamma = \left(\frac{1 + \bar{z}^T\bar{s}}{2}\right)^{1/2}, \qquad \bar{w} = \frac{1}{2\gamma}\left(\bar{s} + \begin{bmatrix}\bar{z}_0 \\ -\bar{z}_1\end{bmatrix}\right).$$

Then the NT-scaling is given by

$$W_{\text{SOC}} \triangleq \eta \begin{bmatrix} \bar{w}_0 & \bar{w}_1^T \\ \bar{w}_1 & I_{n-1} + (1+\bar{w}_0)\bar{w}_1\bar{w}_1^T \end{bmatrix}, \qquad \eta \triangleq \left((s_0^2 - s_1^T s_1)/(z_0^2 - z_1^T z_1)\right)^{1/4}. \tag{9.8}$$

9.2.2.3 Composition of Scalings

We now consider the NT-scaling W for the general problem (SOCP). First, define the cone $\boldsymbol{K}$ as follows.

Definition 9.1 (Product Cone of Positive Orthants and Second-Order Cones)**.** *Without loss of generality, the cone $\boldsymbol{K}$ in* (SOCP) *is the direct product of N symmetric cones,*

$$\boldsymbol{K} \triangleq \boldsymbol{Q}^1 \times \boldsymbol{Q}^1 \times \cdots \times \boldsymbol{Q}^1 \times \boldsymbol{Q}^{n_{L+1}} \times \cdots \times \boldsymbol{Q}^{n_N}, \tag{9.9}$$

where the first L cones correspond to the positive orthant $\boldsymbol{Q}^1$, and the remaining $N-L$ cones are second-order cones $\boldsymbol{Q}^{n_i}$, $n_i > 1$, $i = L+1, \ldots, N$.

The final scaling matrix W associated with the cone $\boldsymbol{K}$ as defined in Definition 9.1 and the variables $s, z \in \operatorname{int} \boldsymbol{K}$ is a block diagonal matrix composed of the scalings for each cone $\boldsymbol{Q}^{n_i}$,

$$W = \operatorname{blkdiag}(W_1, W_2, \ldots, W_N), \tag{9.10}$$

where the first L blocks $W_{1:L}$ are defined by (9.7) and the remaining $N-L$ ones by (9.8). Note that $W = W^T$ for the NT scaling for second-order cones. The following lemma can be proved by the properties of symmetric cones and by the definition of the Nesterov-Todd scaling point, cf. [58, 100].

Lemma 9.1 (Positive Definiteness of Nesterov-Todd Scalings [100])**.** *Let the symmetric cone $\boldsymbol{K}$ be defined as in Definition 9.1 and $s, z \in \operatorname{int} \boldsymbol{K}$. Then the scaling matrix W in* (9.10) *associated with s, z is positive definite.*

9.2.3 Search Directions

In predictor-corrector methods, the overall search direction is the sum of three directions (cf. Section 2.2.2.3). The *affine search* direction aims at directly satisfying the KKT conditions, i.e. $\mu = 0$ in (9.3), and offers progress (in terms of distance to the optimal solution) at the current step. The *centering direction* does not make progress in the current step, but brings the current iterate closer to the central path, such that larger progress can be made at the next affine step. The amount of centering is determined by a parameter $\sigma \in [0, 1]$, for

which Mehrotra proposed an efficient heuristic (2.62). In addition, second-order information from the central path is employed by computing a *corrector direction*, which compensates for the nonlinearities at the predicted affine step, cf. (2.61) for the non-conic setting. The centering and the corrector direction can be computed together in a so-called *combined* direction.

We detail in the following on the different search directions, for which we propose an efficient numerical solution method in Section 9.3.

9.2.3.1 Affine Search Direction

For the affine-scaling search direction, we set $\mu = 0$ and obtain the following linear system by linearizing (9.3):

$$\begin{bmatrix} 0 \\ 0 \\ \Delta s_a \\ \Delta \kappa_a \end{bmatrix} - \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x_a \\ \Delta y_a \\ \Delta z_a \\ \Delta \tau_a \end{bmatrix} = - \begin{bmatrix} r_x \\ r_y \\ r_z \\ r_\tau \end{bmatrix} \tag{9.11a}$$

$$\lambda \circ \left(W \Delta z_a + W^{-1} \Delta s_a\right) = -\lambda \circ \lambda \tag{9.11b}$$

$$\kappa \Delta \tau_a + \Delta \kappa_a \tau = -\kappa \tau \tag{9.11c}$$

The RHS of (9.11a) are the (negative) residuals

$$\begin{bmatrix} r_x \\ r_y \\ r_z \\ r_\tau \end{bmatrix} \triangleq \begin{bmatrix} 0 \\ 0 \\ s \\ \kappa \end{bmatrix} - \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \tau \end{bmatrix} . \tag{9.12}$$

Equality (9.11b) can be derived as follows. The first order Taylor approximation of the (conic) complementarity condition with scaling (9.3b) is given by

$$\begin{aligned} \left(W^{-1}s\right) \circ (Wz) &\approx \left(W^{-1}s\right) \circ (Wz) + \left(W^{-1}s\right) \circ (W \Delta z_a) + \left(W^{-1}\Delta s_a\right) \circ (Wz) \\ &= \lambda \circ \lambda + \lambda \circ (W \Delta z_a) + \left(W^{-1}\Delta s_a\right) \circ \lambda \\ &= \lambda \circ \lambda + \lambda \circ (W \Delta z_a) + \lambda \circ \left(W^{-1}\Delta s_a\right) \\ &= \lambda \circ \lambda + \lambda \circ \left(W \Delta z_a + W^{-1}\Delta s_a\right) . \end{aligned} \tag{9.13}$$

Setting (9.13) to zero (we want the affine search direction to give zero duality gap) yields (9.11b). Similarly, setting

$$\kappa \tau \approx \kappa \tau + \tau \Delta \kappa_a + \Delta \tau_a \kappa , \tag{9.14}$$

to zero yields (9.11c).

Lemma 9.2 (Properties of Affine Search Directions [129, §7.2])**.** *The affine search direction as computed by* (9.11) *satisfies*

$$s^T \Delta z_a + z^T \Delta s_a = -s^T z \,, \tag{9.15a}$$

$$\kappa \Delta \tau_a + \tau \Delta \kappa_a = -\kappa \tau \,, \tag{9.15b}$$

$$\Delta s_a^T \Delta z_a + \Delta \tau_a \Delta \kappa_a = 0 \,. \tag{9.15c}$$

Proof. See [129, §7.2]. □

Note in particular (9.15c), i.e. the primal and dual directions satisfy the complementarity condition.

Corollary 9.1. *For a step length α^{aff} in the affine search direction, the predicted duality measure μ^{aff} is given by*

$$\mu^{aff} = \left(1 - \alpha^{aff}\right) \mu \tag{9.16}$$

Proof. Starting from

$$D\mu^{\text{aff}} = (s + \alpha^{\text{aff}} \Delta s_a)^T (z + \alpha^{\text{aff}} \Delta z_a) + (\kappa + \alpha^{\text{aff}} \Delta \kappa_a)(\tau + \alpha^{\text{aff}} \Delta \tau_a) \,, \tag{9.17}$$

expanding the product and using the results of Lemma 9.2 yields the claim. □

As a result, Mehrotra's rule (2.62) simplifies to

$$\sigma = \left(1 - \alpha^{\text{aff}}\right)^3 . \tag{9.18}$$

9.2.3.2 Centering

The centering direction is obtained by solving the linearized version of (9.3) for a path parameter $\sigma\mu$, $\sigma = (0, 1)$, where σ determines the amount of centering. The resulting linear system determining the centering direction is

$$\begin{bmatrix} 0 \\ 0 \\ \Delta s_c \\ \Delta \kappa_c \end{bmatrix} - \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x_c \\ \Delta y_c \\ \Delta z_c \\ \Delta \tau_c \end{bmatrix} = \sigma\mu \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_\tau \end{bmatrix} \tag{9.19a}$$

$$\lambda \circ \left(W \Delta z_c + W^{-1} \Delta s_c\right) = \sigma\mu \mathbf{e} \tag{9.19b}$$

$$\kappa \Delta \tau_c + \Delta \kappa_c \tau = \sigma\mu \,. \tag{9.19c}$$

We use Mehrotra's rule (2.62) to determine the amount of centering σ based on the predictor search direction as given in (9.18).

9.2.3.3 Correction for Nonlinearities

As in the linear case described in Section 2.2.2.2, nonlinearities in the central path can be approximately corrected by estimating the error in the complementarity condition of a full affine-scaling step, which is given by

$$\begin{aligned}\left(W^{-1}(s+\Delta s_a)\right)\circ\left(W(z+\Delta z_a)\right) &= \left(W^{-1}\Delta s_a\right)\circ\left(W\Delta z_a\right)\\ &\quad+\underbrace{\lambda\circ\lambda+\lambda\circ\left(W\Delta z_a+W^{-1}\Delta s_a\right)}_{=0\ (9.11)},\end{aligned}\tag{9.20}$$

and, similarly,

$$(\tau+\Delta\tau_a)(\kappa+\Delta\kappa_a)=\Delta\tau_a\Delta\kappa_a+\underbrace{\tau\kappa+\Delta\tau_a\kappa+\tau\Delta\kappa_a}_{=0\ (9.11)}.\tag{9.21}$$

As a result, the linearization leads to an overall error of $(W^{-1}\Delta s_a)\circ(W\Delta z_a)+\Delta\tau_a\Delta\kappa_a$. The corrector direction compensates for this non-zero terms by changing the right hand side of (9.19b) and (9.19c) to

$$\lambda\circ\left(W\Delta z_c+W^{-1}\Delta s_c\right)=\sigma\mu-\left(W^{-1}\Delta s_a\right)\circ\left(W\Delta z_a\right)\tag{9.22a}$$

$$\kappa\Delta\tau_c+\Delta\kappa_c\tau=\sigma\mu-\Delta\tau_a\Delta\kappa_a\tag{9.22b}$$

Note that this correction is an approximation, since we assume that a full the step in the affine direction is taken. Hence the correction is inaccurate but nevertheless it has proven very effective in practice.

9.2.3.4 Combined search direction

The three search direction computations of the affine, centering and correction step can be combined into a single operation by summing up the right hand sides of (9.11), (9.19) and (9.22):

$$\begin{bmatrix}0\\0\\\Delta s\\\Delta\kappa\end{bmatrix}-\begin{bmatrix}0&A^T&G^T&c\\-A&0&0&b\\-G&0&0&h\\-c^T&-b^T&-h^T&0\end{bmatrix}\begin{bmatrix}\Delta x\\\Delta y\\\Delta z\\\Delta\tau\end{bmatrix}=-(1-\sigma)\begin{bmatrix}r_x\\r_y\\r_z\\r_\tau\end{bmatrix}\tag{9.23a}$$

$$\lambda\circ\left(W\Delta z+W^{-1}\Delta s\right)=-\lambda\circ\lambda+\sigma\mu-\left(W^{-1}\Delta s_a\right)\circ\left(W\Delta z_a\right)\tag{9.23b}$$

$$\kappa\Delta\tau+\Delta\kappa\tau=-\tau\kappa+\sigma\mu-\Delta\tau_a\Delta\kappa_a\tag{9.23c}$$

using the relation [129, §7.2]

$$(r_x,r_y,r_z,r_\tau)=\mu\,(q_x,q_y,q_z,q_\tau)\ .\tag{9.24}$$

Remark 9.1 (Linear Convergence)**.** *It can be shown that the duality measure μ and the residuals decrease linearly [129, §7.2], i.e.*

$$\mu^+ = (1 - \alpha(1-\sigma))\mu\,. \tag{9.25}$$
$$(r_x, r_y, r_z, r_\tau)^+ = (1 - \alpha(1-\sigma))(r_x, r_y, r_z, r_\tau)\,. \tag{9.26}$$

Consequently, if $0 < \sigma_{min} \leq \sigma < 1$ for all steps and an appropriate line search is used that ensures $0 < \alpha < 1$, convergence of the algorithm is ensured.

9.2.3.5 Obtaining Linear Systems with Symmetric Indefinite Coefficient Matrix

The main computational burden in interior point methods is the computation of the search directions (9.11) and (9.23). In principle, these equations differ only in their RHS, which allows one to reuse the factorization of the coefficient matrix. The two linear systems (9.11) and (9.23) are of the form

$$\begin{bmatrix} 0 \\ 0 \\ \Delta s \\ \Delta\kappa \end{bmatrix} - \begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & 0 & h \\ -c^T & -b^T & -h^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\tau \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \\ d_z \\ d_\tau \end{bmatrix} \tag{9.27a}$$
$$\lambda \circ \left(W\Delta z + W^{-1}\Delta s\right) = d_s \tag{9.27b}$$
$$\kappa\Delta\tau + \Delta\kappa\tau = d_\kappa \tag{9.27c}$$

We eliminate Δs and $\Delta\kappa$ by using (9.27b) and (9.27c), respectively:

$$\Delta s = W\left(\lambda \setminus d_s - W\Delta z\right) \tag{9.28a}$$
$$\Delta\kappa = \left(d_\kappa - \kappa\Delta\tau\right)/\tau \tag{9.28b}$$

where the operator $\setminus$ is the inverse operator to the vector product $\circ$ defined in (2.15), such that if $u \circ v = w$ then $u \setminus w = v$ (we provide an efficient formula for this step in (9.57)). These equations can be used to rewrite (9.27a):

$$\begin{bmatrix} 0 & A^T & G^T & c \\ -A & 0 & 0 & b \\ -G & 0 & W^2 & h \\ -c^T & -b^T & -h^T & \kappa/\tau \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\tau \end{bmatrix} = - \begin{bmatrix} d_x \\ d_y \\ d_z - W\left(\lambda \setminus d_s\right) \\ d_\tau - d_\kappa/\tau \end{bmatrix} \tag{9.29}$$

The coefficient matrix in (9.29) can be made symmetric by reducing (9.29) to two linear systems with the same symmetric indefinite coefficient matrix as follows. First, rewrite (9.29)

as

$$\begin{bmatrix} 0 & A^T & G^T \\ -A & 0 & 0 \\ -G & 0 & W^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} + \Delta\tau \begin{bmatrix} c \\ b \\ h \end{bmatrix} = - \begin{bmatrix} d_x \\ d_y \\ d_z - W(\lambda \setminus d_s) \end{bmatrix} \tag{9.30a}$$

$$\Delta\tau = \left(-d_\tau + d_\kappa/\tau + c^T\Delta x + b^T\Delta y + h^T\Delta z\right)\tau/\kappa \tag{9.30b}$$

We can now solve two linear systems,

$$\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^2 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \end{bmatrix} = \begin{bmatrix} -c \\ b \\ h \end{bmatrix} \tag{9.31a}$$

and

$$\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^2 \end{bmatrix} \begin{bmatrix} \Delta x_2 \\ \Delta y_2 \\ \Delta z_2 \end{bmatrix} = \begin{bmatrix} d_x \\ -d_y \\ -d_z + W(\lambda \setminus d_s) \end{bmatrix} \tag{9.31b}$$

which recover (9.30a) by a linear combination: Take $\Delta\tau$ times (9.31a), add it to (9.31b) and define

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \Delta x_2 \\ \Delta y_2 \\ \Delta z_2 \end{bmatrix} + \Delta\tau \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \end{bmatrix} . \tag{9.32}$$

The remaining quantity $\Delta\tau$ is then obtained by inserting (9.32) into (9.30b):

$$\begin{aligned} &\Delta\tau\kappa/\tau = \left[-d_\tau + d_\kappa/\tau + c^T(\Delta x_2 + \Delta\tau\Delta x_1) + b^T(\Delta y_2 + \Delta\tau\Delta y_1) + h^T(\Delta z_2 + \Delta\tau\Delta z_1)\right] \\ \Leftrightarrow &\Delta\tau\left[\kappa/\tau - c^T\Delta x_1 - b^T\Delta y_1 - h^T\Delta z_1\right] = -d_\tau + d_\kappa/\tau + c^T\Delta x_2 + b^T\Delta y_2 + h^T\Delta z_2 \end{aligned}$$

which gives

$$\Delta\tau = \frac{-d_\tau + d_\kappa/\tau + c^T\Delta x_2 + b^T\Delta y_2 + h^T\Delta z_2}{\kappa/\tau - c^T\Delta x_1 - b^T\Delta y_1 - h^T\Delta z_1} . \tag{9.33}$$

Note that step (9.31a) is the same for the predictor and corrector step, hence it has to be solved only once, while (9.31b) is solved twice with different RHS in the predictor and corrector step. In summary, we have to solve a system of linear equations of type (9.31a)/(9.31b) three times.

Table 9.1: RHS for affine and combined search direction. Δs_a and Δz_a are affine directions.

RHS	affine	combined (centering & corrector)
b_x	r_x	$(1-\sigma)r_x$
b_y	r_y	$(1-\sigma)r_y$
b_z	$-r_z + s$	$-(1-\sigma)r_z + W(\lambda \setminus d_s)$
b_s	$\lambda \circ \lambda$	$\lambda \circ \lambda + \left(W^{-1}\Delta s_a\right) \circ (W\Delta z_a) - \sigma\mu\mathbf{e}$
b_τ	r_τ	$(1-\sigma)r_\tau$
b_κ	$\tau\kappa$	$\tau\kappa + \Delta s_a \Delta z_a - \sigma\mu$

9.2.3.6 Summary

We summarize here the search direction computation for further reference in the discussion of the ECOS solver. A search direction is computed by solving two linear systems,

$$K(\Delta x_1, \Delta y_1, \Delta z_1) = (-c, b, h) \quad \text{and} \tag{9.34a}$$

$$K(\Delta x_2, \Delta y_2, \Delta z_2) = (b_x, b_y, b_z), \tag{9.34b}$$

where

$$K \triangleq \begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^2 \end{bmatrix} \tag{9.34c}$$

and combining the results to

$$\Delta\tau = \frac{b_\tau - b_\kappa/\tau + c^T\Delta x_2 + b^T\Delta y_2 + h^T\Delta z_2}{\kappa/\tau - c^T\Delta x_1 - b^T\Delta y_1 - h^T\Delta z_1} \tag{9.35a}$$

$$\Delta x = \Delta x_1 + \Delta\tau\Delta x_1 \tag{9.35b}$$

$$\Delta y = \Delta y_1 + \Delta\tau\Delta y_1 \tag{9.35c}$$

$$\Delta z = \Delta z_1 + \Delta\tau\Delta z_1 \tag{9.35d}$$

$$\Delta s = -W(\lambda \setminus b_s + W\Delta z), \tag{9.35e}$$

$$\Delta\kappa = -(b_\kappa + \kappa\Delta\tau)/\tau. \tag{9.35f}$$

with right hand sides $(b_x, b_y, b_z, b_s, b_\kappa, b_\tau)$ for the affine and centering-corrector direction summarized in Table 9.1. For the affine direction, (9.35e) simplifies to

$$\Delta s_a = -W(\lambda + W\Delta z_a).$$

Since the right hand side of (9.34a) is the same for both directions, only three linear systems have to be solved per interior point iteration. This is the computational bottleneck of the primal-dual interior point algorithm. Next, we propose an efficient solution method for linear equations of type (9.34a) and (9.34b) in Section 9.3.

9.3 Efficient Search Direction Computation

As outlined in Section 9.2.3.6, the computational burden of the primal-dual interior point method implemented in ECOS is the solution of linear systems of the form

$$\underbrace{\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^2 \end{bmatrix}}_{=K} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{9.36}$$

for three different right-hand sides (b_x, b_y, b_z). For typical problems, the cost of solving the linear system (9.36) amounts to more than $90\,\%$ of the total computational cost. While efficient sparse solvers for symmetric indefinite systems exist, they are rather complex in terms of memory requirements, code size, code execution flow and memory access patterns. For this reason, usual codes for solving (9.36) that run on desktop computers are generally not well suited for embedded applications, and one of the main contributions of this part of the thesis is to present an alternative concept for solving (9.36) that can be implemented on embedded systems.

In Section 9.3.1, an overview of our method for an efficient computation of search directions in (9.36) is presented. It is conceptually similar to the approach used in CVXGEN [87] for QPs. The major differences lie in solving a different symmetric indefinite linear system, and in extending the approach to handle Nesterov-Todd scalings, which are dense matrices when second-order cone constraints are present, which we expand to larger but sparse matrices. The latter is discussed in Section 9.3.2.

9.3.1 Sparse LDL Factorization with Fixed Ordering

9.3.1.1 Conceptual Overview

For small to medium sized problems of up to several tens of thousands of variables, direct methods are the preferred choice for solving (9.36), since the factorization of the coefficient matrix, which is the most expensive part in the computation, can be reused to solve for different right hand sides. Working with the indefinite system (9.36) has the advantage that sparsity in A and G is preserved and can be directly exploited.

Remark 9.2. *In most existing interior point codes, the linear system* (9.36) *is reduced by eliminating* Δz*:*

$$\Delta z = -W^{-2}\left(b_z - G\Delta x\right), \tag{9.37}$$

which gives rise to the linear system

$$\begin{bmatrix} G^T W^{-2} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} b_x + G^T W^{-2} b_z \\ b_y \end{bmatrix}. \tag{9.38}$$

The (1,1) block in (9.38) *can be potentially dense if G has a dense row (or if W^{-2} is dense). This unnecessarily increases the fill-in*[2] *and with it the amount of computations needed to compute the factorization. Additionally, computing the product $G^T W^{-2} G$ is potentially expensive when standard matrix-matrix multiplication is used or requires additional linear algebra manipulations to be carried out efficiently, in turn increasing the code complexity. We therefore do* not *eliminate Δz, but work with the unreduced symmetric indefinite system* (9.36) *instead.*

As described in Section 3.1, a general symmetric indefinite factorization requires symmetric interchanges of rows and columns for numerical stability, cf. Section 3.1.1. As a result, in classical LDL factorization, the effective *permutation* of the matrix is data dependent. From a code perspective, this involves dynamic memory allocation, extensive branching, complex data structures and expensive access patterns, and is therefore not well suited for embedded platforms.

In order to obtain an efficient solver for embedded systems with simple code of small size, the goal is to use a fixed permutation, which depends only on the sparsity pattern of the coefficient matrix in (9.36). This has the key advantage that the *structure* of the factors L and D does not change from iteration to iteration. The challenge lies in ensuring that the LDL decomposition exists for all permutations of the coefficient matrix in (9.36) and all possible instances of the data. As will be shown in the following section, these properties are achieved by regularization, turning K into a quasi-definite matrix, for which a stable factorization exists for any permutation, see Theorem 3.1. To recover the solution of (9.36) from the perturbed solution of the regularized system, we employ the iterative refinement technique from Section 3.3.

Using these techniques, the permutation P matrix in (3.4), i.e. the elimination ordering for the LDL factorization, can be computed to approximately minimize the fill-in in the unit lower triangular factor L. In turn, this reduces the number of floating point operations for computing the factorization.

To summarize, our strategy for solving (9.36) efficiently is as follows:

- employ the indefinite matrix K in (9.36) to exploit sparsity in A and G,
- use a fill-in reducing ordering on K,

[2] the additional number of non-zeros in the Cholesky factor of a matrix when compared to the original matrix

- perform regularization of K to ensure the existence of a factorization for *any* static ordering
- use iterative refinement (cf. Section 3.3) to recover the solution to the unregularized system (9.36).

We detail on these aspects in the following subsections. Overall, this strategy offers the key benefit of computing the permutation and symbolic factorization in an initialization stage, which is then used in the individual interior point iterations to obtain the numerical factorization.

9.3.1.2 Permutations to Reduce Fill-in

Finding the best elimination ordering P that minimizes the fill in the LDL factor L (cf. (3.4)) is hard, but good heuristics exist. We use the approximate minimum degree (AMD) code [4] to compute fill-in reducing orderings of K. AMD is a heuristic that performs well in practice and has low computational complexity. Other heuristics that work well in practice are for example SYMAMD [33] or graph partitioning algorithms from the METIS library [74]. ECOS can easily be extended to use these orderings instead of AMD.

9.3.1.3 Static Regularization: Turning K into a Quasi-Definite Matrix

We can modify K from (9.36) to obtain a quasi-definite matrix by perturbing the diagonal elements of the (1,1) and (2,2) blocks as follows:

$$\tilde{K} \triangleq \underbrace{\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^2 \end{bmatrix}}_{\triangleq K} + \underbrace{\begin{bmatrix} \delta I_n & 0 & 0 \\ 0 & -\delta I_p & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\triangleq \Delta K} = \left[\begin{array}{c|cc} \delta I_n & A^T & G^T \\ \hline A & -\delta I_p & 0 \\ G & 0 & -W^2 \end{array}\right] \tag{9.39}$$

where δ is a regularization parameter and is in the order of 10^{-7}. Note that due to Lemma 9.1, the (3,3) block containing the squared Nesterov-Todd scalings W is negative definite. As a result of regularization, $\tilde{K}$ has the structure of a quasi-definite matrix as defined in Definition 3.1. It then follows from Theorem 3.1 that the regularized matrix $\tilde{K}$ can be factored in a stable manner.

There exists a tradeoff for the regularization parameter δ: large values perturb the linear system significantly, while small values make the coefficient matrix barely quasi-definite, so numerical problems are more likely to occur during the factorization. In [51], the use of quasi-definite matrices in barrier methods for LPs has been investigated, introducing an effective condition number for quasi-definite systems. Based on this analysis, the following has been concluded in the context of linear programming [51]:

> "With reasonable values of [...] δ, we can expect $PKP^T = LDL^T$ to be stable until the iterates are close to the optimal solution",

where reasonable values of δ were found to be in the range of $\delta \approx 10^{-6}...10^{-8}$ [51]. From this analysis, we cannot expect the regularization strategy to give high-accuracy solutions to problem (SOCP), as the quasi-definite system to be solved is increasingly ill-conditioned as the interior point algorithm approaches the solution. For this reason, it is suggested in [51] to switch to standard Bunch-Parlett factorization (cf. Section 3.1.1) or to a Cholesky factorization of the positive definite system (obtained by block reduction, cf. Section 3.2.1) if it is detected that the quasi-definite matrix (9.39) is too ill-conditioned. In our numerical experiments, we have observed that the solver is generally stable for accuracies that would be typically used in embedded applications, and therefore omit the fallback strategy for the sake of code simplicity.

9.3.1.4 Dynamic Regularization

When the iterates are close to optimality, the scaling matrix W in the (3,3) block becomes nearly singular, and the effect is even more eminent in the matrix W^2. In order to further enhance numerical stability despite roundoff errors due to finite precision arithmetics, we apply dynamic regularization in addition to static regularization. Specifically, we alter D_{ii} during the factorization whenever it becomes too small or it changes sign:

$$D_{ii} \leftarrow S_i\delta \quad \text{if} \quad S_i D_{ii} \leq \epsilon,$$

with parameters $\epsilon \approx 10^{-14}$, $\delta \approx 10^{-7}$. The signs S_i of the diagonal elements are known in advance,

$$S = (\mathbf{1}_n, -\mathbf{1}_p, -\mathbf{1}_L, \rho)\,,$$

where ρ is the sign vector associated with the second-order cones $\boldsymbol{Q}^{n_i}$, $n_i > 1$. For the matrix K as defined in (9.36), $\rho \triangleq -\mathbf{1}_{N-L}$. For the sparse representation of K introduced in Section 9.3.2.1 and used in the current implementation of ECOS,

$$\rho \triangleq (\rho_{L+1}, \rho_{L+2}, \ldots, \rho_N) \quad \text{with} \quad \rho_i \triangleq (-\mathbf{1}_{m_i}, -1, 1)$$

for each second-order cone.

9.3.1.5 Iterative Refinement

The iterative refinement technique from Section 3.3 is used to recover the solution to system (9.36) from an approximate solution obtained by solving (9.36) using the regularized

coefficient matrix $\tilde{K}$ (9.39) instead of K. In the current implementation of ECOS, we terminate the refinement procedure if

$$\|r\|_\infty/(1+\|b\|_\infty) \leq \epsilon_{IR} \tag{9.40}$$

holds, where ϵ_{IR} is in the order of 10^{-14} and r is the residual of the linear system to be solved (cf. Algorithm 3.5). This approach tends to terminate the refinement procedure earlier than the original criterion in Algorithm 3.5, while producing solutions to (9.36) of reasonable accuracy. We additionally check whether a refinement step has been successful in reducing the residual norm by a certain factor, say, by a factor of two; otherwise the procedure is terminated. The reasoning behind this strategy is based on computational experience: if the progress in reducing the residual is slow, it is usually better to stop refining the solution, because the computational cost is not justified by the additional accuracy that is expected from the next step of refinement.

9.3.2 Structure Exploitation of Second-Order Cone NT-Scalings

While the strategy outlined in the previous Section 9.3.1 has been successfully applied in CVXGEN [87] to the more compact linear system (9.38) for solving QPs, there is an additional difficulty when solving problems with second-order cone constraints: The Nesterov-Todd scalings (9.8) for the second order cone, which propagate to the (3,3) block in the KKT matrix K as $-W_{\text{soc}}^2$, are *dense* matrices involving a diagonal plus rank one term. Not exploiting this particular structure would yield to a significant number of non-zeros in both the KKT matrix and in its LDL factor L even for problems with sparse data, which would deteriorate the performance of the solver. Exploitation of the diagonal plus rank one structure present in W_{SOC} is therefore essential for solving problems with second-order cones of large dimension efficiently.

As one of the main contributions of this part of the thesis, we show how this can be achieved by expanding the square of the Nesterov-Todd scaling matrices appearing in the (3,3) block of the KKT matrix into a sparse matrix. This expansion yields $O(n)$ non-zero entries in the KKT matrix per second-order cone $\boldsymbol{Q}^n$, instead of the usual $O(n^2)$. Consequently (and more importantly), the resulting LDL factor L contains *significantly* fewer non-zeros, which allows for the efficient solution of problems with second-order cones of large dimensions.

We present the procedure for a single second order cone in the following for simplicity, but the construction can be carried out for each cone separately, as the NT-scaling W is a block diagonal matrix.

9.3.2.1 Sparse Expansion of NT-Scalings in KKT System

It is easily verified that the square of the second-order cone scaling matrix (9.8) is given by

$$V \triangleq W_{\mathrm{SOC}}^2 = \eta^2 \begin{bmatrix} \bar{w}_0^2 + \|\bar{w}_1\|_2^2 & c\bar{w}_1^T \\ c\bar{w}_1 & I_{n-1} + d\bar{w}_1\bar{w}_1^T \end{bmatrix}, \tag{9.41a}$$

$$c \triangleq 1 + \bar{w}_0 + \frac{\|\bar{w}_1\|_2^2}{1+\bar{w}_0}, \tag{9.41b}$$

$$d \triangleq 1 + \frac{2}{1+\bar{w}_0} + \frac{\|\bar{w}_1\|_2^2}{(1+\bar{w}_0)^2}. \tag{9.41c}$$

The following theorem is the main theoretical contribution of this chapter.

Theorem 9.1. *Let W be a Nesterov-Todd scaling for the second-order cone $\boldsymbol{Q}^n$ as defined in* (9.8)*, and $V = W^2$ as defined in* (9.41)*.*

(i) The square of the second-order cone scaling matrix, $V = W^2$, can be equivalently represented as the sum of a diagonal plus two rank one terms:

$$V = \eta^2(\mathcal{D} + uu^T - vv^T) \tag{9.42a}$$

with

$$\mathcal{D} \triangleq \begin{bmatrix} a & \\ & I_{n-1} \end{bmatrix}, \quad u \triangleq \begin{bmatrix} u_0 \\ u_1\bar{w}_1 \end{bmatrix}, \quad v \triangleq \begin{bmatrix} 0 \\ v_1\bar{w}_1 \end{bmatrix}, \tag{9.42b}$$

$$a \triangleq \left(\bar{w}_0^2 + \|\bar{w}_1\|_2^2 - \frac{c^2\|\bar{w}_1\|_2^2}{1+d\|\bar{w}_1\|_2^2}\right)/2, \tag{9.42c}$$

$$u_0 \triangleq \sqrt{\bar{w}_0^2 + \|\bar{w}_1\|_2^2 - a}, \quad u_1 \triangleq c/u_0, \quad v_1 \triangleq \sqrt{\frac{c^2}{u_0^2} - d}, \tag{9.42d}$$

where u_0, u_1, v_1, a, c, d are nonnegative scalars.

(ii) Let $\mathcal{D}$, u, and v be defined as above. Then the matrix

$$\tilde{V} \triangleq \eta^2 \begin{bmatrix} \mathcal{D} & v & u \\ v^T & 1 & 0 \\ u^T & 0 & -1 \end{bmatrix} \in \boldsymbol{R}^{(n+2)\times(n+2)} \tag{9.43}$$

is quasi-definite. We call $\tilde{V}$ the expanded, sparse representation of V.

Proof. Although (i) is easily verified by inserting the given definitions and carrying out simple algebra, we present a constructive proof in the following which gives further insight in the particular choices made. Let us start with the Ansatz that V should be represented

by (9.42a), where $\mathcal{D}, u$ as in (9.42b) and $v \triangleq (v_0, v_1\bar{w}_1)$ (we neglect the factor η^2 for notational convenience):

$$V = \begin{bmatrix} \bar{w}_0^2 + \|\bar{w}_1^2\|_2^2 & c\bar{w}_1^T \\ c\bar{w}_1 & I_{n-1} + d\bar{w}_1\bar{w}_1^T \end{bmatrix} = \begin{bmatrix} a & \\ & D \end{bmatrix} + \begin{bmatrix} u_0 \\ u_1\bar{w}_1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1\bar{w}_1^T \end{bmatrix}^T - \begin{bmatrix} v_0 \\ v_1\bar{w}_1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1\bar{w}_1^T \end{bmatrix}^T$$
$$= \begin{bmatrix} a + u_0^2 - v_0^2 & (u_0u_1 - v_0v_1)\bar{w}_1^T \\ (u_0u_1 - v_0v_1)\bar{w}_1 & D + (u_1^2 - v_1^2)\bar{w}_1\bar{w}_1^T \end{bmatrix}.$$

By comparison, $D = I_{n-1}$ follows and furthermore

$$a + u_0^2 - v_0^2 = \bar{w}_0^2 + \|\bar{w}_1\|_2^2 \tag{9.44a}$$
$$u_0u_1 - v_0v_1 = c \tag{9.44b}$$
$$u_1^2 - v_1^2 = d\,. \tag{9.44c}$$

For quasi-definiteness of $\tilde{V}$ in (9.43), the following additional condition is required:

$$\mathcal{D} - vv^T \succ 0 \Leftrightarrow \begin{bmatrix} a - v_0^2 & -v_0v_1\bar{w}_1^T \\ -v_0v_1\bar{w}_1 & I_{n-1} - v_1^2\bar{w}_1\bar{w}_1^T \end{bmatrix} \succ 0 \tag{9.45a}$$
$$\Leftrightarrow \begin{cases} a - v_0^2 > 0 \\ I_{n-1} - v_1^2\bar{w}_1\bar{w}_1^T \succ 0 \\ I_{n-1} - v_1^2\bar{w}_1\bar{w}_1^T - \frac{1}{a-v_0^2}v_0^2v_1^2\bar{w}_1\bar{w}_1^T \succ 0 \end{cases} \tag{9.45b}$$

where the last condition in (9.45b) arises from the Schur complement and can be written more compactly as

$$I_{n-1} - \frac{av_1^2}{a - v_0^2}\bar{w}_1\bar{w}_1^T \succ 0\,. \tag{9.46}$$

Without loss of generality, we can choose $v_0 = 0$. Conditions (9.44) and (9.45b) then simplify to

$$a + u_0^2 = \bar{w}_0^2 + \|\bar{w}_1\|_2^2 \tag{9.47a}$$
$$u_0u_1 = c \tag{9.47b}$$
$$u_1^2 - v_1^2 = d \tag{9.47c}$$
$$a > 0 \tag{9.47d}$$
$$I - v_1^2\bar{w}_1\bar{w}_1^T \succ 0 \tag{9.47e}$$

From (9.47b),

$$u_1 = \frac{c}{u_0}, \quad u_0 \neq 0. \tag{9.48a}$$

Inserting this into (9.47c) yields

$$v_1 = \pm\sqrt{\frac{c^2}{u_0^2} - d}, \quad \frac{c^2}{u_0^2} - d > 0\,. \tag{9.48b}$$

From (9.47a)

$$a = \bar{w}_1^2 + \|\bar{w}_1\|_2^2 - u_0^2 \tag{9.48c}$$

and due to (9.47d), the condition

$$u_0^2 < \bar{w}_0^2 + \|\bar{w}_1\|_2^2 \tag{9.48d}$$

follows. In order to fulfil (9.47e), we must have

$$\frac{1}{v_1^2} > \|\bar{w}_1\|_2^2 \quad \Leftrightarrow \quad \frac{u_0^2}{c^2 - d u_0^2} > \|\bar{w}_1\|_2^2 \quad \Leftrightarrow \quad u_0^2 > \frac{c^2\|\bar{w}_1\|_2^2}{1 + d\|\bar{w}_1\|_2^2} \tag{9.48e}$$

Combining (9.48e) with (9.48d), we obtain

$$\frac{c^2\|\bar{w}_1\|_2^2}{1 + d\|\bar{w}_1\|_2^2} < u_0^2 < \bar{w}_0^2 + \|\bar{w}_1\|_2^2\,. \tag{9.49}$$

It remains to show that the interval $(c^2\|\bar{w}_1\|_2^2/(1 + d\|\bar{w}_1\|_2^2),\ \bar{w}_0^2 + \|\bar{w}_1\|_2^2)$ is non-empty, i.e. we can always chose u_0^2 such that (9.49) holds. Subtracting the lower bound from the upper bound in (9.49) and using the definition of c and d in (9.41) yields

$$\Delta u_0^2 \triangleq \bar{w}_0^2 + \|\bar{w}_1\|_2^2 - \frac{c^2\|\bar{w}_1\|_2^2}{1 + d\|\bar{w}_1\|_2^2} \tag{9.50a}$$

$$= \frac{(\bar{w}_0^2 + \|\bar{w}_1\|_2^2)(1 + d\|\bar{w}_1\|_2^2) - c^2\|\bar{w}_1\|_2^2}{1 + d\|\bar{w}_1\|_2^2} \tag{9.50b}$$

$$= \frac{(\bar{w}_0^2 + \bar{w}_0 - \|\bar{w}_1\|_2^2)^2}{1 + \bar{w}_0^2} > 0\,. \tag{9.50c}$$

This shows that u_0^2 can always be chosen to satisfy (9.49), as the interval

$$\left(c^2\|\bar{w}_1\|_2^2/(1 + d\|\bar{w}_1\|_2^2),\ \bar{w}_0^2 + \|\bar{w}_1\|_2^2\right) \tag{9.51}$$

is non-empty. For numerical robustness, we choose the center point of the interval:

$$u_0^2 = \frac{c^2\|\bar{w}_1\|_2^2}{1 + d\|\bar{w}_1\|_2^2} + \frac{1}{2}\Delta u_0^2 > 0, \tag{9.52}$$

which gives the particular expression for u_0 in (9.42d), and concludes the proof. □

Using the result of Theorem 9.1, we propose a sparse representation of the last equation in (9.36),

$$G_i \Delta x - W_{\text{soc}}^2 \Delta z = b_z, \tag{9.53}$$

where $W \equiv W_{\text{SOC}}$ since we consider only second order cones in this section, by introduction of two scalar variables q and t as follows:

$$\begin{bmatrix} G_i \\ 0 \\ 0 \end{bmatrix} \Delta x - \underbrace{\eta^2 \begin{bmatrix} \mathcal{D} & v & u \\ v^T & 1 & 0 \\ u^T & 0 & -1 \end{bmatrix}}_{=\tilde{V}} \begin{bmatrix} \Delta z \\ q \\ t \end{bmatrix} = \underbrace{\begin{bmatrix} b_{z,i} \\ 0 \\ 0 \end{bmatrix}}_{\triangleq \tilde{b}_{z,i}}. \tag{9.54}$$

where G_i is the part of G corresponding to the SOC in question and $b_{z,i}$ the corresponding right hand side from Table 9.1. It is easily verified via basic linear algebra that (9.54) yields the same solution Δz_i as solving the dense equation (9.53). This yields to the following main practical outcome of this section.

Corollary 9.2. *Let all second-order cone blocks in the (3,3) block of the regularized KKT matrix* (9.39) *be expanded to their sparse representation as in* (9.54)*. The resulting lifted KKT matrix is quasi-definite, and the symmetric indefinite LDL factorization with diagonal D is numerical stable for any permutation.*

Proof. Due to part (ii) of Theorem 9.1, the expanded representation is quasi-definite, and since the (3,3) block in (9.39), $-W^2$, is block-diagonal, the resulting block-wise expanded matrix is also quasi-definite. As a result, the lifted KKT matrix is quasi-definite, and the claim on factorability for any permutation follows from Theorem 3.1. □

The lifted KKT matrix, which we denote by $\tilde{K}$, is of increased dimension – two rows and columns are added per second-order cone –, but significantly more sparse than the original (regularized) matrix (9.39), given that A and G are sparse. This yields more sparse LDL factors and thereby significant computational savings if second-order cone constraints with large dimensions are involved. This result is illustrated in Figure 9.1 for the KKT matrix of a portfolio optimization problem introduced in Section 9.5.1.

9.3.2.2 Fast Scaling Operations

In the following, we give a formula for the fast multiplication of a vector $z \in \boldsymbol{K}$ by a NT-scaling W_{SOC} for second-order cones, which exploits the diagonal plus rank one block in (9.8). The NT-scaling can be applied as follows:

$$W_{\text{SOC}}\, z = \eta \begin{bmatrix} \bar{w}_0 z_0 + \zeta \\ z_1 + (z_0 + \zeta/(1+\bar{w}_0))\, \bar{w}_1 \end{bmatrix}, \tag{9.55}$$

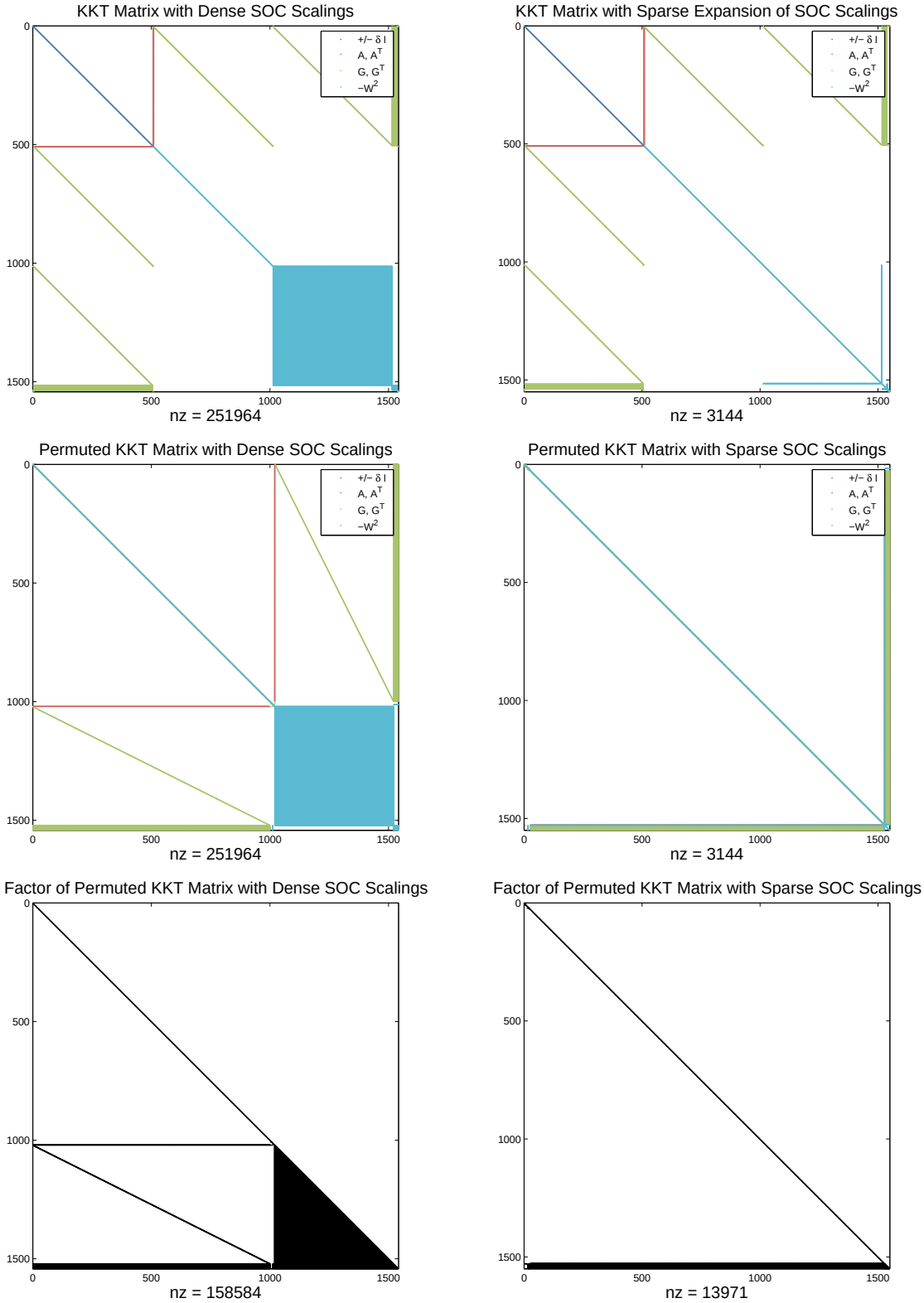


Figure 9.1: Sparsity patterns of the KKT matrix of dimension 1549, before (top) and after permutation (middle) as computed by AMD [4], and the sparsity pattern of the associated Cholesky factors (bottom) for the third portfolio optimization problem in Section 9.5. Left column: dense NT-scalings for second-order cones, right column: with sparse expansion according to Theorem 9.1. "nz" denotes the number of non-zeros.

where $\zeta \triangleq \bar{w}_1^T z_1$. Similarly, we can apply the inverse scaling W_{SOC}^{-1} by using

$$W_{\text{SOC}}^{-1} z = \frac{1}{\eta} \begin{bmatrix} \bar{w}_0 z_0 - \zeta \\ z_1 + (-z_0 + \zeta/(1+\bar{w}_0))\, \bar{w}_1 \end{bmatrix}. \tag{9.56}$$

By employing (9.55) and (9.56), a multiplication or left-division by W require only $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$ floating point operations, where n is the size of the cone under consideration, as in the LP case. This is significant if the dimension of the cone is large.

9.3.2.3 Fast Computation of Inverse Conic Vector Product

By exploiting the structure of the conic product $\circ$ as defined in (2.15), an efficient method for computing its *inverse*, which is defined as $v = u \setminus w$ for $u \circ v = w$, can be derived. For $u, v, w \in \boldsymbol{Q}^n$, $n > 1$, the formula is

$$\begin{gathered} v = u \setminus w \triangleq \frac{1}{\varrho} \begin{bmatrix} u_0 w_0 - \nu \\ (\nu/u_0 - w_0) u_1 + (\varrho/u_0)\, w_1 \end{bmatrix}, \\ \text{with } \varrho = u_0^2 - u_1^T u_1\,, \quad \nu = u_1^T w_1\,. \end{gathered} \tag{9.57}$$

For $u, v, w \in \boldsymbol{Q}^1$, $v = u \setminus w \triangleq w/u$, i.e. the usual scalar division. As a result, computing the inverse conic product can be performed in $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$ floating point operations that would occur in a naive implementation.

9.4 Implementation Details

Various aspects regarding the implementation of ECOS are discussed in the following, starting with the computation of initial values for the primal and dual variables in Section 9.4.1 and stopping criteria in Section 9.4.2. Some remarks on the internal structure of the ECOS code, which can be found at `github.com/ifa-ethz/ecos`, and an algorithm summary are provided in Section 9.4.3.

9.4.1 Initialization

The initialization procedure determines values for the primal variables x_0, s_0 and the dual variables y_0, z_0 to be used as the starting point for the self-dual embedding algorithm. Given these points, we set $\kappa_0 = \tau_0 = 1$ and $\mu_0 = (s_0^T z_0 + \kappa\tau)/(D+1)$ as in (2.28). In order to find x_0, y_0, z_0, s_0, the standard initialization procedure from CVXOPT is used (cf. [129, §7.3]). We include this for completeness here.

9.4.1.1 Primal variable x

For the initial value of the primal variable x, we solve the problem

$$\begin{aligned} \min_{r,x} \ & \frac{1}{2}\|r\|_2^2 \\ \text{s.t. } & r = h - Gx \\ & Ax = b\,, \end{aligned} \tag{9.58}$$

which computes a feasible point x with respect to the equality constraints $Ax = b$ that has minimum residual norm with respect to the inequalities. The solution to problem (9.58) is obtained by solving the linear system

$$\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -I \end{bmatrix} \begin{bmatrix} x \\ y \\ -r \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ h \end{bmatrix}, \tag{9.59}$$

which we regularize as in Sections 9.3.1.3 and 9.3.1.4 to make the coefficient matrix quasi-definite, and we apply iterative refinement to recover the solution to (9.59).

9.4.1.2 Slack s

The initial value of the slacks is determined using the residual value r resulting from (9.59). Note that if $r \succ_{\mathcal{K}} 0$ then the inequalities are strictly satisfied by x and we can set $s = r$. Otherwise we achieve $s \succ_K 0$ as follows: by examining the largest residual, we find α_p such that $r + \alpha_p \boldsymbol{e} \succeq_{\mathcal{K}} 0$. The initial value of s results to

$$s = \begin{cases} r & \text{if } r \succ_{\mathcal{K}} 0 \\ r + (1 + \alpha_p)\boldsymbol{e} & \text{otherwise} \end{cases}. \tag{9.60}$$

9.4.1.3 Dual variables y and z

In order to initialize the dual variables, the following equality constrained least-squares problem is solved:

$$\begin{aligned} \min_{y,\bar{z}} \ & \frac{1}{2}\|\bar{z}\|_2^2 \\ \text{s.t. } & A^T y + G^T \bar{z} + c = 0 \end{aligned} \tag{9.61}$$

This can be interpreted as a least squares solution to the dual equality constraints, which does not necessarily satisfy the inequality constraint $\bar{z} \succeq_{\mathcal{K}} 0$. The solution to (9.61) is

obtained by solving the linear system

$$\begin{bmatrix} 0 & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -I \end{bmatrix} \begin{bmatrix} x \\ y \\ \bar{z} \end{bmatrix} = \begin{bmatrix} -c \\ 0 \\ 0 \end{bmatrix} \tag{9.62}$$

using regularization, sparse LDL and iterative refinement. In order to ensure dual feasibility with respect to the inequality constraints, i.e. to have dual feasible initial variables $z \succeq_K 0$, we find α_d such that $\bar{z} + \alpha_d \boldsymbol{e} = 0$ and then set the initial Lagrange multipliers z to

$$z = \begin{cases} \bar{z} & \text{if } \bar{z} \succ_{\mathcal{K}} 0 \\ \bar{z} + (1 + \alpha_d)\boldsymbol{e} & \text{otherwise} \end{cases} . \tag{9.63}$$

9.4.2 Stopping Criteria

Our stopping criteria are similar to those of CVXOPT [7] and SeDuMi [121], although they differ in the way measures for primal and dual infeasibility are computed. First, define the quantities

$$r_x^0 \triangleq \max\{1\,,\|c\|_2\}, \qquad r_y^0 \triangleq \max\{1\,,\|b\|_2\}, \qquad r_z^0 \triangleq \max\{1\,,\|h\|_2\} \tag{9.64a}$$

and

$$\varrho \triangleq \max\{-c^T x\,, -b^T y - h^T z\}\,. \tag{9.64b}$$

At iteration k, the solver returns the following flags (and the corresponding iterates (x, y, z, s)):

- `ECOS_OPTIMAL` (optimal solution found) if $\varrho > 0$,

$$\max\left\{\frac{\|A^T y + G^T z + c\|_2}{r_x^0}, \frac{\|Ax - b\|_2}{r_y^0}, \frac{\|Gx + s - h\|_2}{r_z^0}\right\} < \texttt{FEASTOL} \tag{9.65a}$$

and

$$\left\{s^T z < \texttt{ABSTOL} \quad \text{or} \quad \frac{s^T z}{\varrho} < \texttt{RELTOL}\right\} . \tag{9.65b}$$

Condition (9.65a) ensures that the equality constraints in the primal problem (SOCP) and the dual problem (CP-D) are satisfied. Condition (9.65b) checks whether the complementarity gap $s^T z$ of the current iterate is below a certain absolute value, or whether the relative gap $s^T z/\varrho$ is small enough. It is useful to check both the absolute value and the relative value of the complementarity gap in order to ensure that optimality is also detected for problems with large objectives, which would lead to a large absolute but a small relative duality gap.

Parameter	MAXIT	FEASTOL	ABSTOL	RELTOL
Value	50	1E–5	1E–6	1E–6

Table 9.2: Default parameter values for stopping criteria used in ECOS.

- ECOS_PINF (primal infeasible) if $h^T z + b^T y < 0$ and

$$\frac{\|A^T y + G^T z\|_2}{r_x^0} < \texttt{FEASTOL}\,. \tag{9.65c}$$

  This condition tests whether $A^T y + G^T z = 0$ is approximately satisfied, which together with $h^T z + b^T y < 0$ yields a certificate of primal infeasibility (cf. Lemma 2.1, part 2, (i)).
- ECOS_DINF (dual infeasible) if $c^T x < 0$ and

$$\max\left\{\frac{\|Ax\|_2}{r_y^0}, \frac{\|Gx + s\|_2}{r_z^0}\right\} < \texttt{FEASTOL}\,. \tag{9.65d}$$

  This condition tests whether $Ax = 0$ is approximately satisfied, which together with $c^T x < 0$ yields a certificate of dual infeasibility (cf. Lemma 2.1 part 2, (ii)).
- ECOS_MAXIT (Maximum number of iterations reached) if $k =$ MAXIT. The algorithm terminates if a predefined number of iterations is reached.

The default parameter values that suffice for typical embedded applications are given in Table 9.2. If none of these conditions is fulfilled, ECOS proceeds with an iteration of the interior point method.

9.4.3 Implementation Overview

From a user point of view, ECOS implements three functions: *Setup*, *Solve* and *Cleanup*:

- *Setup* allocates memory and creates the upper triangular part of the expanded sparse KKT matrix $\tilde{K}$ as well as the sign vector S used for dynamic regularization. Then, a permutation of $\tilde{K}$ is computed using AMD [4], and $\tilde{K}$ and S are permuted accordingly. Last, a symbolic factorization of $P^T \tilde{K} P$ is carried out to determine the data structures needed for numerical factorization in *Solve*. For both the symbolic and numerical factorization, we use Tim Davis' *sparseLDL* code [31], which was modified to perform dynamic regularization and iterative refinement.

Algorithm 9.1 Primal-dual Path Following Method Implemented in ECOS

Require: c, G, h, A, b, P, $\tilde{K}_p = P\tilde{K}P^T$, $S_P \triangleq PS$, δ, ϵ, γ, `FEASTOL`, `ABSTOL`, `RELTOL`
Ensure: Certificate of optimality or infeasibility or #iterations = `MAXIT`
1: $k = 0$
2: $(x, y, z, s)_0 \leftarrow$ Initialize variables (cf. Section 9.4.1)
3: **loop**
4: Calculate residuals (9.12)
5: Evaluate stopping criteria (cf. Section 9.4.2):
6: **if** optimal **or** primal/dual infeasible **or** $k ==$ `MAXIT` **then**
7: break;
8: **end if**
9: $\tilde{K}_p \leftarrow$ Update scalings (9.7), (9.8), (9.43)
10: Factor $\tilde{K}_P$ (regularized sparse LDL using δ, ϵ and S_P, cf. Sections 9.3.1.3 and 9.3.1.4)
11: **Affine search direction:**
12: Solve $\tilde{K}_P\,(\Delta x_1, \Delta y_1, \Delta z_1) = P(-c, b, h)$ (1-3 steps of IR)
13: Solve $\tilde{K}_P\,(\Delta x_2, \Delta y_2, \Delta z_2) = P(b_x, b_y, b_z)$ (Left column in Table 9.1, 1-3 steps of IR)
14: Combine solutions (cf. (9.35)) to $\left(\Delta s^{\text{aff}}, \Delta z^{\text{aff}}, \Delta\tau^{\text{aff}}, \Delta\kappa^{\text{aff}}\right)$
15: **Line search for affine direction:**
16: $\alpha^{\text{aff}} \triangleq \max\left\{\alpha \in (0,1) \mid (s,z) + \alpha(\Delta s^{\text{aff}}, \Delta z^{\text{aff}}) \succ_{\mathcal{K}} 0, (\tau,\kappa) + \alpha(\Delta\tau^{\text{aff}}, \Delta\kappa^{\text{aff}}) > 0\right\}$
17: **Centering parameter:** $\sigma = \left(1 - \alpha^{\text{aff}}\right)^3$ (Mehrotra's rule)
18: **Combined search direction:**
19: Solve $\tilde{K}_P\,(\Delta x_2, \Delta y_2, \Delta z_2) = P(b_x, b_y, b_z)$ (Right column in Table 9.1, 1-3 steps of IR)
20: Combine solutions of line 12 and 19 (cf. (9.35)) to $(\Delta x, \Delta y, \Delta s, \Delta z, \Delta\tau, \Delta\kappa)$
21: **Line search for combined direction:**
22: $\alpha \triangleq \gamma \max\{\alpha \in (0,1) \mid (s,z) + \alpha(\Delta s, \Delta z) \succ_{\mathcal{K}} 0, (\tau,\kappa) + \alpha(\Delta\tau, \Delta\kappa) > 0\}$
23: **Update variables:** $(x, y, s, z, \tau, \kappa)_{k+1} = (x, y, s, z, \tau, \kappa)_k + \alpha\,(\Delta x, \Delta y, \Delta s, \Delta z, \Delta\tau, \Delta\kappa)$
24: **end loop**

- The *Solve* procedure in ECOS is summarized in Algorithm 9.1, which in the current implementation comprises about 800 lines of ANSI C-code only. The matrix $\tilde{K}$ is the KKT matrix K from (9.36), but with expanded (3,3) blocks for the Nesterov-Todd scalings as defined in Theorem 9.1.
- *Cleanup* frees all memory allocated by *Setup*.

Only *Setup* and *Cleanup* need memory management functionality such as `malloc` and `free`, while *Solve* uses only static memory allocation. Therefore, once the problem has been set up, different problem *instances* can be solved without the need for dynamic memory allocation (and without the *Setup* overhead).

9.5 Examples and Benchmarks

9.5.1 Portfolio Optimization

We consider a simple long-only portfolio optimization problem [19, p. 185–186], where the relative weights of assets are chosen to maximize risk-adjusted return. The problem to be solved is

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma(x^T \Sigma x) \\ \text{subject to} & \mathbf{1}^T x = 1, \quad x \geq 0, \end{array} \tag{9.66}$$

where the variable $x \in \boldsymbol{R}^n$ represents the portfolio, $\mu \in \boldsymbol{R}^n$ is the vector of expected returns, $\gamma > 0$ is the risk-aversion parameter, and $\Sigma \in \boldsymbol{R}^{n \times n}$ is the asset return covariance, given in factor model form,

$$\Sigma = FF^T + D.$$

Here $F \in \boldsymbol{R}^{n \times m}$ is the factor-loading matrix, and $D \in \boldsymbol{R}^{n \times n}$ is a diagonal matrix (of *idiosyncratic risk*). The number of factors in the risk model is m, which we assume is substantially smaller than n, the number of assets.

Problem (9.66) is a QP and can be converted into an SOCP of the form

$$\begin{array}{ll} \text{minimize} & -\mu^T x + \gamma(t + s) \\ \text{subject to} & \mathbf{1}^T x = 1, \quad x \geq 0 \\ & \|D^{1/2} x\|_2 \leq u, \quad \|F^T x\|_2 \leq v \\ & \|(1 - t, 2u)\|_2 \leq 1 + t \\ & \|(1 - s, 2v)\|_2 \leq 1 + s \end{array} \tag{9.67}$$

with variables $x \in \boldsymbol{R}^n$ and $(t, s, u, v) \in \boldsymbol{R}^4$.

A run time comparison of existing SOCP solvers and ECOS for the solution of random instances of (9.67) with accuracy 10^{-6} is given in Figure 9.2. The solvers MOSEK and Gurobi were run in single-threaded mode (the problems were solved more quickly than with multiple threads), while the current versions of SDPT3 and SeDuMi do not have the option to change the number of threads. For these solvers, we observed a CPU usage of more than 100%, which indicates that SDPT3 and SeDuMi make use of multiple threads. We switched off printing and used standard values for all other solver options. The number of iterative refinement steps for ECOS was limited to two.

As Figure 9.2 shows, ECOS outperforms most established solvers for small problems, and is overall competitive for medium-sized problems up to several thousands of variables even when compared with the two commercial solvers Gurobi and MOSEK, with the latter being the overall fastest conic solver for this example.

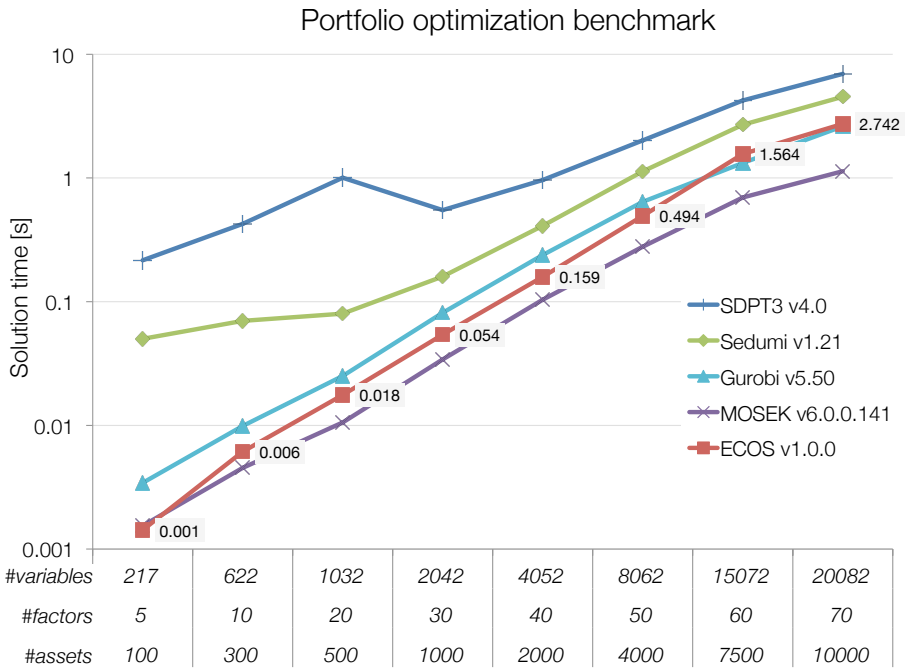


Figure 9.2: Timing results for portfolio problem (9.67) on a MacBook Pro with Intel Core i7 2.6 GHz CPU.

9.5.2 DIMACS Challenge Tests

In order to test for numerical robustness, ECOS was applied to all problems that are LPs, QPs or SOCPs from the seventh DIMACS challenge. A description of the test problems is given in [107]. The DIMACS problems are collected from real-world applications that are numerically challenging, and many of them are of considerable size. An early independent benchmarking of MOSEK, SDPT3 and SeDuMi (and many other codes) can be found in [96] along with a more detailed description of error measures and problem features.

We present the results for ECOS in Table 9.3, in comparison to SeDuMi version 1.21. SeDuMi was run with an accuracy of `EPS` $= 10^{-6}$ and `BIGEPS` $= 10^{-5}$ (see the SeDuMi manual [121] for details). Both SeDuMi and ECOS solve 13 out of 16 problems returning the flag `OPTIMAL`. For ECOS, the three unsolved problems have unscaled data, while their scaled variants are solved successfully. This indicates that scaling is important in practice; see e.g. [41, §4.12] for a good introduction to scaling. For most problems, the ECOS solve times are lower than those of SeDuMi using comparable accuracies. The biggest difference can be observed in problem 14, for which SeDuMi needs 76 iterations, whereas ECOS takes about half as many interior point steps (36). The overall computation time for the successfully solved problems are 23 seconds for SeDuMi and 10 seconds for ECOS, which is a speedup factor of about 2.3x.

No.	Name	#vars	rows A	Status		$\lvert f^\star - f\rvert/\lvert f^\star\rvert$		$\lVert Ax-b\rVert/(1+\lVert b\rVert_1)$		# iterations		time [sec]	
				SeDuMi	ECOS	SeDuMi	ECOS	SeDuMi	ECOS	SeDuMi	ECOS	SeDuMi	ECOS
1	nb	2383	123	OK	OK	2.79e-07	9.17e-08	1.77e-07	7.01e-10	18	19	0.57	0.72
2	nb_L1	3176	915	OK	OK	3.23e-05	5.10e-08	4.89e-09	2.87e-16	15	17	0.89	0.67
3	nb_L2	4195	123	numerr	OK	1.70e-01	1.18e-08	4.42e-01	8.43e-12	10	13	1.12	1.15
4	nb_L2_bessel	2641	123	OK	OK	2.04e-05	1.45e-05	1.28e-09	6.08e-12	13	12	0.58	0.56
5	nql30	6302	3680	OK	OK	7.31e-06	3.01e-05	2.67e-09	3.67e-12	11	18	0.34	0.17
6	nql60	25202	14560	OK	OK	4.75e-04	3.96e-04	1.20e-09	5.31e-13	11	19	0.99	1.09
7	qssp30	7566	3691	OK	OK	1.18e-04	8.83e-08	1.91e-09	3.93e-12	18	17	0.49	0.20
8	qssp60	29526	14581	OK	OK	3.31e-04	1.56e-06	8.14e-10	8.38e-13	24	20	2.75	1.64
9	sched_100_100_orig	18240	8338	numerr	numerr	1.00e+00	1.09e-06	9.85e-02	9.85e-02	104	41	8.30	0.06
10	sched_100_100_scaled	18238	8337	OK	OK	2.46e-05	2.46e-05	7.37e-09	5.27e-07	57	46	2.90	1.26
11	sched_100_50_orig	9746	4844	OK	numerr	2.17e-07	3.37e-05	2.13e-09	5.04e+00	38	31	1.72	0.40
12	sched_100_50_scaled	9744	4843	OK	OK	1.85e-05	1.86e-05	4.26e-09	4.16e-08	38	37	1.28	0.46
13	sched_200_100_orig	37889	18087	numerr	numerr	9.97e-01	5.27e-06	1.21e-01	2.83e-03	150	49	19.96	3.88
14	sched_200_100_scaled	37887	18086	OK	OK	5.93e-06	9.84e-06	2.53e-09	2.85e-07	75	36	10.90	2.85
15	sched_50_50_orig	4979	2527	OK	OK	3.68e-08	3.94e-08	5.51e-09	1.18e-06	41	39	0.80	0.23
16	sched_50_50_scaled	4977	2526	OK	OK	5.79e-08	5.60e-08	1.57e-08	6.09e-07	28	26	0.49	0.16

Table 9.3: Results of DIMACS test on MacBook Pro with Intel Core i7 2.6 GHz CPU.

9.5.3 Soft-Constrained Tracking MPC with Stability Guarantees

In the following, the performance of the ECOS solver is investigated for an example from the framework of model predictive control that requires solving an optimization problem with second-order cone constraints. The motivation for this problem arises from the fact that in many practical implementations of MPC, state constraints are usually relaxed to ensure feasibility of the optimization problem, sacrificing closed-loop stability guarantees. One formulation that allows one to relax state constraints while maintaining closed-loop stability is presented in [143]. The MPC problem is formulated for tracking the reference $(x_{\text{ref}}, u_{\text{ref}})$, and the relaxation is achieved through slack variables ϵ_i which, if non-zero, incur a cost. We assume polyhedral constraint sets on x and u, i.e. $\mathbb{U} \triangleq \{u \in \boldsymbol{R}^{n_u} \mid Gu \leq g\}$ with $G \in \boldsymbol{R}^{m_u \times n_u}$, $g \in \boldsymbol{R}^{m_u}$ and $\mathbb{X} \triangleq \{x \in \boldsymbol{R}^{n_x} \mid Fx \leq f$ with $F \in \boldsymbol{R}^{m_x \times n_x}$, $f \in \boldsymbol{R}^{m_x}$ and a horizon length of N. We choose ∞-norm penalties on the slack variables. The soft-constrained MPC problem to be solved is given by [143]:

$$
\begin{aligned}
\min \sum_{i=0}^{N-1} & (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) + \rho \epsilon_i^2 \\
& + (x_N - x_s)^T P (x_N - x_s)^T + \rho \epsilon_N^2 + r_\epsilon \epsilon_N + r_x \|x_s - x_{\text{ref}}\|_\infty + r_u \|u_s - u_{\text{ref}}\|_\infty \\
\text{s.t. } & x_0 = x \,, && \text{(9.68a)} \\
& x_{i+1} = A x_i + B u_i \,, \quad i = 0, \ldots, N-1 \,, && \text{(9.68b)} \\
& x_s = A x_s + B u_s \,, && \text{(9.68c)} \\
& G u_i \leq g \,, \quad i = 0, \ldots, N-1 \,, && \text{(9.68d)} \\
& F x_i \leq f + \epsilon_N + \epsilon_i \,, \quad i = 0, \ldots, N-1 \,, && \text{(9.68e)} \\
& (1+\beta) F x_s \leq f + \epsilon_N \,, && \text{(9.68f)} \\
& ((x_N - x_s), u_s)^T P_E ((x_N - x_s), u_s) \,, && \text{(9.68g)} \\
& c_j (x_N - x_s)^T P_T (x_N - x_s) + F_j x_s \leq f_j + \epsilon_N \,, j = 1, \ldots, m_x && \text{(9.68h)} \\
& \epsilon_i \geq 0 && \text{(9.68i)}
\end{aligned}
$$

For more details on how to compute the parameters of this problem, the reader is referred to [143, §9.3].

In the following, we present computation times for solving instances of problem (9.68) for the oscillating masses benchmark system from Chapter 8. Unlike the portfolio problem in Section 9.5.1, where we have many variables and essentially just one large second-order cone constraint, this benchmark has relatively few variables but many small- to medium-sized second-order cone constraints. In this sense, this benchmark is therefore complementary to the portfolio problem. In addition, we expect the ordering of the KKT matrix to be more difficult than in the portfolio problem due to the complex non-zero patterns and inter-

dependencies of variables. We reformulate the objective into a linear function by using the epigraph formulation

$$x^T Q x \leq t \;\Leftrightarrow\; \left\| \begin{array}{c} Q^{1/2} x \\ (t-1)/2 \end{array} \right\|_2 \leq (t+1)/2\,, \tag{9.69}$$

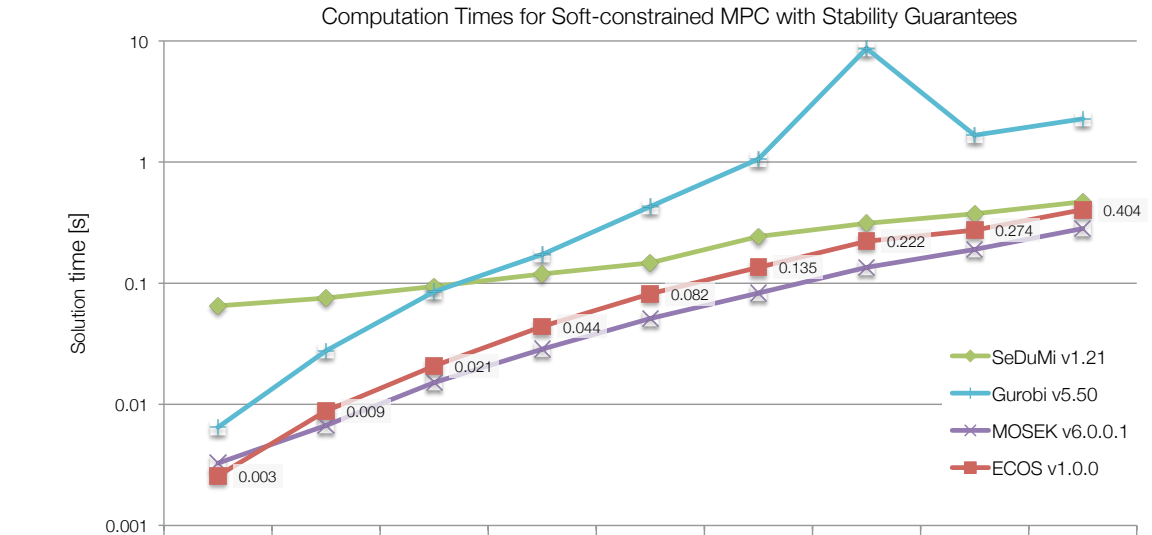
where we use a sparse Cholesky decomposition to compute $Q^{1/2}$.

The computation times for solving (9.68) to an accuracy of $\texttt{FEASTOL} = \texttt{RELTOL} = \texttt{ABSTOL} = 10^{-6}$ using various solvers is shown in Figure 9.3. We choose $Q = 3I_{n_x}$ and $R = I_{n_u}$, $s = r_\epsilon = r_x = r_u = 100$ and the constraints $-4 \leq x_i \leq 4$ and $-0.5 \leq u_i \leq 0.5$. For each test problem, the presented computation time is the average value of the time taken by each solver for 20 randomly sampled initial values. It can be seen that despite the complex constraint structure, the overall solve speed of ECOS is well below the sampling time of 0.5 seconds, being about an order of magnitude faster than SeDuMi for the smaller problems, and about a factor of five to ten faster than Gurobi for this example. Most importantly, the performance of ECOS scales very well with increasing problem sizes, being consistently close in performance to the overall fastest solver MOSEK. In this test, the Gurobi solve times are prohibitively large for bigger problems. The spike in the computation time for 14 masses or 28 states, respectively, is due to numerical issues (a jump in the number of iterations is observed for this problem). Interestingly, this happens only for some initial states, while all other solvers, including ECOS, show a significantly lower variation in the number of iterations, and hence also in the solution time.

The presented numbers show that ECOS is suitable to solve even complex MPC problems in the milliseconds range, which is sufficient for most mechanical systems with fast dynamics.

9.6 Conclusion

We have presented ECOS, an efficient implementation of an interior-point solver for SOCPs targeting embedded systems. Using a newly proposed, dynamically regularized symmetric indefinite KKT system, search directions can be computed in a stable manner by sparse LDL factorization with fixed pivoting. This avoids dynamic memory allocation during solves and results in only 800 lines of ANSI C source code for the solver, including all linear algebra functionality. This makes ECOS suitable and attractive for use on embedded systems; in fact, it could be certified for code safety (such as for no divisions by zero) with reasonable effort. Despite its simplicity, ECOS is competitive to established desktop solvers for small and medium-sized problems.



non-zeros in [A;G]	*861*	*2844*	*6693*	*15208*	*26411*	*42113*	*63085*	*90090*	*123901*
SOC constraints	*10x3 | 10x6 | 10x3 | 1x10 | 8x5*	*10x5 | 10x10 | 10x3 | 1x20 | 16x9*	*10x7 | 10x14 | 10x3 | 1x30 | 24x13*	*10x9 | 10x18 | 10x3 | 1x40 | 32x17*	*10x11 | 10x22 | 10x3 | 1x50 | 40x21*	*10x13 | 10x26 | 10x3 | 1x60 | 48x25*	*10x15 | 10x30 | 10x3 | 1x70 | 56x29*	*10x17 | 10x34 | 10x3 | 1x80 | 64x33*	*10x19 | 10x38 | 10x3 | 1x90 | 72x37*
Linear constraints	*120*	*252*	*384*	*516*	*648*	*780*	*912*	*1044*	*1176*
# of variables	*97*	*163*	*229*	*295*	*361*	*427*	*493*	*559*	*625*
# of inputs	*1*	*3*	*5*	*7*	*9*	*11*	*13*	*15*	*17*
# of states	*4*	*8*	*12*	*16*	*20*	*24*	*28*	*32*	*36*
horizon length	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*	*10*
# of masses	*2*	*4*	*6*	*8*	*10*	*12*	*14*	*16*	*18*

Figure 9.3: Computation times for solving the soft-constrained MPC tracking problem (9.68) for the oscillating masses system on a MacBook Pro with Intel Core i7 CPU at 2.6 GHz. Solution times are averaged for 20 randomly sampled (feasible) initial states for different number of masses in the chain. The notation kxn denotes that the problem has k SOCs of size n.

10 Discussion and Outlook

The goal of this thesis was to make sophisticated optimal decision making methods in the context of optimal control available on embedded systems. We have presented both suboptimal explicit methods, which construct a decision function from simulation data with the optimal controller, and iterative optimization methods supporting convex LPs, QPs, QCQPs and SOCPs for a direct implementation on embedded platforms.

The results show that standard machine learning algorithms work remarkably well in practice for learning controllers from data samples. We have addressed a key challenge for the use of learning methods in control and have derived tractable constraints that can be introduced into any learning formulation based on convex optimization to preserve the desired system theoretic guarantees, ensuring closed-loop constraint satisfaction and stability. An interesting future direction is to investigate the use of different learning techniques, e.g. using polynomial or piece-wise affine functions as basis functions to further incorporate knowledge of the resulting controller structure.

For the synthesis of approximate explicit controllers for binary decisions, we have proposed to use AdaBoost classifiers with decision stumps as weak learners, which result in simple if-then-else rules that are meaningful to human system operators and easy to implement on resource constrained platforms. Since the simulation study shows promising results and demonstrates high performance and significant complexity reduction, the next step will be to investigate the performance in practice and implement the method in a building control system. An interesting question beyond the scope of the considered application is how to appropriately select the training data such as to resemble practically relevant scenarios. This is particularly challenging, as the state space is high dimensional and complete coverage of all operating conditions is infeasible. While we have shown how theoretical guarantees can be imposed for learning continuous inputs, the extension to binary decisions represents a challenge to be addressed in future work. Another interesting direction for future research is *feature design*, addressing the question of which data is necessary to infer a certain decision, and whether it is possible to engineer new features that are effective and easy to obtain. For example, average values of simple weather forecasts from public news might be sufficient for an energy efficient control of buildings, making expensive, high-resolution forecast data from meteorological services obsolete in control practice.

The second part of the thesis was concerned with iterative optimization methods for solving convex optimization problems efficiently on embedded platforms. One approach has been to restrict the class of supported problems to multistage problems and to tailor primal-dual interior point methods to the specific problem structure. A code generator system was established that analyzes the given problem, and exploits as much structure as possible to generate a highly efficient solver with small code size. The approach can be taken further in various directions: First, an intuitive modeling language that allows a user to model multistage problems without detailed knowledge about the problem structure would considerably facilitate the use of the proposed method, possibly increasing the degree of structure exploitation beyond the user knowledge. Automatic structure recognition of stage variables, their grouping, etc. are graph partitioning problems for which efficient algorithms exist. Second, exploitation of parallelism, both on multi-core CPUs and on FPGAs could be further developed to increase the level of parallelism in the generated solvers. Third, enhancing the code generator system with sparse linear algebra routines would improve the performance of the generated solvers for large-scale sparse multistage problems.

In addition to an embedded solver for problems with quadratic constraints, we have presented a very efficient interior point solver for optimization over second-order cones, ECOS, and introduced a novel sparse representation of the symmetric indefinite KKT matrix, which can be factored in a numerically stable manner for any fixed permutation. The design of ECOS paves the way for a code generation system, which would execute the *Setup* routine at code generation time, allocating memory and computing the elimination ordering, and would write out the *Solve* procedure with all dimensions defined at compilation time. In this context, the *Setup* routine could be enhanced with different ordering schemes, possibly investing more time offline for finding a good ordering of the KKT matrix. If the code has fixed dimensions for all `for`-loops, compiler optimizations such as loop unrolling, which are otherwise not possible, are enabled. Our experience with the FORCES code generator in Chapter 8 showed that such automatic compiler optimizations can significantly accelerate the solver computations.

Part IV

Appendix

A Definitions and Facts from Convex Analysis

A.1 Self-Dual Embeddings

Lemma A.1. *Problem* (HSD) *is self-dual.*

Proof. The dual problem of (HSD) is given by

$$\begin{array}{ll} \max\limits_{\substack{u,v,\gamma,\\ w \succeq_K 0}} \ \min\limits_{\substack{x,y\\ z \succeq_K 0\\ (\tau,\kappa)\geq 0}} & u^T(G^T z + A^T y + \tau\cdot c) + v^T(Ax - \tau\cdot b) \\ & \quad + w^T(Gx - \tau\cdot h) \\ & \quad + \gamma(-c^T x - b^T y - h^T z - \kappa) \end{array} \tag{A.1}$$

By collecting the primal variables, we can rewrite (A.1) to

$$\begin{array}{ll} \max\limits_{\substack{u,v,\gamma,\\ w \succeq_K 0}} \ \min\limits_{\substack{x,y\\ (s,z) \succeq_K 0\\ (\tau,\kappa)\geq 0}} & x^T(G^T w + A^T v - \gamma\cdot c) + y^T(Au - \gamma\cdot b) \\ & \quad + z^T(Gu - \gamma\cdot h) \\ & \quad + \tau(c^T u - b^T v - h^T w) \\ & \quad + \kappa(-\gamma) \end{array} \tag{A.2}$$

Each primal variable now multiplies an expression of dual variables, which can be put as appropriate constraints to the outer maximization instead. The expression of dual variables has to lie in the dual cone K^*, e.g. in $K^* = \{0\}$ for $K = \mathcal{R}^n$, for instance. Note that the LP- and second order cone are self-dual. Carrying out this step yields

$$\begin{array}{lll} \max\limits_{u,v,w,\gamma} & 0 & \\ \text{subject to} & G^T w + A^T v - \gamma\cdot c = 0 & \\ & Au - \gamma\cdot b = 0 & \\ & Gu - \gamma\cdot h \succeq_K 0 & \\ & c^T u - b^T v - h^T w \geq 0 & \\ & w \succeq_K 0 & \\ & -\gamma \geq 0 & \end{array} \tag{A.3}$$

Finally, by rewriting the inequalities in terms of equalities and simple inequalities and by changing to minimization, we arrive at

$$\begin{aligned} \min_{u,v,w,\mu,\alpha,\beta} \quad & 0 \\ \text{subject to} \quad & G^T w + A^T v - \gamma \cdot c = 0 \\ & Au - \gamma \cdot b = 0 \\ & Gu - \mu - \gamma \cdot h = 0 \\ & c^T u - b^T v - h^T w - \beta = 0 \\ & (w, \mu) \succeq_{\boldsymbol{K}} 0 \\ & (-\gamma, \beta) \geq 0 \end{aligned} \tag{A.4}$$

which, using the substitutions $u = -x$, $\mu = s$, $v = y$, $w = z$, $-\gamma = \tau$ and $\beta = \kappa$ is equal to the initial primal problem (HSD).

A.2 Theorem of Alternatives for Generalized Inequalities

Lemma A.2 (Farkas' Lemma for Generalized Inequalities [19, Example 2.26])**.** *Let $\boldsymbol{K} \subseteq \boldsymbol{R}^m$ be a proper cone. For any matrix $A \in \boldsymbol{R}^{m \times n}$ and vectors $x \in \boldsymbol{R}^n$, $b \in \boldsymbol{R}^m$ and $\lambda \in \boldsymbol{R}^m$ exactly one of the following statements is true:*

(i) $Ax \preceq_{\boldsymbol{K}} b$, *or*

(ii) $\lambda \neq 0\,, \quad \lambda \succeq_{\boldsymbol{K}^*} 0\,, \quad A^T\lambda = 0\,, \quad \lambda^T b \leq 0.$

□

B Results and Formulations in Model Predictive Control

B.1 Reference Tracking

In reference tracking, the goal is to steer the output $y \in \boldsymbol{R}^{n_y}$ of system (4.2),

$$y = Cx + Du\,, \tag{B.1}$$

to a desired target y_t. The associated target steady states (x_t, u_t) satisfy the relation [79]

$$\begin{bmatrix} A - I_{n_x} & B & 0 \\ C & D & -I_{n_y} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \\ y_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\,. \tag{B.2}$$

Assumption 4.1 ensures that (B.2) has a well-defined solution [97]. In particular, a minimal parameterization of (x_t, u_t, y_t) is

$$(x_t, u_t, y_t) = M\theta\,, \tag{B.3}$$

where $\theta \in \boldsymbol{R}^{n_\theta}$ is a parameter defining all solutions and $M \in \boldsymbol{R}^{n_x+n_u+n_y \times n_\theta}$. The matrix M is a basis for the nullspace of the coefficient matrix in (B.2) and can be calculated by a singular value decomposition.

The usual way to incorporate reference tracking into the basic MPC formulation (4.7) is to shift the origin to the target steady state (x_t, u_t) [97]:

$$\min\ (x_N - x_t)^T P (x_N - x_t) + \sum_{i=0}^{N-1} (x_i - x_t)^T Q (x_i - x_t) + (u_i - u_t)^T R (u_i - u_t) \tag{B.4a}$$

$$\text{s.t. } x_0 = x\,, \tag{B.4b}$$

$$x_{i+1} = Ax_i + Bu_i\,, i = 0, \ldots, N-1\,, \tag{B.4c}$$

$$(x_i, u_i) \in \mathbb{X} \times \mathbb{U}\,, i = 0, \ldots, N-1\,, \tag{B.4d}$$

$$x_N \in \mathbb{X}_f(x_t, u_t)\,. \tag{B.4e}$$

The particular difficulty with this formulation is that, in general, the terminal constraint set $\mathbb{X}_f$ depends on the target steady state (x_t, u_t). Therefore, unlike for (4.9), no stability guarantees can be given for (B.4), since recursive feasibility is no longer ensured when the target (x_t, u_t) changes. One approach to overcome these difficulties has been proposed by Limon et al. in [79], where by introduction of an artificial steady-state (x_s, u_s), which is an optimization variable, the terminal set remains constant:

$$\min\ (x_N - x_s)^T P (x_N - x_s) + \sum_{i=0}^{N-1} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) \quad \text{(B.5a)}$$

$$+ (x_s - x_t)^T T (x_s - x_t) \quad \text{(B.5b)}$$

$$\text{s.t. } x_0 = x\,, \quad \text{(B.5c)}$$

$$x_s = A x_s + B u_s\,, \quad \text{(B.5d)}$$

$$x_{i+1} = A x_i + B u_i\,, i = 0, \ldots, N-1\,, \quad \text{(B.5e)}$$

$$(x_i, u_i) \in \mathbb{X} \times \mathbb{U}\,, i = 0, \ldots, N-1\,, \quad \text{(B.5f)}$$

$$(x_s, u_s) \in \mathbb{X} \times \mathbb{U}\,, \quad \text{(B.5g)}$$

$$(x_N, x_s, u_s) \in \mathbb{X}_f^{\text{tr}}\,, \quad \text{(B.5h)}$$

where $T \in \boldsymbol{S}_{++}^{n_x}$, $\mathbb{X}_f^{\text{tr}}$ is a PI set for tracking. Recursive feasibility and closed loop stability for the receding horizon control law $\kappa(x, x_t, u_t)^{\text{tr}} \triangleq u_0^*(x, x_t, u_t)$ can be shown. For further details see [79] and [143, §5.4.1].

B.2 Multistage Problem Matrices for Robust Real-time Tracking Problem

In the following, we give the matrices necessary to convert

B.2.1 Cost function

The following matrices model the cost function $1/2 z_i^T H_i z + f_i^T z_i$ in the multi-stage framework (note that f_0 is a parameter):

$$H_0 = \begin{bmatrix} Q + P & -QM_x & & & & \\ -M_x^T Q & M_x^T Q M_x + M_u^T R M_u + T_\theta & -M_u^T R & & & \\ & 0 & & & & \\ & -RM_u & R & & & \\ & & & 0 & & \\ & & & & 0 & \end{bmatrix} \quad \text{(B.6a)}$$

$$f_0 = \begin{bmatrix} -x^T P & -\theta_r^T T_\theta & 0 & 0_{n_u}^T & 0 & 0 \end{bmatrix}^T \tag{B.6b}$$

$$H_i = \begin{bmatrix} Q & -QM_x & & & & \\ -M_x^T Q & M_x^T Q M_x + M_u^T R M_u & -M_u^T R & & & \\ & & 0 & & & \\ & -RM_u & & R & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix}, \quad i = 1, \ldots, N-1 \tag{B.6c}$$

$$f_i = 0_{n_z}^T, \quad , i = 1, \ldots, N-1 \tag{B.6d}$$

$$H_N = \begin{bmatrix} P & -PM_x & & \\ -M_x^T P & P & & \\ & & 0 & \\ & & & 0 \end{bmatrix}, \tag{B.6e}$$

$$f_N = 0_{n_x+n_\theta+2}^T. \tag{B.6f}$$

B.2.2 Equality constraints

B.2.2.1 Dynamics

- $i = 0$:

$$\underbrace{\begin{bmatrix} 0 & 0_{n_\theta}^T & 1 & 0_{n_u}^T & 0 & 0 \\ A & 0_{n_x \times n_\theta} & 0_{n_x} & B & 0_{n_x} & 0_{n_x} \\ 0_{n_\theta \times n_x} & I_{n_\theta} & 0_{n_\theta} & 0_{n_\theta \times n_u} & 0_{n_\theta} & 0_{n_\theta} \\ 0_{n_x}^T & 0_{n_\theta}^T & 1 & 0_{n_u}^T & -1 & -1 \end{bmatrix}}_{C_i} z_0 \tag{B.7a}$$

$$+ \underbrace{\begin{bmatrix} 0 & 0_{n_\theta}^T & 0 & 0_{n_u}^T & 0 & 0 \\ -I_{n_x \times n_x} & 0_{n_x \times n_\theta} & 0_{n_x} & 0_{n_x \times n_u} & 0_{n_x} & 0_{n_x} \\ 0_{n_\theta \times n_x} & -I_{n_\theta} & 0_{n_\theta} & 0_{n_\theta \times n_u} & 0_{n_\theta} & 0_{n_\theta} \\ 0_{n_x}^T & 0_{n_x}^T & -1 & 0_{n_u}^T & 0 & 0 \end{bmatrix}}_{D_1} z_1 = \underbrace{\begin{bmatrix} \Pi \\ 0_{n_x} \\ 0_{n_\theta} \\ 0 \end{bmatrix}}_{c_0}$$

Note that c_0 is a parameter.

- $i = 1, \ldots, N-2$:

$$\underbrace{\begin{bmatrix} A & 0_{n_x \times n_\theta} & 0_{n_x} & B & 0_{n_x} & 0_{n_x} \\ 0_{n_\theta \times n_x} & I_{n_\theta} & 0_{n_\theta} & 0_{n_\theta \times n_u} & 0_{n_\theta} & 0_{n_\theta} \\ 0_{n_x}^T & 0_{n_\theta}^T & 1 & 0_{n_u}^T & -1 & -1 \end{bmatrix}}_{C_i} z_i \tag{B.7b}$$

$$+ \underbrace{\begin{bmatrix} -I_{n_x \times n_x} & 0_{n_x \times n_\theta} & 0_{n_x} & 0_{n_x \times n_u} & 0_{n_x} & 0_{n_x} \\ 0_{n_\theta \times n_x} & -I_{n_\theta} & 0_{n_\theta} & 0_{n_\theta \times n_u} & 0_{n_\theta} & 0_{n_\theta} \\ 0_{n_x}^T & 0_{n_x}^T & -1 & 0_{n_u}^T & 0 & 0 \end{bmatrix}}_{D_{i+1}} z_{i+1} = \underbrace{\begin{bmatrix} 0_{n_x} \\ 0_{n_\theta} \\ 0 \end{bmatrix}}_{c_{i+1}}$$

- $i = N$:

$$C_1 z_{N-1} + \underbrace{\begin{bmatrix} -I_{n_x \times n_x} & 0_{n_x \times n_\theta} & 0_{n_x} & 0_{n_x} \\ 0_{n_\theta \times n_x} & -I_{n_\theta} & 0_{n_\theta} & 0_{n_\theta} \\ 0_{n_x}^T & 0_{n_x}^T & -1 & 0 \end{bmatrix}}_{D_N} z_N = \underbrace{\begin{bmatrix} 0_{n_x} \\ 0_{n_\theta} \\ 0 \end{bmatrix}}_{c_N}$$

B.2.3 Inequality constraints

- i=0:

$$x_0^T T x_0 + (-2x^T T) x_0 \leq r_1(x) \tag{B.8a}$$

$$\underline{x} \leq x_0 \leq \bar{x}, \quad \underline{u} \leq u_0 \leq \bar{u} \tag{B.8b}$$

$$\frac{1}{2} \begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix}^T \begin{bmatrix} Q + P & -QM_x \\ -M_x^T Q & M_x^T Q M_x + T_\theta \end{bmatrix} \begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix} + \begin{bmatrix} -x^T P \\ -\theta_r^T T_\theta \end{bmatrix}^T \begin{bmatrix} x_0 \\ \theta_0 \end{bmatrix} - \gamma_0 \leq r_2(x, \theta_r) \tag{B.8c}$$

$$\frac{1}{2} \begin{bmatrix} \theta_0 \\ u_0 \end{bmatrix}^T \begin{bmatrix} M_u^T R M_u & -M_u^T R \\ -M_u R & R \end{bmatrix} \begin{bmatrix} \theta_0 \\ u_0 \end{bmatrix} - \delta_0 \leq 0 \tag{B.8d}$$

where

$$r_1(x) \triangleq 1 - x^T T x \tag{B.9a}$$

$$r_2(x, \theta_r) \triangleq -\frac{1}{2} x^T P x - \frac{1}{2} \theta_r^T T_\theta \theta_r \tag{B.9b}$$

are parameters.

- i=1,...,N-1:

$$\underline{x} \leq x_i \leq \bar{x}, \quad \underline{u} \leq u_i \leq \bar{u} \tag{B.10a}$$

$$\frac{1}{2}\begin{bmatrix} x_i \\ \theta_i \end{bmatrix}^T \begin{bmatrix} Q & -QM_x \\ -M_x^T Q & M_x^T Q M_x \end{bmatrix} \begin{bmatrix} x_i \\ \theta_i \end{bmatrix} - \gamma_i \leq 0 \tag{B.10b}$$

$$\frac{1}{2}\begin{bmatrix} \theta_i \\ u_i \end{bmatrix}^T \begin{bmatrix} M_u^T R M_u & -M_u^T R \\ -R M_u & R \end{bmatrix} \begin{bmatrix} \theta_i \\ u_i \end{bmatrix} - \delta_i \leq 0 \tag{B.10c}$$

- i=N:

$$\frac{1}{2}\begin{bmatrix} x_N \\ \theta_N \end{bmatrix}^T \begin{bmatrix} P & -PM_x \\ -M_x^T P & M_x^T P M_x \end{bmatrix} \begin{bmatrix} x_N \\ \theta_N \end{bmatrix} - \gamma_N \leq 0 \tag{B.11a}$$

$$\begin{bmatrix} x_N \\ \theta_N \end{bmatrix}^T P_E \begin{bmatrix} x_N \\ \theta_N \end{bmatrix} \leq 1 \tag{B.11b}$$

$$\gamma_N - J_N \leq 0 \tag{B.11c}$$

$$\begin{bmatrix} M_x^T & M_u^T & -M_x^T & -M_u^T \end{bmatrix}^T \theta_N \leq \begin{bmatrix} \bar{x}^T & \bar{u}^T & -\underline{x}^T & -\underline{u}^T \end{bmatrix}^T \tag{B.11d}$$

Bibliography

[1] B. Acikmese and S. R. Ploen. Convex programming approach to powered descent guidance for Mars landing. *Journal of Guidance, Control and Dynamics*, 30(5):1353–1366, 2007.

[2] A. Alessio and A. Bemporad. A survey on explicit model predictive control. In *Nonlinear model predictive control*, pages 345–369. Springer, 2009.

[3] A. A. Alizadeh et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–511, 2000.

[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. on Math. Software*, 30(3):381–388, 2004.

[5] E. Andersen and K. Andersen. The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. *High Performance Optimization*, 33:197–232, 2000.

[6] M. Andersen, J. Dahl, Z. Liu, and Lieven Vandenberghe. Interior-point methods for large-scale cone programming. In *Optimization for Machine Learning*. MIT Press, 2011.

[7] M. S. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: A python package for convex optimization, version 1.1.5. abel.ee.ucla.edu/cvxopt, 2012.

[8] M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM Journal on Matrix Analysis and Applications*, 10(2):165–190, 1989.

[9] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[10] F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[11] G. Beccuti, G. Papafotiou, and L. Harnefors. Multivariable predictive control of voltage source converter HVDC transmission systems. In *IEEE International Symposium on Industrial Electronics (ISIE)*, pages 3145 –3152, July 2010.

[12] A. Bemporad and C. Filippi. Suboptimal explicit mpc via approximate multiparametric quadratic programming. In *IEEE Conf. Decision and Control*, volume 5, pages 4851 –4856 vol.5, 2001.

[13] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[14] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The Explicit Linear Quadratic Regulator for Constrained Systems. *Automatica*, 38(1):3–20, Jan. 2002.

[15] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace. Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations. *Automatic Control, IEEE Transactions on*, 56(12):2883–2897, 2011.

[16] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

[17] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations research*, 50(1):3–15, 2002.

[18] F. Borrelli. *Constrained Optimal Control of Hybrid Systems*, volume 290. Springer, 2003.

[19] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[20] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[21] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[22] J. Bunch and B. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, 1971.

[23] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[24] E. J. Candes and P. A. Randall. Highly robust error correction byconvex programming. *Information Theory, IEEE Transactions on*, 54(7):2829–2840, 2008.

[25] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[26] M. Chu, Y. Zinchenko, S. G. Henderson, and M. B. Sharpe. Robust optimization for intensity modulated radiation therapy treatment planning under uncertainty. *Physics in Medicine and Biology*, 50(23):5463, 2005.

[27] B. Coffey. *Using Building Simulation and Optimization to Calculate Lookup Tables for Control*. PhD thesis, UC Berkeley: Center for the Built Environment, 2012.

[28] O. C. Consortium. OptiControl webpage. `http://www.opticontrol.ethz.ch`, July

2011.

[29] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[30] F. Cucker and D.-X. Zhou. *Learning Theory: an Approximation Theory Viewpoint*. Cambridge University Press, Cambridge, 2007.

[31] T. A. Davis. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Transactions on Mathematical Software (TOMS)*, 31(4):587–591, 2005.

[32] T. A. Davis. *Direct methods for sparse linear systems*, volume 2. Society for Industrial and Applied Mathematics, 2006.

[33] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):377–380, 2004.

[34] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[35] A. Domahidi. FORCES: Fast optimization for real-time control on embedded systems. `http://forces.ethz.ch`, Oct. 2012.

[36] A. Domahidi, E. Chu, and S. Boyd. Ecos: An socp solver for embedded systems. *European Control Converence*, 2013.

[37] A. Domahidi, F. Ullmann, M. Morari, and C. Jones. Learning near-optimal decision rules for energy efficient building control. In *IEEE Conference on Decision and Control*, pages 7571 – 7576, Maui, HI, USA, Dec. 2012.

[38] A. Domahidi, M. Zeilinger, M. Morari, and C. Jones. Learning a feasible and stabilizing explicit model predictive control law by robust optimization. In *IEEE Conference on Decision and Control*, pages 513–519, Orlando, FL, USA, Dec. 2011.

[39] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *Conference on Decision and Control (CDC)*, pages 668–674, Maui, HI, USA, Dec. 2012.

[40] I. S. Duff. MA57 – a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw.*, 30:118–144, June 2004.

[41] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, 1986.

[42] A. El-Keyi, T. Kirubarajan, and A. B. Gershman. Robust adaptive beamforming based on the kalman filter. *Signal Processing, IEEE Transactions on*, 53(8):3032–3041, 2005.

[43] J. Faraut and A. Korányi. *Analysis on Symmetric Cones*. Oxford University Press, 1994.

[44] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18:816–830, 2008.

[45] H. J. Ferreau, P. Ortner, P. Langthaler, L. del Re, and M. Diehl. Predictive control of a real-world diesel engine using an extended online active set strategy. *Annual Reviews in Control*, 31(2):293–301, 2007.

[46] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In P. Vitányi, editor, *Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin / Heidelberg, 1995.

[47] D. Frick, A. Domahidi, M. Vukov, S. Mariethoz, M. Diehl, and M. Morari. Moving horizon estimation for induction motors. In *Sensorless Control for Electrical Drives (SLED), 2012 IEEE Symposium on*, pages 1–6. IEEE, 2012.

[48] M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, Dec. 2001.

[49] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29:58–81, March 2003.

[50] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computations*, 28:505–535, 1974.

[51] P. E. Gill, M. A. Saunders, and J. R. Shinnerl. On the stability of cholesky factorization for symmetric quasidefinite systems. *SIAM Journal on Matrix Analysis and Applications*, 17(1):35–46, 1996.

[52] D. Goldfarb and G. Iyengar. Robust portfolio selection problems. *Mathematics of Operations Research*, 28(1):1–38, 2003.

[53] D. Goldfarb and K. Scheinberg. A product-form cholesky factorization method for handling dense columns in interior point methods for linear programming. *Mathematical Programming*, 99(1):1–34, 2004.

[54] D. Goldfarb and K. Scheinberg. Product-form cholesky factorization in interior point methods for second-order cone programming. *Mathematical programming*, 103(1):153–179, 2005.

[55] P. J. Goulart, E. C. Kerrigan, and J. M. Maciejowski. Optimization over state feedback policies for robust control with constraints. *Automatica*, 42(4):523–533, 2006.

[56] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming. `http://cvxr.com/cvx`, Apr. 2011.

[57] P. Grieder and M. Morari. Complexity Reduction of Receding Horizon Control. In *IEEE Conference on Decision and Control*, pages 3179–3184, Maui, Hawaii, Dec.

2003.

[58] G. Gu, M. Zangiabadi, and C. Roos. Full Nesterov-Todd step infeasible interior-point method for symmetric optimization. *Eur. Jnl. of Operational Research*, 214(3):473–484, 2011.

[59] Gurobi Optimization, Inc., Houston, Texas. *Gurobi optimizer reference manual version 5.0*, July 2012.

[60] M. Gwerder, D. Gyalistras, F. Oldewurtel, B. Lehmann, K. Wirth, V. Stauch, and J. Tödtli. Potential assessment of rule-based control for integrated room automation. In *Clima - RHEVA World Congress*, Antalya, Turkey, May 2010.

[61] D. Gyalistras and M. Gwerder. Use of weather and occupancy forecasts for optimal building climate control (opticontrol): Two years progress report. Technical report, ETH Zurich, Switzerland and Siemens Building Technologies Division, Siemens Switzerland Ltd., Zug, Switzerland, 2009.

[62] D. Gyalistras, M. Gwerder, F. Oldewurtel, C. Jones, M. Morari, B. Lehmann, K. Wirth, and V. Stauch. Analysis of Energy Savings Potentials for Integrated Room Automation. In *Clima - RHEVA World Congress*, Antalya, Turkey, May 2010.

[63] L. Hermes and J. Buhmann. Feature selection for support vector machines. In *Int. Conf. Pattern Recognition*, volume 2, pages 712–715, 2000.

[64] L. Hermes and J. Buhmann. Feature selection for support vector machines. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 712 –715 vol.2, 2000.

[65] J. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2001.

[66] B. Houska, H. J. Ferreau, and M. Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[67] International Business Machines Corp. (IBM). IBM ILOG CPLEX Optimization Studio. `http://www.ibm.com/software/commerce/optimization/cplex-optimizer`, July 2013.

[68] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. Fpga implementation of an interior point solver for linear model predictive control. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 316–319. IEEE, 2010.

[69] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded predictive control on an fpga using the fast gradient method. In *Proceedings of the European Control Conference*, Zurich, Switzerland, July 2013.

[70] C. N. Jones, P. Grieder, and S. Rakovic. A Logarithmic-Time Solution to the Point

Location Problem for Parametric Linear Programming. *Automatica*, 42(12):2215–2218, Dec. 2006.

[71] C. N. Jones and M. Morari. The Double Description Method for the Approximation of Explicit MPC Control Laws. In *IEEE Conf. Decision and Control*, Cancun, Mexico, Dec. 2008.

[72] C. N. Jones and M. Morari. Approximate Explicit MPC using Bilevel Optimization. In *European Control Conference*, Budapest, Hungary, Aug. 2009.

[73] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 3:373–395, 1984.

[74] G. Karypis and V. Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. `http://glaros.dtc.umn.edu/gkhome/views/metis`, 1998.

[75] R. B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, 1996.

[76] A. Kelman and F. Borrelli. Bilinear model predictive control of a hvac system using sequential quadratic programming. In *Proceedings of the IFAC World Congress*, 2011.

[77] M. Kvasnica, J. Löfberg, M. Herceg, L. Cirka, and M. Fikar. Low-complexity polynomial approximation of explicit mpc via linear programming. In *American Control Conference (ACC), 2010*, pages 4713–4718. IEEE, 2010.

[78] D. Lam, C. Manzie, and M. Good. Model predictive contouring control. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 6137–6142. IEEE, 2010.

[79] D. Limon, I. Alvarado, T. Alamo, and E. Camacho. Mpc for tracking piecewise constant references for constrained linear systems. *Automatica*, 44(9):2382–2387, 2008.

[80] A. Liniger. Autonomous drift control. Master's Thesis, Automatic Control Laboratory, ETH Zurich, December 2012.

[81] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1-3):193–228, 1998.

[82] J. Löfberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD*, Taipei, Taiwan, 2004.

[83] P. J. Lute. *The use of predictions in temperature control in buildings*. Delft University of Technology, Faculty of Mechanical Engineering and Marine Technology, 1992.

[84] Y. Ma, A. Kelman, A. Daly, and F. Borrelli. Predictive control for energy efficient buildings with thermal storage: Modeling, stimulation, and experiments. *Control Systems, IEEE*, 32(1):44 –64, Feb. 2012.

[85] S. Mariéthoz, A. Domahidi, and M. Morari. Sensorless explicit model predictive control

of permanent magnet synchronous motors. In *IEEE International Electric Machines and Drives Conference (IEMDC)*, pages 1250 –1257, May 2009.

[86] S. Mariéthoz, A. Domahidi, and M. Morari. High-Bandwidth Explicit Model Predictive Control of Electrical Drives. *IEEE Trans. on Industry Applications*, 48(6):1980 – 1992, Dec. 2012.

[87] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13:1–27, 2012.

[88] J. Mattingley, Y. Wang, and S. Boyd. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems*, 31(3):52 –65, June 2011.

[89] P. May-Ostendorp, G. P. Henze, C. D. Corbin, B. Rajagopalan, and C. Felsmann. Model-predictive control of mixed-mode buildings with rule extraction. *Building and Environment*, 46(2):428 – 437, 2011.

[90] D. Mayne, M. Seron, and S. Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219 – 224, 2005.

[91] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.

[92] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, November 1992.

[93] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[94] S. Merkli. Extension of forces solver for the use of programmable logic. Semester project report, Automatic Control Laboratory, ETH Zurich, July 2013.

[95] H. Michalska and D. Mayne. Moving horizon observers and observer-based control. *IEEE Trans. Automatic Control*, 40(6):995 –1006, June 1995.

[96] H. D. Mittelmann. An independent benchmarking of sdp and socp solvers. *Mathematical Programming*, 95(2):407–430, 2003.

[97] K. R. Muske and J. B. Rawlings. Model predictive control with linear models. *AIChE Journal*, 39(2):262–287, 1993.

[98] Y. Nesterov. *Introductory Lectures on Convex Optimization. A Basic Course*. Springer, 2004.

[99] Y. Nesterov and A. Nemirovski. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial Mathematics, 1994.

[100] Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[101] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd

edition, 2006.

[102] F. Oldewurtel. *Stochastic Model Predictive Control for Building Climate Control*. PhD thesis, ETH Zurich, Zurich, Switzerland, Aug. 2011.

[103] F. Oldewurtel, A. Parisio, C. Jones, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and M. Morari. Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, 45:15–27, Feb. 2012.

[104] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*, volume 30. Society for Industrial and Applied Mathematics, 1987.

[105] G. Pannocchia, J. B. Rawlings, and S. J. Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43(5):852–860, May 2007.

[106] T. Parisini and R. Zoppoli. A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10):1443–1451, 1995.

[107] G. Pataki and S. Schmieta. The dimacs library of semidefinite-quadratic-linear programs. *Computational Optimization Research, Columbia University, NY*, 1999.

[108] P. Patrinos and A. Bemporad. An accelerated dual gradient-projection algorithm for linear model predictive control. In *Conference on Decision and Control (CDC), Maui, HI, USA*, pages 662–667, December 2012.

[109] L. Pérez-Lombard, J. Ortiz, and C. Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394 – 398, 2008.

[110] S. Prívara, J. Široký, L. Ferkl, and J. Cigler. Model predictive control of a building heating system: The first experience. *Energy and Buildings*, 43:564 – 572, 2011.

[111] P.Tondel, T.A.Johansen, and A.Bemporad. Evaluation of piecewise affine control via binary search tree. *Automatica*, 39(5):945–950, 2003.

[112] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of Interior-Point methods to model predictive control. *Journal of Optimization Theory and Applications*, 99(3):723–757, Dec. 1998.

[113] S. Richter. *Computational Complexity Certification of Gradient Methods for Real-Time Model Predictive Control*. PhD thesis, ETH Zurich, Switzerland, 2012.

[114] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

[115] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[116] P. Scokaert, D. Mayne, and J. Rawlings. Suboptimal model predictive control (feasibility implies stability). *Automatic Control, IEEE Transactions on*, 44(3):648–654, Mar. 1999.

[117] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding

to a change in one element of a given matrix. *Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[118] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[119] S. Sojoudi and J. Lavaei. Physics of power networks makes hard optimization problems easy to solve. In *Power and Energy Society General Meeting, 2012 IEEE*, pages 1–8. IEEE, 2012.

[120] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[121] J. F. Sturm. Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[122] S. Summers, C. N. Jones, J. Lygeros, and M. Morari. A Multiscale Approximation Scheme for Explicit Model Predictive Control with Stability, Feasibility, and Performance Guarantees. In *IEEE Conf. on Decision and Control*, Shanghai, China, Dec. 2009.

[123] S. Summers, C. N. Jones, J. Lygeros, and M. Morari. A Multiscale Approximation Scheme for Explicit Model Predictive Control with Stability, Feasibility, and Performance Guarantees. In *IEEE Conference on Decision and Control*, 2009.

[124] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):202–210, 2005.

[125] K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 - a MATLAB software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999.

[126] F. Torrisi and A. Bemporad. Hysdel-a tool for generating computational hybrid models for analysis and synthesis problems. *Control Systems Technology, IEEE Transactions on*, 12(2):235 – 249, Mar. 2004.

[127] F. Torrisi, A. Bemporad, G. Bertini, P. Hertach, D. Jost, and D. Mignone. HYSDEL - User Manual. Technical report, Automatic Control Laboratory, ETH Zurich, Aug. 2002.

[128] F. Ullmann and S. Richter. FiOrdOs: Code generation for first-order methods. `http://fiordos.ethz.ch`, May 2012.

[129] L. Vandenberghe. The CVXOPT linear and quadratic cone program solvers. Online: `http://cvxopt.org/documentation/coneprog.pdf`, March 2010.

[130] R. Vanderbei. Symmetric quasidefinite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.

[131] A. Vezhnevets. Gml adaboost toolbox for matlab. `http://www.inf.ethz.ch/personal/vezhneva/index.html#code`, March 2010.

[132] M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice Hall, 1993.

[133] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[134] S. A. Vorobyov, A. B. Gershman, and Z.-Q. Luo. Robust adaptive beamforming using worst-case performance optimization: A solution to the signal mismatch problem. *Signal Processing, IEEE Transactions on*, 51(2):313–324, 2003.

[135] Y. Wang and S. Boyd. Fast Model Predictive Control Using Online Optimization. In *Proceedings of the 17th World Congress*, Seoul, 2008. IFAC.

[136] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Trans. on Control Systems Technology*, 18(2):267–278, March 2010.

[137] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267 –278, Mar. 2010.

[138] S. J. Wright. Applying new optimization algorithms to model predictive control. *Chemical Process Control-V*, 93(316):147–155, 1997.

[139] S. J. Wright. *Primal-dual Interior-point Methods*. Society for Industrial Mathematics, Dec. 1997.

[140] S.-P. Wu, S. Boyd, and L. Vandenberghe. Fir filter design via spectral factorization and convex optimization. In *Applied and Computational Control, Signals, and Circuits*, pages 215–245. Springer, 1999.

[141] Y. Ye, M. J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19(1):53–67, Feb. 1994.

[142] E. L. Yip. A note on the stability of solving a rank-p modification of a linear system by the Sherman-Morrison-Woodbury formula. *SIAM Journal on Scientific and Statistical Computing*, 7(2):507–513, 1986.

[143] M. N. Zeilinger. *Real-Time Model Predictive Control*. PhD thesis, ETH Zurich, Switzerland, 2011.

[144] M. N. Zeilinger, D. M. Raimondo, A. Domahidi, M. Morari, and C. N. Jones. On real-time robust model predictive control. *Automatica (to appear)*, 2013.

[145] P. Zometa and R. Findeisen. μAO-MPC: Microcontroller applications online model predictive control. `http://www.et.uni-magdeburg.de/syst/research/muAO-MPC/`, 2012.

[146] P. Zometa and R. Findeisen. Code generation of linear model predictive control for real-time embedded applications. In *Proceedings of the American Control Conference*, Washington, D.C., USA, 2013.