Mixed-Integer Optimal Control of Fast Dynamical Systems

Bartolomeo Stellato

St Edmund Hall
University of Oxford

A dissertation submitted for the degree of

Doctor of Philosophy

November 2017

To my parents,

Nicola and Daniela

Abstract

Many applications in engineering, computer science and economics involve *mixed-integer optimal control problems*. Solving these problems in real-time is a challenging task because of the explosion of integer combinations to evaluate. This thesis focuses on the development of new algorithms for mixed-integer programming with an emphasis on optimal control problems of fast dynamical systems with discrete controls.

The first part proposes two reformulations to reduce the computational complexity. The first reformulation avoids integer variables altogether. By considering a sequence of switched dynamics, we analyze the switching time optimization problem. Even though it is a continuous smooth problem, it is non-convex and the cost function and derivatives are hard to compute. We develop a new efficient method to compute the cost function and its derivatives. Our technique brings up to two orders of magnitude speedups with respect to state-of-the-art tools. The second approach reduces the number of integer decisions. In hybrid model predictive control (MPC) the computational complexity grows exponentially with the horizon length. Using approximate dynamic programming (ADP) we reduce the horizon length while maintaining good control performance by approximating the tail cost offline. This approach allows, for the first time, the application of such control techniques to fast dynamical systems with sampling times of only a few microseconds.

The second part investigates embedded branch-and-bound algorithms for mixed-integer quadratic programs (MIQPs). A core component of these methods is the solution of continuous quadratic programs (QPs). We develop OSQP, a new robust and efficient general-purpose QP solver based on the alternating direction method of multipliers (ADMM) and able, for the first time, to detect infeasible problems. We include OSQP into a custom branch-and-bound algorithm suitable for embedded systems. Our extension requires only a single matrix factorization and exploits warm-starting, thereby greatly reducing the number of ADMM iterations required. Numerical examples show that our algorithm solves small to medium scale MIQPs more quickly than commercial solvers.

Acknowledgments

This wonderful journey would have never been possible without the support and encouragement of many exceptional people.

I would like to start by thanking Bart Van Parys who encouraged me to pursue my doctorate in Oxford. I really enjoyed the time we spent together during his visit and I am very grateful for his valuable academic and life advices. I am incredibly happy we will meet soon after my defense!

My deepest gratitude goes to my mentor, Paul Goulart, for believing in me from the start. His close guidance, patience and attention to details were invaluable to me. During the biggest struggles in my research, I could always ask him for help. With his enthusiasm he was always able to motivate and energize me during our meetings. I really appreciated all the times he supported me during these years. I will always remember our video calls when it was very late at night for him in Oxford and we had 8 hours time difference. I am also grateful for his promptness in the moments of need. When there were urgent deadlines he always read my papers very quickly and, yet, extremely carefully. He never commanded, but instead he motivated me to follow my research interests with all the freedom I needed. I am very much indebted to Paul. This dissertation and my future career would not have been the same without him.

I am especially thankful to Stephen Boyd for hosting me at Stanford. Visiting his group has been an enlightening and enriching experience. I feel privileged to have had the chance to work with him so closely and to share his contagious enthusiasm and joy at work. His passion for simplicity, clarity and his unique style are a true inspiration for me. I am also grateful to him for sharing his invaluable suggestions on research topics that really matter and his passion for open-source software and open-access research.

I am furthermore grateful to Alberto Bemporad for always welcoming me with open doors at IMT Lucca. I really enjoyed working with him and having our very long meetings in Lucca racking our brains to solve complex research problems.

I wish to thank the EU Marie Curie project TEMPO for the financial support. Thanks to this funding scheme I had the unique opportunity to pursue my doctorate without any constraint. I am grateful to Eric Kerrigan, Tor Arne Johansen, Moritz Diehl and all the other professors

in the TEMPO project for organizing all the workshops and events and for the great feedback they gave on my work. I am also thankful to ABB Switzerland for funding the TEMPO project and to Tobias Geyer for his support on power electronics during my first year.

All the faculty members in the control group encouraged and supported me in different ways. I wish to thank Kostas Margellos for being such a positive fresh spirit in the group and for his honest, helpful and friendly career and life advices. I am also grateful to Sina Ober-Blöbaum for our very pleasant and interesting collaboration. Big thanks go also to Stephen Duncan for his support and for helping out in any moment of need.

I am sincerely grateful to Goran Banjac for being my closest friend in this amazing experience. I realized the incredible amount of time we spent together only now that it is over. We managed to get along well after sharing the project, the office, the flat in Oxford, the flat in Palo Alto, the office in Stanford and also the project Stanford! I am so grateful for his patience and support for all my problems in life and research. His honesty, impeccable precision (also with appointment times!) and organizational skills will always be an inspiration for me. I am sure that after such an experience we will stay close even when living in different continents.

I would like to thank Paola D'Andrea for being such an amazing flatmate during my first year and for being such an understanding, caring and irreplaceable friend. She was always there in the moments of need despite distances and different time zones. I will deeply miss our coffees and intense chats.

Giovanni Licitra deserves a huge thank you for being such a cheerful, honest and humble friend. Even if we were far apart in these years, Giovanni was always there, just a phone call away. I greatly enjoyed our time together in the TEMPO meetings and our great adventures in Sicily. I am so looking forward to meet Isabella in person!

A special thank you goes to Emilia Vanni for her warm-hearted support, especially in the stressful moments when writing up my thesis. I am so grateful to her for the joy and the positive energy she always brings to people and for always finding the silver lining, no matter what happens.

I am also grateful to all the incredible people in the control group in Oxford. I would like to thank Nikitas Rontsis for being a great office

mate and friend, and for sharing all the coding struggles together. Moritz Schulze-Darup deserves the price for the greatest (and craziest) party maker and control theorist ever! A big thank you goes to Dhruva Raman, Xiaojing Chen, Mohamadreza Ahmadi, Ross Drummond, Michael Garstka and all the other members of the control group for making it such an enjoyable place to work.

I would like to thank the amazing people I met during my research trips. Steven Diamond, Nicholas Moehle and Enzo Busseti made my research visit at Stanford such a unique experience. I wish to thank also Youngsuk Park for being such a great office mate and for suggesting amazing places to visit in California. Also, Stanford visit would have never happened without the professional and extremely efficient support from Douglas Chaffee. Thanks a lot Doug! Many thanks also to Ugo Rosolia and to Jacopo Guanetti for our great trip from Berkeley to the Yosemite Park. I wish to thank also Vihang Naik for showing me around IMT Lucca and for being such a great host. Thanks also to Miles Lubin for inviting me to MIT, introducing me to the great Julia community and for hanging out in Boston.

My Oxford friends were the most important part of my life during these years. I would like to thank Jan for being such a great flatmate, for sharing tasty brunches during the weekend and for all the startup suggestions. Big thanks go to Markus Wulfmeier for our deep discussions and for being such a fantastic beer and coffee buddy from the first day I arrived. I thank also Felix Schupp and Shannon Christyna for organizing great dinners and parties at their place. Many thanks also to Lina Gridvainyte, Ben Lange, Matthias Mer, Chloe Ann for all our times together.

I enjoyed meeting all the fantastic people at St Edmund Hall. I really appreciated the effort and enthusiasm of Linde Wester and Kusal Lokuge in organizing amazing events in the college in these years. I am also thankful to Alex Blakes, Thomas Cosnahan, Rosin Huggins, Isabel Wassing for making the MCR such a cozy and welcoming environment. A big thank you goes also to Daniel Barba Cancho for all the moments we spent together.

I would like to thank also all my Italian friends in Oxford. I am greatly indebted to Federico Danieli for all the incredibly tasty dinners and for being such a welcoming host in his flat. It was great to live just 50 me-

ters apart! Big thanks also to Noemi Picco for her support, the amazing barbecues at her place and for the awesome dinners at St John's College SCR. Luca Bertinetto, Andrea Tucci, Roberto Soleti, Angela Diana, Giovanna Granata, Gabriele Abbati, thank you for the great dinners, football matches, TV shows and drinks in these years! I wish to thank also my friends Claudio Caletti, Michele Fontana, and Silvia Abruzzi who always supported me from Cremona. Claudio, your computer science and tech suggestions were so helpful to me!

The employees at The Rickety Press deserve special thanks for all the delicious brunches, the great drinks and the tasty burgers. Thank you for creating such a fantastic atmosphere.

Many thanks also to Iris Ballestreros, Niels van Duijkeren, Robin Verschueren, and all the other TEMPO fellows for all the adventures during our trips. Our presentation skills workshop videos will always be a secret!

Last, but most importantly, I cannot express in words how grateful I am to my parents and my family for their unconditional love and support. I would have never been able to accomplish this without having them by my side.

Bartolomeo Stellato
Oxford,
November 2017

Contents

1

Introduction

In recent years, there has been an explosion in computational capabilities of modern processors and a dramatic decrease in the cost of computing hardware. In fractions of a second, we can now solve on cheap hardware complex decision making problems that were almost impossible to solve just 20 years ago. These advances drove revolutions in several science end engineering fields such as control theory, machine learning, scheduling and finance.

Mathematical optimization is a comprehensive framework for formulating and solving complex decision making problems. With these techniques we can systematically compute the best decisions minimizing a performance index or minimizing a cost while satisfying constraints. Together with other branches of science and engineering, mathematical optimization has seen tremendous advances in the last 20 years.

There currently exist many efficient algorithms able to solve optimization problems with continuous variables in a wide range of applications [137]. A popular example is model predictive control (MPC), where the optimal control input is computed in real-time at each sampling in-

stance. MPC has been successfully applied in both the industry and academic sectors in the past 20 years, becoming a standard framework for optimal control. Historically, MPC was applied to slow processes with sampling times in the order of minutes or hours because of the limited computing resources available. In the last decade, there has been a great reduction in computation time for continuous problems arising in linear MPC from hours to milliseconds [178] or even microseconds [100]. We can now apply MPC to systems with fast dynamics in robotics, signal processing and power electronics.

The last 20 years have also seen great improvements in solving optimization problems with integer variables [26, 135]. These problems arise, for instance, when only a limited number of options or configurations is available. Examples include power distribution, scheduling, mechanical systems, portfolio investments and many others. If we consider the speedups of algorithmic developments together with hardware improvements, we can now solve medium-sized problems with mixed-integer variables 200 billion times faster than 20 years ago [22]. However, when dealing with integer decisions, the complexity required to compute the optimal solutions grows dramatically and is much higher than for problems with only continuous decisions. In the worst case we have to check all the possible integer combinations which grows exponentially in the problem dimensions [134]. For this reason, despite the recent advances, it is still difficult to obtain optimal integer solutions within the seconds time scale for medium-sized problems with hundreds or thousands of integer variables. There exists no reliable solution method for mixed-integer optimization problems for high-speed real-time applications. One example is MPC of hybrid systems where integer inputs describe the logic governing the dynamics. Compared to linear MPC, we are still not able to comfortably apply hybrid MPC to fast dynamical systems with milliseconds sampling time. Therefore, even though we can now solve these problems much faster than before, we still need to improve integer programming techniques to tackle fast dynamical systems.

This thesis focuses on the development of new algorithms for mixed-integer programming with an emphasis on optimal control problems of fast dynamical systems. The main goal is to apply real-time optimal control schemes to certain classes of systems with integer inputs by reducing the

computation time.

There are currently two main approaches to deal with integer optimization problems. On the one hand, *exact solution methods* focus on finding and certifying the globally optimal solution for a given problem. If there is enough computation time and hardware available, these approaches return the optimal solution which cannot be improved. However, when the number of integer decisions is too high or when there is not enough computing time available, exact solution methods become prohibitive. In these cases we need to resort to *approximations to reduce the complexity*, *i.e.*, heuristics. These methods return suboptimal solutions that perform very well in many practical applications. However, in some cases the returned solution might be highly suboptimal or even infeasible. Based on this reasoning, we need to take into account an important trade-off between the computation time available and the quality of the solutions. Depending on the application, we construct tractable tailored methods to deal with integer optimal control problems. In this case, tractability does not mean polynomial time solvability but instead "the ability to solve problems of realistic size in times that are appropriate for the applications we consider" [22].

This thesis is structured in two parts. The first part deals with approximations to reduce the complexity arising from integer decisions in optimization problems. We develop two approaches: one avoiding integer variables and based on switching time optimization, and the other reducing the number of integer variables and based on approximate dynamic programming (ADP). The second part investigates an algorithm for the exact solution of mixed-integer quadratic programs (MIQPs). We first develop a novel efficient solver for quadratic programs (QPs). Then, we embed it into a branch-and-bound algorithm simplifying all the unnecessary computations. The next sections introduce the thesis topics more in detail.

Figure 1.1 illustrates the chapters involved for each topic. The contributions are ordered by the generality of the integer problems involved. The first approach, based on switching time optimization, tackles optimal control problems where the goal is to switch between dynamics in a predefined order. The second approach extends this case to control problems where the dynamics can be in any order that needs to be decided by the

solution algorithm. The third approach, based on exact solution methods and MIQPs, can deal with any problem of that form arising in several application areas that are not necessarily control systems.

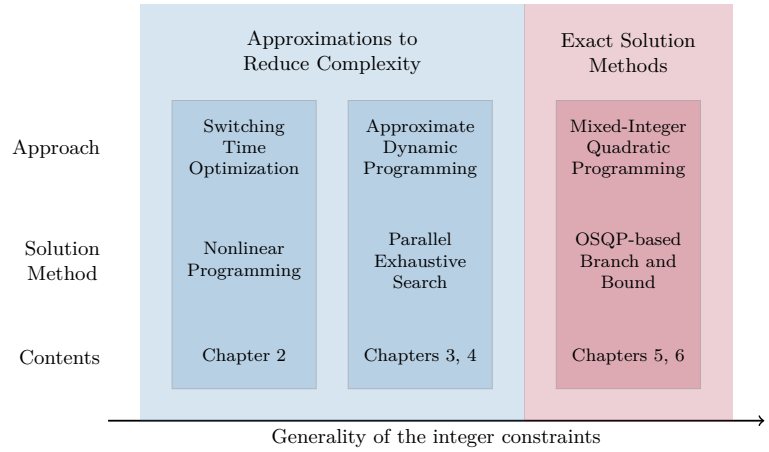


Figure 1.1: Thesis outline in terms of generality of the integer constraints.

1.1 Approximations to reduce complexity

In this part of the thesis we focus on approximations to make the problems tractable for the application considered. By restricting the generality of the problem addressed, we reduce the complexity together with the computation time required.

1.1.1 Switching time optimization for switched dynamical systems

Chapter 2 investigates the switching time optimization problem. Given a fixed sequence of nonlinear dynamics, we analyze the problem of finding the optimal times to switch between them in order to minimize a predefined cost function. With this technique we can reformulate a hard integer

optimization problem as a continuous, smooth, non-convex optimization problem whose locally optimal solutions are easy to find.

Recent approaches compute the optimal switching times using iterative optimization methods. First-order [64] and second-order [101] algorithms have been proposed in the literature to solve this problem. However, current techniques suffer from the computational complexity of the multiple numerical integrations required to compute the cost function, the gradient and the Hessian of the optimization problem.

In Chapter 2 we present a novel efficient technique to formulate and solve switching time optimization problems. We develop easily computable expressions for the cost function, the gradient and the Hessian, sharing the most expensive computations. In this way, once the cost function is evaluated, there is no significant increase in complexity in computing its derivatives. In the case of linear dynamics we show that our method can be even further simplified and the computations parallelized to greatly reduce the computation time.

We implemented our method in the Julia package SwitchTimeOpt. This tool provides a simple interface allowing the user to easily define and efficiently solve switching time optimization problems. SwitchTimeOpt supports a wide variety of nonlinear solvers that can be easily interchanged. Numerical examples show that our method exhibits up to $100\times$ improvements in computation time over tailored state-of-the-art nonlinear optimal control software tools.

1.1.2 Approximate dynamic programming for integer optimal control

Chapters 3 and 4 introduce a novel method for MPC of hybrid systems with fast linear dynamics and integer inputs. To address the computational issues of performing the optimization over long prediction horizons, we cast the problem into the framework of ADP [20, 21]. The infinite horizon value function is approximated with a quadratic underestimator by solving a semidefinite program (SDP) offline [29]. This allows us to shorten the controller horizon to reduce the algorithm complexity by applying an estimated tail cost to the last stage while maintaining good control performance. Moreover, in contrast to common formulations where the input

effort is reduced indirectly via penalization of the input switchings over the controller horizon [80], in this work we augment the system dynamics to directly estimate the switching frequency. This allows the user to easily define the desired frequency to be tracked a priori without complicated tuning of the cost function.

We apply our approach to a variable-speed drive system consisting of a voltage source inverter connected to a medium-voltage induction machine. The plant is modeled as a linear system with a switched three-phase input with equal switching steps for all phases.

We implemented our algorithm on a small size Xilinx Zynq field-programmable gate array (FPGA) (xc7z020) in fixed-point arithmetic. We show via processor-in-the-loop (PIL) experiments that our method, even with very short prediction horizons, outperforms state-of-the-art approaches [80] with much longer planning horizons in terms of current distortion and switching frequency, while achieving computation times under 25 µs.

1.2 Exact solution methods

In this part we focus on the solution of general MIQPs using the branch-and-bound algorithm. The solution of QPs is an essential part of MIQPs branch-and-bound methods. Regardless of the integer programming techniques used, the majority of time needed to solve MIQPs is spent solving individual QP subproblems. In addition, there are two important features we must have in QP solvers for MIQP algorithms.

The first key feature is warm-starting. Even though the number of QPs solved grows exponentially in the worst case, they are all very similar and only a few elements in the problem data change between parent and child nodes. Thus, the solutions are in general very close and starting the QP solver from the solution of each parent node can greatly simplify the number of iterations involved. To see this, consider for simplicity two QPs with strictly convex cost and slightly different bounds. In this case, the solutions are expected to be close, especially when few constraints are active at the optimum. For this reason we need a QP solution algorithm able to be easily warm-started to exploit the fact that the solutions of

consecutive QPs are usually close.

The second feature is infeasibility detection. Many problems in the search tree are usually infeasible, because we alter the bounds on the integer variables throughout the search. If a problem is infeasible, we would like to detect it as soon as possible and to prune its child nodes to reduce the search space.

We develop a novel efficient QP solver based on first-order methods that is suitable for being embedded in a MIQPs solver. We then construct a tailored branch-and-bound method that exploits the QP solution algorithm to save computation time.

1.2.1 The OSQP solver

Chapter 5 describes OSQP, our novel general-purpose QP solver based on the alternating direction method of multipliers (ADMM). OSQP is very robust and requires no assumptions on problem data such as positive definiteness of the objective function or linear independence of the constraints.

OSQP does not require an initial feasible solution and can detect infeasible problems directly from the algorithm iterates. Therefore, it provides a very simple framework to detect infeasible problems without resorting to Phase-I/II methods such as primal active set methods where an auxiliary feasibility problem is solved first. This problem can be computationally very expensive and before OSQP several approaches have been proposed in the literature to avoid it. One example is the primal-dual active set method implemented in qpOASES [65] where no auxiliary problem is needed.

OSQP is based on a novel operator splitting technique that requires the solution of a quasi-definite linear system with the same coefficient matrix in each iteration. Our algorithm is division free once the initial matrix factorization is carried out, making it suitable for real-time applications in embedded systems. The method also supports factorization caching and warm-starting, making it particularly efficient when solving parametrized problems arising in finance, control, and machine learning. Our open-source C implementation has a small footprint, is library-free,

and has been tested on many problem instances arising in a wide variety of application areas. It also supports code generation of an embeddable version of the solver featuring only static memory allocation. Our solver is typically faster than interior-point methods, and sometimes much more when factorization caching and warm-starting is used.

1.2.2 An MIQP solver based on OSQP

In Chapter 6 we extend the OSQP solver developed in Chapter 5 to the solution of MIQPs. We present a robust branch-and-bound algorithm that requires no dynamic memory allocation and is division free after a first initial factorization is performed at the beginning of the search tree. We exploit factorization caching and warm-starting techniques to reduce the computational cost of the QP relaxations at each branch-and-bound node and over the repeated solution of parametrized MIQPs. These problems arise in many applications such as control, portfolio optimization and machine learning. With a simple high-level Python implementation, we show that our method is competitive with established commercial solvers.

1.3 Publications

The work presented in this thesis relies on the following publications. If not otherwise specified, I am the main author of the contributions presented.

Chapter 2 is based on

- B. Stellato, S. Ober-Blöbaum, and P. Goulart. Second-order switching time optimization for switched dynamical systems. *IEEE Transactions on Automatic Control*, 62(10):5407–5414, October 2017
- B. Stellato, S. Ober-Blöbaum, and P. Goulart. Optimal control of switching times in switched linear systems. In *IEEE Conference on Decision and Control (CDC)*, pages 7228–7233, December 2016.

Chapters 3 and 4 are based on

- B. Stellato, T. Geyer, and P. Goulart. High-speed finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, 32(5):4007–4020, May 2017
- B. Stellato and P. Goulart. Real-time FPGA implementation of direct MPC for power electronics. In *IEEE Conference on Decision and Control (CDC)*, pages 1471–1476, December 2016
- B. Stellato and P. Goulart. High-speed direct model predictive control for power electronics. In *European Control Conference (ECC)*, pages 129–134, July 2016.

Chapter 5 is based on

- B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An Operator Splitting Solver for Quadratic Programs. *ArXiv e-prints*, November 2017, 1711.08013
- G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd. Embedded code generation using the OSQP solver. In *IEEE Conference on Decision and Control (CDC) (To appear)*, 2017
- G. Banjac, P. Goulart, B. Stellato, and S. Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. *SIAM Journal on Optimization (Submitted)*, June 2017.

This work was co-authored with Goran Banjac. We worked together the formulation and algorithm. Afterwards, I focused on the numerical implementation, the interfaces, and the extensive numerical testing. Goran Banjac focused more on the infeasibility detection while also contributing to the code.

Chapter 6 is based on

- B. Stellato, V. Naik, A. Bemporad, P. Goulart, and S. Boyd. Embedded mixed-integer quadratic optimization using the OSQP solver. In *European Control Conference (ECC) (Submitted)*, 2018.

Part I

Approximations to Reduce Complexity

2

Optimal Switching Times for Switched Dynamical Systems

2.1 Switched systems

Switched systems are a particular class of hybrid systems consisting of several continuous subsystems where a switching law defines the active system at each time instant. A recent survey on computational methods for switched systems control appears in [185].

In this chapter we focus on optimal control of autonomous switched systems where the sequence of continuous dynamics is fixed. In particular, we study the problem of computing the optimal switching instants at which the dynamics must change in order to minimize a given cost function. This problem is usually referred to as *switching time optimization*.

This topic has been studied extensively in the last decade. In [153] the authors provide a method to construct an offline mapping of the optimal switching times for linear dynamics from the initial state of the system. Even though this approach seems appealing at first sight, it suffers from the high storage requirements typical for explicit control approaches [18]

as the dimension of the system and the number of possible switchings increase.

More recent approaches focus on finding optimal switching times using iterative optimization methods. In [64] an expression for the gradient of the cost function with respect to the switching times is derived for the case of nonlinear systems. A first-order method based on Armijo stepsizes is then adopted to find the optimal switching times. An extension for discrete-time nonlinear systems is given in [67]. However, first-order methods are very sensitive to the problem data and can exhibit slow convergence [137]. To overcome these limitations Johnson and Murphey [101] derived an expression for the Hessian of the cost function for nonlinear dynamics and adopted a second-order method to compute the optimal switching times finding significant improvements on the number of iterations compared to the first-order method in [64]. However, both these first and second-order approaches suffer from the computational complexity of multiple numerical integrations required to solve the differential equations used to define the cost function, the gradient and the Hessian (in the second-order case). Note that the Hessian definition in [101] requires an additional set of integrations to be performed.

Computational effort. There has been very limited focus in the literature on the computational effort required by the switching time optimization and the multiple integration routines. In [54] the authors present a convergence analysis of a second-order method for switched nonlinear systems similar to the one in [101] without considering the overall computation time. In [37] the switching time optimization problem for linear time-varying dynamics is formulated so that only a set of differential equations needs to be solved before the optimization procedure. Once the integration is performed, the steepest descent direction can be computed directly without solving any further differential equations. However, in [37] the authors do not provide a closed-form expression for the Hessian and only a steepest descent algorithm is adopted.

2.1.1 Our approach

In this work we present a novel method to solve switching time optimization problems efficiently for linear and nonlinear dynamics. We develop efficiently computable expressions for the cost function, the gradient and the Hessian, exploiting shared terms in the most expensive computations. In this way, at each iteration of the optimization algorithm there is no significant increase in complexity in computing the gradient or the Hessian once the cost function is evaluated. These easily computable expressions are obtained thanks to linearizations of the system dynamics around equally spaced grid points, and then integrated via independent matrix exponentials. In the case of linear dynamics, our method can be greatly simplified and the matrix exponentials usually decomposed into independent scalar exponentials that can be parallelized to further reduce the computation times.

Our method has been implemented in the open-source Julia package SwitchTimeOpt with a simple interface that allows the user to easily define and solve switching time optimization problems. SwitchTimeOpt supports a wide variety of nonlinear solvers which can be quickly interchanged.

We provide three examples to benchmark the performance of our method. The first, from [37], is a system with two unstable switched dynamics whose optimal switching times are obtained in only a few milliseconds with our approach. The second is the so-called Lotka-Volterra fishing problem [176] with nonlinear dynamics, integer control inputs and constant steady state values to be tracked. The third is a double-tank system that first appeared in [121] and was used in the switching time optimization setting in [8]. In the final example we apply our algorithm to find the optimal switching times to track a time-varying reference level of the liquid in one of the tanks. In both the nonlinear examples our method, which is implemented in the high-level language Julia, exhibits up to two orders of magnitude improvements over tailored state-of-the-art nonlinear optimal control software tools.

2.2 Problem statement

Consider a switched autonomous system switching between N modes whose dynamics can be expressed as

$$\dot{x}(t) = f_i(x(t)), \quad \forall t \in [\tau_i, \tau_{i+1}), \quad i = 0, \dots, N, \tag{2.1}$$

with $f_i : \mathbf{R}^{n_x} \to \mathbf{R}^{n_x}$, $\forall i$. We assume the solution $x(t) \in \mathbf{R}^n$ to (2.1) to always exist and be unique. Note that this assumption is stronger than requiring that all the functions f_i are uniformly Lipschitz continuous because of degenerate behaviors that could be introduced by the switchings. We set the initial state as $x(0) \coloneqq x_0$.

We refer to the times τ_i as the *switching times*, and define also the *switching intervals*

$$\delta_i \coloneqq \tau_{i+1} - \tau_i, \quad i = 0, \dots, N,$$

so that each

$$\tau_i = \sum_{j=0}^{i-1} \delta_j.$$

In this Chapter, we take the set of switching intervals $\delta \coloneqq \{\delta_i\}_{i=0}^{N}$ as decision variables to be optimized, but occasionally use the switching times $\tau \coloneqq \{\tau_i\}_{i=0}^{N+1}$ for convenience of notation. We define the final time as $T_\delta \coloneqq \sum_i \delta_i = \tau_{N+1}$, with initial time $\tau_0 \coloneqq 0$.

Our goal is to find optimal switching intervals $\delta^\star$ minimizing an objective function in Bolza form, *i.e.*,

$$\underbrace{\int_0^{T_\delta} x(t)^T \overline{Q} x(t) \mathrm{d}t}_{\mathcal{L}(\delta)} + \underbrace{x(T_\delta)^T \overline{E} x(T_\delta)}_{\psi(\delta)}. \tag{2.2}$$

The Lagrange term $\mathcal{L}$ penalizes the integral between 0 and T_δ of the quadratic state penalty weighted by matrix $\overline{Q} \in \mathbf{S}_+^{n_x}$. The Mayer term ψ penalizes the final state at time T_δ with weights defined by matrix $\overline{E} \in \mathbf{S}_+^{n_x}$.

We include a set of constraints on the switching intervals

$$\Delta = \left\{ \delta \in \mathbf{R}_+^{N+1} \mid 0 \leq \underline{b}_i \leq \delta_i \leq \bar{b}_i,\ i = 0, \ldots, N \ \wedge\ T_\delta = T \right\},$$

which requires all switching times to be nonnegative and the final time T_δ to be equal to some desired final time T. In addition, in case the i^{th} dynamics must be active for a minimum or maximum time, we allow lower and upper bounds $\underline{b}_i$ and $\bar{b}_i$, respectively. If neither minimum nor maximum constraints are imposed for interval δ_i, we set $\underline{b}_i = 0$ and $\bar{b}_i = \infty$.

The switching time optimization problem then takes the form

$$\begin{array}{ll} \text{minimize} & \int_0^{T_\delta} x(t)^T \overline{Q} x(t) \mathrm{d}t + x(T_\delta)^T \overline{E} x(T_\delta) \\ \text{subject to} & \dot{x}(t) = f_i(x(t)), \quad t \in [\tau_i, \tau_{i+1}), \quad i = 0, \ldots, N \\ & x(0) = x_0 \\ & \delta \in \Delta. \end{array} \tag{$\mathcal{P}$}$$

In some degenerate cases this problem can show non strict minima allowing the optimal δ_i to live in intervals that are not singletons. These situations can cause numerical optimization solvers to stall and not progress in the iterations. To enforce strict minima, many regularization techniques could be applied. For instance, we could add a quadratic regularization term $\epsilon \|\delta\|_2^2$ with $\epsilon > 0$ to the objective function so that we equally penalize all the intervals. In this way, with a small ϵ we prefer solutions where the time different dynamics are active for similar amount of time. For simplicity, we exclude regularization terms from our derivations, but they could be taken into account without a significant increase in complexity.

Switching order. Although we restrict ourselves to the case where the switching order of the $N + 1$ modes is prescribed, we allow the system dynamics to be the same for different i. If we set some $\delta_i = 0$, then the i^{th} interval collapses and the dynamics switch directly from the $(i-1)^{\text{th}}$ to the $(i+1)^{\text{th}}$ mode. This allows some dynamics to be bypassed and an arbitrary switching order realized without recourse to integer optimization; see [76]. For example, given N_{dyn} different dynamics, one can cycle through all of

them in the same predefined ordering N_{cyc} times for a total of $N_{\text{dyn}}N_{\text{cyc}}$ intervals and $N_{\text{dyn}}N_{\text{cyc}}-1$ switching times, thereby allowing the dynamics to be visited in arbitrary order. We illustrate the use of this approach in the examples in Section 2.6.

Cost function. The cost function in problem $(\mathcal{P})$ is non-convex in general, but it is smooth [54] and its first and second derivatives can be used efficiently within a nonlinear optimization method, *e.g.*, sequential quadratic programming (SQP) or interior-point methods to obtain locally optimal switching times. In order to obtain an algorithm implementable in real-time, we derive tractable formulations of the cost function, the gradient and the Hessian based on linearizations of the system dynamics. We show that this approach offers significant improvements in computational efficiency relative to competing approaches in the literature.

2.3 Preliminaries

This section introduces the preliminary notions and definitions that is used in the rest of the Chapter.

2.3.1 Time grid

In order to integrate the switched nonlinear dynamics (2.1), we define an evenly spaced "background" grid of n_{grid} time-points from 0 to the final time T, and hold these background grid points fixed regardless of the choice of switching times τ_i. Note that, depending on the intervals δ, the switching times τ can be in different positions relative to the background grid while maintaining the ordering $\tau_i \leq \tau_{i+1}$.

We subdivide each interval δ_i according to the background grid points falling between τ_i and τ_{i+1}, with τ_i^j denoting the j^{th} grid point after the switching time τ_i. The number of such background grid points between switching times τ_i and τ_{i+1} is denoted by n_i, which is itself a function of the switching times τ. Note that we set $n_{N+1} = 0$.

For notational convenience, we define $\tau_i^0 \coloneqq \tau_i$ and $\tau_i^{n_i+1} \coloneqq \tau_{i+1}$ for $i = 0, \ldots, N$. We further define a partitioning of the switching intervals such that δ_i^j is the j^{th} subdivision of interval i, so that

$$\delta_i \coloneqq \sum_{j=0}^{n_i} \delta_i^j \quad \text{and} \quad \tau_i^k \coloneqq \tau_i + \sum_{j=0}^{k-1} \delta_i^j, \quad \forall k \leq n_i. \tag{2.3}$$

In subsequent sections we define a number of vector and matrix quantities to be associated with the time instants τ_i^j, and adopt complementary notation, *e.g.*, the vector x_i^j and matrix M_i^j are associated with the j^{th} grid point after switching time τ_i. Likewise $x_i^{n_i+1} \coloneqq x_{i+1}, x_i^0 \coloneqq x_i$ and $M_i^{n_i+1} \coloneqq M_{i+1}, M_i^0 \coloneqq M_i$.

A portion of the grid is presented in Figure 2.1 where the smaller ticks represent the background grid points.

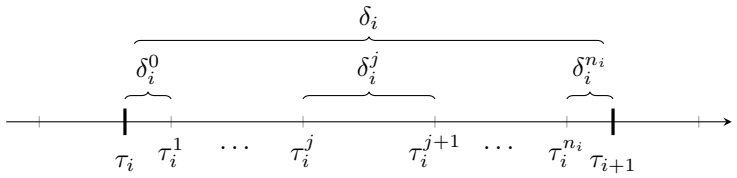


Figure 2.1: Switching times within the time grid.

2.3.2 Dynamics linearization

In order to make the computations of the cost function and its derivatives numerically efficient, we linearize the dynamics around each time instant of the background grid and each switching time.

For a given time instant τ_i^j we consider the linearized dynamics around the state $x_i^j \coloneqq x(\tau_i^j)$ with $j = 0, \ldots, n_i + 1$ (to simplify the notation we consider $x_i = x_i^0 = x(\tau_i)$) by writing

$$\begin{aligned} \dot{x}(t) &\approx f_i(x_i^j) + J_{f_i}(x_i^j)(x(t) - x_i^j) \\ &= J_{f_i}(x_i^j)x(t) + \left(f_i(x_i^j) - J_{f_i}(x_i^j)x_i^j\right) \end{aligned} \tag{2.4}$$

where

$$J_{f_i}(x_i^j) = \left.\frac{\partial f_i(x_i^j)}{\partial x}\right|_{x_i^j = x(\tau_i^j)} \tag{2.5}$$

is the Jacobian of the i^{th} nonlinear dynamics evaluated at x_i^j.

We can obtain an approximate linear model by augmenting the dynamics with an additional constant state so that

$$\dot{x}(t) = A_i^j x(t), \quad t \in [\tau_i^j, \tau_i^{j+1}), \tag{2.6}$$

where

$$A_i^j = \begin{bmatrix} J_{f_i}(x_i^j) & f_i(x_i^j) - J_{f_i}(x_i^j)x_i^j \\ 0 & 0 \end{bmatrix} \tag{2.7}$$

and $x(t)$ is an augmented version of the previous state definition, *i.e.*, $x(t) \coloneqq (x(t), 1)$.

Following the dynamics augmentation, let us define also the augmented cost function weights $Q \coloneqq \mathbf{blkdiag}(\overline{Q}, 0)$ and $E \coloneqq \mathbf{blkdiag}(\overline{E}, 0)$.

2.3.3 Definitions

We now present some definitions required to develop our main result. Note that the order of the matrix products is always from the left.

Definition 2.1 (State evolution)**.** The matrix $\Phi(t, \tau_i^j)$ is the state transition matrix of the linearized system from τ_i^j to t, and is defined as:

$$\Phi(t, \tau_i^j) \coloneqq e^{A_l^m (t - \tau_\ell^m)} \left(\prod_{p=0}^{m-1} e^{A_\ell^p \delta_\ell^p} \right) \left(\prod_{q=i+1}^{\ell-1} \prod_{p=0}^{n_q} e^{A_q^p \delta_q^p} \right) \left(\prod_{p=j}^{n_i} e^{A_i^p \delta_i^p} \right), \tag{2.8}$$

where τ_ℓ and τ_ℓ^m are the last switching time and the last grid point before t respectively.

Given a time instant τ_i^j and a time $t \in \mathbf{R}_+$ such that $t \geq \tau_i^j$ we can compute the state $x(t)$ as

$$x(t) = \Phi(t, \tau_i^j) x_i^j.$$

Observe that if we consider the transition between two switching times τ_i and τ_ℓ with $\tau_i \leq \tau_\ell$, the state transition matrix in (2.8) simplifies to

$$\Phi(\tau_\ell, \tau_i) = \prod_{q=i}^{\ell-1} \prod_{p=0}^{n_q} e^{A_q^p \delta_q^p}, \tag{2.9}$$

which is used extensively in most of the computations in the remainder of this Chapter.

Definition 2.2 (Cost-to-go matrices)**.** Given the time τ_i^j, define matrix $P_i^j \in \mathbf{S}_+^{n_x}$ as

$$P_i^j \coloneqq \int_{\tau_i^j}^{T_\delta} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t, \tag{2.10}$$

where $\Phi(t, \tau_i^j)$ is the state transition matrix in Definition 2.1. Define the matrix $F_i^j \in \mathbf{S}_+^{n_x}$ as

$$F_i^j \coloneqq \Phi(T_\delta, \tau_i^j)^T E \Phi(T_\delta, \tau_i^j). \tag{2.11}$$

Define the sum of these two matrices as

$$S_i^j \coloneqq P_i^j + F_i^j, \quad i = 0, \ldots, N+1. \tag{2.12}$$

Following the convention described in Section 2.3.1, we denote $P_i^0 \coloneqq P_i, F_i^0 \coloneqq F_i, S_i^0 \coloneqq S_i$ and $P_i^{n_i+1} \coloneqq P_{i+1}, F_i^{n_i+1} \coloneqq F_{i+1}, S_i^{n_i+1} \coloneqq S_{i+1}$.

Definition 2.3 (Matrices C)**.** Given matrices S_i with $i = 0, \ldots, N+1$ and $A_i^{n_i}$ with $i = 0, \ldots, N$, define matrices $C_i \in \mathbf{S}_+^{n_x}$ as

$$C_i = Q + \left(A_i^{n_i}\right)^T S_{i+1} + S_{i+1} A_i^{n_i}, \quad i = 0, \ldots, N. \tag{2.13}$$

We now define the auxiliary matrices needed for our computations.

Definition 2.4 (Auxiliary matrices)**.** We define the matrix exponential of the linearized system between time instants τ_i^j and τ_i^{j+1} with $i = 0, \ldots, N$ and $j = 0, \ldots, n_i$ as

$$\mathcal{E}_i^j \coloneqq e^{A_i^j \delta_i^j}, \tag{2.14}$$

Moreover, we define the matrices $M_i^j \in \mathbf{S}_+^{n_x}$ as

$$M_i^j \coloneqq \int_0^{\delta_i^j} e^{A_i^{jT} \eta} Q e^{A_i^j \eta} \mathrm{d}\eta. \tag{2.15}$$

2.4 Numerical solution method

We now describe a novel iterative method to compute the optimal switching times. Our computations can be applied to any second-order algorithm to solve switching time optimization problems.

2.4.1 Iterative algorithm

We can approximate the problem $(\mathcal{P})$ as

$$\begin{aligned} \text{minimize} \quad & \textstyle\int_0^{T_\delta} x(t)^T Q x(t) \mathrm{d}t + x(T_\delta)^T E x(T_\delta) \\ \text{subject to} \quad & \dot{x}(t) = A_i^j x(t), \quad t \in [\tau_i^j, \tau_i^{j+1}), \\ & \qquad i = 0, \ldots, N, \quad j = 0, \ldots, n_i \\ & x(0) = x_0 \\ & \delta \in \Delta. \end{aligned} \tag{$\mathcal{P}_{\text{lin}}$}$$

We make use of problem $(\mathcal{P}_{\text{lin}})$ to approximate the original problem $(\mathcal{P})$ at each iteration of a standard second-order nonlinear programming routine such as IPOPT [177]. By linearizing the system dynamics around the state trajectory we can directly construct problem $(\mathcal{P}_{\text{lin}})$.

In the remainder of the chapter we focus on the numerical evaluation of the cost function J, the gradient ∇J and the Hessian H_J for problem $(\mathcal{P}_{\text{lin}})$, all of which can be computed efficiently.

We sketched a prototype algorithm in Algorithm 1. Note that in lines 3 and 4 the act of linearizing problem $(\mathcal{P})$ and computing $J(\delta)$ produces the majority of the computational work, with the benefit that the cost function derivatives can be then computed efficiently in line 4.

2.4.2 Main result

We are now in the position to derive the cost function and its first and second derivatives for Problem $(\mathcal{P}_{\text{lin}})$.

Theorem 2.1 (Cost function J, gradient ∇J, and Hessian H_J)**.** The following holds:

Algorithm 1 Solve switching time optimization problem ($\mathcal{P}$)

```
function SwitchingTimeOptimization
    while Termination conditions not met do
        Linearize problem (P)
        Compute J(δ), ∇J(δ) and H_J(δ) for (P_lin)
        Perform one NLP solver iteration obtaining a new δ
    end while
end function
```

(i) The cost function J is the following quadratic function of the initial state

$$J(\delta) = x_0^T S_0 x_0. \tag{2.16}$$

(ii) The gradient ∇J of the cost function can be computed as

$$\nabla J(\delta)_i = \frac{\partial J(\delta)}{\partial \delta_i} = x_{i+1}^T C_i x_{i+1}, \quad i = 0, \ldots, N. \tag{2.17}$$

(iii) The Hessian H_J of the cost function can be computed as

$$\begin{aligned} H_J(\delta)_{i,\ell} &= \frac{\partial^2 J(\delta)}{\partial \delta_i \partial \delta_\ell} \\ &= \begin{cases} 2x_{\ell+1}^T C_\ell \Phi(\tau_{\ell+1}, \tau_{i+1}) A_i^{n_i} x_{i+1} & \ell \geq i \\ H_J(\delta)_{\ell,i} & \ell < i, \end{cases} \end{aligned} \tag{2.18}$$

where $i, \ell = 0, \ldots, N$.

The proof can be found in Appendix A.1.

Regardless of the second-order optimization method employed, most of the numerical operations needed to compute $J(\delta), \nabla J(\delta)$ and $H_J(\delta)$ at each iteration are shared. It is therefore necessary to perform them only once per solver iteration.

2.4.3 State propagation and matrix exponentials

The auxiliary matrices in Definition 2.4 can be computed with the following single matrix exponential

$$Z_i^j = e^{G_i^j \delta_i^j} \coloneqq \begin{bmatrix} Z_{i,1}^j & Z_{i,2}^j \\ 0 & Z_{i,3}^j \end{bmatrix}, \quad \text{with} \quad Z_{i,1}^j, Z_{i,2}^j, Z_{i,3}^j \in \mathbf{R}^{n_x \times n_x}. \tag{2.19}$$

and matrices G_i^j being defined as

$$G_i^j \coloneqq \begin{bmatrix} -A_i^{jT} & Q \\ 0 & A_i^j \end{bmatrix}, \quad \text{with} \quad G_i^j \in \mathbf{R}^{2n_x \times 2n_x}. \tag{2.20}$$

After computing Z_i^j, matrices $\mathcal{E}_i^j$ and M_i^j can be obtained as

$$\mathcal{E}_i^j = Z_{i,3}^j \quad \text{and} \quad M_i^j = Z_{i,3}^{jT} Z_{i,2}^j. \tag{2.21}$$

For more details, see [171, Theorem 1].

In Algorithm 2 we describe the subroutine to propagate the state, linearize the dynamics and obtain matrices $\mathcal{E}_i^j$ and M_i^j. At every instant

Algorithm 2 Linearize, compute matrix exponentials and propagate

```
1:  function LinMatExpProp
2:      for i = 0, ..., N do
3:          for j = 0, ..., n_i do
4:              A_i^j ← (2.7)                          ▷ Linearize Dynamics
5:              Z_i^j ← (2.19)                         ▷ Matrix Exponential
6:              E_i^j, M_i^j ← (2.21)
7:              x_i^{j+1} ← E_i^j x_i^j
8:          end for
9:      end for
10:     return x_i,    i = 0, ..., N + 1
11:     return M_i^j, E_i^j,    j = 0, ..., n_i    i = 0, ..., N
12: end function
```

τ_i^j the dynamics are linearized, matrices $\mathcal{E}_i^j, M_i^j$ are computed and the state x_i^j is propagated. Note that as we described in Section 2.3.1, we consider $x_i^0 \coloneqq x_i$ and $x_i^{n_i+1} \coloneqq x_{i+1}$.

Computing matrix exponentials. There are several methods to compute the matrix exponentials as discussed in [127] and [128]. In our work we use "Method 3" in [128, Section 3], being the scaling and squaring method explained in detail in [94] which is in the main linear algebra library of the Julia language. The scaling and squaring method is the most common method used for computing the matrix exponential because of its efficiency and precision. However, in the case of linear dynamics discussed in Section 2.5, the matrices A_i^j are always constant and many operations can be precomputed increasing the speed of the algorithm.

Exponential integrators. The matrix exponentials employed in this section are an implementation of the first-order forward Euler exponential integrator [95]. Exponential integrators perform well in many cases of stiff systems. However, most common numerical methods for exponential integration reduce the operations required by computing directly the product of a matrix exponential and a vector. In our case we not only need to propagate the dynamics, but also to compute the cost function integral. Thus, we need to compute the matrix exponentials Z_i^j which are then used to compute $\mathcal{E}_i^j$ and M_i^j from (2.19).

2.4.4 State Transition Matrices

From Theorem 2.1 we need to compute the state transition matrices between the switching instants. They can be computed recursively using Definition 2.1 which, combined with the definition of the matrix exponentials in (2.14), can be written as

$$\Phi(\tau_\ell, \tau_i) = \prod_{q=i}^{\ell-1} \prod_{p=0}^{n_q} \mathcal{E}_q^p. \tag{2.22}$$

Note that we need to compute the state transition matrices only for all $\ell \geq i$ so that the transition goes forward in time.

2.4.5 Matrices S

To obtain the cost function and its first and second derivatives we need to compute the matrices S_i. Given the matrices $\mathcal{E}_i^j$ and M_i^j, the matrices S_i can be obtained using the following result:

Proposition 2.1 (S recursion)**.** Matrix S_i^j with $i = 0, \dots, N+1$ and $j = 0, \dots, n_i$ satisfy the recursion

$$S_N = E \tag{2.23}$$

$$S_i^j = M_i^j + \mathcal{E}_i^{jT} S_i^{j+1} \mathcal{E}_i^j. \tag{2.24}$$

We describe the proof in Appendix A.2. Note that we defined $S_i^0 \coloneqq S_i$ and $S_i^{n_i+1} \coloneqq S_{i+1}$ as discussed in Section 2.3.1.

2.4.6 Complete algorithm to compute $J(\delta), \nabla J(\delta)$ and $H_J(\delta)$

We now outline the complete algorithm to linearize problem $(\mathcal{P})$ and compute the cost function, the gradient and the Hessian of $(\mathcal{P}_{\text{lin}})$ with respect to the switching intervals in Algorithm 3.

Algorithm 3 Compute $J(\delta), \nabla J(\delta)$ and $H_J(\delta)$

function ComputeCostFunctionAndDerivatives
Shared Precomputations:
  $x_i, \mathcal{E}_i^j, M_i^j \leftarrow$ LinMatExpProp ▷ Algorithm 2
  $S_i \leftarrow$ Compute S ▷ Proposition 2.1
  $C_i \leftarrow$ (2.13) ▷ Definition 2.3
  $\Phi(\tau_l, \tau_i) \leftarrow$ Compute Φ ▷ (2.22)
Compute $J(\delta), \nabla J(\delta)$ and $H_J(\delta)$ ▷ Theorem 2.1
  $J(\delta) \leftarrow$ (2.16)
  $\nabla J(\delta) \leftarrow$ (2.17)
  $H_J(\delta) \leftarrow$ (2.18)
end function

After performing the shared precomputations, we can compute the cost function and its derivatives using Theorem 2.1, with no significant

increase in computation to obtain also the Hessian in order to apply a second-order method.

2.5 Linear switched systems

When the system has linear switched dynamics of the form

$$\dot{x}(t) = A_i x(t), \quad t \in [\tau_i, \tau_{i+1}), \quad i = 0, \dots, N \tag{2.25}$$

the computations can be greatly simplified. In Algorithm 1 there is no need to resort to an auxiliary problem with linearized dynamics. In this case the main result in Theorem 2.1 applies directly to the cost function and derivatives of the original problem $(\mathcal{P})$.

There is no need for a linearization grid when dealing with linear systems. Thus, we simplify all the results for nonlinear dynamics by removing the indices j by setting $n_i = 0$ with $i = 0, \dots, N+1$.

Since the dynamics matrices do not change during the optimization, we precompute the matrices in $G_i = G_i^0$ in (2.19) offline. In addition, if some of the G_i are diagonalizable, they can be factorized offline as

$$G_i = Y_i \Lambda_i Y_i^{-1}, \quad i = 0, \dots, N, \tag{2.26}$$

where Λ_i are the diagonal matrices of eigenvalues and Y_i are the nonsingular matrices of right eigenvectors. The matrix exponentials (2.19) can then be computed online as simple scalar exponentials of the diagonal elements of Λ_i

$$Z_i = Y_i e^{\Lambda_i \delta_i} Y_i^{-1}, \quad i = 0, \dots, N, \tag{2.27}$$

which corresponds to "Method 14" in [127, Section 6] and [128]. Matrices Y_i and Y_i^{-1} can be precomputed offline. Note that the scalar exponentials are independent and can be computed in parallel to minimize the computation times. If G_i are not diagonalizable, we compute the matrix exponentials as in the nonlinear system case with the scaling and squaring method [127, Section 3].

Further improvements in computational efficiency can be obtained in the case of linear dynamics by executing the main for loop in Algorithm 2 in parallel since there is no need to propagate the state and iteratively linearize the system.

2.6 Implementation and examples

All algorithms and examples described in this Chapter have been implemented in the open-source package SwitchTimeOpt in the Julia language, and are publicly available at

`https://github.com/oxfordcontrol/SwitchTimeOpt.jl`

This package allows the user to easily define and efficiently solve switching time optimization problems for linear and nonlinear systems. SwitchTimeOpt supports a wide variety of nonlinear solvers through the MathProgBase interface including IPOPT [177] or KNITRO [36].

For complete documentation of the configurable options for defining problem ($\mathcal{P}$) and the package functionalities we refer the reader to the project website.

For each of the examples described in this section, we interfaced SwitchTimeOpt with the IPOPT solver [177] on a late 2013 Macbook Pro with Intel Core i7 and 16GB of RAM. All the examples are initialized with τ_i equally spaced between between 0 and T. All the examples are solved with the default IPOPT options.

2.6.1 Unstable switched dynamics

Consider the switched system from [37] described by the two unstable dynamics

$$A_1 = \begin{bmatrix} -1 & 0 \\ 1 & 2 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix}. \tag{2.28}$$

Note that A_1 and A_2 have no common eigenvectors. The system transitions happen $N = 5$ times between 0 and $T = 1$ according to the mode sequence $\{1, 2, 1, 2, 1, 2\}$. The cost function matrices are $\overline{Q} = I$ and $\overline{E} = 0$. The approach converges to precision 10^{-8} in roughly 3.5 ms producing the optimal switching times

$$\tau^\star = (0.100, 0.297, 0.433, 0.642, 0.767)$$

which correspond to the same solution obtained in [37]. However, no timing is reported in that work.

```
N = 5  # Number of switching times

# System dynamics
A = zeros(nx, nx, N+1)
A[:,:,1] = [-1 0; 1  2]
A[:,:,2] = [ 1 1; 1 -2]
for i = 3:N+1
  A[:,:,i] = A[:,:,mod(i+1,2)+1]
end

m = stoproblem(x0, A)  # Define problem
solve!(m)              # Solve problem

# Obtain results and timings
tauopt = gettau(m)
Jopt = getobjval(m)
soltime = getsoltime(m)
```

Listing 2.1: SwitchTimeOpt code for the linear example.

To show the implementation ease of our software, we report in Listing 2.1 the code needed to produce this example.

2.6.2 Lotka-Volterra type fishing problem

Problem description. The Lotka-Volterra fishing problem has been studied for almost a century following D'Ancona and Volterra's observation of an unexpected decrease in fishing quotas after World War I [176]. This system has been analyzed from an integer optimal control point of view in [150] and included in a library of standard integer optimal control benchmark problems for nonlinear systems in [149]. Lotka-Volterra systems possess the nonlinear dynamics

$$\dot{x}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_1(t) - x_1(t)x_2(t) - c_1 x_1(t)u(t) \\ -x_2(t) + x_1(t)x_2(t) - c_2 x_2(t)u(t) \end{bmatrix},$$

defining the behavior of the biomass of the prey $x_1(t)$, assumed to grow exponentially, and the predator $x_2(t)$, assumed to decrease exponentially. In addition, there is a coupling term describing the interaction of the

biomasses when the predator eats the prey. The control action is the binary variable $u(t) \in \{0, 1\}$ modeling the decision to fish $u(t) = 1$ or not to fish $u(t) = 0$ at time t. We choose $c_1 = 0.4$ and $c_2 = 0.2$ defining the number of prey and predators caught when fishing occurs.

When no changes in the control action occur, *i.e.*, we are either never fishing or always fishing, the system shows an oscillating behavior which can lead one of the biomasses to disappear [150], destroying the ecosystem. The goal is to responsibly fish in order to bring both the biomasses from an initial value of $x_0 = (0.5, 0.7)$ to the steady state value $(1, 1)$ within the time $T = 12$. In other words, the optimal control problem is a tracking problem where we penalize the deviations from the reference values $x_r(t) = (1, 1)$ by deciding when to start and stop fishing.

Given an integer input sequence $\{u_i\}_{i=0}^{N},\ u_i \in \{0, 1\}$ and N switching times τ_i, the nonlinear dynamics can be described as a switched system of the form

$$\dot{x}(t) = f_i(x(t)) = \begin{bmatrix} x_1(t) - x_1(t)x_2(t) - c_1 x_1(t)u_i \\ -x_2(t) + x_1(t)x_2(t) - c_2 x_2(t)u_i \end{bmatrix},$$

with $t \in [\tau_i, \tau_{i+1}),\ i = 0, \ldots, N$.

The complete optimal control problem can be written as

$$\begin{array}{ll} \text{minimize} & \int_0^{T_\delta} \|x(t) - x_r(t)\|_2^2 \mathrm{d}t \\ \text{subject to} & \dot{x}(t) = f_i(x(t)), \quad t \in [\tau_i, \tau_{i+1}), \\ & \qquad\qquad\qquad\quad i = 0, \ldots, N \\ & x(0) = x_0 \\ & \delta \in \Delta. \end{array} \tag{2.29}$$

We can easily write this problem into the state-regulation form $(\mathcal{P})$ by augmenting the state with $x_r(t)$ with $\dot{x}_r(t) = 0$ and minimizing the deviations between $x(t)$ and $x_r(t)$.

We consider a sequence of $N = 8$ switchings between the two possible input values $\{u_i\}_{i=0}^{N} = \{0, 1, 0, 1, 0, 1, 0, 1, 0\}$ giving a total of 9 dynamics.

Numerical results. We run the algorithm for 20 iterations for increasing number of fixed-grid points $100, 150, 200$ and 250. The optimal switching

times for $n_{\text{grid}} = 200$ are

$$\tau^\star = (2.446, 4.150, 4.533, 4.799, 5.436, 5.616, 6.969, 7.033), \tag{2.30}$$

and the state behavior is displayed in Figure 2.2. The linearized system is also plotted as a dot-dashed green line showing an almost indistinguishable curve.

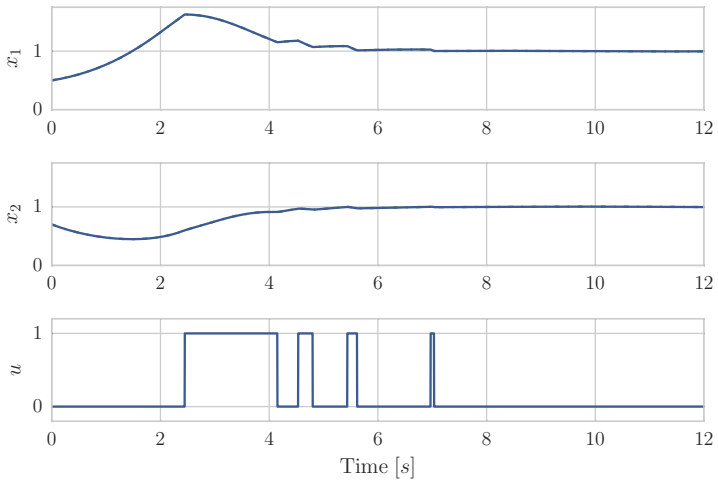


Figure 2.2: Fishing problem. States and input behaviors at the optimal switching times $\tau^\star$. The states of the simulated nonlinear system (blue line) and the linearized system (dot-dashed green line) show a very close match.

The complete results are shown in Table 2.1. The system is simulated at the optimal intervals $\delta^\star$ with an ode45 integrator obtaining the cost function value $J_{\text{ode45}}(\delta^\star)$ and with the grid linearizations obtaining $J(\delta^\star)$ – their values converge as the number of grid points increases. The latter can be seen from the value of $\Delta J(\delta^\star) = \|J_{\text{ode45}}(\delta^\star) - J(\delta^*)\|/\|J_{\text{ode45}}(\delta^\star)\|$ which decreases as the grid becomes finer. The number of cost function evaluations $n_{J,\text{eval}}$ and the computation time are also shown in Table 2.1.

Table 2.1: Results for Lotka-Volterra fishing problem after 20 iterations.

n_{grid}	$J_{\text{ode45}}(\delta^\star)$	$J(\delta^\star)$	$\Delta J\,[\%]$	$n_{J,\text{eval}}$	Time [s]
100	1.3500	1.3508	0.065	177	0.65
150	1.3454	1.3459	0.033	56	0.27
200	1.3456	1.3459	0.016	51	0.29
250	1.3454	1.3455	0.010	54	0.38

For the solver IPOPT, increasing the number of grid points does not necessarily mean a higher computation time, because the latter is strictly related to the number of cost function evaluations which varies depending on the line search steps. We notice that, as the grid becomes finer, *i.e.*, from $n_{\text{grid}} = 100$ to 150, the linear approximation is more precise and the number of line search steps required is lower.

Our results are very close to the solutions in [150] which are obtained with multiple shooting approach discretizing the problem a priori in 60 time instants leading to a mixed-integer optimization problem with 2^{60} possible input combinations. In [150] the authors deal with the required computational complexity by applying several heuristics. Their best cost function value is 1.3451 and is obtained after solving the integer optimal control problem, applying a sum-up-rounding heuristic defined in [150] as "Rounding 2" and using the result to solve a switching time optimization problem with multiple shooting. Even though no timings are provided in [150], timing benchmarks for the multiple shooting approach applied to this problem are provided in the report [148, Section 5.5] where the execution times are approximately 10 times slower than the ones obtained in this work. Note that the implementations in [150] and [148] use the software package MUSCOD-II [96] which is an optimized C++ implementation of the multiple shooting methods, while our approach has been implemented on the high-level language Julia.

2.6.3 Double-tank system

Problem description. The problem of controlling two interconnected tanks using hybrid control appeared in [121]. The authors of [8] applied switching time optimization to obtain the optimal inputs. This example has also been used in [174] and [41] for relaxations in switched control systems. The plant is shown in Figure 2.3.

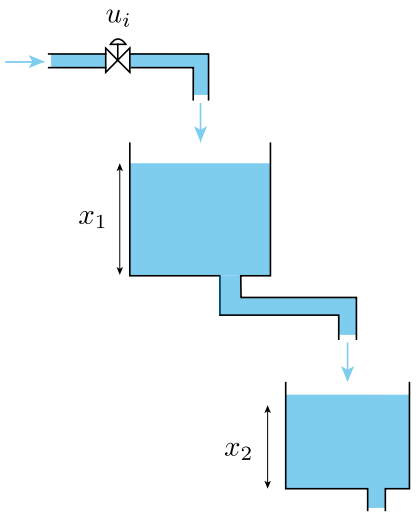


Figure 2.3: Double tank system.

We can write the system dynamics in the form

$$\dot{x}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -\sqrt{x_1(t)} + u(t) \\ \sqrt{x_1(t)} - \sqrt{x_2(t)} \end{bmatrix}, \tag{2.31}$$

where x_1 and x_2 are the fluid levels in the upper and lower tanks respectively. The control action $u(t)$ is the flow into the upper tank which is linked to the valve opening. We assume the input to be either $u_{\min}$

or $u_{\max}$. The goal of the control problem is to track the reference level $x_r(t) = 3 - 0.05t$ with the second tank (the tank is slowly emptying) over the time window from 0 to T. The initial state is $x_0 = (2, 2)$.

The optimal switching times problem has the same form as (2.29), but in this case the reference varies over time. We can bring the problem into state-regulation form by augmenting the state with $x_r(t)$ such that $\dot{x}_r(t) = -0.05$ and $x_r(0) = 3$ and minimizing deviations between $x_2(t)$ and $x_r(t)$.

Numerical results. We run the algorithm for 15 iterations for grid points $10, 30, 50$ and 100. The optimal switching times for $n_{\text{grid}} = 10$ are

$$\begin{aligned}\tau^\star = (0.0, 4.18, 4.92, 4.93, 5.57, 6.12, 6.48, 6.9,\\ 7.26, 7.67, 8.04, 8.43, 8.81, 9.19, 9.56).\end{aligned} \tag{2.32}$$

The behavior of the water levels is simulated with an ode45 integrator and displayed together with the valve opening in Figure 2.4. The linearized system states' behavior is plotted as a dot-dashed green line which coincides with the result from the nonlinear integrator. The dotted black line in the second plot represents the reference water level to be tracked.

The complete results are shown in Table 2.2. The nonlinear system is simulated at the optimal intervals $\delta^\star$ obtaining the cost function $J_{\text{ode45}}(\delta^\star)$ and with the grid linearizations giving $J(\delta^\star)$. The normalized absolute value of their difference tends to 0 as the number of grid points increases. Even if the number of objective function evaluations is not monotonically increasing in the number of fixed grid points, we see an increasing execution time due to the required computations.

Although [8] does not report computation times, in [174, Section 5.2] the authors report execution times on the order of 30 s on an Intel Xeon, 12 core, 3.47 GHz, 92 GB RAM. Our approach is approximately 200 to 550 times faster on a standard laptop. Moreover, the problem described here is slightly more general since the reference is time-varying.

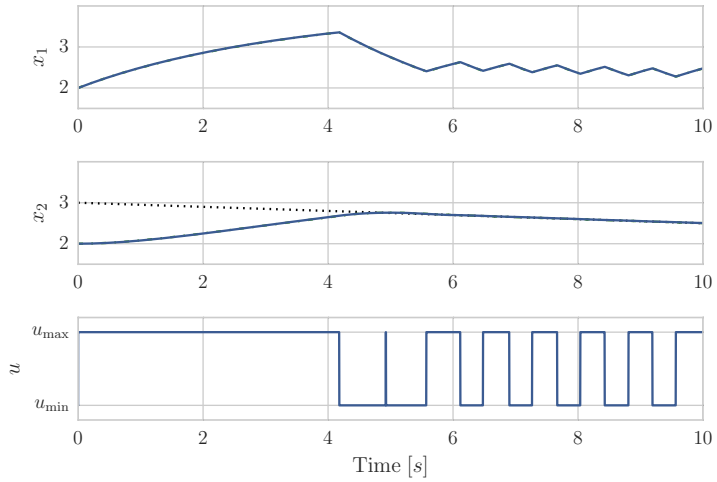


Figure 2.4: Double-tank system. States and input behaviors at the optimal switching times $\tau^\star$. The states of the simulated nonlinear system (blue line) and the linearized system (dot-dashed green line) show a very close match. The dotted black line in the second plot defines the reference to be tracked.

Table 2.2: Results for Double-tank problem after 15 iterations.

n_{grid}	$J_{\text{ode45}}(\delta^\star)$	$J(\delta^\star)$	$\Delta J\,[\%]$	$n_{J,\text{eval}}$	Time [s]
10	1.8595	1.8495	0.537	39	0.05
30	1.8582	1.8573	0.049	39	0.09
50	1.8582	1.8578	0.021	49	0.11
100	1.8582	1.8580	0.010	33	0.12

2.7 Conclusions

We presented a novel method for computing the optimal switching times for linear and nonlinear switched systems. By reformulating the problem with the switching intervals as optimization variables, we derived efficiently computable expressions for the cost function, the gradient and the Hessian which share the most expensive computations. At each iteration of the optimization algorithm, once the cost function value is obtained, there is no significant increase in complexity in computing the gradient and the Hessian. In addition, we showed that in the case of linear dynamics many operations can be performed offline and many online operations parallelized greatly reducing the computation times.

We implemented our method in a new open-source Julia package SwitchTimeOpt which allows the user to quickly define and solve optimal switching time problems. An example with linear dynamics showed that our method can solve switching time optimization problems in millisecond time scales. We also showed with two nonlinear dynamics examples that our high-level Julia implementation can solve these problems with up to two orders of magnitude improvements over state-of-the-art approaches.

3

ADP for Integer Optimal Control

The goal of this chapter is to compute a value function approximation for infinite horizon integer optimal control problems. By exploiting the results from [179] focusing on linear systems with continuous inputs, we formulate the value function approximation problem as a semidefinite program tailored to our specific class of linear systems with integer inputs. In Chapter 4 we will apply the results of this Chapter to reduce the complexity of integer optimal control problems arising in power electronics.

3.1 Optimal control of hybrid linear systems

Consider a discrete-time linear time-invariant (LTI) dynamical system with dynamics

$$x(k+1) = Ax(k) + Bu(k), \quad k = 0, 1, \ldots,$$

where $x(k) \in \mathbf{R}^n$ is the state and $u(k) \in \mathbf{Z}^m$ is the integer input. We consider state feedback control policies [20] of the form

$$u(k) = \phi(x(k)), \quad k = 0, 1, \ldots,$$

where $\phi : \mathcal{X} \to \mathcal{U}$ is the *control policy*.

The *optimal control problem* consists of choosing $u(k) = \phi(x(k))$ to minimize the infinite horizon discounted cost

$$J_\phi = \sum_{k=0}^{\infty} \gamma^k \ell(x(k), u(k)),$$

where $\ell : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$ is the stage cost and $\gamma \in (0, 1)$ is the discount factor. We denote the optimal cost by $J^\star$, *i.e.*, the infimum of J_ϕ over all policies ϕ.

Infinite values of ℓ encode the state and input constraints,

$$(x, u) \in \mathcal{X} \times \mathcal{U} = \{(x, u) \mid \ell(x, u) < \infty\},$$

where $\mathcal{X} \in \mathbf{R}^n$ are the state constraints and $\mathcal{U} \in \mathbf{Z}^m$ are the input constraints.

In the rest of this Chapter we assume no constraints on the state, *i.e.*, $\mathcal{X} = \mathbf{R}^n$ and integer control sets with finite cardinality K. Moreover, we assume a stage cost of the form

$$\ell(z, v) = z^T Q z + l_v, \tag{3.1}$$

where $Q \in \mathbf{S}^n_+$ and $l_v \in \mathbf{R}$. The input cost l_v can be seen as the cost of choosing the input v. We chose this class of systems and this cost because it both simplifies the derivations and will come into play in Chapter 4. Note that state constraints can be included by introducing additional convex constraints in the resulting optimization problem to estimate the value function [179, Section 6].

3.2 Dynamic programming

Let $V^\star$ be the *value function* of the optimal control problem, $V^\star : \mathcal{X} \to \mathbf{R} \cup \{\infty\}$, given by

$$V^\star(z) = \inf_u \left\{ \sum_{k=0}^{\infty} \gamma^k \ell(x(k), u(k)) \right\},$$

subject to the dynamics (3.1) and $x(0) = z$.

The main idea behind dynamic programming is that the function $V^\star$ is the unique solution to the equation

$$V^\star(z) = \inf_u \left\{ \ell(z, u) + \gamma V^\star \left(Az + Bu \right) \right\} \quad \forall z \in \mathcal{X},$$

known as the Bellman equation. The value $V^\star$ is the cost of applying the optimal policy when the initial state is z. The right-hand side can be written more compactly as

$$V^\star = \mathcal{T} V^\star,$$

where $\mathcal{T}$ is usually referred to as the *Bellman operator*

$$(\mathcal{T} q)(z) \coloneqq \inf_u \left\{ \ell(z, u) + \gamma q \left(Az + Bu \right) \right\} \quad \forall z \in \mathcal{X},$$

for any $q : \mathcal{X} \to \mathbf{R}$. Once $V^\star$ is known, the optimal control policy for our problem starting at state z can be found as

$$\phi^\star(z) = \operatorname*{argmin}_u \left\{ \ell(z, u) + \gamma V^\star \left(Az + Bu \right) \right\},$$

for all $z \in \mathcal{X}$.

3.2.1 Properties of the Bellman operator

There are several useful properties of the Bellman operator $\mathcal{T}$ [20, 21, 179].

Monotonicity. For functions $f, g : \mathcal{X} \to \mathbf{R}$, we have

$$f \leq g \quad \Longrightarrow \quad \mathcal{T}f \leq \mathcal{T}g,$$

where we consider the inequality elementwise, *i.e.*, $f(x) \leq g(x)$, $\forall x \in \mathcal{X}$.

Value iteration convergence. For any function $f : \mathcal{X} \to \mathbf{R}$, we have that

$$V^\star(x) = \lim_{k\to\infty} (\mathcal{T}^k f)(x), \quad \forall x \in \mathcal{X}.$$

This means that by applying iteratively the Bellman operator to any function, we converge pointwise to the value function. The methods computing $V^\star$ using this technique are called value iteration.

3.3 Approximate dynamic programming

Unfortunately, solutions to the Bellman equation can only be computed analytically in a limited number of special cases, *e.g.*, when the state and inputs have small dimensions or when the system is linear and unconstrained and the cost function is quadratic [103]. For more complicated problems, dynamic programming is limited by the so-called curse of dimensionality; storage and computation requirements tend to grow exponentially with the problem dimensions. In the case of integer inputs, it is intractable to compute the optimal infinite horizon cost and policy, hence systematic methods for approximating the optimal value function offline are needed.

Approximate dynamic programming (ADP) [21] consists of various techniques for estimating $V^\star$ using knowledge from the system dynamics, fitted data through machine learning or iterative learning through simulations. A common alternative, is to replace the value function with an approximation $\hat{V}$ [21]. The resulting policy can be written as

$$\hat{\phi}(z) = \operatorname*{argmin}_u \left\{ \ell(z,u) + \gamma \hat{V}\left(Az + Bu\right) \right\},$$

for all $z \in \mathcal{X}$. We describe $\hat{\phi}$ as an *ADP policy*. The goal of ADP is to find a policy with close to optimal performance that can be easily evaluated.

3.4 Bellman inequality

In this section show how to compute an underestimator of the true value function by first relaxing the Bellman equality and then restricting the approximated function to a specific family of functions.

We follow the approach developed in [50] and [179] where the authors relax the Bellman equation into an inequality

$$\hat{V}(z) \leq \inf_u \left\{ \ell(z,u) + \gamma \hat{V}\left(Az + Bu\right) \right\}, \quad \forall z \in \mathcal{X}, \tag{3.2}$$

or, equivalently, using the Bellman operator

$$\hat{V} \leq \mathcal{T}\hat{V}. \tag{3.3}$$

The set of functions $\hat{V}$ that satisfy the Bellman inequality are underestimators of the optimal value function $V^\star$. This happens because, if $\hat{V}$ satisfies (3.3), then by the monotonicity of the operator $\mathcal{T}$ and value iteration convergence from Section 3.2.1 we can write

$$\hat{V} \leq \mathcal{T}\hat{V} \leq \mathcal{T}\left(\mathcal{T}\hat{V}\right) \leq \cdots \leq \lim_{i\to\infty} \mathcal{T}^i \hat{V} = V^\star.$$

The Bellman inequality (3.3) is therefore a sufficient condition for underestimating V^*.

3.5 Iterated Bellman inequalities

In [179] the authors show that the conservatism of the approximation can be reduced by iterating (3.2). The iterated Bellman inequality is defined as

$$\hat{V} \leq \mathcal{T}^M \hat{V},$$

where $M > 1$ is an integer defining the number of iterations. Satisfaction of this inequality is equivalent to the existence of functions $\hat{V}_i$ such that

$$\hat{V} \leq \mathcal{T}\hat{V}_1, \quad \hat{V}_1 \leq \mathcal{T}\hat{V}_2, \quad \dots \hat{V}_{M-1} \leq \mathcal{T}\hat{V}.$$

By defining $\hat{V}_0 \coloneqq \hat{V}_M \coloneqq \hat{V}$, we can rewrite the iterated inequality as

$$\hat{V}_{i-1} \leq \mathcal{T}\hat{V}_i, \quad i = 1, \dots, M, \tag{3.4}$$

where $\hat{V}_i$ are the iterates of the value function.

To make the problem tractable, we restrict the iterates to the finite-dimensional subspace spanned by the basis functions $V^{(j)}$ as defined in [50],[179], *i.e.*,

$$\hat{V}_i = \sum_{j=1}^{K} \alpha_{ij} V^{(j)}, \quad i = 0, \dots, M-1. \tag{3.5}$$

Given the inequalities (3.4), we can write the problem of finding the best value function underestimator as

$$\begin{array}{ll} \text{maximize} & \int_{\mathcal{X}} \hat{V}(z) c(\mathrm{d}z) \\ \text{subject to} & \hat{V}_{i-1}(z) \leq \inf_u \{\ell(z,u) + \gamma \hat{V}_i(Az + Bu)\}, \\ & \qquad \forall z \in \mathcal{X}, i = 1, \dots, M, \\ & \hat{V}_0 = \hat{V}_M = \hat{V}, \end{array} \tag{3.6}$$

where c is a non-negative measure over the state space. On the chosen subspace (3.5), the iterated inequalities are convex in the coefficients α_{ij}. To see this, note that the left-hand side is affine in α_{ij}. Moreover, for a fixed u the argument in the infimum in the right-hand side is affine in α_{ij} while the pointwise infimum of affine functions is concave.

The solution to (3.6) is the function spanned by the chosen basis that maximizes the c-weighted 1-norm defined in the cost function while satisfying the iterated Bellman inequality [50, Lemma 1]. Hence, c can be regarded as a distribution giving more importance to regions of the state space where we would like a better approximation.

3.6 Semidefinite program reformulation

Following the approach in [179], we make use of quadratic candidate functions of the form

$$\hat{V}_i(z) = z^T P_i z + 2q_i^T z + r_i, \quad i = 0, \ldots, M, \tag{3.7}$$

where $P_i \in \mathbf{S}_+^n, q_i \in \mathbf{R}^n, r_i \in \mathbf{R}$. This choice is motivated by the fact that in our case the optimal value function is piecewise quadratic or, more specifically, the pointwise minimum of quadratic functions [184]. Hence, we expect a quadratic approximator to have a shape that is compatible with the original value function.

If we define $\mu \in \mathbf{R}^n$ and $\Sigma \in \mathbf{S}_+^n$ as the mean and covariance of measure c respectively, by using quadratic candidate functions the cost of problem (3.6) can be written as

$$\int_{\mathcal{X}} \hat{V}(z) c(\mathrm{d}z) = \mathbf{tr}(P_0 \Sigma) + 2q_0^T \mu + r_0.$$

We now focus on rewriting the constraint in (3.6) as a linear matrix inequality (LMI) [29]. We first remove the infimum on the right-hand side by imposing the constraint for every admissible $u \in \mathcal{U}$ and obtain

$$\hat{V}_{i-1} \leq \ell(z,u) + \gamma \hat{V}_i(Az + Bu), \quad \forall z \in \mathcal{X}, \forall u \in \mathcal{U}, \quad i = 1, \ldots, M.$$

We can then rewrite the constraint as

$$\begin{bmatrix} z \\ 1 \end{bmatrix}^T M_i(u) \begin{bmatrix} z \\ 1 \end{bmatrix} \geq 0, \quad \forall z \in \mathcal{X}, \forall u \in \mathcal{U}, \quad i = 1, \ldots, M, \tag{3.8}$$

where

$$M_i(u) = L + \gamma G_i(u) - S_{i-1} \in \mathbf{S}^n. \tag{3.9}$$

The matrices defining the quadratic form are

$$S_{i-1} \coloneqq \begin{bmatrix} P_{i-1} & q_{i-1} \\ q_{i-1}^T & r_{i-1} \end{bmatrix}, \quad L \coloneqq \begin{bmatrix} Q & 0_{n\times 1} \\ 0_{n\times 1}^T & 0 \end{bmatrix}$$
$$G_i(u) \coloneqq \begin{bmatrix} \Psi^{(i)} & \Phi^{(i)}(u) \\ \Phi^{(i)}(u)^T & \Gamma^{(i)}(u) \end{bmatrix},$$

with

$$\begin{aligned}\Psi^{(i)} &\coloneqq A^T P_{i-1} A,\\ \Phi^{(i)}(u) &\coloneqq A^T P_i B u + A^T q_i,\\ \Gamma^{(i)}(u) &\coloneqq u^T B^T P_i B u + 2 q_i^T B u + r_i.\end{aligned}$$

Using the non-negativity condition of quadratic forms [172], it is easy to see that (3.8) holds if and only if $M_i(u)$ is positive semidefinite. Hence, we can rewrite problem (3.6) as the following semidefinite program (SDP)

$$\begin{array}{ll}\text{maximize} & \mathbf{tr}(P_0\Sigma) + 2q_0^T\mu + r_0\\ \text{subject to} & M_i(u) \succeq 0, \quad \forall u \in \mathcal{U}, \quad i = 1, \ldots, M,\\ & \hat{V}_0 = \hat{V}_M = \hat{V}, \quad P_i \in \mathbf{S}_+,\end{array} \tag{3.10}$$

which can be solved efficiently using a standard SDP solver, *e.g.*, MOSEK [130]. Once the solution is obtained, we can define our value function underestimator as

$$\hat{V}(z) \coloneqq z^T P_0 z + 2q_0^T z + r_0. \tag{3.11}$$

We will use this approach in Chapter 4 when reducing the complexity of integer optimal control problems with fast dynamics.

4

High-Speed Hybrid MPC for Power Electronics

Among the control strategies adopted in power electronics, model predictive control (MPC) [143] has recently gained popularity due to its various advantages [45]. MPC has been shown to outperform traditional control methods mainly because of its ease in handling time-domain constraint specifications, which can be imposed by formulating the control problem as a constrained optimization problem. Due to its structure, MPC can be applied to a variety of power electronics topologies and operating conditions providing a higher degree of flexibility than traditional approaches.

With the recent advances in convex optimization techniques [137], it has been possible to apply MPC to very fast constrained linear systems with continuous inputs by solving convex quadratic optimization problems within microseconds [100]. However, when dealing with nonlinear systems or with integer inputs, the optimal control problems are no longer convex and it is harder to find optimal solutions. For this reason, there are still two orders of magnitude difference in achievable computation time compared to results obtained for linear systems [100] and further advances are required to apply these methods to very fast power electronics systems.

4.1 Model predictive control in power electronics

In power electronics, many conventional control strategies applied in industry are based on proportional-integral controllers (PIs) providing continuous input signals to a modulator that manages conversion to discrete switch positions. Direct MPC [78] instead combines the current control and the modulation problem into a single computational problem, providing a powerful alternative to conventional PI controllers. With direct MPC, the manipulated variables are the switch positions, which lie in a discrete and finite set, giving rise to a switched system. Therefore, this approach does not require a modulator and is often referred to as *finite control set* MPC.

Since the manipulated variables are restricted to be integer-valued, the optimization problem underlying direct MPC is provably $\mathcal{NP}$-hard [23]. In power electronics these optimization problems are often solved by complete enumeration of all the possible solutions, which grow exponentially with the prediction horizon [142]. Since long horizons are required to ensure stability and good closed-loop performance [129], direct MPC quickly becomes intractable for real-time applications. As a consequence, in cases when reference tracking of the converter currents is considered, the controller horizon is often restricted to one [45]. Despite attempts to overcome the computational burden of integer programs in power electronics [107], the problem of implementing these algorithms on embedded systems remains open.

4.1.1 Sphere decoding

A recent technique introduced in [80] and benchmarked in [81] reduces the computational burden of direct MPC when increasing the prediction horizon. In that work the optimization problem was formulated as an integer least-squares (ILS) problem and solved using a tailored branch-and-bound algorithm, described as *sphere decoding* [93], generating the optimal switching sequence at each time step. Although this approach appears promising relative to previous work, the computation time required to perform the sphere decoding algorithm for long horizons (*e.g.*,

$N = 10$), is still far slower than the sampling time typically required, *i.e.*, $T_s = 25\,\mu s$.

Some approaches have been studied to improve the computational efficiency of the sphere decoding algorithm. In particular, in [105] a method based on a lattice reduction algorithm decreased the average computational burden of the sphere decoding. However, the worst case complexity of this new reformulation is still exponential in the horizon length. In [106], heuristic search strategies for the sphere decoding algorithm were studied at the expense of suboptimal control performance. Even though a floating point complexity analysis of the algorithms is presented in these works, no execution times and no details about fixed-point implementation are provided. Furthermore, there currently exists no embedded implementation of a direct MPC algorithm for current control achieving comparable performance to formulations with long prediction horizons.

4.1.2 Our approach

In this chapter we introduce a different method to deal with the direct MPC problem. In contrast to common formulations [114] where the switching frequency is controlled indirectly via penalization of the input switches over the controller horizon, in this work the system dynamics are augmented to directly estimate the switching frequency. Our approach allows the designer to set the desired switching frequency a priori by penalizing its deviations from this estimate. Thus, the cost function tuning can be performed more easily than with the approach in [80] and [81], where a tuning parameter spans the whole frequency range with no intuitive connection to the desired frequency. To address the computational issues of long prediction horizons, we formulate the tracking problem as a regulation one by augmenting the state dynamics and cast it in the framework of approximate dynamic programming (ADP) [21] introduced in Chapter 3. The infinite horizon value function is approximated using the approach in [50] and [179] by solving an semidefinite program (SDP) [172] offline. This enables us to shorten the controller horizon by applying the estimated tail cost to the last stage to maintain good control performance. In [24] the authors applied a similar approach to stochastic systems with

continuous inputs, describing the control law as the "iterated greedy policy".

As a case study, we applied our method to a variable-speed drive system consisting of a three-level neutral point clamped voltage source inverter connected to a medium-voltage induction machine. The plant is modeled as a linear system with a switched three-phase input with equal switching steps for all phases.

Closed loop simulations in MATLAB in steady state operation showed that our method, with very short prediction horizons, gives better performance than the approach in [80] and [81] with much longer planning horizons.

We have implemented our algorithm on a small size Xilinx Zynq FPGA (xc7z020) in fixed-point arithmetic and verified its performance with processor-in-the-loop (PIL) tests of both steady-state and transient performance. The results achieve almost identical performance to closed-loop simulations and very fast computation times, allowing us to comfortably run our controller within the 25 µs sampling time.

Coordinates and per-unit system. In this work we use normalized quantities by adopting the per-unit system (pu). The time scale is also normalized using the base angular velocity ω_b that in this case is $2\,\pi 50\,\mathrm{rad/s}$, *i.e.*, one time unit in pu corresponds to $1/\omega_b\,\mathrm{s}$.

In this chapter we make use of vectors in the three-phase system (abc) and in the stationary orthogonal coordinates ($\alpha\beta$). If not specified, vectors are in the orthogonal coordinates $\alpha\beta$ reference frame. For more details see B.1.

4.2 Drive system case study

We consider a variable-speed drive system as shown in Figure 4.1, consisting of a three-level neutral point clamped voltage source inverter driving a medium-voltage induction machine. The total dc-link voltage V_{dc} is assumed constant and the neutral point potential N fixed.

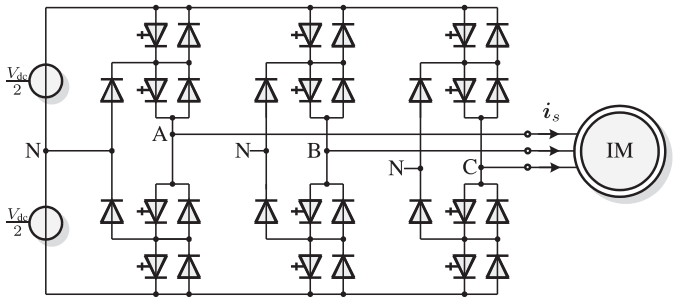


Figure 4.1: Three-level three-phase neutral point clamped (NPC) voltage source inverter driving an induction motor with a fixed neutral point potential.

In modern techniques to control variable-speed drive systems, the control is split into two cascaded loops. The outer loop controls the machine speed by manipulating the torque reference. The inner loop controls the torque and the fluxes by manipulating the voltages applied to the stator windings of the machine. Our approach focuses on the inner loop. The reference torque is converted into stator currents references that must be tracked and the controller manipulates the stator voltages by applying different inverter switch positions.

We can write the system dynamics as the following discrete-time linear time-invariant (LTI) system

$$\begin{aligned} x_{\text{ph}}(k+1) &= A_{\text{ph}}x_{\text{ph}}(k) + B_{\text{ph}}u_{\text{sw}}(k) \\ y_{\text{ph}}(k) &= D_{\text{ph}}x_{\text{ph}}(k), \end{aligned} \tag{4.1}$$

where the state vector $x_{\text{ph}} = (i_s, \psi_r) \in \mathbf{R}^4$ includes the stator current and rotor flux in the $\alpha\beta$ reference frame. The output vector is the stator current, *i.e.*, $y_{\text{ph}} = i_s \in \mathbf{R}^2$. The input $u_{\text{sw}} = (u_a, u_b, u_c) \in \{-1, 0, 1\}^3$ represents the switch positions in the three phase legs corresponding to different voltages $\{-V_{\text{dc}}/2, 0, V_{\text{dc}}/2\}$ applied to the stator windings. The sampling time is $T_s = 25\,\mu\text{s}$. See Appendix B for a detailed derivation.

4.3 Model predictive current control

Our control scheme must address two conflicting objectives simultaneously. On the one hand, the distortion of the stator currents i_s causes iron and copper losses in the machine leading to thermal losses. Because of the limited cooling capability of the electrical machine, the stator current distortions have to be kept as low as possible. On the other hand, high frequency switching of the inputs u_{sw} produces high power losses and stress on the semiconductor devices. Owing to the limited cooling capability in the inverter, we therefore should minimize the switching frequency of the integer inputs.

The controller sampling time plays an important role in the distortion and switching frequency tradeoff. Depending on the precision required in defining the inverter switching times, the controller is discretized with larger (*e.g.*, 125 µs) or smaller (*e.g.*, 25 µs) sampling times. Larger sampling times define a more coarse discretization grid leading to less precise definition of the switching instants, but more available time to perform the computations during the closed-loop cycles. Lower sampling times, on the other hand, lead to improved controller accuracy while reducing the allowed computing time. However, for the same switching frequency, longer sampling times produce higher distortions. Ideally, we should choose the sampling time as low as possible to have the highest possible accuracy.

4.3.1 Machine design based approaches

The effect of the inverter switchings on the torque ripples can be improved during the machine design. In particular, increasing the time constants of the stator and the rotor τ_s and τ_r can reduce the amplitude of the torque ripples by decreasing the derivative of the currents i_s and fluxes φ_r. This is achieved naturally when dealing with machines with higher power. Thanks to the flexibility of model based controller designs such as MPC, different machine dynamics influencing the torque ripples are automatically taken into account by the controller, which adapts the optimal input computation depending on the plant parameters. Thus, any improvements during the machine design can be optimally exploited by

adapting the internal model dynamics in the controller.

Another similar approach is to include LCL filters between the inverter and the motor to decrease the high frequency components of the currents; see [152]. These approaches allow operation at lower switching frequencies with low THD at the same time. However, it is sometimes impossible to change the machine's physical configuration and it is necessary to operate the inverter at high switching frequencies to satisfy high performance requirements in terms of stator currents distortion.

4.3.2 Total harmonic distortion

The current distortion is measured via the total harmonic distortion (THD). Given an infinitely long time-domain current signal i and its fundamental component $i^\star$ of constant magnitude, the THD is proportional to the root mean square value of the difference $i - i^\star$. Hence, we can write for one phase current

$$THD \sim \lim_{M\to\infty} \sqrt{\frac{1}{M}\sum_{k=0}^{M-1} (i(k) - i^\star(k))^2}, \tag{4.2}$$

with $M \in \mathbf{Z}_{++}$. For three-phase currents the THD is proportional to the mean value of (4.2) over the phases. It is of course not possible to calculate the THD in real-time within our controller computations because of finite storage constraints.

To overcome this problem, we will embed the desired currents into the state dynamics. According to (4.2), the THD is proportional to the stator current ripple which can be written in the $\alpha\beta$ coordinate system [80] as

$$THD \sim \lim_{M\to\infty} \sum_{k=0}^{M-1} \|e_i(k)\|_2^2, \tag{4.3}$$

where we have introduced the error signal $e_i := i_s - i_s^\star$. It is straightforward to show [60] that the stator current reference during steady-state operation at rated frequency is given by

$$i_s^\star(k) = (\sin(k), -\cos(k)). \tag{4.4}$$

Hence, in order to minimize the THD, we minimize the squared 2-norm of vector e_i over all future time steps. We also introduce a discount factor $\gamma \in (0, 1)$ to normalize the summation preventing it from going to infinity due to persistent tracking errors. The cost function related to THD minimization is therefore

$$\sum_{k=0}^{\infty} \gamma^k \left\| e_i(k) \right\|_2^2 . \tag{4.5}$$

In order to construct a regulation problem, we include the oscillating currents from (4.4) as two additional uncontrollable states $x_{\text{osc}} = i_s^\star$ within our model of the system dynamics. We then model the ripple signal e_i as an output defined by the difference between two pairs of system states.

4.3.3 Switching frequency

The switching frequency of the inverter can be identified by computing the average frequency of each active semiconductor device. As displayed in Figure 4.1, the total number of switches in all three phases is 12, and, for each switching transition by one step up or down in a phase, one semiconductor device is turned on. Thus, the number of *on* transitions occurring between time step $k-1$ and k is given by the 1-norm of the difference of the inputs vectors: $\|u_{\text{sw}}(k) - u_{\text{sw}}(k-1)\|_1$.

Given a time interval centered at the current time step k from $k-M$ to $k+M$, it is possible to estimate the switching frequency by counting the number of *on* transitions over the time interval and dividing the sum by the interval's length $2MT_s$. We then can average over all the semiconductor switches by dividing the computed fraction by 12. At time k, the switching frequency estimate can be written as

$$f_{\text{sw,M}}(k) \coloneqq \frac{1}{12 \cdot 2MT_s} \sum_{i=-M}^{M} \left\| u_{\text{sw}}(k+i) - u_{\text{sw}}(k+i-1) \right\|_1, \tag{4.6}$$

which corresponds to a non-causal finite impulse response (FIR) filter of order $2M$. The true average switching frequency is the limit of this

quantity as the window length goes to infinity

$$f_{\mathrm{sw}} \coloneqq \lim_{M\to\infty} f_{\mathrm{sw,M}}(k), \tag{4.7}$$

and does not depend on time k.

The f_{sw} computation brings similar issues as the THD. In addition to finite storage constraints, the part of the sum regarding the future signals produces a non-causal filter that is impossible to implement in a real-time control scheme.

To overcome the difficulty of dealing with the filter in (4.6), we consider only the past input sequence, with negative time shift giving a causal FIR filter estimating f_{sw}. Note that we take into account future input sequences in (4.6) inside the controller prediction. The FIR filter could be estimated with a high order LTI system so that there is one state per time step from 0 to $-M$. However, M is usually in the order of thousands and the resulting filter dimension would be prohibitive for embedded controllers with limited computational capabilities. Instead, this filter is approximated with an infinite impulse response (IIR) one whose dynamics can be modeled as a LTI system. This approach can also be interpreted as a model order reduction technique for FIR filters [102].

Let us define three binary phase inputs denoting whether each phase switching position changed at time k or not, *i.e.*,

$$p(k) \coloneqq (p_a(k), p_b(k), p_c(k)) \in \{0,1\}^3, \tag{4.8}$$

with

$$p_s(k) = \left\| u_s(k) - u_s(k-1) \right\|_1, \; s \in \{a,b,c\}. \tag{4.9}$$

The FIR filter to approximate can be then written as

$$\hat{f}_{\mathrm{sw}}(k+1) = \frac{1}{12T_s M} \sum_{i=0}^{M} p_a(k-i) + p_b(k-i) + p_c(k-1),$$

where $\hat{f}_{\mathrm{sw}}(k)$ is the estimated switching frequency. We can approximate this IIR filter with the following first-order FIR filter

$$\hat{f}_{\mathrm{sw}}(k+1) = a_1 \hat{f}_{\mathrm{sw}}(k) + \frac{1-a_1}{12T_s} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} p(k)$$

with pole $a_1 = 1 - 1/r_1$ describing the memory of the filter and $r_1 >> 0$. Our numerical simulations showed that a higher order approximation can provide more accurate and smooth frequency estimators. Thus, we construct a second-order IIR filter by extending the last one obtaining

$$x_{\text{flt}}(k+1) = \underbrace{\begin{bmatrix} a_1 & 0 \\ 1-a_1 & a_2 \end{bmatrix}}_{A_{\text{flt}}} x_{\text{flt}}(k) + \underbrace{\frac{1-a_2}{12T_s}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{B_{\text{flt}}} p(k) \tag{4.10}$$

$$\hat{f}_{\text{sw}}(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} x_{\text{flt}}(k), \tag{4.11}$$

where $x_{\text{flt}}(k)$ is the filter state. The two poles at $a_1 = 1 - 1/r_1$ and $a_2 = 1 - 1/r_2$ with $r_1, r_2 >> 0$ can be tuned to shape the behavior of the filter. Increasing a_1, a_2 makes the estimate smoother, while decreasing a_1, a_2 gives faster estimation with more noisy values.

We denote the difference between the approximation $\hat{f}_{\text{sw}}$ and the target frequency $f^\star_{\text{sw}}$ by $e_{\text{sw}} \coloneqq \hat{f}_{\text{sw}} - f^\star$. The quantity to be minimized in order to bring the switching frequency estimate as close to the target as possible is therefore

$$\sum_{k=0}^{\infty} \delta \gamma^k \left\| e_{\text{sw}}(k) \right\|_2^2, \tag{4.12}$$

where $\delta \in \mathbf{R}_+$ is a design parameter included to reflect the importance of this part of the cost relative to the THD component.

Finally, we can augment the state space to include the filter dynamics and the target frequency by adding the states $(x_{\text{flt}}, f^\star_{\text{sw}})$ so that the control inputs try to drive the difference between two states to zero. Since the physical states are expressed in the pu system with values around 1, in order to have these augmented states within the same order of magnitude we will normalize them by the desired frequency $f^\star_{\text{sw}}$ defining $x_{\text{sw}} \coloneqq ((1/f^\star_{\text{sw}})x_{\text{flt}}, 1)$ and the matrices $A_{\text{sw}} \coloneqq \mathbf{blkdiag}(A_{\text{flt}}, 1)$, $B_{\text{sw}} \coloneqq (B_{\text{flt}}, 0)$.

4.3.4 Model predictive control formulation

States. Let us define the complete augmented state as

$$x(k) \coloneqq (x_{\text{ph}}(k), x_{\text{osc}}(k), x_{\text{sw}}(k), u_{\text{sw}}(k-1)), \tag{4.13}$$

with $x(k) \in \mathbf{R}^9 \times \{-1, 0, 1\}^3$ and total state dimension $n = 12$. Vector x_{ph} represents the physical system from Section 4.2, x_{osc} defines the oscillating states of the sinusoids to track introduced in Section 4.3.2, and $u_{\text{sw}}(k-1)$ are additional states used to keep track of the physical switch positions at the previous time step. The vector x_{sw} are the states related to the switching filter from Section 4.3.3.

Inputs. We denote the inputs as

$$u(k) \coloneqq (u_{\text{sw}}(k), p(k)), \tag{4.14}$$

with $u(k) \in \mathbf{R}^m$ and input dimension $m = 6$. The vector u_{sw} are the physical switch positions and p are the three binary inputs entering in the frequency filter from Section 4.3.3. To simplify the notation let us define the matrices G and T to obtain $u_{\text{sw}}(k)$ and $p(k)$ from $u(k)$ respectively, *i.e.*, such that $u_{\text{sw}}(k) = Gu(k)$ and $p(k) = Tu(k)$. Similarly, to obtain $u_{\text{sw}}(k-1)$ from $x(k)$ we define a matrix W so that $u_{\text{sw}}(k-1) = Wx(k)$.

Augmented system dynamics. Given the state x definition in (4.13) and the input u definition (4.14) we can define the dynamics

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k), \end{aligned} \tag{4.15}$$

where $A \coloneqq \mathbf{blkdiag}(A_{\mathrm{ph}}, A_{\mathrm{osc}}, A_{\mathrm{sw}}, 0)$ and

$$B \coloneqq \begin{bmatrix} B_{\mathrm{ph}} & 0 \\ 0 & 0 \\ 0 & B_{\mathrm{flt}} \\ I & 0 \end{bmatrix},$$

$$C \coloneqq \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \delta & -\delta & 0 & 0 & 0 \end{bmatrix}.$$

They consist of the physical model (4.1) together with the augmentations in Sections 4.3.3 and 4.3.2.

MPC Problem. We can write the MPC problem with horizon $N \in \mathbf{Z}_{++}$ as

$$\begin{array}{ll} \text{minimize} & \sum_{k=0}^{N-1} \gamma^k \ell(x(k)) + \gamma^N \hat{V}(x(N)) \\ \text{subject to} & x(k+1) = Ax(k) + Bu(k) \\ & x(0) = x_0 \\ & x(k) \in \mathcal{X}, \quad u(k) \in \mathcal{U}(x(k)). \end{array} \tag{4.16}$$

The stage cost combines the THD and the switching frequency penalties in (4.5) and (4.12) respectively

$$\ell(x(k)) = \|Cx(k)\|_2^2 = \|e_i(k)\|_2^2 + \delta \|e_{\mathrm{sw}}(k)\|_2^2 .$$

The function $\hat{V}$ is an approximation of the infinite horizon tail cost. The matrices A, B and C define the extended system dynamics (4.15) and the output vector.

We define the input constraints set as

$$\mathcal{U}(x(k)) \coloneqq \Big\{ \|Tu(k)\|_\infty \leq 1, \tag{4.17}$$

$$-Tu(k) \leq Gu(k) - Wx(k) \leq Tu(k), \tag{4.18}$$

$$Gu(k) \in \{-1, 0, 1\}^3 \Big\}, \tag{4.19}$$

where constraint (4.17) defines the relationship between u_{sw} and p from (4.8) and (4.9). Constraint (4.18) together with (4.17) defines the switching constraints $\|u_{\text{sw}}(k) - u_{\text{sw}}(k-1)\|_\infty \leq 1$ imposed to avoid shoot-throughs in the inverter positions that could damage the components. Finally, (4.19) enforces integrality of the switching positions.

The number of switching sequence combinations grows exponentially with the horizon length N, *i.e.*, $3^{3N} = 27^N$. The problem therefore becomes extremely difficult to solve for even modest horizon lengths.

Observe that the controller tuning parameters are δ, which defines the relative importance of the THD and f_{sw} components of the cost function, and r_1, r_2, which shape the switching frequency estimator.

Tail cost approximation. We compute the tail cost $\hat{V}$ using ADP-based methods developed in Chapter 3 by solving an SDP offline. The reader can find the detailed derivation of the optimization problem in Appendix B.5. The resulting quadratic function of the form (3.11) can be used directly in problem (4.16) providing good predictive behavior with a limited number of initial computation steps.

4.3.5 Control loop

The complete block diagram is shown in Figure 4.2. The desired torque $T^\star$ determines the currents x_{osc} by setting the initial states of the oscillator "OSC". The motor speed ω_r and the stator currents i_s are measured directly from the machine. They enter in the observer "OBS" providing the physical states of the motor x_{ph}. The auxiliary inputs p are fed into the filter "FLT" estimating the switching frequency in x_{sw}. The switch positions u_{sw} go through a one step delay and are exploited again by the MPC formulation.

Following a receding horizon control strategy, at each stage k we solve the problem (4.16), obtaining the optimal sequence $\{u^\star(k)\}_{k=0}^{N-1}$ from which only $u^\star(0)$ is applied to the switches. At the next stage $k+1$, given new vectors $x_{\text{osc}}(k), x_{\text{ph}}(k), u_{\text{sw}}(k-1)$ and $x_{\text{sw}}(k)$ as in Figure 4.2, we solve a new optimization problem providing an updated optimal switch-

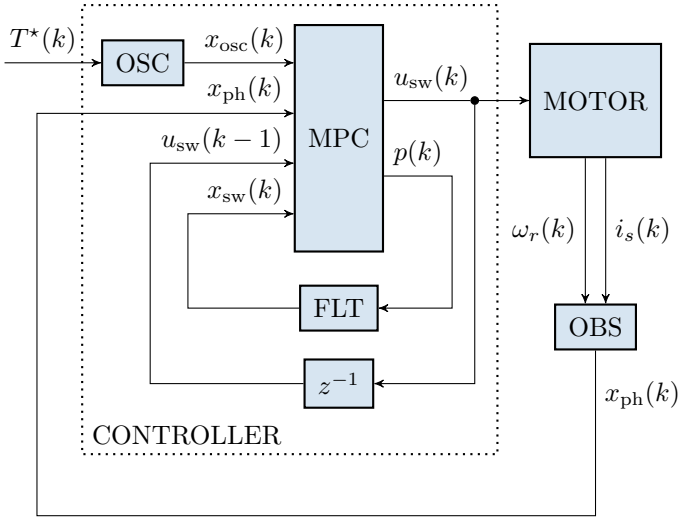


Figure 4.2: Block diagram of the control loop. The controller within the dotted line receives the desired torque $T^\star(k)$ and the motor states $x_{\mathrm{ph}}(k)$ providing the switch position $u_{\mathrm{sw}}(k)$.

ing sequence, and so on. The whole control algorithm, appearing within the dotted line, runs within 25 µs.

4.3.6 Integer optimization problem

Since we consider short horizons, we adopt a condensed MPC formulation of problem (4.16) with only input variables, producing a purely integer program. In this way all possible discrete input combinations can be evaluated directly. With a sparse formulation including the continuous states within the variables, it would be necessary to solve a mixed-integer program requiring more complex computations.

Let us define the input sequence over the horizon N starting at time 0 as

$$u = (u(0), u(1), \ldots, u(N-1)) \in \mathbf{R}^{6N}, \tag{4.20}$$

where we have dropped the time index from u to simplify the notation. We can rewrite problem (4.16) as a parametric integer quadratic program in the initial state x_0,

$$\begin{array}{ll} \text{minimize} & u^T Q u + 2f\,(x_0)^T\, u \\ \text{subject to} & Fu \leq g(x_0) \\ & u \in \mathbf{Z}^{6N}. \end{array} \tag{4.21}$$

In Appendix B.6 we describe the algebraic manipulations performed to derive this problem.

4.4 Framework for performance evaluation

To benchmark our algorithm we consider a neutral point clamped voltage source inverter connected to a medium-voltage induction machine and a constant mechanical load. We consider the same model as in [80]: a 3.3 kV and 50 Hz squirrel-cage induction machine rated at 2 MV A with a total leakage inductance of 0.25 pu. On the inverter side, we assume the dc-link voltage $V_{\mathrm{dc}} = 5.2\,\mathrm{kV}$ to be constant and the potential of the neutral point to be fixed. The base quantities of the pu system are the

following: $V_b = \sqrt{2/3}V_{\text{rat}} = 2694\,\text{V}, I_b = \sqrt{2}I_{\text{rat}} = 503.5\,\text{A}$ and $f_b = f_{\text{rat}} = 50\,\text{Hz}$. Quantities $V_{\text{rat}}, I_{\text{rat}}$ and f_{rat} refer to the rated voltage, current and frequency respectively. The detailed parameters are provided in Table 4.1. The switching frequency is typically between 200 and 350 Hz for medium-voltage inverters [81]. If not otherwise stated, all simulations were done at rated torque, nominal speed and fundamental frequency of 50 Hz.

Table 4.1: Rated values and parameters of the drive [81]

Induction Motor				Inverter	
Voltage	3300 V	R_s	0.0108 pu	V_{dc}	1.930 pu
Current	356 A	R_r	0.0091 pu	x_c	11.769 pu
Real power	1.587 MW	X_{ls}	0.1493 pu		
Apparent power	2.035 MW	X_{lr}	0.1104 pu		
Frequency	50 Hz	X_m	2.3489 pu		
Rotational speed	596 rpm				

We consider an idealized model with the semiconductors switching instantaneously. As such, we neglect second-order effects like deadtimes, controller delays, measurement noise, observer errors, saturation of the machine's magnetic material, variations of the parameters and so on. This is motivated by the fact that, using a similar model, previous simulations [79] showed a very close match with the experimental results in [140]. All the steady-state simulations in the following sections were also performed with model mismatch of $\pm 1\,\%$ in all the parameters of Table 4.1 showing negligible variations in the THD. However, we omit these benchmarks since an exhaustive sensitivity analysis is out of the scope of this chapter.

4.5 Achievable performance in steady-state

We performed closed loop simulations in steady-state operation in MATLAB to benchmark the achievable performance in terms of THD

and switching frequency. The system was simulated for 4 periods before recording to ensure it reaches steady-state operation. We computed THD and switching frequency over simulations of 20 periods. The discount factor was chosen as $\gamma = 0.95$ and the switching frequency filter parameters as $r_1 = r_2 = 800$ in order to get a smooth estimate. We chose weighting δ such that the switching frequency is around 300 Hz. The infinite horizon estimation SDP (B.7) was formulated using YALMIP [120] with $M = 50$ Bellman iterations and was solved offline using MOSEK [130]. Note that in case of a change in the systems parameters, *e.g.*, the dc-link voltage or the rotor speed, the tail cost has to be recomputed. However, it possible to precompute offline and store several quadratic tail costs for different possible parameters and evaluate the desired one online without significant increase in complexity.

For comparison, we simulated the drive system also with the direct MPC controller described in [80] (denoted DMPC) tuned in order to have the same switching frequency by adjusting the weighting parameter λ_u.

The integer optimization problems were solved using GUROBI Optimizer [92]. Numerical results with both approaches appear in Table 4.2. Note that the choice of the solver does not influence the THD or the switching frequency and we would have obtained the same results with another optimization software.

Table 4.2: Simulation Results with ADP and with DMPC from [80] at switching frequency 300 Hz

	ADP		DMPC [80]	
	δ	THD [%]	λ_u	THD [%]
$N = 1$	4	5.24	0.00235	5.44
$N = 2$	5.1	5.13	0.00690	5.43
$N = 3$	5.5	5.10	0.01350	5.39
$N = 10$	10	4.80	0.10200	5.29

Our method, with a horizon of $N = 1$ provided both a THD improvement over the DMPC formulation in [80] with $N = 10$ and drastically better numerical speed. This shows how choosing a meaningful cost function

can provide good control performance without recourse to long horizons. Moreover, we also performed a comparison with longer horizons $N = 2$, $N = 3$ and $N = 10$. Our method, with horizon $N = 10$ would give an even greater reduction in THD to 4.80 %. However, it would be impossible to compute the solutions within the required sampling time since the total number of combinations would be $27^{10} \approx 210$ trillion.

4.6 FPGA implementation

4.6.1 Hardware setup

We implemented the control algorithm on a Xilinx Zynq (xc7z020) [183], a low cost field-programmable gate array (FPGA), running at approximately 150 MHz mounted on the Zedboard evaluation module [6]. Figure 4.3 shows the hardware used in the experiments. We coded the control algorithm in C++ using the PROTOIP Toolbox [110]. The FPGA vendor's High-Level Synthesis tool Xilinx Vivado HLS [182] was used to convert the written code to VHSIC hardware description language (VHDL) defining the Programmable Logic connections.

4.6.2 Algorithm description

We now present a detailed description of how the controller within the dotted lines in Figure 4.2 was implemented on the FPGA.

We implemented the updates in OSC and FLT as simple matrix multiplications. The solver for the integer problem (4.21) was implemented with a simple exhaustive search algorithm for three reasons: first, the tail cost approximation provides good performance with very few horizon steps while considering a relatively small number of input combinations; second, the structure of the problem allows us to evaluate both the inequalities and cost function for multiple input sequences in parallel; third, the FPGA logic is particularly suited for highly pipelined and/or parallelized operations, which are at the core of exhaustive search.

To exploit the FPGA architecture, we implemented our algorithm in fixed-point arithmetic using custom data types defined in Vivado HLS [182]. In particular, we used 4 integer and 0 fractional bits to describe the integer inputs and 2 integer and 22 fractional bits to describe the states and the cost function values. This choice is given by the minimum number of bits necessary to describe these quantities from floating-point simulations in Section 4.5. Note that the exhaustive search algorithm does not suffer from any accumulation of rounding error because it consists entirely of independent function evaluations, in contrast to iterative optimization algorithms [137].

We provide pseudo-code for our method in Algorithm 4. From Figure 4.2,the controller receives the required torque $T^\star(k)$ and the motor states $x_{\text{ph}}(k)$ and returns the switch positions $u_{\text{sw}}(k)$. From line 3 to line 7 the oscillator OSC and the filter FLT serve to compute the new initial state x_0 for the optimization algorithm. Note that if there is a change in the required torque then we reset the oscillator states $x_{\text{osc}}(k)$ to match the new $T^\star(k)$. Line 8 precomputes the vectors in problem (4.21) depending on x_0.

The main loop iterating over all input combinations is split into two subloops: loop 1, which is completely decoupled and can be parallelized; and loop 2, which can only be sequential.

Loop 1. Loop 1 from line 9 to 15 computes the cost function values for every combination i and stores it into vector J. All the possible input sequence combinations are saved in the static matrix U. For every loop cycle, sequence i is saved into variable u. Then, in line 10, we update the value of $p(k)$ inside u with $u_{\text{sw}}(k-1)$ according to (4.8) and (4.9). If u does not satisfy the constraint $Fu \leq g(x_0)$, then the we set the related cost to a high value J_{ub} (line 11). Note that even though it would bring considerable speed improvements, we do not precompute offline the quadratic part $u^T Qu$ of the cost and the left side of the inequality Fu since it would also require enumeration over inputs at the previous control cycle used in line 10. Each iteration of this loop is independent from the others and can therefore be parallelized efficiently.

Algorithm 4 Controller Algorithm

function ComputeMPCInput($T^\star(k), x_{\mathrm{ph}}(k)$)
Data: $x_{\mathrm{osc}}(k-1)$, $x_{\mathrm{sw}}(k-1)$, $p(k-1)$ and $u_{\mathrm{sw}}(k-1)$
Parameters: $U \in \mathbf{Z}^{6\times 27^N}$, $J_{\mathrm{ub}} \in \mathbf{R}$
Initialize: $J \in \mathbf{R}^{27^N}$, $J_{\min} \in \mathbf{R}$ and $i_{\min} \in \mathbf{Z}$
Execute oscillator and filter to obtain initial state:
    $x_{\mathrm{osc}}(k) \leftarrow A_{osc} x_{\mathrm{osc}}(k-1)$
    if change in $T^\star(k)$ **then**
        $x_{\mathrm{osc}}(k) \leftarrow$ Reset according to (B.3)
    end if
    $x_{\mathrm{sw}}(k) \leftarrow A_{\mathrm{sw}} x_{\mathrm{sw}}(k-1) + B_{\mathrm{sw}} p(k-1)$
    $x_0 \leftarrow (x_{\mathrm{ph}}(k), x_{\mathrm{osc}}(k), x_{\mathrm{sw}}(k), u_{\mathrm{sw}}(k-1))$
Precompute vectors:
    $f(x_0) \leftarrow$ from (B.13), $g(x_0) \leftarrow$ from (4.17), (4.18) and (4.19).
Loop 1 – Compute cost function values:
    for $i = 1, \ldots, 27^N$ **do**
        $u \leftarrow U_{(:,i)}$, $u_{(4:6)} \leftarrow \|u_{(1:3)} - u_{\mathrm{sw}}(k-1)\|_1$
        $J_{(i)} \leftarrow u^T Q u + 2 f(x_0)^T u$
        if $Fu > g(x_0)$ **then**
            $J_{(i)} \leftarrow J_{\mathrm{ub}}$
        end if
    end for
Loop 2 – Find minimum:
    $J_{\min} \leftarrow J_{\mathrm{ub}}$, $i_{\min} \leftarrow 1$
    for $i = 1, \ldots, 27^N$ **do**
        if $J_{(i)} \leq J_{\min}$ **then**
            $J_{\min} \leftarrow J_{(i)}$, $i_{\min} \leftarrow i$
        end if
    end for
Return results
    return $u_{\mathrm{sw}}(k) \leftarrow U_{(1:3,i_{\min})}$, $p(k) \leftarrow U_{(4:6,i_{\min})}$
end function

Loop 2. Loop 2 from line 16 to 21 is a simple loop iterating over the computed cost function values to find the minimum and save it into $J_{\min}$. Every iteration depends sequentially on $J_{\min}$ which is accessed and can be modified at every i. Thus, in this form it is not possible to parallelize this loop.

4.6.3 Circuit generation

In Vivado HLS [182] it is possible to specify directives to optimize the circuit synthesis according to the resources available on the target board. Loop 1 and loop 2 were pipelined and the preprocessing operations from line 3 to 8 parallelized. We generated the circuit for the algorithm 4 with horizons $N = 1$ and $N = 2$ at frequency 150 MHz (clock cycle of 7 ns). The resources usage and the timing estimates are displayed in Table 4.3. Since timing constraints were met, there was no need to parallelize loop 1 to reduce computation time.

Table 4.3: Resources Usage and Timing Estimates for Implementation on the Xilinx Zynq FPGA (xc7z020) running at 150 MHz

		$N = 1$	$N = 2$
FPGA Resources	LUT	15127 (28 %)	31028 (58 %)
	FF	11156 (10 %)	20263 (19 %)
	BRAM	6 (2 %)	21 (7 %)
	DSP	89 (40 %)	201 (91 %)
Clock Cycles		371	1953
Delay		2.60 µs	13.67 µs

Note that for $N = 2$ we are using already 91 % of the DSP multipliers. This is due to the limited amount of resources available on the low-cost hardware chosen.

4.7 Processor-in-the-loop tests

We performed PIL experiments using the controller FPGA fixed-point implementation developed in Section 4.6 and the machine model described in Section 4.4.

We operated the control loop using the PROTOIP toolbox [110]: the plant model was simulated on a Macbook Pro 2.8 GHz Intel Core i7 with 16GB of RAM while the control algorithm was entirely executed on the Zedboard development board described in Section 4.6.1.

4.7.1 Steady State

We benchmarked the steady-state operation to compare the performance to the results obtained in Table 4.2. We chose the same controller parameters as in Section 4.5.

The PIL tests for horizon $N = 1$ are shown in Figure 4.4 in the pu system. The three-phase stator currents are displayed over a fundamental period in Figure 4.4a, the three spectra are shown in Figure 4.4c with THD of 5.23 % and the input sequences are plotted in Figure 4.4b.

From the experimental benchmarks with horizon $N = 1$ and $N = 2$ we obtained THD = 5.23 % and THD = 5.14 % respectively. As expected, these results are very close to the simulated ones in Table 4.2. The slight difference ($\sim 0.01\,\%$) comes from the fixed-point implementation of the oscillator OSC and the filter FLT in Figure 4.2.

4.7.2 Transients

One of the main advantages of direct MPC is the fast transient response [80]. We tested torque transients in PIL with the same tuning parameters as in the steady state benchmarks. At nominal speed, reference torque steps were imposed; see Figure 4.5b. These steps were translated into different current references to track, as shown in Figure 4.5a, while the computed inputs are shown in Figure 4.5c.

The torque step from 1 to 0 in the per unit system presented an extremely short settling time of 0.35 ms, similar to deadbeat control ap-

proaches [45]. This was achieved by inverting the voltage applied to the load. Since we prohibited switchings between -1 and 1 in (4.17) and (4.18), the voltage inversion was performed in $2T_s$ via an intermediate zero switching position.

Switching from 0 to 1 torque produced much slower response time of approximately 3.5 ms. This is due to the limited available voltage in the three-phase admissible switching positions. As shown in Figure 4.5c, during the second step at time 20 ms, the phases b and c saturated at the values $+1$ and -1 respectively for the majority of the transient providing the maximum available voltage that could steer the currents to the desired values.

These results match the simulations of the formulation in [80] in terms of settling time showing that our method possesses the fast dynamical behavior during transients typical of direct current MPC.

As noted in [80], having a longer horizon or a better predictive behavior does not significantly improve the settling times. This is because the benefit of longer prediction obtainable by extending the horizon or adopting a powerful final stage costs is reduced by the saturation of the inputs during the transients.

4.7.3 Execution Time

To show that the controller is able to run on cheap hardware within $T_s = 25\,\mu s$, we measured the time the FPGA took to execute Algorithm 4 for horizon $N = 1$ and $N = 2$. Since there are no available DMPC sphere decoding algorithm execution times, we compared our results to the time needed to solve the DMPC formulation in [81] for the same horizon lengths on a Macbook Pro with Intel Core i7 2.8 GHz and 16GB of RAM using the commercial integer program solver GUROBI Optimizer [92] which implements an efficient branch-and-bound algorithm. The results are shown in Figure 4.6.

The FPGA execution times were 5.76 µs and 17.27 µs for horizon $N = 1$ and $N = 2$ respectively. Note that they presented a slight overhead of approximately 3.5 µs compared to the estimates in Table 4.3 since the measured times included the time needed to exchange the input-output

data from the FPGA to the ARM processor through RAM. Without the overhead, the estimated FPGA computing times obtained by the circuit generation are exact; see [183].

Note that the time needed by the FPGA to compute the control algorithm is deterministic with zero variance. This makes our PIL implementation particularly suited for real-time applications. Furthermore, it is important to underline that the method we propose is the *only* method available that can produce integer optimal solutions to this problem achieving this performance in 25 µs sampling time.

GUROBI optimizer needed (621.20 ± 119.98) µs and (750.40 ± 216.15) µs to complete the operations for horizons $N = 1$ and $N = 2$ respectively. The non-negligible standard deviation appeared because of the branch-and-bound algorithm implemented in GUROBI. However, since we are considering real-time applications, we are interested in the worst case number of visited nodes which is always the whole tree of combinations, *i.e.*, 27^N. Note that the DMPC formulation was solved in [81] using a different branch-and-bound algorithm based on the sphere decoding algorithm [93], but the worst case number of visited nodes cannot be easily reduced because of the $\mathcal{NP}$-hardness of the problem.

4.8 Conclusions

This Chapter proposed a new computationally efficient direct MPC scheme for current reference tracking in power converters. We extended the problem formulation in [80] and [81] in order to include a switching frequency estimator in the system state and rewrite the optimal control problem as a regulation one. To reduce the horizon length and decrease the computational burden while preserving good control performances, we estimated the infinite horizon tail cost of the MPC problem formulation using ADP.

Steady-state simulation results showed that with our method requiring short horizons, it is possible to obtain better performance than the direct MPC formulation in [81] with long horizons. This is due to the predictive behavior of the tail cost function obtained with ADP.

The control algorithm was implemented in fixed-point arithmetic on

the low size Xilinx Zynq FPGA (xc7z020) for horizons $N = 1$ and $N = 2$. PIL tests during steady-state operation showed an almost identical performance to the simulation results. We also performed transient simulations where our proposed approach exhibited the same very fast dynamic response as the direct MPC described in [81]. Moreover, we showed that our algorithm can run within the sampling time of 25 µs by measuring the execution time on the FPGA. Results showed that only 5.76 µs and 17.27 µs are required to run our controller for horizons $N = 1$ and $N = 2$ respectively.

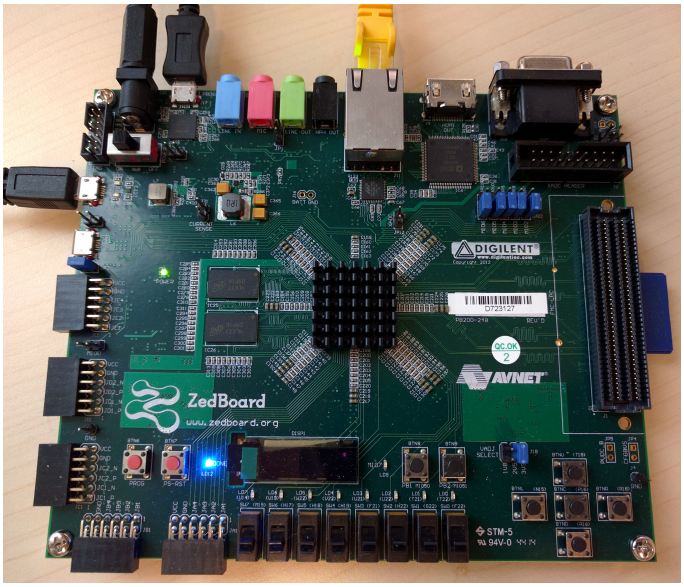
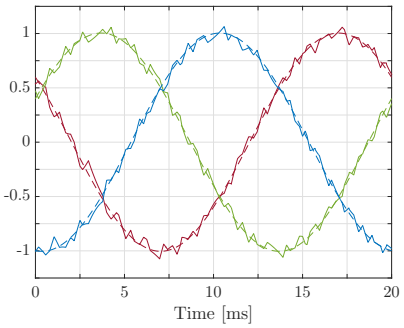
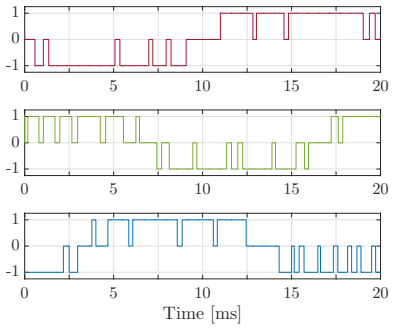


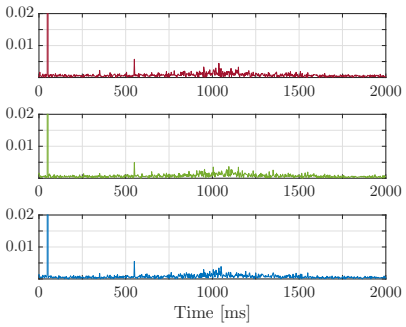
Figure 4.3: Zedboard Evaluation Board used for PIL Tests. The controller runs on the FPGA while the plant is simulate on the laptop. The states and input vectors are passed via the ethernet cable (yellow). The micro-usb cable on the left side provides a UART interface with the laptop used to print if there are any problems in the communication. The cable in the top left corner is connected to the power supply while the other micro-usb cable next to it provides access to the USB-JTAG interface to program the FPGA module.



(a) Three-phase stator currents (solid lines) with their references (dashed lines).

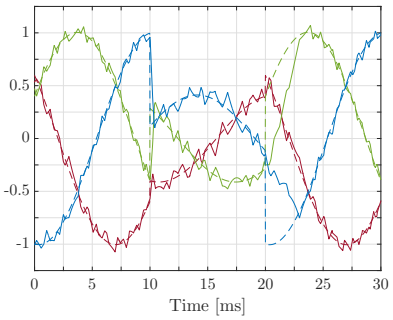


(b) Three-phase switch positions inputs.

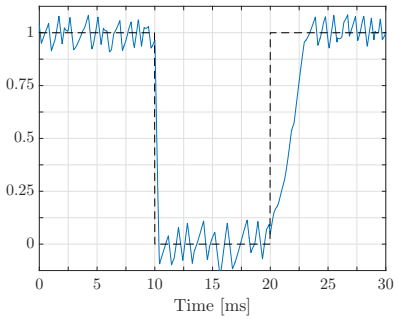


(c) Stator current spectrum.

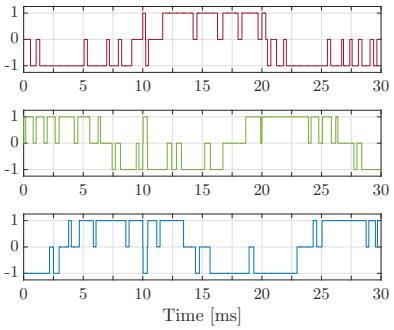
Figure 4.4: Waveforms produced during PIL Tests by the direct model predictive controller at steady state operation, at full speed and rated torque. Horizon of $N = 1$ is used. The switching frequency is approximately 300 Hz and the current THD is 5.23 %.



(a) Three-phase stator currents (solid lines) with their references (dashed lines).



(b) Actual (solid line) and reference (dashed line) torques.



(c) Three-phase switch positions inputs.

Figure 4.5: Reference torque steps produced by the direct model predictive controller in PIL tests with horizon $N = 1$.

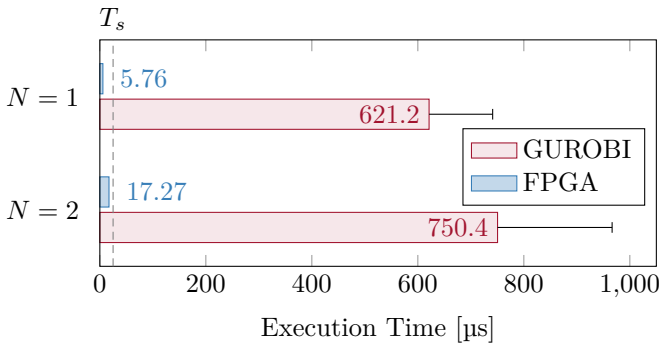


Figure 4.6: Execution times required by the Xilinx Zynq FPGA (xc7z020) to execute our controller based on an ADP formulation (blue) and using GUROBI Optimizer [92] to solve the formulation in [80] on a Macbook Pro with Intel Core i7 2.8 GHz and 16GB of RAM.

Part II

Exact Solution Methods

5

An Operator Splitting Solver for Quadratic Programs

5.1 Introduction

In this Chapter we propose a new solver for quadratic program (QP) based on first-order methods. The proposed algorithm and implementation will be used in Chapter 6 to develop an exact solution method for mixed-integer quadratic programs (MIQPs) based on branch-and-bound.

5.1.1 The problem

Consider the following optimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & Ax \in \mathcal{C}, \end{array} \tag{5.1}$$

where $x \in \mathbf{R}^n$ is the decision variable. The objective function is defined by a positive semidefinite matrix $P \in \mathbf{S}_+^n$ and a vector $q \in \mathbf{R}^n$, and the

constraints by a matrix $A \in \mathbf{R}^{m \times n}$ and a nonempty, closed and convex set $\mathcal{C} \subseteq \mathbf{R}^m$. We will refer to it as a *general (convex) quadratic program.*

If the set $\mathcal{C}$ takes the form

$$\mathcal{C} = [l, u] := \{z \in \mathbf{R}^m \mid l_i \leq z_i \leq u_i,\ i = 1, \ldots, m\},$$

with $l_i \in \{-\infty\} \cup \mathbf{R}$ and $u_i \in \mathbf{R} \cup \{+\infty\}$, we can write problem (5.1) as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \end{array} \tag{5.2}$$

which we will denote as a *quadratic program.* Linear equality constraints can be encoded by setting $l_i = u_i$ for some or all of the elements in (l, u). Note that any linear program (LP) can be written in this form by setting $P = 0$. We characterize the size of (5.2) with the tuple (n, m, N) where N is the total number of nonzero entries in P and A, *i.e.*, $N := \mathrm{nnz}(P) + \mathrm{nnz}(A)$.

Applications. Optimization problems of the form (5.1) arise in a huge variety of applications in engineering, finance, operations research and many other fields. Applications in machine learning include support vector machine (SVM) [44], lasso [168, 38] and Huber fitting [98, 99]. Financial applications of (5.1) include portfolio optimization [42, 122, 31, 28] [33, §4.4.1]. In the field of control engineering, model predictive control (MPC) [143, 75] and moving horizon estimation (MHE) [3] techniques require the solution of a QP at each time instant. Several signal processing problems also fall into the same class [33, §6.3.3][123]. In addition, the numerical solution of QP subproblems is an essential component in nonconvex optimization methods such as sequential quadratic programming (SQP) [137, Chap 18] and mixed-integer optimization using branch-and-bound algorithms [15, 68].

5.1.2 Solution methods

Convex QPs have been studied since the 1950s [70], following from the seminal work on LPs started by Kantorovich [104]. Several solution meth-

ods for both LPs and QPs have been proposed and improved upon throughout the years.

Active set methods. Active set methods were the first popular QP solution algorithms [180] obtained from an extension of Dantzig's simplex method for solving LPs [47]. Active set algorithms select an active set (*i.e.*, a set of binding constraints) and then iteratively adapt it by adding and dropping constraints from the index of active ones [137, §16.5]. New active constraints are added based on the cost function gradient and the current dual variables. Active set methods for QPs differ from the simplex method for LPs because the iterates are not necessarily vertices of the feasible region. These methods can easily be warm-started to reduce the number of active set recalculations required. However, the major drawback of active set methods is that the worst-case complexity grows exponentially with the number of constraints, since it may be necessary to investigate all possible active sets before reaching the optimal one [112]. Modern implementation of active set methods for the solution of QPs can be found in most commercial solvers such as MOSEK [130] and GUROBI [92], and in the open-source solver qpOASES [65].

Interior-point methods. Interior-point algorithms gained popularity in the 1980s as a method for solving LPs in polynomial time [108, 83]. In the 90s these techniques were extended to general convex optimization problems, including QPs [136]. Interior-point methods model the problem constraints as parametrized penalty functions, also referred to as *barrier functions*. At each iteration an unconstrained optimization problem is solved for varying barrier function parameters until the optimum is achieved; see [33, Chap. 11] and [137, §16.6] for details. Primal-dual interior-point methods, in particular the Mehrotra predictor-corrector [125] method, became the algorithms of choice for practical implementation [181] because of their good performance across a wide range of problems. However, interior-point methods are not easily warm-started and do not scale well for very large problems. Interior-point methods are currently the default algorithms in the commercial solvers MOSEK [130],

FORCES [57], GUROBI [92] and CVXGEN [124] and in the open-source solvers OOQP [77].

First-order methods. First-order optimization methods date back to the 1950s, when they were already applied to solving quadratic programs [70]. These methods iteratively compute an optimal solution using only first-order information about the cost function. Operator splitting techniques such as the Douglas-Rachford splitting [119, 58] are a particular class of first-order methods which model the optimization problem as a sum of nonlinear operators.

In recent years, the operator splitting method known as the alternating direction method of multipliers (ADMM) [74, 87] has received particular attention because of its very good practical convergence behavior; see [32] for a survey. ADMM can be seen as a variant of the classical alternating projections algorithm [13] for finding a point in the intersection of two convex sets, and can also be shown to be equivalent to the Douglas-Rachford splitting [74]. ADMM has been shown to reliably provide modest accuracy solutions to QPs in a relatively small number of computationally inexpensive iterations. It is therefore well suited to applications such as embedded optimization or large-scale optimization, wherein high accuracy solutions are typically not required due to noise in the data and arbitrariness of the cost function. ADMM steps are computationally very cheap and simple to implement, and thus ideal for embedded processors with limited computing resources such as embedded control systems [100, 139, 155]. ADMM is also compatible with distributed optimization architectures enabling the solution of very large scale problems [32].

A drawback of first-order methods is that they are typically unable to detect primal and/or dual infeasibility. In order to address this shortcoming, a homogeneous self-dual embedding was proposed in [138] in conjunction with ADMM for solving conic optimization problems, including QPs, and implemented in the open-source solver SCS. A further drawback of ADMM is that the number of iterations required to converge is highly dependent on the problem data and on the user's choice of the algorithm's step-size parameters. Despite some recent theoretical results [85, 10], it remains unclear how to select those parameters to optimize the algorithm

convergence. For this reason, even though there are several benefits in using ADMM techniques for solving optimization problems, there exists no reliable general-purpose QP solver based on operator splitting methods.

5.1.3 Embedded applications

An important feature of QP solvers is the possibility of being deployed on embedded hardware for real-time applications.

Real-time applications impose special requirements on the solvers used [124]. First, embedded solvers must be reliable even in the presence of poor quality data, and should avoid exceptions caused by division by zero or memory faults caused by dynamic memory allocation. Second, the solver should be implementable on low-cost embedded platforms with very limited memory resources. In particular, solvers should have very small compiled footprint, should consist only of basic algebraic operations, and should not be linked to any external libraries, which also makes the solver easily verifiable. Finally, real-time applications typically require that the solver is fast and able to correctly identify infeasible problems.

On the other hand, optimization problems arising in embedded applications have certain features that can be exploited when designing an embedded solver [124]. First, embedded optimization is typically applied to the repeated solution of parametrized problems in which the problem data, but not its dimensions or sparsity pattern, change between problem instances. For such problems, the solver initialization and some part of its computations can be performed offline during the solver design phase. Second, requirements on the solution accuracy in embedded applications are often moderate because of noise in the data and arbitrariness of the objective function. Also, in embedded applications one can typically assume that problems are reasonably scaled. As an example, the authors in [178] show that acceptable control performance of an MPC controller is achievable even when using a very low accuracy solver. This argument supports the use of first-order optimization methods, which are known to return solution of medium accuracy with low computational cost.

Related work. In some cases solution of a parametrized convex optimization problem can be precomputed offline using multi-parametric programming techniques [18, 169]. However, the memory required for storing such solutions grows exponentially with the problem dimensions, making this approach applicable only to relatively small problems.

In the last decade tools for generating custom online solvers for parametric problems have attracted increasing attention. CVXGEN [124] is a code generation software tool for small-scale parametric QPs. The generated solver is fast and reliable, but its main disadvantage is that the code size grows rapidly with the problem dimensions. This issue is overcome in FORCES [57] where the code size of the compiled code is broadly constant with respect to the problem dimensions. In HPMPC [73] tailored solvers for MPC are combined with high-performance optimized libraries for linear algebra. ECOS [56, 39] and Bsocp [59] are embedded solvers for a wider class of second-order cone program programs (SOCPs). All of the aforementioned solvers are based on primal-dual interior-point methods that are tailored for their specific problem classes.

In contrast, qpOASES [65] is based on a parametric active-set method which can effectively use a priori information to speed-up computation of a QP solution. On the other hand, since qpOASES is based on dense linear algebra it cannot exploit sparsity in the problem data. Moreover, as noted in Section 5.1.2 the computational complexity of active-set methods grows exponentially with the number of constraints.

FiOrdOs [170] uses first-order gradient methods as the basis for the embedded solvers it generates. In the case of a general QPs, these methods require a Lipschitz constant of the gradient of the objective function in order to compute the stepsize. Alternatively, FiOrdOs implements an adaptive rule for the stepsize selection, but it requires a new matrix factorization each time the stepsize is updated. QPgen [85] uses optimal preconditioning of the problem data that can improve performance of first-order methods considerably. The main disadvantage of FiOrdOs and QPgen is their inability to detect infeasible problems.

5.1.4 Our approach

In this work we present a new general-purpose QP solver based on ADMM that is able to provide high accuracy solutions. The proposed algorithm is based on a novel splitting requiring the solution of a quasi-definite linear system that is always solvable for any choice of problem data. We therefore pose no requirements such as strict convexity of the cost function or linear independence of the constraints. Since the linear system matrix coefficients remain the same at every iteration, we perform only a single initial factorization at the beginning of the algorithm. Once the initial factorization is computed, the algorithm is *division-free*. In contrast to other first-order methods, our approach is able to return primal and dual solutions when the problem is solvable or to provide certificates of primal and dual infeasibility without resorting to the homogeneous self-dual embedding.

To obtain high quality solutions, we perform *solution polishing* on the iterates obtained from ADMM. By identifying the active constraints from the final dual variable iterates, we construct an ancillary equality-constrained QP whose solution is equivalent to that of the original QP (5.1). This ancillary problem is then solved by computing the solution of a single linear system with usually much lower dimensions than the one solved during the ADMM iterations. When the polishing phase is successful, the resulting solution of our method has accuracy comparable or higher than interior-point methods.

Our algorithm can be efficiently warm-started to reduce the number of iterations. Moreover, if problem matrices do not change, the quasi-definite system factorization can be reused across multiple solves, greatly improving the computation time. This feature is particularly useful when solving parametric QPs where only a few elements of the problem data change. Examples illustrating the effectiveness of the proposed algorithm in parametric programs arising in embedded applications appear in [12].

We have implemented our method in the open-source "Operator Splitting Quadratic Program" (OSQP) solver. OSQP is written in C and can be compiled to be library free. OSQP is robust against noisy and unreliable problem data, has a very small code footprint, and is suitable for both embedded and large-scale applications. Our software features

complete code generation functionality able to produce tailored C code suitable for embedded hardware and requiring only static memory allocation. Numerical benchmarks for both desktop and embedded versions of our solver show that our algorithm is able to provide up to one order of magnitude computational time improvements over existing commercial and open-source solvers on a wide variety of applications.

5.2 Optimality conditions

We will find it convenient to rewrite problem (5.1) by introducing an additional decision variable $z \in \mathbf{R}^m$, to obtain the equivalent problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & z \in \mathcal{C}. \end{array} \tag{5.3}$$

We can write the optimality conditions of problem (5.3) as [145, Thm. 6.12][33]

$$Ax = z, \tag{5.4}$$

$$Px + q + A^T y = 0, \tag{5.5}$$

$$z \in \mathcal{C}, \quad y \in N_{\mathcal{C}}(z), \tag{5.6}$$

where $y \in \mathbf{R}^m$ is a Lagrange multiplier associated with the constraint $Ax = z$ and $N_{\mathcal{C}}(z)$ denotes the normal cone of $\mathcal{C}$ at z. If there exist $x \in \mathbf{R}^n$, $z \in \mathbf{R}^m$ and $y \in \mathbf{R}^m$ that satisfy the conditions above, then we say that (x, z) is a *primal* and y is a *dual* solution of problem (5.3). We define the primal and dual residuals of problem (5.1) as

$$r_{\text{prim}} := Ax - z, \tag{5.7}$$

$$r_{\text{dual}} := Px + q + A^T y. \tag{5.8}$$

Quadratic programs. In case of QPs of the form (5.2), condition (5.6) reduces to

$$l \leq z \leq u, \qquad y_+^T(z - u) = 0, \qquad y_-^T(z - l) = 0, \tag{5.9}$$

where $y_+ \coloneqq \max(y, 0)$ and $y_- \coloneqq \min(y, 0)$ componentwise.

5.2.1 Certificates of primal and dual infeasibility

From the *theorem of strong alternatives* [33, Section 5.8.2], exactly one of the following sets is nonempty

$$\begin{aligned}\mathcal{P} &= \{x \in \mathbf{R}^n \mid Ax \in \mathcal{C}\}, \\ \mathcal{D} &= \left\{y \in \mathbf{R}^m \mid A^T y = 0, \quad S_{\mathcal{C}}(y) < 0\right\},\end{aligned}$$

where $S_{\mathcal{C}}$ is the support function of $\mathcal{C}$, provided that some type of constraint qualification holds [33]. This means that the problem is either feasible, *i.e.*, $\mathcal{P}$ is nonempty, or the set $\mathcal{D}$ is nonempty. In other words, any variable $y \in \mathcal{D}$ serves as a *certificate* that problem (5.1) is primal infeasible.

Quadratic programs. In case $\mathcal{C} = [l, u]$, certifying primal infeasibility of (5.2) amounts to finding a vector $y \in \mathbf{R}^m \setminus \{0\}$ such that

$$A^T y = 0, \quad u^T y_+ + l^T y_- < 0. \tag{5.10}$$

Similarly, it can be shown that a vector $x \in \mathbf{R}^n$ satisfying

$$Px = 0, \quad q^T x < 0, \quad (Ax)_i \begin{cases} = 0 & l_i, u_i \in \mathbf{R} \\ \geq 0 & u_i = +\infty,\ l_i \in \mathbf{R} \\ \leq 0 & l_i = -\infty,\ u_i \in \mathbf{R} \end{cases} \tag{5.11}$$

is a certificate of dual infeasibility for problem (5.2).

5.3 Solution with ADMM

Our method solves the problem (5.3) using the alternating direction method of multipliers (ADMM) [32]. In the proposed novel splitting the

subproblems in each algorithm step are always solvable independently from the problem data. By introducing auxiliary variables $\tilde{x} = x$ and $\tilde{z} = z$, we can rewrite problem (5.3) as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^T P\tilde{x} + q^T\tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x},\tilde{z}) + \mathcal{I}_{\mathcal{C}}(z) \\ \text{subject to} & (\tilde{x},\tilde{z}) = (x,z), \end{array} \tag{5.12}$$

where $\mathcal{I}_{Ax=z}$ and $\mathcal{I}_{\mathcal{C}}$ are the indicator functions of the sets $\{(x,z) \in \mathbf{R}^n \times \mathbf{R}^m \mid Ax = z\}$ and $\mathcal{C}$, respectively. For a set $\mathcal{X}$, we define the indicator function as $\mathcal{I}_{\mathcal{X}}(x) \coloneqq 1$ if $x \in \mathcal{X}$ and 0 otherwise.

An iteration of ADMM for solving problem (5.12) consists of the following steps:

$$\begin{aligned} (\tilde{x}^{k+1}, \tilde{z}^{k+1}) \leftarrow & \underset{(\tilde{x},\tilde{z}):A\tilde{x}=\tilde{z}}{\operatorname{argmin}} \tfrac{1}{2}\tilde{x}^T P\tilde{x} + q^T\tilde{x} \\ & \qquad + \tfrac{\sigma}{2}\|\tilde{x} - x^k + \tfrac{1}{\sigma}w^k\|_2^2 + \tfrac{\rho}{2}\|\tilde{z} - z^k + \tfrac{1}{\rho}y^k\|_2^2 \end{aligned} \tag{5.13}$$

$$x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \tfrac{1}{\sigma}w^k \tag{5.14}$$

$$z^{k+1} \leftarrow \Pi\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \tfrac{1}{\rho}y^k\right) \tag{5.15}$$

$$w^{k+1} \leftarrow w^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1}\right) \tag{5.16}$$

$$y^{k+1} \leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\right) \tag{5.17}$$

where $\sigma > 0$ and $\rho > 0$ are the *step-size parameters* and $\alpha \in (0,2)$ is the *relaxation parameter*. Π denotes the Euclidean projection onto set $\mathcal{C}$ defined as $\Pi(x) \coloneqq \operatorname{argmin}_{z\in\mathcal{C}} \|x - z\|_2$. The iterates w^k and y^k are associated with the dual variables of the equality constraints $\tilde{x} = x$ and $\tilde{z} = z$, respectively. Observe from steps (5.14) and (5.16) that $w^{k+1} = 0$ for all k, and consequently the w-iterate and the step (5.16) can be disregarded.

5.3.1 Solving the linear system

Evaluating the ADMM step (5.13) involves solving the equality constrained quadratic optimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^T P\tilde{x} + q^T\tilde{x} + \frac{\sigma}{2}\|\tilde{x} - x^k\|_2^2 + \frac{\rho}{2}\|\tilde{z} - z^k + \frac{1}{\rho}y^k\|_2^2 \\ \text{subject to} & A\tilde{x} = \tilde{z}. \end{array} \tag{5.18}$$

The optimality conditions for this equality constrained QP are

$$P\tilde{x}^{k+1} + q + \sigma(\tilde{x}^{k+1} - x^k) + A^T\nu^{k+1} = 0, \tag{5.19}$$

$$\rho(\tilde{z}^{k+1} - z^k) + y^k - \nu^{k+1} = 0, \tag{5.20}$$

$$A\tilde{x}^{k+1} = \tilde{z}^{k+1}, \tag{5.21}$$

where $\nu^{k+1} \in \mathbf{R}^m$ is a Lagrange multiplier associated with the constraint $Ax = z$. By eliminating the variable $\tilde{z}^{k+1}$ from (5.20), the above linear system reduces to

$$\begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho}y^k \end{bmatrix}, \tag{5.22}$$

where $\tilde{z}^{k+1}$ is recoverable as

$$\tilde{z}^{k+1} = z^k + \tfrac{1}{\rho}(\nu^{k+1} - y^k).$$

We will refer to the coefficient matrix in (5.22) as the *KKT matrix*. This matrix has always full rank thanks to the parameters σ and ρ introduced in our splitting. Therefore, (5.22) has always a unique solution independently from the problem data. Our splitting allows to solve the linear system in (5.22) using tailored formulations for either a *direct method* or an *indirect method*.

Direct method. The direct method finds an exact solution of the linear system (5.22) by first computing a factorization of the KKT matrix and then performing forward and backward substitution. Since the KKT matrix remains the same for every iteration of ADMM, we need to perform the factorization only once prior to the first iteration and cache the factors so that we can reuse them in subsequent iterations. This approach is very efficient when the factorization cost is considerably higher than the solve cost, so that each iteration is computed quickly.

The KKT matrix is quasi-definite, *i.e.*, it can be written as a 2-by-2 block-symmetric matrix where the $(1,1)$-block is positive definite, and the $(2,2)$-block is negative definite. It therefore always has a well defined

LDL^T factorization, with L being a lower triangular matrix with unit diagonal elements and D a diagonal matrix with nonzero diagonal elements [173]. Note that once the factorization is carried out, computing the solution of (5.22) can be made division-free by storing the D^{-1} instead of D.

When the KKT matrix is sparse and quasidefinite, efficient algorithms can be used for computing a suitable permutation for which the factorization of the permuted KKT matrix results in a sparse factor L [49] without regard for the actual non-zero values appearing in the KKT matrix. The LDL^T factorization is therefore decomposed into two steps. In the first step we compute a permutation matrix P and the sparsity pattern of the factor L. The latter is referred to as the *symbolic factorization* and requires only the sparsity pattern of the KKT matrix. In the second step, referred to as the *numerical factorization* step, we determine the values of nonzero elements in L and D. Note that we do not need to update the symbolic factorization if the nonzero entries of the KKT matrix change but the sparsity pattern remain the same.

Indirect method. We can find the solution of (5.22) by solving instead the following linear system

$$\left(P + \sigma I + \rho A^T A\right) \tilde{x}^{k+1} = \sigma x^k - q + A^T(\rho z^k - y^k)$$

obtained by eliminating ν^{k+1} from (5.22). We then compute $\tilde{z}^{k+1}$ as $\tilde{z}^{k+1} = A\tilde{x}^{k+1}$. Note that the coefficient matrix in the above linear system is always positive definite. The linear system can therefore be solved with an iterative schemes such as the conjugate gradient method [88, 137]. When the linear system is solved up to some predefined accuracy, we terminate the method. We can also warm-start the method using the linear system solution at the previous iteration of ADMM to speed-up its convergence.

5.3.2 Final algorithm

By simplifying the ADMM iterations according to the previous discussion, we obtain Algorithm 5. Steps 4, 5, 6 and 7 of Algorithm 5 are very cheap to evaluate since they involve only vector addition and subtraction, scalar-vector multiplication and projection onto a box. Moreover, they are component-wise separable and can be easily parallelized. The most computationally expensive part is solving the linear system in Step 3, which can be done using direct or indirect methods as discussed in Section 5.3.1.

Algorithm 5

1: **given** initial values x^0, z^0, y^0 and parameters $\rho > 0, \sigma > 0, \alpha \in (0, 2)$
2: **repeat**
3: $(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow \begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho} y^k \end{bmatrix}$
4: $\tilde{z}^{k+1} \leftarrow z^k + \frac{1}{\rho}(\nu^{k+1} - y^k)$
5: $x^{k+1} \leftarrow \alpha \tilde{x}^{k+1} + (1 - \alpha) x^k$
6: $z^{k+1} \leftarrow \Pi\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k + \frac{1}{\rho} y^k\right)$
7: $y^{k+1} \leftarrow y^k + \rho\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k - z^{k+1}\right)$
8: **until** termination condition is satisfied

5.3.3 Convergence and infeasibility detection

We show in this section that proposed algorithm generates a sequence of tuples (x^k, z^k, y^k) that in the limit satisfy the optimality conditions (5.4)–(5.6) if the problem (5.1) is feasible, or provide primal or dual infeasibility certificates otherwise.

It is a well known fact that ADMM for solving problems of the form (5.12) is equivalent to Douglas-Rachford splitting applied to a specific problem reformulation [63, 86].

In particular, as shown in [86] Algorithm 5 is equivalent to

$$(\tilde{x}^k, \tilde{z}^k) \leftarrow \underset{(\tilde{x},\tilde{z}):A\tilde{x}=\tilde{z}}{\operatorname{argmin}} \tfrac{1}{2}\tilde{x}^T P\tilde{x} + q^T\tilde{x} + \tfrac{\sigma}{2}\|\tilde{x} - x^k\|_2^2 + \tfrac{\rho}{2}\|\tilde{z} - \left(2\Pi(v^k) - v^k\right)\|_2^2 \tag{5.23}$$

$$x^{k+1} \leftarrow x^k + \alpha\left(\tilde{x}^k - x^k\right) \tag{5.24}$$

$$v^{k+1} \leftarrow v^k + \alpha\left(\tilde{z}^k - \Pi(v^k)\right) \tag{5.25}$$

where

$$z^k = \Pi(v^k) \quad \text{and} \quad y^k = \rho\left(v^k - \Pi(v^k)\right). \tag{5.26}$$

Observe that iterates z^k, y^k satisfy the optimality condition (5.6) by construction [14, Prop. 6.46].

We now show that the primal and dual residuals defined in (5.7) and (5.8) converge to zero if the original problem is solvable. The solution of the optimization problem in (5.23) satisfies the following optimality conditions that are equivalent to (5.19)–(5.21),

$$(P + \sigma I)\tilde{x}^k + q - \sigma x^k + \rho A^T\left(\tilde{z}^k - 2z^k + v^k\right) = 0 \tag{5.27}$$

$$A\tilde{x}^k = \tilde{z}^k. \tag{5.28}$$

From a general convergence theory of Douglas-Rachford splitting (see, *e.g.*, [14, Cor. 27.4]) it follows that if problem (5.1) is solvable, then as $k \to \infty$

$$x^k - \tilde{x}^k \to 0 \quad \text{and} \quad z^k - \tilde{z}^k \to 0. \tag{5.29}$$

It follows from the above conditions and equality (5.28) that

$$Ax^k - z^k = \underbrace{A\tilde{x}^k - \tilde{z}^k}_{=0} + A\underbrace{(x^k - \tilde{x}^k)}_{\to 0} - \underbrace{(z^k - \tilde{z}^k)}_{\to 0},$$

which means that the primal residual converges to zero, *i.e.*,

$$r^k_{\text{prim}} = Ax^k - z^k \to 0. \tag{5.30}$$

Similarly, we have

$$\begin{aligned} &Px^k + q + A^T y^k \\ &= (P + \sigma I)\tilde{x}^k + q - \sigma x^k + \rho A^T\left(\tilde{z}^k - 2z^k + v^k\right) \\ &\quad + (P + \sigma I)(x^k - \tilde{x}^k) + \rho A^T(z^k - \tilde{z}^k), \end{aligned}$$

which taken together with (5.27) and (5.29) implies that the dual residual converges to zero, *i.e.*,

$$r^k_{\text{dual}} = Px^k + q + A^T y^k \to 0. \tag{5.31}$$

Quadratic programs infeasibility. If problem (5.1) is primal and/or dual infeasible, then the sequence of iterates (x^k, z^k, y^k) generated by Algorithm 5 will not necessarily converge. However, in the case $\mathcal{C} = [l, u]$, the sequence

$$(\delta x^k, \delta z^k, \delta y^k) \coloneqq (x^k - x^{k-1}, z^k - z^{k-1}, y^k - y^{k-1})$$

will always converge [11], where the δ notation is used to indicate the difference between successive iterates. If the problem is primal infeasible, then $\delta y \coloneqq \lim_{k\to\infty} \delta y^k$ will satisfy conditions (5.10), whereas $\delta x \coloneqq \lim_{k\to\infty} \delta x^k$ will satisfy conditions (5.11) if it is dual infeasible. For more details we refer the reader to [11].

5.3.4 Termination criteria

We can define reasonable termination criteria for Algorithm 5 so that the iterations will stop when either a primal-dual solution or a certificate of primal or dual infeasibility is found up to some tolerance.

For feasible problems, a reasonable termination criterion for detecting optimality is that the norms of the residuals r^k_{prim} and r^k_{dual} are smaller than some tolerance levels $\epsilon_{\text{prim}} > 0$ and $\epsilon_{\text{dual}} > 0$ [32], *i.e.*,

$$\|r^k_{\text{prim}}\|_\infty \leq \epsilon_{\text{prim}} \quad \text{and} \quad \|r^k_{\text{dual}}\|_\infty \leq \epsilon_{\text{dual}}. \tag{5.32}$$

We set the tolerance levels as

$$\begin{aligned}
\epsilon_{\text{prim}} &\coloneqq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|Ax^k\|_\infty, \|z^k\|_\infty\} \\
\epsilon_{\text{dual}} &\coloneqq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|Px^k\|_\infty, \|A^T y^k\|_\infty, \|q\|_\infty\}.
\end{aligned}$$

Quadratic programs infeasibility. If $\mathcal{C} = [l, u]$, we check the following conditions for primal infeasibility

$$\|A^T \delta y^k\|_\infty \leq \epsilon_{\text{pinf}} \|\delta y^k\|_\infty,$$
$$u^T (\delta y^k)_+ + l^T (\delta y^k)_- \leq -\epsilon_{\text{pinf}} \|\delta y^k\|_\infty,$$

where $\epsilon_{\text{pinf}} > 0$ is some tolerance level. Similarly, we define the following criterion for detecting dual infeasibility

$$\|P\delta x^k\|_\infty \leq \epsilon_{\text{dinf}} \|\delta x^k\|_\infty, \quad q^T \delta x^k \leq -\epsilon_{\text{dinf}} \|\delta x^k\|_\infty,$$
$$(A\delta x^k)_i \begin{cases} \in [-\epsilon_{\text{dinf}}, \epsilon_{\text{dinf}}] \, \|\delta x^k\|_\infty & u_i, l_i \in \mathbf{R} \\ \geq \epsilon_{\text{dinf}} \|\delta x^k\|_\infty & u_i = +\infty \\ \leq -\epsilon_{\text{dinf}} \|\delta x^k\|_\infty & l_i = -\infty, \end{cases}$$

for $i = 1, \ldots, m$ where $\epsilon_{\text{dinf}} > 0$ is some tolerance level. Note that $\|\delta x^k\|_\infty$ and $\|\delta y^k\|_\infty$ appear in the right-hand sides to avoid any division while considering normalized vectors δx^k and δy^k in the termination criteria.

5.4 Problem data scaling

A known weakness of first-order methods is their inability to deal effectively with ill-conditioned problems, and their convergence rate can vary significantly when data are badly scaled.

Preconditioning is a common heuristic which aims to reduce the number of iterations in first-order methods [137, Chap 5],[82, 19, 141, 84, 85]. The optimal choice of preconditioners has been studied for at least two decades and remains an active area of research [109, Chap 2],[90, Chap 10]. For example, the optimal diagonal preconditioner required to minimize the condition number of a matrix can be found exactly by solving a semidefinite program [29]. However, this computation is typically *more* complicated than solving the original QP, and is therefore unlikely to be worth the effort since preconditioning is only a heuristic to minimize the number of iterations.

In order to keep the preconditioning procedure simple, we instead make use of a simple heuristic called *matrix equilibration* [34, 164, 69, 52].

Our goal is to rescale the problem data to reduce the condition number of the symmetric matrix $M \in \mathbf{S}^{n+m}$ defined as

$$M := \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix}. \tag{5.33}$$

In particular, we use *symmetric matrix equilibration* by computing the diagonal matrix $S \in \mathbf{S}^{n+m}_{++}$ to decrease the condition number of SMS. We can write matrix S as

$$S = \begin{bmatrix} D & \\ & E \end{bmatrix}, \tag{5.34}$$

where $D \in \mathbf{S}^n_{++}$ and $E \in \mathbf{S}^m_{++}$ are both diagonal. In addition, we would like to normalize the cost function to prevent the dual variables from being too large. We can achieve this by multiplying the cost function by an appropriate scalar $c > 0$.

Preconditioning effectively modifies problem (5.1) into the following

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\bar{x}^T \bar{P}\bar{x} + \bar{q}^T \bar{x} \\ \text{subject to} & \bar{A}\bar{x} \in E\mathcal{C}, \end{array} \tag{5.35}$$

where $\bar{x} = D^{-1}x$, $\bar{P} = cDPD$, $\bar{q} = cDq$, $\bar{A} = EAD$ and $E\mathcal{C} := \{Ez \in \mathbf{R}^m \mid z \in \mathcal{C}\}$. The dual variables of the new problem are $\bar{y} = cE^{-1}y$. Note that when $\mathcal{C} = [l, u]$ the Euclidean projection onto $E\mathcal{C} = [El, Eu]$ is as easy to evaluate as the projection onto $\mathcal{C}$.

The main idea of the equilibration procedure is to scale the rows of matrix M so that they all have equal ℓ_p norm. It is possible to show that finding such a scaling matrix S can be cast as a convex optimization problem [9]. However, it is computationally more convenient to solve this problem with heuristic iterative methods, rather than continuous optimization algorithms such as interior-point methods. We refer the reader to [34] for more details on matrix equilibration.

Ruiz equilibration. In this work we apply a variation of the Ruiz equilibration [147]. This technique was originally proposed to equilibrate square

Algorithm 6 Modified Ruiz equilibration

initialize $c = 1$, $S = I$, $\delta = 0$, $\bar{P} = P, \bar{q} = q, \bar{A} = A, \bar{l} = l, \bar{u} = u$
while $\|1 - \delta\|_\infty > \epsilon_{\text{equil}}$ **do**
  for $i = 1, \ldots, n + m$ **do**
    $\delta_i \leftarrow 1/\sqrt{\|M_{:i}\|_\infty}$ ▷ M equilibration
  end for
  $\bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u} \leftarrow$ Scale $\bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u}$ using $\mathbf{diag}(\delta)$
  $\gamma \leftarrow 1/\max\{\text{mean}(\|\bar{P}_{:i}\|_\infty), \|\bar{q}\|_\infty\}$ ▷ Cost scaling
  $\bar{P} \leftarrow \gamma \bar{P}, \bar{q} \leftarrow \gamma \bar{q}$
  $S \leftarrow \mathbf{diag}(\delta) S, c \leftarrow \gamma c$
end while
return S, c

matrices, showing fast linear convergence superior to other methods such as the Sinkhorn-Knopp equilibration [154]. Ruiz equilibration converges in few tens of iterations even in cases when Sinkhorn-Knopp equilibration takes thousands of iterations [113]. The steps are outlined in Algorithm 6 and differ from the original Ruiz algorithm by adding a cost scaling step that takes into account very large values of the cost. The first part is the usual Ruiz equilibration step. Since M is symmetric, we focus only on the columns $M_{:i}$ and apply the scaling to both sides of M. At each iteration, we compute the ∞-norm of each column and normalize that column by the inverse of its square root. The second part is a cost scaling step. Scalar γ is the current cost normalization coefficient, taking into account the maximum between the average norm of the columns of $\bar{P}$ and $\bar{q}$. We normalize the problem data $\bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u}$ in place at each iteration using the current values of δ and γ.

Scaled termination criteria. Although we rescale our problem in the form (5.35), we would still like to apply the stopping criteria defined in Section 5.3.4 to an unscaled version of our problem. The primal and dual residuals in (5.32) can be rewritten in terms of the scaled problem as

$$r^k_{\text{prim}} = E^{-1}(\bar{A}\bar{x}^k - \bar{z}^k), \qquad r^k_{\text{dual}} = c^{-1}D^{-1}(\bar{P}\bar{x}^k + \bar{q} + \bar{A}^T\bar{y}^k),$$

and the tolerances levels as

$$\epsilon_{\text{prim}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|E^{-1}\bar{A}\bar{x}^k\|_\infty, \|E^{-1}\bar{z}^k\|_\infty\}$$
$$\epsilon_{\text{dual}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}} c^{-1} \max\{\|D^{-1}\bar{P}\bar{x}^k\|_\infty, \|D^{-1}\bar{A}^T\bar{y}^k\|_\infty, \|D^{-1}\bar{q}\|_\infty\}.$$

Quadratic programs infeasibility. When $\mathcal{C} = [l, u]$, the primal infeasibility conditions become

$$\left\|D^{-1}\bar{A}^T\delta\bar{y}^k\right\|_\infty \leq \epsilon_{\text{pinf}}\|E\delta\bar{y}^k\|_\infty,$$
$$\bar{u}^T(\delta\bar{y}^k)_+ + \bar{l}^T(\delta\bar{y}^k)_- \leq -\epsilon_{\text{pinf}}\|E\delta\bar{y}^k\|_\infty,$$

where the primal infeasibility certificate is $E\delta\bar{y}^k$. The dual infeasibility criteria are

$$\|D^{-1}\bar{P}\delta\bar{x}^k\|_\infty \leq c\epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty, \quad \bar{q}^T\delta\bar{x}^k \leq -c\epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty,$$

$$(E^{-1}\bar{A}\delta\bar{x}^k)_i \begin{cases} \in [-\epsilon_{\text{dinf}}, \epsilon_{\text{dinf}}]\,\|D\delta\bar{x}^k\|_\infty & u_i, l_i \in \mathbf{R} \\ \geq \epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty & u_i = +\infty \\ \leq -\epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty & l_i = -\infty, \end{cases}$$

where the dual infeasibility certificate is $D\delta\bar{x}^k$.

5.5 Solution polishing

Operator splitting methods are typically used for obtaining a solution of an optimization problem with a low or medium accuracy. However, even if a solution is not very accurate we can often guess from an approximate primal-dual solution which constraints are active at optimality. When dealing with QPs of the form (5.2), we can obtain high accuracy solutions from the final ADMM iterates by solving an additional system of equations.

Given a dual solution y of the problem, we define the sets of lower and upper-active constraints

$$\mathcal{L} \coloneqq \{i \in \{1, \ldots, m\} \mid y_i < 0\}, \tag{5.36}$$
$$\mathcal{U} \coloneqq \{i \in \{1, \ldots, m\} \mid y_i > 0\}. \tag{5.37}$$

According to (5.9) we have that $z_{\mathcal{L}} = l_{\mathcal{L}}$ and $z_{\mathcal{U}} = u_{\mathcal{U}}$, where $l_{\mathcal{L}}$ denotes the vector composed of elements of l corresponding to the indices in $\mathcal{L}$. Similarly, we denote by $A_{\mathcal{L}}$ the matrix composed of rows of A corresponding to the indices in $\mathcal{L}$.

If the sets of active constraints are known *a priori*, then a primal-dual solution (x, y, z) can be found by solving the following linear system

$$\begin{bmatrix} P & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & & \\ A_{\mathcal{U}} & & \end{bmatrix} \begin{bmatrix} x \\ y_{\mathcal{L}} \\ y_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \tag{5.38}$$
$$y_i = 0, \quad i \notin (\mathcal{L} \cup \mathcal{U}), \tag{5.39}$$
$$z = Ax. \tag{5.40}$$

We can then apply the aforementioned procedure to obtain a candidate solution (x, y, z). If (x, y, z) satisfies the optimality conditions (5.4)–(5.6), then our guess is correct and (x, y, z) is a primal-dual solution of problem (5.3). This approach is referred to as *solution polishing*.

However, the linear system (5.38) is not necessarily solvable even if the sets of active constraints $\mathcal{L}$ and $\mathcal{U}$ have been correctly identified. This can happen, *e.g.*, if the solution is degenerate, *i.e.*, if it has one or more redundant active constraints. We can make the solution polishing procedure more robust by solving instead the following linear system

$$\begin{bmatrix} P + \delta I & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & -\delta I & \\ A_{\mathcal{U}} & & -\delta I \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y}_{\mathcal{L}} \\ \hat{y}_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \tag{5.41}$$

where $\delta > 0$ is a regularization parameter with value $\delta \approx 10^{-6}$. Since the regularized matrix in (5.41) is quasi-definite, the linear system (5.41) is always solvable.

By using regularization, we actually solve a perturbed linear system and thus introduce a small error to the polished solution. If we denote by K and $(K + \Delta K)$ the coefficient matrices in (5.38) and (5.41), respectively, then we can represent the two linear systems as $Kt = g$ and $(K + \Delta K)\hat{t} = g$. To compensate for this error, we apply an *iterative refinement* procedure [61], *i.e.*, we iteratively solve

$$(K + \Delta K)\Delta\hat{t}^k = g - K\hat{t}^k \tag{5.42}$$

and update $\hat{t}^{k+1} := \hat{t}^k + \Delta\hat{t}^k$. The sequence $\{\hat{t}^k\}$ converges to the true solution t, provided that it exists. Observe that, compared to solving the linear system (5.41), iterative refinement requires only a backward and a forward-solve, and does not require another matrix factorization.

5.6 Parametric programs

In application domains such as control, statistics, finance, problem (5.1) is solved repeatedly for varying data. For these problems, usually referred to as *parametric programs*, we can speed up the repeated OSQP calls by re-using certain computations across multiple solves.

We distinguish between the case in which only the vectors or both vector and matrices change between subsequent instances of (5.1). We assume that the problem dimensions n and m and the sparsity patterns of P and A are fixed.

Vectors as parameters. If the vectors q, l, and u are the only parameters that vary, then the KKT coefficient matrix in Algorithm 5 does not change across different instances of the parametric program. If a direct method is used, we therefore perform and store its factorization only once before the first solution and reuse it across all subsequent iterations. Since the matrix factorization is computationally the most expensive step of the algorithm, this approach reduces significantly the amount of time OSQP takes to solve subsequent problems. This class of problems arises very frequently in many applications, including linear MPC and MHE [143, 3], lasso [168, 38], and portfolio optimization [31, 28, 122].

Matrices and vectors as parameters. We separately consider the case in which the values (but not the locations) of the nonzero entries of matrices P and A are updated. In this case, in a direct method, we need to refactor the matrix in Algorithm 5. However, since the sparsity pattern does not change we need only to recompute the *numerical factorization* while reusing the *symbolic factorization* from the previous solution as explained in Section 5.3.1. This allows for a modest reduction in the computation time. This class of problems encompasses applications such as nonlinear MPC and MHE [53] and SQP [137].

Warm-starting. In contrast to interior-point methods, OSQP is easily initialized by providing an initial guess of both the primal and dual solutions to the QP. This approach is known as *warm-starting* and is particularly effective when the subsequent QP solutions do not vary significantly, which is the case for most *parametric programs* applications. We can warm-start the ADMM iterates from the previous OSQP solution $(x^\star, y^\star)$ by setting $(x^0, z^0, y^0) \leftarrow (x^\star, Ax^\star, y^\star)$.

5.7 OSQP

We have implemented our proposed approach in the "Operator Splitting Quadratic Program" (OSQP) solver, an open-source software package in the C language. OSQP can solve any QP of the form (5.2) and makes no assumptions about the problem data other than convexity. OSQP is available online at

`http://osqp.readthedocs.io`.

Users can call OSQP from C, C++, Python, MATLAB and Julia, and also via parsers such as CVXPY [51, 2], YALMIP [120] and JuMP [62].

To exploit the data sparsity pattern, OSQP accepts matrices in Compressed-Sparse-Column (CSC) format [49]. We implemented the direct linear system solution described in Section 5.3.1 using the SuiteSparse package [4, 48] via a sparse permuted LDL^T decomposition.

```
import osqp

# Create an OSQP object
m = osqp.OSQP()

# Solver initialization
m.setup(P, q, A, l, u, settings)

# Generate code
m.codegen('code', project_type='Makefile',
                  parameters='vectors')
```

Listing 5.1: A simple Python script for generating the code for a given parametric QP.

The default values for the OSQP termination tolerances described in Section 5.3.3 are

$$\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-3}, \quad \epsilon_{\text{pinf}} = \epsilon_{\text{dinf}} = 10^{-4}.$$

The default step-size parameters ρ, σ and the relaxation parameter α are set to

$$\rho = 0.1, \quad \sigma = 10^{-6}, \quad \alpha = 1.6.$$

OSQP reports timing information including the total computation time, the fraction spent on preprocessing operations such as scaling or matrix factorization, and the fraction spent on the ADMM iterations. If the solver is called multiple times reusing the same matrix factorization, it will report only the ADMM solve time as total computation time.

5.7.1 Code generation

The OSQP implementation features a complete code generation framework.

Listing 5.1 shows a simple Python script that generates code for a given problem family. To generate a solver, the end-user must provide the problem data and (optionally) configure the solver settings. The end-user also has some flexibility to customize the solver prior to code generation. For instance, if the setting `check_termination` is set to a positive integer N, then the solver will terminate when one of the termination

```
#include "osqp.h"
#include "workspace.h"

int main(int argc, char **argv) {

    // Solve problem
    osqp_solve(&workspace);

    return 0;
};
```

Listing 5.2: A simple C program that loads the problem data from header file `workspace.h` and solves the problem.

criteria is satisfied by checking every N iterations, or when the maximum number of iterations is reached, whichever happens first. Checking the termination criteria in is computationally expensive since it involves several matrix-vector multiplications, and may slow down the code execution considerably. If the user instead sets `check_termination` to 0, then the algorithm will run for the maximum number of iterations without checking the termination criteria. For the complete list of solver settings we refer the reader to [156].

To generate the code, the `codegen` method is called with the specified name of a directory where the generated code is stored. Using the keyword argument `project_type` the user can define the build environment, *e.g.*, Makefiles or several supported IDEs such as Eclipse, Apple Xcode or Microsoft Visual Studio. The keyword argument `parameters` allows the user to specify which of the data are parameters. The option `vectors` assumes that only vectors q, l and u in problem 5.1 are parameters, while the option `matrices` allows matrices P and A to be parameters as well.

Generated files. Figure 5.1 shows the tree structure of the generated code. Directories `<dir_name>/src/osqp` and `<dir_name>/include` contain the solver source code. The generated code is self-contained, has small footprint and does not perform dynamic memory allocation, and is thus suitable for embedded applications. `CMakeLists.txt` is a CMake configuration file that manages the compilation process in a compiler- and

```
<dir_name>
├── include
│   └── [*.h]
├── src
│   ├── osqp
│   │   └── [*.c]
│   └── example.c
└── CMakeLists.txt
```

Figure 5.1: The tree structure of the generated code. The main program is stored in `example.c`.

```
// Update linear cost
osqp_update_lin_cost(&workspace, &q_new);

// Update lower bound
osqp_update_lower_bound(&workspace, &l_new);

// Update upper bound
osqp_update_upper_bound(&workspace, &u_new);
```

Listing 5.3: Function calls in C for updating vectors of a parametric QP.

platform-independent manner [111].

The main program is stored in `example.c` whose content is shown in Listing 5.2. The program loads the problem data from the header file `workspace.h`, and solves the problem. For solving a different instance of parametric problems, the problem data must be updated. Listing 5.3 provides illustrative function calls for updating vectors in problem 5.1; for the complete documentation we refer the reader to [156].

5.8 Numerical examples

5.8.1 Desktop

We benchmarked the desktop version of the OSQP solver against the open-source interior-point solver ECOS [56], the open-source active-set solver qpOASES [65], and the commercial interior-point solvers GUROBI [92] and MOSEK [130].

We executed the OSQP solver with default settings and polishing disabled. Note that the solution returned by the other solvers is with high accuracy while OSQP returns a lower accuracy solution. Hence, runtime benchmarks are not completely fair since OSQP might take more time than interior-point methods if a high accuracy is required. On the other hand, we used the direct light single-threaded linear systems solver SuiteSparse package [4, 48] where other solvers such as GUROBI and MOSEK use more advanced multi-threaded linear system solvers.

We consider the returned primal-dual solution $(x^\star, y^\star)$ by each solver to be optimal if the following optimality conditions are satisfied with $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-3}$,

$$\begin{aligned}
&\|(Ax^\star - u)_+ + (Ax^\star - l)_-\|_\infty \leq \epsilon_{\text{prim}},\\
&\|Px^\star + q + A^T y^\star\|_\infty \leq \epsilon_{\text{dual}},\\
&\|\min((y^\star)_+, |u - Ax^\star|)\|_\infty \leq \epsilon_{\text{slack}},\\
&\|\min(-(y^\star)_-, |Ax^\star - l|)\|_\infty \leq \epsilon_{\text{slack}},
\end{aligned}$$

where ϵ_{prim} and ϵ_{dual} are defined in Section 5.3.4 and $\epsilon_{\text{slack}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}}\|Ax^\star\|_\infty$.

All the experiments were carried out on a system with 32 2.2 GHz cores and 512 GB of RAM, running Linux. The code for all the numerical examples here presented is available online at

`https://github.com/oxfordcontrol/osqp_benchmarks`.

5.8.2 Benchmark problems

We considered QPs in the form (5.2) from 6 problem classes ranging from standard random programs to applications in the areas of finance and machine learning. For each problem class, we generated 10 different instances for 20 different problem dimensions giving a total of 1200 problem instances. We describe generation for each class in Appendix C. All instances were either obtained from real data or from realistic non-trivial random data.

Throughout all the problem classes, n ranges between 10^1 and 10^4, m between 10^2 and 10^5, and N between 10^2 and 10^8.

Failure rates. Figure 5.2 describes the failure rates for all the solvers across the generated problems. Note that when OSQP fails, the returned solutions always satisfy the complementary slackness conditions and remain relevant for many practical applications where high accuracy is not needed.

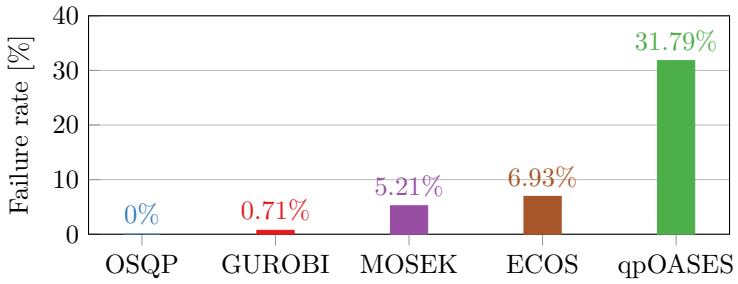


Figure 5.2: Failure rates

Computation times. We show in Figure 5.7 the computation time across all the problem classes for GUROBI and OSQP. Each problem class is

represented using a different symbol. OSQP manages to achieve similar timings as GUROBI across all the problem classes. In some cases such as equality constrained QPs, it also outperforms GUROBI.

Performance profiles. Figure 5.3 compares the performance profiles [55] of all the solvers tested. We define the performance ratio

$$r_{p,s} := \frac{t_{p,s}}{\min_s t_{p,s}},$$

where $t_{p,s}$ is the time it takes for solver s to solve problem instance p. If solver s fails at solving problem p, we set $r_{p,s} = \infty$. The performance profile plots the function $f_s : \mathbf{R} \to [0, 1]$ defined as

$$f_s(\tau) = \frac{1}{n_p} \sum_p \mathcal{I}_{\leq \tau}(r_{p,s}),$$

where $\mathcal{I}_{\leq \tau}(r_{p,s}) = 1$ if $r_{p,s} \leq \tau$ or 0 otherwise. The value $f_s(\tau)$ corresponds to the fraction of problems solved within τ times from the best solver. GUROBI outperforms all the solvers except for the very few cases when it fails outright. OSQP is the second best solver with no failures reported for these problem classes. ECOS and MOSEK manage to perform similarly even though ECOS runs single threaded. qpOASES does not manage to compute the optimal solutions of many instances since the number of active constraints combinations is too large for an active-set method.

5.8.3 Polishing

We executed the OSQP solver also with polishing enabled. Polishing succeeded in 44.33 % of the times providing a high-accuracy solution with a median of 1.07× computation time compared to the OSQP solution without it. When polishing succeeds, the solution is as accurate or even more than the one obtained with any other solver. Note that by decreasing the tolerances ϵ_{abs} and ϵ_{rel} we can increase the percentage of times polishing succeeds.

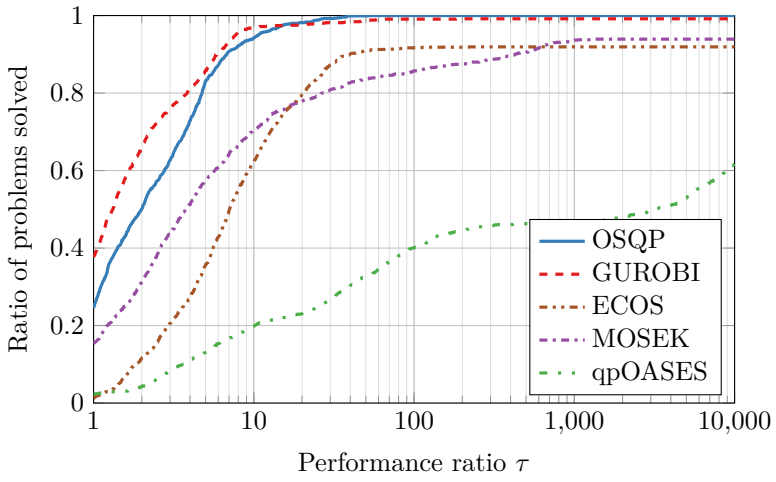


Figure 5.3: Performance profiles

5.8.4 Warm-start and factorization caching

To show the benefits of warm-starting and factorization caching, we solved a sequence of QPs with the data varying according to some parameters.

Lasso regularization path. We solved a lasso problem described in Appendix C.4 with varying λ in order to choose a regressor with good validation set performance.

We solved one problem instance with $n = 50$ features, $m = 5000$ data points, and λ logarithmically spaced taking 100 values between $\lambda_{\max} = \|A^T b\|_\infty$ and $0.01\lambda_{\max}$.

Since the parameters only enter linearly in the cost, we can reuse the matrix factorization and enable warm-starting to reduce the computation time as discussed in Section 5.6.

Portfolio back test. Consider the portfolio optimization problem in Appendix C.3 with $n = 3000$ assets and $k = 100$ factors.

We run a 4 years back test to compute the optimal assets investment depending on varying expected returns and factor models [28]. We solved 240 QPs per year giving a total of 960 QPs. Each month we solved 20 QPs corresponding to the trading days. Every day, we updated the expected returns μ by randomly generating another vector with $\mu_i \sim \mathcal{N}(0.9\hat{\mu}_i, 0.1)$, where $\hat{\mu}_i$ comes from the previous expected returns. The risk model was updated every month by updating the nonzero elements of D and F according to $D_{ii} \sim \mathcal{U}[0.9\hat{D}_{ii}, 0.1\sqrt{k}]$ and $F_{ij} \sim \mathcal{N}(0.9\hat{F}_{ij}, 0.1)$ where $\hat{D}_{ii}$ and $\hat{F}_{ij}$ come from the previous risk model.

As discussed in Section 5.6, we exploited the following computations during the QP updates to reduce the computation times. Since μ enters only linearly in the constraint bounds, we can reuse the matrix factorization and enable warm-starting. Since the sparsity pattern of of D and F does not change during the monthly updates, we can reuse the symbolic factorization and exploit warm-starting to speedup the computations.

Results. We show the results in Figure 5.4. For the lasso example, warm-starting and factorization caching bring an average reduction in computation time of 13.31× going from 404.7 ms to 30.4 ms. In the portfolio example, we obtain an average improvement of 8.44× from 277.9 ms to 32.9 ms.

5.8.5 Code generation

We benchmarked the generated solvers against the open-source code generation tools CVXGEN [124], FiOrdOs [170], the open-source solver qpOASES [65], and the commercial solver GUROBI [92]. All the solvers were selected with their default options. We performed benchmarks on a Macbook Pro 2.8GHz Intel Core i7 with 16GB RAM running Python 3.5. The code to reproduce the examples is available at

`https://github.com/oxfordcontrol/osqp_codegen_benchmarks`.

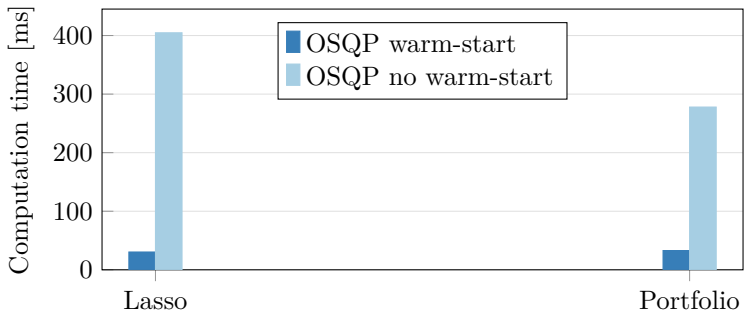


Figure 5.4: OSQP warm-start and factorization caching benchmarks.

Portfolio example Consider the portfolio optimization problem in C.3. In order to obtain Pareto optimal portfolios, one needs to solve the resulting QP for varying risk-aversion parameter γ. Since the parameter appears only in the linear cost, one does not need to perform any matrix factorization once the code is generated. Moreover, seeing that the optimal solution does not significantly differ with small changes in γ, we can make use of warm-starting and get a range of Pareto optimal portfolios with minimal computational effort.

We generate 11 values of γ equally spaced on a logarithmic scale between 10^{-2} and 10^{2}. For each solver and each dimension n we solve the generated problem for the 11 values of γ and average the execution time.

The results are shown in Figure 5.5. OSQP consistently outperforms all other methods tested. CVXGEN is not able to generate the problem when $n > 120$ since the resulting coefficient matrix has more than 4000 nonzero elements. Note that CVXGEN, FiOrdOs and qpOASES exploit the simple bounds on variable x, while OSQP and GUROBI use the formulation 5.1. Table 5.1 shows the sizes of executable files generated by the OSQP solver as a function of the number of assets. The size of the compiled code for all the tested examples does not exceed 0.55 MB, which includes problem data.

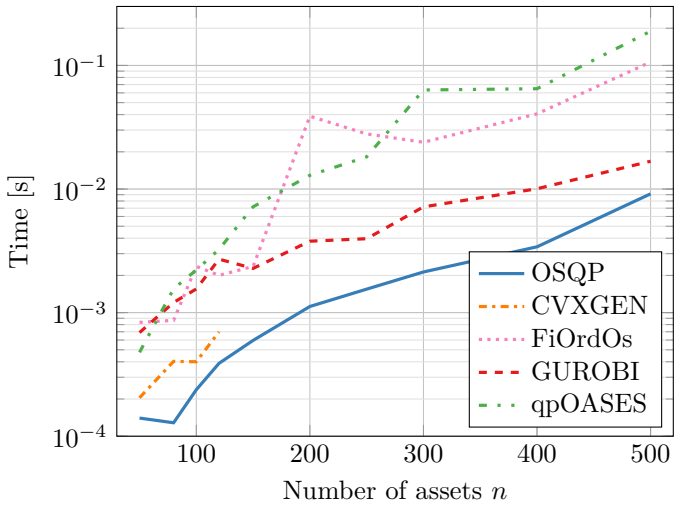


Figure 5.5: Average time to solve the portfolio code generation example.

Table 5.1: Sizes of executable file of OSQP solver for the portfolio code generation example.

Assets n	Dimension N	File size [kB]
50	235	46
80	496	58
100	720	70
120	984	82
150	1455	102
200	2440	142
250	3675	190
300	5160	246
400	8880	382
500	13600	554

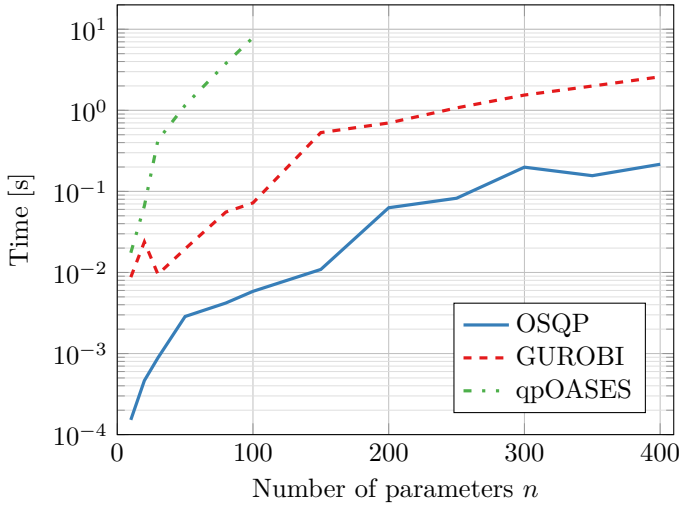


Figure 5.6: Average time to solve the lasso code generation example.

Lasso regularization path. Consider the lasso problem described in Appendix C.4 for varying weighting parameter λ as for the desktop OSQP examples in Section 5.8.4.

The results are shown in Figure 5.6. FiOrdOs does not converge within $50,000$ iterations for this problem type and CVXGEN is not able to generate code for $n > 10$. OSQP performs clearly better than GUROBI and qpOASES for all the parameters considered. Note that qpOASES is not able to find a solution in less than 10 seconds for $n > 100$.

5.9 Conclusions

In this Chapter, we presented a novel general-purpose QP solver based on ADMM called OSQP.

Our method is very robust and needs only a single quasi-definite matrix factorization before the first iteration. All the subsequent iterations

are much cheaper and division-free because they involve only forward-backward solves and vector-vector operations. OSQP is the first QP solver based on first-order methods able to reliably detect infeasible problems directly from the algorithm iterates. We showed that OSQP can outperform most state-of-the-art commercial and academic solvers in our publicly available benchmark set of 1200 problems from different application areas. Moreover, thanks to warm-starting and factorization caching our algorithm can obtain around $10\times$ speedups for parametric problems where only part of the data change.

We also implemented a code generation feature for OSQP. After the problem setup, our solver can generate tailored C code suitable for embedded applications. This code requires no dynamic memory allocation and it is library free. Since our algorithm is also division free, the generated solver can be safely deployed into safety-critical systems. Code generation benchmarks showed that the embedded version of our solver is able to outperform state-of-the-art code generation tools.

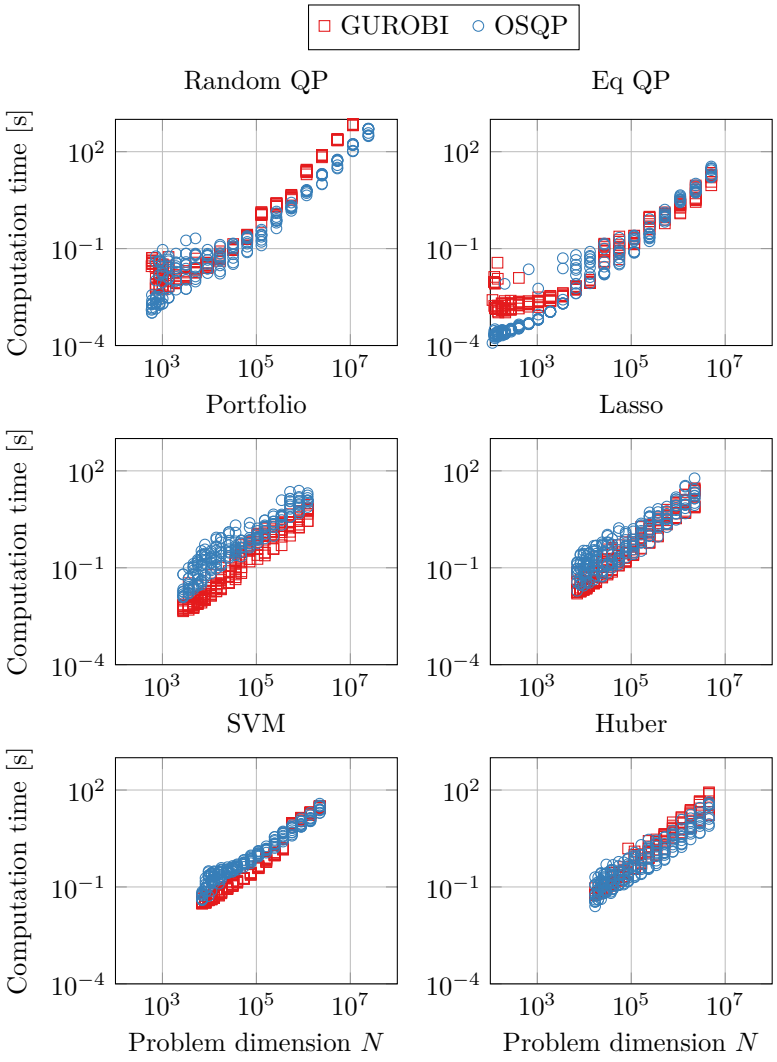


Figure 5.7: Computation time vs problem dimension for OSQP and GUROBI.

6

An MIQP Solver based on OSQP

6.1 Introduction

In this Chapter we develop a tailored branch-and-bound algorithm for mixed-integer quadratic programs (MIQPs) by extending the OSQP algorithm introduced in Chapter 5. Our algorithm exploits the particular structure of the alternating direction method of multipliers (ADMM) iterations in OSQP by greatly reducing the computations involved with a combination of offline precomputations and warm-starting.

6.1.1 The problem

We are interested in solving the following MIQP

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}x^T Px + q^T x \\ \text{subject to} \quad & l \leq Ax \leq u, \\ & x_i \in \mathbf{Z}, \quad \forall i \in \mathbf{I} \end{aligned} \tag{6.1}$$

with respect to the decision vector $x \in \mathbf{R}^n$. The objective function is defined by the symmetric positive semidefinite matrix $P \in \mathbf{S}_+^n$ and vector $q \in \mathbf{R}^n$, and the linear constraints by the matrix $A \in \mathbf{R}^{m \times n}$ and vectors $l \in (\mathbf{R} \cup \{-\infty\})^m$ and $u \in (\mathbf{R} \cup \{+\infty\})^m$. The set $\mathbf{I}$ denotes the elements of x constrained to take integer values, with $p \coloneqq |\mathbf{I}|$ their total number. We refer to the cost function of (6.1) as $f(x) \coloneqq \frac{1}{2}x^T Px + q^T x$.

Problems arising in many application domains can be expressed in the form (6.1), including portfolio optimization [25, 27, 175], hybrid vehicle control [131], regressor selection [22], hybrid model predictive control [17], geolocalization [166], and power systems [80, 157].

Problem (6.1) is $\mathcal{NP}$-hard in general since it includes mixed-integer linear programs (MILPs) as a special case [134]. Nevertheless, in the last two decades both hardware and software improvements have brought several orders of magnitude improvements in computation time [26, 135]. However, state-of-the art-solvers are still not well-suited for solving MIQPs on embedded platforms with low memory resources.

6.1.2 Solution methods

Exact solution. There are many methods for computing the optimal solution to (6.1) exactly [118, 15]. When the set of discrete variables is finite, the simplest approach is *exhaustive-search*, consisting of the enumeration of all possible integer combinations. The *branch-and-bound* algorithm instead searches for the optimal solution over a tree by repetitively partitioning the feasible region of integer variables into sub-domains. This technique was first introduced in the 1960s [117] to solve MILPs, and later

extended to mixed-integer nonlinear programs (MINLPs) [46]. *Branch-and-cut* [163] methods combine the benefits of branch-and-bound with *cutting plane* [89, 40] methods by iteratively introducing additional constraints to reduce the feasible region and thereby the number of nodes explored in the search tree. Other approaches such as *outer approximation* or *generalized Benders' decomposition* exploit the structure of the problem by alternating between the solution of a convex relaxation and of an MILP containing the feasible region. However, *branch-and-bound* is generally considered the most efficient algorithm available to solve problems of the form (6.1) [68], and is currently implemented in most commercial solvers [92].

Heuristics. Several heuristics have been proposed to compute suboptimal solutions to problem (6.1) when there is insufficient time or computing power to solve it to optimality. The *relax-and-round* heuristic solves a continuous relaxation of the MIQP and then rounds the fractional components to their closest integers. More advanced heuristics such as the *feasibility pump* [66] search for a solution by iteratively solving linear programs (LPs). Recently, an ADMM-based heuristic [165] has shown promising timing results relative to commercial solvers in computing good quality feasible solutions. A similar method has been proposed together with accelerated dual gradient projection [132]. The main downside of these heuristics is that they are not guaranteed to find a good solution, or even a feasible one.

6.1.3 Embedded systems

The main focus of this work is on embedded applications where the available time and computational power are both limited and the same problem is solved many times for varying parameters. In these cases the problem structure can be exploited to accelerate subsequent solutions.

A significant part of the research on embedded optimization to date has focused on tools for convex optimization problems. Examples include the solvers CVXGEN [124], ECOS [56] and FiOrdOs [170] and the com-

mercial solver FORCES Pro [57]. Significant advances were also obtained by applying first-order methods to solve optimal control problems on FPGAs [100, 146].

However, reliable numerical tools are still needed for solving MIQPs on embedded systems. Some progress has been made in developing MIQP solvers that are narrowly tailored to control applications such as hybrid model predictive control, including those based on interior-point optimization [71, 57], active set methods [7] and first-order methods [72]. In [16, 132], general branch-and-bound solvers based on nonnegative least squares and dual gradient projection were presented whose performance is competitive with commercial solvers in relatively small problems, but they require strict convexity of the objective function. Hybrid model predictive control (MPC) problems usually present a positive semidefinite matrix P because some of the auxiliary integer variables are not penalized. Thus, MIQP solvers requiring strict convexity of the objective function must add a regularization term to P which on the one hand lowers the solution accuracy and on the other slows down the inner quadratic program (QP) solvers.

6.1.4 Our approach

We propose a new robust branch-and-bound algorithm that exploits the particular structure of the OSQP [156] solver described in Chapter 5 to efficiently compute the solutions to MIQPs of the form (6.1).

OSQP is suited for MIQPs due to its efficiency and robustness. It is able to recognize primal and dual infeasible problems and does not require any assumption on the problem data such as specific structure, positive definiteness of the matrix P or linear independence of the constraints. The OSQP solver is also easily warm-started and thus efficiently employed within branch-and-bound schemes so that only a limited number of iterations are required to solve each subproblem.

The proposed MIQP algorithm extends OSQP by embedding it within a branch-and-bound method. In addition, it exploits the particular OSQP iterations to greatly reduce the numerical operations performed. The resulting method requires only a single quasi-definite matrix factorization

that is performed offline, cached and reused in all ADMM iterations of all QP subproblems solved during branch-and-bound. Moreover, the same factorization together with the current optimal solution can also be reused in subsequent optimizations arising in parametric optimization. The initial factorization combined with warm-starting allows us to compute the solutions very efficiently. Note that the same performance would not be achievable by adopting other QP solvers in our branch-and-bound routine because their inner iterations cannot be exploited in the same manner. Moreover, other QP solution algorithms such as interior point methods, cannot exploit warm-starting.

Our method is suitable for embedded systems since, following an initial matrix factorization that can be performed offline, it does not require dynamic memory allocation and is division-free.

We have prototyped the algorithm in Python interfacing to the fast OSQP solver binaries. Numerical results show that our approach is faster than commercial packages in solving small/medium-scale MIQPs arising in embedded optimization.

6.2 Branch-and-bound solver based on OSQP

The branch-and-bound algorithm computes the optimal solution $x^\star$ by exploring the integer combinations in a tree [30],[15, Sec 3.1]. The search is performed by repeatedly solving QPs of the form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T Px + q^T x \\ \text{subject to} & l \leq Ax \leq u, \\ & \underline{x}_i \leq x_i \leq \overline{x}_i, \quad \forall i \in \mathbf{I}. \end{array} \qquad (\mathrm{QP}(\underline{x}, \overline{x}))$$

Each $\mathrm{QP}(\underline{x}, \overline{x})$ is uniquely identified by the lower and upper bounds $(\underline{x}, \overline{x})$ imposed on the integer variables. An example branch-and-bound tree is shown in Figure 6.1.

The algorithm starts by solving the continuous relaxation of (6.1) at the root node, *i.e.*, $\mathrm{QP}(-\infty, \infty)$, obtaining a solution $\tilde{x}$ and a lower bound $f(\tilde{x})$. If $\mathrm{QP}(-\infty, \infty)$ is primal or dual infeasible, then MIQP (6.1) is also primal or dual infeasible, respectively. If the solution satisfies all of the

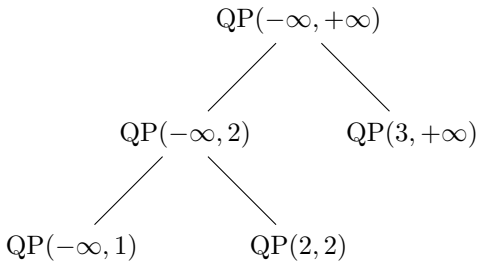


Figure 6.1: Example of branch-and-bound tree with one integer variable.

integer restrictions it is said to be *integer feasible* and is also a global solution of (6.1). Otherwise, the solution is said to be *fractional*. In that case, the algorithm searches over a tree whose nodes are subproblems of the form $\mathrm{QP}(\underline{x}, \overline{x})$ and whose edges are the branching decisions. It is typically unnecessary to search over all possible integer combinations, and the popularity of branch-and-bound is due in part to the fact that many subtrees can typically be pruned before exploration. We next briefly describe the branching and pruning process.

Given a problem $\mathrm{QP}(\underline{x}, \overline{x})$ with fractional solution, we pick a non-integer element $\tilde{x}_i,\ i \in \mathbf{I}$, and branch creating left $(-)$ and right $(+)$ child nodes with the same variable bounds $(\underline{x}, \overline{x})$ as the parent. Then we set

$$(\underline{x}_i^-, \overline{x}_i^-) = (\underline{x}_i, \lfloor \tilde{x}_i \rfloor) \qquad (\underline{x}_i^+, \overline{x}_i^+) = (\lceil \tilde{x}_i \rceil, \overline{x}_i). \tag{6.2}$$

The lower bounds of both children are initialized with the lower bound of their parent, so that lower bounds are monotonically non-decreasing with tree depth. In practice, we do not maintain the whole tree in memory but keep only the leaves in the heap $\mathcal{H}$.

Pruning rules allow us to discard tree branches based on the optimality and feasibility of the current node $\mathrm{QP}(\underline{x}, \overline{x})$. Let us denote the upper bound on the objective value as U and set it to ∞ at initialization. Pruning of branches occurs in three cases:

- If the current node is infeasible, then we prune the subtree since it contains only infeasible problems.

- If the optimal value $f(\tilde{x})$ of the current node is worse than the current upper bound U, *i.e.*, $f(\tilde{x}) > U$, then we prune the node since any integer solution in the subtree will not be better than the current best one.
- If the solution $\tilde{x}$ to the current node is integer feasible, then we can prune the entire subtree because it cannot contain better feasible solutions than $\tilde{x}$. Moreover, if it improves the current upper bound, that is $f(\tilde{x}) < U$, then we can update the optimal solution and the upper bound with $x^\star \leftarrow \tilde{x}$ and $U \leftarrow f(\tilde{x})$.

Since a good upper bound U allows the pruning of unnecessary branches, it is useful to find a good quality feasible solution as quickly as possible. In order to do so, at each iteration we select a vector $\hat{x}$ whose elements $\hat{x}_i$ $i \in \mathbf{I}$ are integer from the solution $\tilde{x}$ of the current node. If $\hat{x}$ satisfies the linear constraints $l \leq A\hat{x} \leq u$, then it is feasible for the original problem (6.1). If, in addition, $f(\hat{x})$ improves the current upper bound U, we can update the best known solution and the upper bound with $x^\star \leftarrow \hat{x}$ and $U \leftarrow f(\hat{x})$.

The complete algorithm description can be found in Algorithm 7. Note that if the QP solver used to solve the subproblems is able to generate dual-feasible solutions, one can stop solving the relaxation prematurely as soon as the corresponding dual cost is larger than the best known upper bound U [68, 16, 132]. In the current form, OSQP is not able to provide dual feasible solutions at each iteration, but only at convergence, *i.e.*, when the dual feasibility condition is satisfied. Hence, we cannot exploit premature pruning at the moment.

6.2.1 Strategic decisions

Algorithm 7 possesses three degrees of freedom that can substantially change its performance depending on the problem instance.

Tree exploration. The way we pick the next node $\mathrm{QP}(\underline{x}, \overline{x})$ to explore from the heap $\mathcal{H}$ determines how the tree exploration progresses. The two

Algorithm 7 MIQP branch-and-bound

initialize $U \leftarrow \infty$, $\mathcal{H} \leftarrow \mathrm{QP}(-\infty, \infty)$
while $\mathcal{H} \neq \emptyset$ **do**
  pick and remove $\mathrm{QP}(\underline{x}, \overline{x})$ from $\mathcal{H}$
  $\tilde{x}$, $f(\tilde{x}) \leftarrow$ solve $\mathrm{QP}(\underline{x}, \overline{x})$
  if $\mathrm{QP}(\underline{x}, \overline{x})$ is *infeasible* **then**
    prune current node
  else if $f(\tilde{x}) > U$ **then**
    prune current node
  else if $\tilde{x}$ is *integer feasible* **then**
    $U \leftarrow f(\tilde{x})$, $x^\star \leftarrow \tilde{x}$
    fathom nodes in $\mathcal{H}$ with lower bound $> U$
  else
    choose integer $\hat{x}$ from $\tilde{x}$
    if $\hat{x}$ is feasible and $f(\hat{x}) < U$ **then**
      $U \leftarrow f(\hat{x})$, $x^\star \leftarrow \hat{x}$
      fathom nodes in $\mathcal{H}$ with lower bound $> U$
    end if
  end if
  branch node $\mathrm{QP}(\underline{x}, \overline{x})$
end while

most common strategies are *best-bound* and *depth-first* [15]. *Best-bound* always chooses the node with the best lower bound, usually resulting in a small number of nodes explored. Its drawback is that large amounts of memory are required in general because, in the worst-case, the whole tree is searched before a feasible solution is found. In contrast, *depth-first* always picks the deepest node (or one of the deepest nodes) in a tree, with the advantage that the heap $\mathcal{H}$ is kept as small as possible. However, *depth-first* search typically visits more total nodes than the *best-bound* approach. In this work we use a hybrid approach where *depth-first* is carried out until a feasible solution is found. Then *best-bound* is used to minimize the number of visited nodes.

Branching variable selection. When branching we must choose amongst the candidate fractional elements of $\tilde{x}$ to determine bounds for the new nodes as in (6.2). The goal is to maximize the increase in the objective function lower bound with the branching so that it becomes easier to fathom nodes in the subtrees. In this work we use a *maximum fractional part branching* rule, *i.e.*, we select the variable with maximum integer violation [35, 68]. More sophisticated branching heuristics like *strong branching* or *pseudocost branching* [15] can be chosen to predict the increase in the value function in the child nodes, but it is not clear that they improve the practical performance [68, 91] and we do not employ them.

Compute an integer solution. Each time we compute a relaxed solution $\tilde{x}$, we also search for an integer feasible solution $\hat{x}$ with the help of a heuristic. The main idea is to quickly find good upper bounds allowing us to prune as many nodes as possible [15]. We use *nearest-neighbor rounding* by computing a simple rounding of the fractional elements of $\tilde{x}$. However, the rounded vector produced by this method may not be feasible, and more sophisticated heuristics can be applied to ensure that a feasible solution will be found; for example *MILP-based rounding* [133] or the *feasibility pump* [66]. We do not to use these heuristics because they require solving several LPs that might be more expensive than just progressing in the tree search, since the required matrix factorization would

be different than the on in Algorithm 5. Note that in the case of purely binary variables, tailored schemes such as *sum-up-rounding* could also be applied [151].

6.3 Exploiting the OSQP solver

In order to solve the subproblems in Algorithm 7, we require an efficient QP solver able to both compute optimal solutions reliably and to detect infeasibility. The OSQP solver presented in Chapter 5 not only satisfies all the requirements to be deployed in our branch-and-bound scheme, but it can be fully integrated into Algorithm 7 to minimize the operations involved using factorization caching and warm-starting.

Factorization caching. The OSQP iterations are summarized in Algorithm 5. The most expensive step is the linear system solution that, if the problem size is not very large, is performed using direct methods. Direct methods consist in two steps: a computationally expensive matrix factorization and easy to perform forward-backward solves. Since the matrix in Algorithm 5 does not change throughout the iterations, we can perform the factorization only once at the beginning of the algorithm to significantly reduce the total computations. This is already part of the OSQP implementation in Section 5.7.

We can exploit these operations even further in Algorithm 7. The linear system matrix does not change as it does not depend on the bounds $\underline{x}$ and $\overline{x}$ of the subproblems defining each node of the branch-and-bound tree. It must therefore be factorized only once at the root node. This factorization is then cached and used in all subsequent ADMM iterations. In addition, it does not change when we solve problem (6.1) for varying vectors q, l, u.

Warm-starting. To reduce the number of iterations required to solve each QP we introduce warm-starting. At each node we warm-start the

OSQP iterates using the parent node solution. Moreover, if we solve several similar MIQPs, we initialize Algorithm 7 with the solution of the previous MIQP if it is feasible.

Even though other algorithms such as active set methods can be warm-started and are currently used in most MIQP algorithms [68], they cannot exploit offline factorization as our method because they do not solve the same linear system at each iteration.

6.4 Numerical results

Our algorithm, named miOSQP, has been implemented in Python and interfaced to the OSQP solver compiled binaries. Timing benchmarks are compared to GUROBI [92] with the default options on a Macbook Pro 2.8GHz Intel Core i7 with 16GB RAM running Python 3.5. For a fair comparison both algorithms are executed single-threaded. Note that there is a variety of other efficient MINLP solvers available commercially and for free. However we compared our method to GUROBI since it is currently the standard reference in performance offering a specific solver for MIQPs. Other solvers such as SCIP [1] tackle more general MINLPs problems. Therefore, including them in the comparison would be unfair. The code together with the examples is available at

`https://github.com/oxfordcontrol/miosqp`.

6.4.1 Random MIQPs

We generated random MIQPs with varying dimensions n, m and number of integer variables p. The entries of P are computed as $P = MM^T$ where $M \in \mathbf{R}^{n \times n}$ is generated from the uniform distribution $\mathcal{U}(0, 1)$ with 70% nonzero elements and the linear part of the cost q with the normal distribution $\mathcal{N}(0, 1)$. The constraints are generated as $A \sim \mathcal{U}(0, 1)$, $l \sim \mathcal{U}(0, 1) - 2$ and $l \sim \mathcal{U}(0, 1) + 2$. Each problem instance is solved 10 times, from which we compute both the average and the maximum execution times together with the standard deviation. The results are shown in

Table 6.1: Timings in ms for random MIQPs with varying n, m and q

			miOSQP			GUROBI		
n	m	p	t_{avg}	t_{std}	t_{max}	t_{avg}	t_{std}	t_{max}
10	5	2	1.45	0.17	1.72	4.20	6.20	22.37
10	100	2	4.10	1.59	7.15	10.82	2.54	15.60
50	25	5	5.34	1.51	8.18	19.57	0.98	21.80
50	200	10	51.09	37.89	159.40	107.04	18.15	157.61
100	50	2	5.67	1.80	8.72	50.90	10.71	69.58
100	200	15	70.76	58.38	238.63	254.85	49.42	383.26
150	100	5	36.46	15.32	67.20	263.75	38.02	343.17
150	300	20	198.37	97.91	429.43	924.46	193.36	1316.03

Table 6.1. miOSQP outperforms GUROBI in all cases with up to 9x improvements.

6.4.2 Power converter control

We consider the hybrid system model of a three-level voltage source converter driving a medium-voltage induction machine described in Chapter 4 [157]. The system dynamics can be described as a discrete-time linear system with integer inputs (4.15). We would like to compute the optimal inputs so that the internal currents track the reference sinusoids. We can model this problem as in (4.16) which can be reformulated as an integer QP (4.21).

Since the computational cost grows exponentially with the horizon length, in Chapter 4 we approximated the infinite horizon cost using the approximate dynamic programming (ADP) approach to shorten the horizon length to $T = 1$ or 2. Numerical examples in Section 4.7 allows good control performance while keeping the number of input combinations manageable to enable *exhaustive search*. This technique becomes prohibitive for longer horizons because the number of input combinations grows exponentially with T. By using our proposed approach we show that the

computation time can be kept in the ms time-scale even with longer horizons while still benefiting from the good predictive behavior of ADP.

We performed closed-loop simulations for horizons $T = 1, 2, \ldots, 5$ for 3 periods each; one for reaching the steady state behavior and two for the actual simulation. Each period has 800 time steps. The computation times are averaged over all the solutions for each simulation. Both GUROBI and our algorithm were warm-started with the solution at the previous time step. The timings are shown in Figure 6.3. The current and input behaviors are shown in Figure 6.2.

The timing comparison shows a consistently better behavior of miOSQP compared to GUROBI: up to more than $2\times$ improvements for shorter horizons and still better performance for longer ones. Note that the problem has 27 integer feasible input combinations per stage which, for horizon 5 amounts to a total of $14,348,907$ worst case number of nodes to be evaluated. However, miOSQP always computes the optimal input after searching only a few hundred nodes. Note that without our offline factorization, the solver would require one factorization for each node and the solution time would greatly increase. Moreover, thanks to the warm-starting capabilities of OSQP only 80 iterations are required on average to solve each individual QP.

6.5 Conclusions

We proposed a new MIQP algorithm based on branch-and-bound combined to the OSQP solver. Thanks to factorization caching and warm-starting, our method is able to efficiently compute globally optimal solutions. Numerical examples showed that our method with a simple, high-level implementation shows better timings than commercial solvers for small to medium-size problems arising in embedded applications.

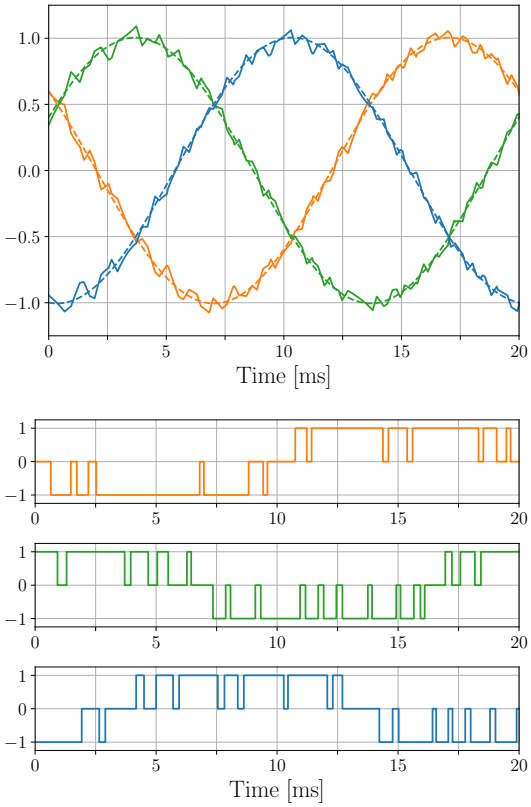


Figure 6.2: Power converter simulation results with miOSQP solver and $N = 3$

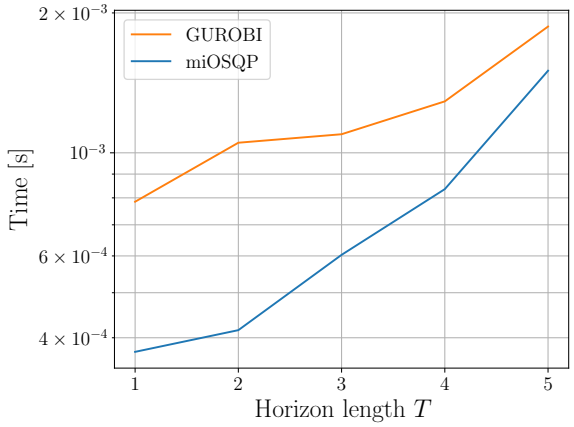


Figure 6.3: Average power converter MIQP solution time comparison between miOSQP and GUROBI

7

Discussion and Outlook

Many applications in process engineering, computer science, operations research, economics, chemistry and physics involve *mixed-integer optimal control problems*. Solving these problems is hard because of the explosion of feasible combinations introduced by the integer variables. Despite the recent tremendous hardware and algorithmic improvements in mathematical optimization, we are still not able to compute the optimal solutions of medium-sized mixed-integer optimization problems within the seconds time-scale. This limitation can become an issue when dealing with real-time applications such as model predictive control (MPC) where we have to compute the optimal solutions reliably within a certain amount of time. This thesis investigated the development of a variety of new algorithms for mixed-integer programming focusing on real-time optimal control of fast dynamical systems.

7.1 Approximations to reduce complexity

In the first part of this thesis we discussed approximations to make the integer optimization problems tractable. These techniques gave great reductions in complexity and computation time at the price of restricting the applicability to certain classes of problems.

7.1.1 Optimal switching times for switched dynamical systems

In Chapter 2 we described a reformulation that simplifies the decisions by removing integer variables. In particular, we focused on the switching time optimization problem: given a sequence of continuous dynamics, when is the optimal time to switch between them? In the recent literature, this problem has been tackled with iterative optimization methods. However, these approaches involve heavy computations consisting of several numerical integrations just to obtain the cost function and its derivatives. Thus, even though no integer variables are present, it becomes impractical to implement these techniques in real-time applications. To address this issue, we introduced an efficient method to solve these optimization problems. We developed easily computable expressions for the cost function, the gradient and the Hessian of the switching time optimization problem by sharing the most expensive computations between them. Moreover, in the case of linear switched systems we showed that most of the computations can be parallelized, thereby reducing the required computation time to just a few milliseconds. We implemented our technique in the Julia package SwitchTimeOpt able to outperform state-of-the-art approaches with up to two orders of magnitude computation time improvements.

Future work. There are several future directions to investigate. First of all, many computations can be parallelized. The state transition matrices in (2.22) can be computed in parallel for every different τ_i. Moreover, given the associative nature of the matrix products, parallel reduction techniques like the prefix-sum [116] could be implemented to reduce the computation time. In the case of linear dynamics, since there is no need to

sequentially propagate the state before the linearizations, the matrix exponentials could be computed completely in parallel. Note that the Julia language already includes several functions to parallelize computations on standard CPUs. We also believe that our approach could benefit greatly from implementations of these parallelizations on graphics processing unit (GPU) or field-programmable gate array (FPGA) architectures. Another research direction is to develop a tailored solver to our method to exploit its structure. For example, interior-point methods such as IPOPT exploit line search routines which could end up evaluating the cost function several times increasing the computation time due to the matrix exponentials computations at each different point. An optimization algorithm taking into account the most expensive computations in our subroutines could greatly improve the performance. Finally, the current work could be extended to more general problem formulations such as optimal control with state constraints.

7.1.2 Approximate dynamic programming for integer optimal control

In Chapters 3 and 4 we investigated a new method to approximate and solve very rapidly hybrid MPC for linear systems with integer inputs. We reduced the computational complexity by shrinking the long control horizon to very few decision steps and by approximating the tail cost offline. Using the framework of approximate dynamic programming (ADP), we computed a quadratic value function underestimator by solving a semidefinite program (SDP). In this way, the problem can be greatly simplified while maintaining good control performance. In addition, we introduced a new way to penalize the input effort usually referred to as the input switching frequency. State-of-the-art formulations penalize the input effort indirectly by reducing the input switchings over the controller horizon. When the horizon is very short, the number of input switchings does not provide a reliable estimate of the input effort. In this work we instead embedded a switching frequency estimator into the state dynamics and we penalized the deviations. With this technique the cost function tuning becomes much easier while the input effort can be better approximated and penalized.

We applied this method to a variable-speed drive system consisting of a voltage source inverted connected to a medium-voltage induction machine. The system was modeled as a linear system with switched three-phase input with equal switching steps for all phases.

We implemented our algorithm on a small size Xilinx Zynq FPGA (xc7z020) in fixed-point arithmetic. With processor-in-the-loop (PIL) tests, we showed that our approach can comfortably run at very high speeds satisfying the sampling time requirements of 25 µs. Moreover, with this method we could also outperform state-of-the-art approaches in terms of performance indices such as the total harmonic distortion (THD).

Future work. Our method can also be applied to more complex schemes such as modular multilevel converters (MMCs) [144]. While it is possible to derive a complete MMC model that could be used in an MPC approach, the number of switching levels per horizon stage increases exponentially with the number of converter levels. As stated in [81], the long horizon predictive power of MPC is expected to be even more beneficial with MMCs. We believe that our method, making use of short computing horizons and long predictions using an approximate value function could be applied effectively to MMCs with more levels. This is because it is still possible to evaluate the multilevel feasible switching combinations on commercially available FPGAs over very short horizons within the required sampling time.

From the system design, there are several symmetries in the model that could be exploited to increase the controller horizon without requiring more computational power. Regarding the frequency estimation, other filters with different orders could be implemented and their parameters chosen optimally by solving an optimization problem instead of performing manual tuning. Moreover, it would be interesting to benchmark other ADP tail cost basis functions (*e.g.*, higher order polynomials or B-splines) to understand which ones best approximate the infinite horizon tail cost and produce the best overall control performance.

7.2 Exact solution algorithms

In this part we directly tackled mixed-integer optimization problems by developing a new algorithm based on branch-and-bound. We first developed a new efficient and robust quadratic program (QP) solver and then embedded it into a branch-and-bound routine eliminating unnecessary computations.

7.2.1 The OSQP solver

In Chapter 5 we presented the OSQP solver, a new QP solver written in C and based on the alternating direction method of multipliers (ADMM). OSQP is very robust and requires no assumptions on problem data apart from convexity. The only expensive operation involved is a single quasi-definite matrix factorization that we can compute just once before the ADMM iterations. In this way all the iterations become cheap to compute and division free. OSQP can exploit warm-starting and factorization caching in parametric problems where only parts of the data change. In addition, OSQP is the first QP solver based on first-order methods capable of detecting infeasible problems. We benchmarked our open-source C implementation against state-of-the-art solvers on a data set of 1200 problems from different application areas. Our method was able to outperform most other solvers with just cold starting. We also showed that warm-starting and factorization caching can bring around $10\times$ speedups in the solution of parametric problems. Chapter 5 also shows a code generation feature we included in OSQP. After the setup phase is performed, we can generate tailored embeddable C code that is library-free, division free and requires no dynamic memory allocation. Thus, it is well suited for embedded applications on safety-critical systems. We benchmarked the code generated version of OSQP against many state-of-the-art embedded solvers showing its the better performance and reliability.

Future work. This solver is still actively developed and will be extended to several applications, benchmarks and additional interfaces.

Even though we showed the performance on various numerical benchmarks, there are many more test cases where it would be interesting to compare it to state-of-the-art solvers. Examples include other test libraries for QPs and linear programs (LPs) [126]. Another interesting benchmark is the infeasible problems detection speed of OSQP compared to other solvers implementing primal-dual interior point or active set methods.

We can extend the OSQP algorithm and implementation in several directions. First of all, we can exploit CPU parallelization. The most expensive part, which is the linear system solution, can be performed with different dedicated software. There are many efficient linear systems solvers freely available such as MUMPS [5] or MKL Pardiso [43]. They exploit multi-threading and advanced pivoting algorithms to speedup the solution. Linking OSQP to these linear systems solvers could bring huge computational gains for solving medium to large scale QPs. Second, different hardware could be exploited to parallelize the algorithm. GPU architectures are designed to massively parallelize linear algebra computations. When solving very large scale QPs with millions of variables, factorizing the linear system matrix becomes prohibitive. In these cases, indirect solution methods become more favorable since they involve only matrix-vector multiplications that can be greatly parallelized. By adopting indirect linear system solution methods such as the conjugate gradient method [88] on a GPU, we would be able to tackle very large scale applications that are inapproachable from any existing academic or industrial QP solver working just on CPUs.

Finally, even though problem data scaling works very well for our current OSQP implementation, there are cases of hard QPs that require a more accurate tuning of the solver step-size ρ. Data sensitivity is a common issue with first-order methods compared to interior-point methods. One approach to deal with this issue is to design acceleration schemes to speedup the convergence such as [86, 167]. Alternatively, we can design a data-driven automatic parameter selection. This can be achieved by solving numerous problems from different classes and fitting a function providing a good ρ value depending on data features such as properties of the matrices, distance of the constraints bounds, and so on.

7.2.2 An MIQP solver based on OSQP

In Chapter 6 we extended the OSQP solver to solve mixed-integer quadratic programs (MIQPs) by developing a tailored branch-and-bound algorithm. OSQP is particularly suited for MIQPs because of its efficiency and robustness. It is also able to detect infeasible problems which are very common in branch-and-bound routines. We obtained substantial computational gains from using OSQP due to factorization caching and warm-starting capabilities. By caching and reusing the matrix factorization at the root node of the search tree, we can save many unnecessary computations. Moreover, by warm-starting each child node with the optimal solution of its parent, we can reduce the number of ADMM iterations required. The proposed branch-and-bound algorithm can also be warm-started and the matrix factorization reused in subsequent optimizations arising in parametric optimization problems from, *e.g.*, hybrid MPC. We prototyped the algorithm in Python, interfacing it to the compiled OSQP binaries. Numerical benchmarks showed that our method with a simple high-level implementation is faster than commercial packages in solving small to medium-scale MIQPs.

Future work. The algorithm still has considerable scope for performance improvement. A parallelized branch-and-bound C implementation will greatly reduce the Python overhead. Moreover, further computation time reductions could be obtained by using premature node pruning. If the solver is able to provide dual-feasible solutions, we can stop solving the current relaxation before convergence as soon as the dual cost is larger than the best known upper bound [68, 16, 132]. We believe that, by obtaining a dual feasible solution from the OSQP iterates, we can apply premature node pruning to reduce the total number of ADMM iterations required. Additional methods such as advanced branching techniques together with cutting planes generation and heuristics can also improve the overall performance by reducing the number of visited nodes.

Notation

Sets

$\mathbf{R}$	Real numbers.
$\mathbf{R}^n$	Real n-vectors ($n \times 1$ matrices).
$\mathbf{R}^{m\times n}$	Real $m \times n$ matrices.
$\mathbf{R}_+$, $\mathbf{R}_{++}$	Nonnegative, positive real numbers.
$\mathbf{Z}$	Integers.
$\mathbf{Z}_+, \mathbf{Z}_{++}$	Nonnegative, positive integers.
$\mathbf{S}^n$	Symmetric $n \times n$ matrices.
$\mathbf{S}^n_+$, $\mathbf{S}^n_{++}$	Symmetric positive semidefinite, positive definite, $n \times n$ matrices.

Norms

$\|\cdot\|$	A norm.
$\|x\|_2$	Euclidean (ℓ_2-) norm of vector x.
$\|x\|_\infty$	Infinity norm of vector x.

Vectors and matrices

$\mathbf{1}$	Vector with all components one.
I	Identity matrix.
X^T	Transpose of matrix X.
$\mathbf{tr}\, X$	Trace of matrix X.
$\mathbf{diag}(x)$	Diagonal matrix with diagonal entries $x_1, \ldots, x_n$.
$\mathbf{blkdiag}(A, B, \ldots)$	Block diagonal matrix with $A, B, \ldots$ matrices as blocks.
$A \otimes B$	Kronecker product between matrices A and B.

Acronyms

ADMM	alternating direction method of multipliers.
ADP	approximate dynamic programming.
FIR	finite impulse response.
FPGA	field-programmable gate array.
GPU	graphics processing unit.
IIR	infinite impulse response.
ILS	integer least-squares.
LMI	linear matrix inequality.
LP	linear program.
LTI	linear time-invariant.
MHE	moving horizon estimation.
MILP	mixed-integer linear program.
MINLP	mixed-integer nonlinear program.
MIQP	mixed-integer quadratic program.
MMC	modular multilevel converter.
MPC	model predictive control.

PI	proportional-integral controller.
PIL	processor-in-the-loop.
pu	per-unit system.
QP	quadratic program.
SDP	semidefinite program.
SOCP	second-order cone program program.
SQP	sequential quadratic programming.
SVM	support vector machine.
THD	total harmonic distortion.
VHDL	VHSIC hardware description language.

Appendices

A

Switching Time Optimization Proofs

A.1 Proof of Theorem 2.1

A.1.1 Auxiliary lemmas

In order to prove Theorem 2.1, we first require the following two lemmas:

Lemma A.1 (State transition matrix derivative)**.** Given two switching times τ_a, τ_{i+1} with $\tau_a \leq \tau_{i+1}$ such that τ_{i+1} does not coincide with any point of the background grid and the switching interval δ_i, the first derivative of the state transition matrix between τ_a and τ_{i+1} with respect to δ_i can be written as

$$\frac{\partial \Phi(\tau_{i+1}, \tau_a)}{\partial \delta_i} = A_i^{n_i} \Phi(\tau_{i+1}, \tau_a). \tag{A.1}$$

Note that in the case when τ_{i+1} coincides with a fixed-grid point, the derivative is not defined since at $\tau_{i+1} + \epsilon$ a new linearization is introduced,

breaking the smoothness of the state transition matrix. Our derivations still hold in that case by considering, instead of the gradient, the subgradient equal to the one-sided limit of the derivative from below.

Proof. We can rewrite $\Phi(\tau_{i+1}, \tau_a)$ using Definition 2.1 as

$$\Phi(\tau_{i+1}, \tau_a) = e^{A_i^{n_i}\left(\delta_i - \sum_{p=0}^{n_i-1} \delta_i^p\right)} \left(\prod_{p=0}^{n_i-1} e^{A_i^p \delta_i^p} \right) \Phi(\tau_i, \tau_a), \tag{A.2}$$

by using the relation

$$\delta_i = \sum_{j=0}^{n_i} \delta_i^j. \tag{A.3}$$

Taking the derivative of (A.2) we obtain:

$$\begin{aligned} \frac{\partial \Phi(\tau_{i+1}, \tau_a)}{\partial \delta_i} &= A_i^{n_i} e^{A_i^{n_i}\left(\delta_i - \sum_{p=0}^{n_i-1} \delta_i^p\right)} \left(\prod_{p=0}^{n_i-1} e^{A_i^p \delta_i^p} \right) \Phi(\tau_i, \tau_a) \\ &= A_i^{n_i} \Phi(\tau_{i+1}, \tau_a), \end{aligned} \tag{A.4}$$

where we made use of the properties of the matrix exponential $e^{X(a+b)} = e^{Xa}e^{Xb}$ and $\frac{\partial}{\partial c} e^{Xc} = Xe^{Xc}$ with $X \in \mathbf{R}^{n_x}$ and $a, b, t \in \mathbf{R}$. ■

The matrices S_i and their first derivatives play an important role in the proof and in the rest of the Chapter. We next derive the first derivative of S_i:

Lemma A.2 (Derivative of Matrices S_i)**.** Given the switching times τ_a and τ_{i+1} so that $\tau_a \leq \tau_{i+1}$ and that τ_{i+1} does not coincide with any point of the background grid and the interval δ_i, the derivative of S_a with respect to δ_i is

$$\frac{\partial S_a}{\partial \delta_i} = \Phi(\tau_{i+1}, \tau_a)^T C_i \Phi(\tau_{i+1}, \tau_a). \tag{A.5}$$

Proof. From (2.12), we can write the derivative as

$$\frac{\partial S_a}{\partial \delta_i} = \frac{\partial P_a}{\partial \delta_i} + \frac{\partial F_a}{\partial \delta_i}. \tag{A.6}$$

Let us analyze the two components separately.

First part. We decompose the part defined by P_a as

$$\begin{aligned}
\frac{\partial P_a}{\partial \delta_i} &= \frac{\partial}{\partial \delta_i}\left(\int_{\tau_a}^{T_\delta} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&= \frac{\partial}{\partial \delta_i}\left(\int_{\tau_a}^{\tau_i} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&\qquad + \frac{\partial}{\partial \delta_i}\left(\int_{\tau_i}^{\tau_{i+1}} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&\qquad\qquad + \frac{\partial}{\partial \delta_i}\left(\int_{\tau_{i+1}}^{T_\delta} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&= \frac{\partial}{\partial \delta_i}\left(\int_{\tau_i}^{\tau_{i+1}} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&\qquad + \frac{\partial}{\partial \delta_i}\left(\int_{\tau_{i+1}}^{T_\delta} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right).
\end{aligned} \tag{A.7}$$

Note that the integral from τ_a to τ_i does not depend on δ_i and its derivative is zero. Taking first the leftmost term in (A.7), the integral from τ_i to τ_{i+1} can be written as

$$\begin{aligned}
&\frac{\partial}{\partial \delta_i}\left(\int_{\tau_a}^{\tau_{i+1}} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) \\
&= \Phi(\tau_i,\tau_a)^T \frac{\partial}{\partial \delta_i}\left(\int_{\tau_i}^{\tau_{i+1}} \Phi(t,\tau_i)^T Q \Phi(t,\tau_i)\mathrm{d}t\right)\Phi(\tau_i,\tau_a) \\
&= \Phi(\tau_i,\tau_a)^T \frac{\partial}{\partial \delta_i}\left(\int_{0}^{\delta i} \Phi(\eta+\tau_i,\tau_i)^T Q \Phi(\eta+\tau_i,\tau_i)\mathrm{d}\eta\right)\Phi(\tau_i,\tau_a) \\
&= \Phi(\tau_{i+1},\tau_a)^T Q \Phi(\tau_{i+1},\tau_a),
\end{aligned} \tag{A.8}$$

where in the second equality we applied the change of variables $\eta = t - \tau_i$ and in the third equality the fundamental theorem of calculus. Next taking the rightmost term in (A.7), the integral from τ_{i+1} to T_δ can be obtained as

$$\frac{\partial}{\partial \delta_i}\left(\int_{\tau_{i+1}}^{T_\delta} \Phi(t,\tau_a)^T Q \Phi(t,\tau_a)\mathrm{d}t\right) =$$

$$
\begin{aligned}
&= \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},\tau_a)^T\left(\int_{\tau_{i+1}}^{T_\delta}\Phi(t,\tau_{i+1})^T Q\Phi(t,\tau_{i+1})\mathrm{d}t\right)\Phi(\tau_{i+1},\tau_a)\right)\\
&= \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},\tau_a)^T P_{i+1}\Phi(\tau_{i+1},\tau_a)\right)\\
&= \frac{\partial}{\partial \delta_i}\Big(\Phi(\tau_{i+1},\tau_a)^T\Big)P_{i+1}\Phi(\tau_{i+1},\tau_a)\\
&\qquad\qquad + \Phi(\tau_{i+1},\tau_a)^T P_{i+1}\frac{\partial}{\partial \delta_i}\Big(\Phi(\tau_{i+1},\tau_a)\Big)\\
&= \Phi(\tau_{i+1},\tau_a)^T\left((A_i^{n_i})^T P_{i+1} + P_{i+1}A_i^{n_i}\right)\Phi(\tau_{i+1},\tau_a). \qquad \text{(A.9)}
\end{aligned}
$$

In the first and second equalities we decomposed the state transition matrices and used the definition of P_{i+1} of (2.10). In the third equality we applied the chain rule, noting that P_{i+1} is independent of δ_i. Then, in the last equality we applied Lemma A.1 to compute the derivatives. We now rewrite (A.7) using (A.8) and (A.9) obtaining

$$
\frac{\partial P_a}{\partial \delta_i} = \Phi(\tau_{i+1},\tau_a)^T\left(Q + (A_i^{n_i})^T P_{i+1} + P_{i+1}A_i^{n_i}\right)\Phi(\tau_{i+1},\tau_a). \qquad \text{(A.10)}
$$

Second part. We now focus on the derivative of F_a in (A.6) which can be written so that

$$
\begin{aligned}
\frac{\partial F_a}{\partial \delta_i} &= \frac{\partial}{\partial \delta_i}\left(\Phi(T_\delta,\tau_a)^T E\Phi(T_\delta,\tau_a)\right)\\
&= \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},\tau_a)^T\left(\Phi(T_\delta,\tau_{i+1})^T E\Phi(T_\delta,\tau_{i+1})\right)\Phi(\tau_{i+1},\tau_a)\right)\\
&= \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},\tau_a)^T F_{i+1}\Phi(\tau_{i+1},\tau_a)\right)\\
&= \frac{\partial}{\partial \delta_i}\Big(\Phi(\tau_{i+1},\tau_a)^T\Big)F_{i+1}\Phi(\tau_{i+1},\tau_a)\\
&\qquad\qquad + \Phi(\tau_{i+1},\tau_a)^T F_{i+1}\frac{\partial}{\partial \delta_i}\Big(\Phi(\tau_{i+1},\tau_a)\Big)\\
&= \Phi(\tau_{i+1},\tau_a)^T\left((A_i^{n_i})^T F_{i+1} + F_{i+1}A_i^{n_i}\right)\Phi(\tau_{i+1},\tau_a), \qquad \text{(A.11)}
\end{aligned}
$$

In the first and second equalities we decomposed the state transition matrices and used the definition of F_{i+1} from (2.11). In the third equality we applied the chain rule, noting that F_{i+1} is independent of δ_i. Then, in the last equality we applied Lemma A.1 to compute the derivatives.

By adding (A.10) and (A.11) as in (A.6) and applying Definition 2.2 we obtain

$$\frac{\partial S_a}{\partial \delta_i} = \Phi(\tau_{i+1}, \tau_a)^T \left(Q + (A_i^{n_i})^T S_{i+1} + S_{i+1} A_i^{n_i} \right) \Phi(\tau_{i+1}, \tau_a). \quad \text{(A.12)}$$

The result follows by using Definition 2.3. ■

A.1.2 Main result

We are now in a position to prove each of the statements in Theorem 2.1 in turn:

Cost function – proof of (i). The cost function in (2.16) can be directly derived from its definition in Problem ($\mathcal{P}_{\text{lin}}$) and Definition 2.2.

Gradient – proof of (ii). The gradient of the cost function can be derived by taking the derivative of (2.16). By considering the component related to δ_i, we can write

$$\begin{aligned} \frac{\partial J(\delta)}{\partial \delta_i} &= \frac{\partial}{\partial \delta_i} \left(x_0^T S_0 x_0 \right) \\ &= x_0^T \frac{\partial S_0}{\partial \delta_i} x_0 \\ &= x_0^T \Phi(\tau_{i+1}, 0)^T C_i \Phi(\tau_{i+1}, 0) x_0 \\ &= x_{i+1}^T C_i x_{i+1}. \end{aligned} \quad \text{(A.13)}$$

In the second equality the initial state has been taken out from the derivative operator since x_0 fixed. In the third equality we applied Lemma A.2 and in the fourth equality we used Definition 2.1 to obtain x_{i+1}. The result holds for $i = 0, \ldots, N$.

Hessian – proof of (iii). The Hessian of the cost function can be derived by taking the derivative of (A.13). Let us first take the derivative with respect to the same interval δ_i writing

$$\begin{aligned}
\frac{\partial^2 J(\delta)}{\partial \delta_i^2} &= \frac{\partial}{\partial \delta_i}\left(x_{i+1}^T C_i x_{i+1}\right) \\
&= \frac{\partial}{\partial \delta_i}\left(x_0^T \Phi(\tau_{i+1},0)^T C_i \Phi(\tau_{i+1},0) x_0\right) \\
&= x_0^T \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},0)^T\right) C_i \Phi(\tau_{i+1},0) x_0 \\
&\qquad + x_0^T \Phi(\tau_{i+1},0)^T C_i \frac{\partial}{\partial \delta_i}\left(\Phi(\tau_{i+1},0)\right) x_0 \\
&= x_0^T \Phi(\tau_{i+1},0)^T \left(A_i^{n_i}\right)^T C_i \Phi(\tau_{i+1},0) x_0 \\
&\qquad + x_0^T \Phi(\tau_{i+1},0)^T C_i A_i^{n_i} \Phi(\tau_{i+1},0) x_0 \\
&= x_{i+1}^T \left(A_i^{n_i}\right)^T C_i x_{i+1} + x_{i+1}^T C_i A_i^{n_i} x_{i+1} \\
&= 2 x_{i+1}^T C_i A_i^{n_i} x_{i+1} \\
&= 2 x_{i+1}^T C_i \Phi(\tau_{i+1},\tau_{i+1}) A_i^{n_i} x_{i+1}. \qquad \text{(A.14)}
\end{aligned}$$

In the third equality we took into account that C_i and x_0 do not depend on δ_i and we applied the chain rule. In the fourth equality we applied Lemma A.1 and in the fifth equality Definition 2.1. Finally, in the fifth and sixth equality we took the transpose of the first term which is a scalar and we used the identity $C_i A_i^{n_i} = C_i \Phi(\tau_{i+1},\tau_{i+1}) A_i^{n_i}$ to get the desired result.

In the case when we take the derivative with respect to δ_ℓ with $\ell > i$, we can write

$$\begin{aligned}
\frac{\partial^2 J(\delta)}{\partial \delta_\ell \partial \delta_i} &= \frac{\partial}{\partial \delta_\ell}\left(x_{i+1}^T C_i x_{i+1}\right) \\
&= \frac{\partial}{\partial \delta_\ell}\left(x_{i+1}^T \left(Q + A_i^{n_i T} S_{i+1} + S_{i+1} A_i^{n_i}\right) x_{i+1}\right) \\
&= x_{i+1}^T A_i^{n_i T} \frac{\partial}{\partial \delta_\ell}\left(S_{i+1}\right) x_{i+1} + x_{i+1}^T \frac{\partial}{\partial \delta_\ell}\left(S_{i+1}\right) A_i^{n_i} x_{i+1}
\end{aligned}$$

$$
\begin{aligned}
&= 2x_{i+1}^T \frac{\partial}{\partial \delta_\ell}\Big(S_{i+1}\Big) A_i^{n_i} x_{i+1} \\
&= 2x_{i+1}^T \Phi(\tau_{\ell+1}, \tau_{i+1})^T C_\ell \Phi(\tau_{\ell+1}, \tau_{i+1}) A_i^{n_i} x_{i+1} \\
&= 2x_{\ell+1}^T C_\ell \Phi(\tau_{\ell+1}, \tau_{i+1}) A_i^{n_i} x_{i+1} \qquad \text{(A.15)}
\end{aligned}
$$

in the second equality we applied Definition 2.3. In the third inequality we brought the terms not depending on δ_ℓ outside of the derivative operator. In the fourth equality we took the transpose of the first element which is a scalar. In the fifth equality we applied Lemma A.2 and finally we obtained $x_{\ell+1}$ from Definition 2.1. ■

A.2 Proof of Proposition 2.1

From Definition 2.2, we can obtain (2.23) by directly setting $i = N + 1$.

The recursion (2.24) can be derived by using (2.10) and (2.11) to rewrite (2.12) as follows:

$$
\begin{aligned}
S_i^j &= \int_{\tau_i^j}^{T_\delta} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t + \Phi(T_\delta, \tau_i^j)^T E \Phi(T_\delta, \tau_i^j) \\
&= \int_{\tau_i^j}^{\tau_i^{j+1}} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t \\
&\qquad + \int_{\tau_i^{j+1}}^{T_\delta} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t + \Phi(T_\delta, \tau_i^j)^T E \Phi(T_\delta, \tau_i^j) \\
&= \int_{\tau_i^j}^{\tau_i^{j+1}} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t + \\
&\qquad + \Phi(\tau_i^{j+1}, \tau_i^j)^T \Bigg(\int_{\tau_i^{j+1}}^{T_\delta} \Phi(t, \tau_i^{j+1})^T Q \Phi(t, \tau_i^{j+1}) \mathrm{d}t \\
&\qquad\qquad + \Phi(T_\delta, \tau_i^{j+1})^T E \Phi(T_\delta, \tau_i^{j+1}) \Bigg) \Phi(\tau_i^{j+1}, \tau_i^j) \\
&= \int_{\tau_i^j}^{\tau_i^{j+1}} \Phi(t, \tau_i^j)^T Q \Phi(t, \tau_i^j) \mathrm{d}t + \Phi(\tau_i^{j+1}, \tau_i^j)^T S_i^{j+1} \Phi(\tau_i^{j+1}, \tau_i^j)
\end{aligned}
$$

$$
\begin{aligned}
&= \int_0^{\delta_i^j} \Phi(\eta+\tau_i^j,\tau_i^j)^T Q \Phi(\eta+\tau_i^j,\tau_i^j)\mathrm{d}\eta \\
&\qquad\qquad + \Phi(\tau_i^{j+1},\tau_i^j)^T S_i^{j+1}\Phi(\tau_i^{j+1},\tau_i^j) \\
&= \int_0^{\delta_i^j} e^{A_i^{jT}\eta} Q e^{A_i^j\eta}\mathrm{d}\eta + e^{A_i^{jT}\delta_i^j} S_i^{j+1} e^{A_i^{jT}\delta_i^j} \\
&= M_i^j + \mathcal{E}_i^{jT} S_i^{j+1} \mathcal{E}_i^j.
\end{aligned}
$$

In the second equality we split the integral in two parts. In the third equality we bring the matrices $\Phi(\tau_i^{j+1},\tau_i^j)$ and $\Phi(\tau_i^{j+1},\tau_i^j)^T$ outside the integrals since they do not depend on t. In the fourth equality we apply (2.12) to obtain S_i^{j+1}. In the fifth equality we applied a change of variables $\eta = t - \tau_i^j$. In the last equality we rewrite the transition matrices using matrix exponentials.

B

Variable-Speed Drive Control

In this appendix we describe the detailed derivation of the drive system introduced in Section 4.2 and composed of an inverter and motor.

B.1 Reference frames

Variables in the three-phases system (abc) are denoted by $v_{abc} = (v_a, v_b, v_c)$. In certain cases it is common practice to express variables the stationary orthogonal $\alpha\beta$ coordinates as $v_{\alpha\beta} = (v_\alpha, v_\beta)$. To transform a vector v_{abc} in the stationary orthogonal $\alpha\beta$ coordinates, it is sufficient to compute

$$v_{\alpha\beta} = P v_{abc}.$$

The inverse operation can be performed as $v_{abc} = P^\dagger v_{\alpha\beta}$. The matrices P and $P^\dagger$ are the Clarke transform and its pseudoinverse respectively, *i.e.*,

$$P = \frac{2}{3}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix}, \quad P^\dagger = 3/2P^T.$$

B.2 Physical model of the inverter

The switch positions in the three phase legs can be described by the integer input variables $u_a, u_b, u_c \in \{-1, 0, 1\}$, leading to phase voltages $\{-V_{\rm dc}/2, 0, V_{\rm dc}/2\}$, respectively. Hence, the output voltage of the inverter is given by

$$v_{\alpha\beta} = (V_{\rm dc}/2)u_{\alpha\beta} = (V_{\rm dc}/2)Pu_{\rm sw}, \tag{B.1}$$

where $u_{\rm sw} = (u_a, u_b, u_c)$.

B.3 Physical model of the machine

Hereafter we derive the state-space model of the squirrel-cage induction machine in the $\alpha\beta$ plane. We express stator current i_s and the rotor flux ψ_r as state variables. The model input is the stator voltage v_s which is equal to the inverter output voltage in (B.1). The model parameters are: the stator and rotor resistances R_s and R_r; the mutual, stator and rotor reactances X_m, X_{ls} and X_{lr}, respectively; the inertia J; and the mechanical load torque T_l. Given these quantities, the continuous-time state equations [115, 97] are

$$\begin{aligned}
\frac{{\rm d}i_s(t)}{{\rm d}t} &= -\frac{1}{\tau_s}i_s(t) + \left(\frac{1}{\tau_r}I - \omega_r\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}\right)\frac{X_m}{D}\psi_r(t) + \frac{X_r}{D}v_s(t)\\
\frac{{\rm d}\psi_r(t)}{{\rm d}t} &= \frac{X_m}{\tau_r}i_s(t) - \frac{1}{\tau_r}\psi_r(t) + \omega_r\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}\psi_r(t)\\
\frac{{\rm d}\omega_r(t)}{{\rm d}t} &= \frac{1}{J}\left(T_e(t) - T_l(t)\right),
\end{aligned} \tag{B.2}$$

where $D \coloneqq X_sX_r - X_m^2$ with $X_s \coloneqq X_{ls} + X_m$ and $X_r \coloneqq X_{lr} + X_m$. Constants $\tau_s \coloneqq X_rD/\left(R_sX_r^2 + R_rX_m^2\right)$ and $\tau_r \coloneqq X_r/R_r$ are the transient stator and the rotor time constants respectively. The electromagnetic torque is given by

$$T(t) \coloneqq \frac{X_m}{X_r}\left(\psi_r(t) \times i_s(t)\right). \tag{B.3}$$

The rotor speed ω_r is assumed to be constant within the prediction horizon. For prediction horizons in the order of a few milliseconds this is a mild assumption.

B.4 Complete model of the physical system

Given the models of the drive and of the induction motor in (B.1) and (B.2) respectively, the state-space model in the continuous time domain can be described as

$$\begin{aligned} \dot{x}_{\mathrm{ph}}(t) &= Dx_{\mathrm{ph}}(t) + Eu_{\mathrm{sw}}(t), \\ y_{\mathrm{ph}}(t) &= Fx_{\mathrm{ph}}(t), \end{aligned} \tag{B.4}$$

where the state vector $x_{\mathrm{ph}} = (i_s, \psi_r) \in \mathbf{R}^4$ includes the stator current and rotor flux in the $\alpha\beta$ reference frame. The output vector is the stator current, *i.e.*, $y_{\mathrm{ph}} = i_s \in \mathbf{R}^2$. The matrices D, E and F are defined as

$$D = \begin{bmatrix} -\dfrac{1}{\tau_s} & 0 & \dfrac{X_m}{\tau_r D} & \omega_r \dfrac{X_m}{D} \\ 0 & -\dfrac{1}{\tau_s} & -\omega_r \dfrac{X_m}{D} & \dfrac{X_m}{\tau_r D} \\ \dfrac{X_m}{\tau_r} & 0 & -\dfrac{1}{\tau_r} & -\omega_r \\ 0 & \dfrac{X_m}{\tau_r} & \omega_r & -\dfrac{1}{\tau_r} \end{bmatrix}$$

$$E = \frac{X_r}{D}\frac{V_{\mathrm{dc}}}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} P, \qquad F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The state-space model of the drive can be converted into the discrete-time domain using exact Euler discretization. By integrating the first equation in (B.4) from $k\hat{T}_s$ to $(k+1)\hat{T}_s$ and keeping u_{sw} constant during each interval, we can obtain the discrete-time model in (4.1) with $A_{\mathrm{ph}} :=$

$e^{D\hat{T}_s}$, $B_{\text{ph}} \coloneqq -D^{-1}\left(I - A_{\text{ph}}\right)E$, $D_{\text{ph}} \coloneqq F$ and $k \in \mathbf{Z}_+$. Although the sampling time is $T_s = 25\,\mu\text{s}$, we use the discretization interval $\hat{T}_s = T_s\omega_b$ for consistency with our per unit system, where ω_b is the base frequency.

B.5 Value function underestimation

In this section we formulate the value function approximation semidefinite program (SDP) for the power system by adapting problem (3.10). The quadratic form $M_i(u)$ can be decomposed as follows.

For every feasible current u_{sw}, and previous $\bar{u}_{\text{sw}}$ semiconductor positions, *i.e.*,

$$(u_{\text{sw}}, \bar{u}_{\text{sw}}) \in \mathcal{M} \coloneqq \left\{(u_{\text{sw}}, \bar{u}_{\text{sw}}) \in \{-1, 0, 1\}^6 \mid \|u_{\text{sw}} - \bar{u}_{\text{sw}}\|_1 \leq 1\right\}, \quad \text{(B.5)}$$

we can define the augmented input

$$\tilde{u} = (u_{\text{sw}}, \|u_{\text{sw}} - \bar{u}_{\text{sw}}\|_1)$$

and the matrix $M_i(\tilde{u})$ using (3.9). We can then decompose the vector z in the quadratic form (3.8) using the augmented state definition (4.13).

$$(z, 1) = (z_{\text{ph}}, z_{\text{osc}}, z_{\text{sw}}, z_u, 1).$$

The matrix $M(\tilde{u})$ can also be decomposed in the same fashion into smaller block matrices as

$$\begin{bmatrix} z_{\text{ph}} \\ z_{\text{osc}} \\ z_{\text{sw},1:2} \\ z_{\text{sw},3} \\ z_u \\ 1 \end{bmatrix}^T \left[\begin{array}{c|ccc} M_{i,11} & M_{i,12} & M_{i,13} & M_{i,14} \\ \hline M_{i,12}^T & M_{i,22} & M_{i,23} & M_{i,24} \\ M_{i,13}^T & M_{i,23}^T & M_{i,33} & M_{i,34} \\ M_{i,14}^T & M_{i,24}^T & M_{i,34}^T & M_{i,44} \end{array}\right] \begin{bmatrix} z_{\text{ph}} \\ z_{\text{osc}} \\ z_{\text{sw},1:2} \\ z_{\text{sw},3} \\ z_u \\ 1 \end{bmatrix} \geq 0,$$

where the dependency on $\tilde{u}$ has been neglected to simplify notation. The first row and first column block matrices have the first and second dimensions respectively equal to the length of vector $(z_{\text{ph}}, z_{\text{osc}}, z_{\text{sw},1:2})$. Since

$z_u = \bar{u}_{\text{sw}}$ are the previous switch positions and $z_{\text{sw},3} = 1$ are the normalized desired switching frequency, we can simplify the quadratic form to

$$\begin{bmatrix} z_{\text{ph}} \\ z_{\text{osc}} \\ z_{\text{sw},1:2} \\ 1 \end{bmatrix}^T \left[\begin{array}{c|c} M_{i,11} & \Psi_{i,1} \\ \hline \Psi_{i,1}^T & \Psi_{i,2} \end{array}\right] \begin{bmatrix} z_{\text{ph}} \\ z_{\text{osc}} \\ z_{\text{sw},1:2} \\ 1 \end{bmatrix} \geq 0, \tag{B.6}$$

where

$$\begin{aligned} \Psi_{i,1} &= M_{i,13} z_u + M_{i,12} + M_{i,14} \\ \Psi_{i,2} &= z_u^T M_{i,33} z_u + 2M_{i,23} z_u + 2M_{i,34}^T z_u \\ &\quad + 2M_{i,24} + M_{i,22}, \end{aligned}$$

Therefore, we will denote the matrix in (B.6) as $\tilde{M}_i(\tilde{u})$ and the quadratic form vectors as $(\tilde{z}, 1)$. We can finally write the final SDP as

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(P_0\Sigma) + 2q_0^T\mu + r_0 \\ \text{subject to} & \tilde{M}_i(\tilde{u}) \succeq 0, \quad \forall \tilde{u} \in \mathcal{M}, \forall \tilde{z} \in \mathbf{R}^8, \quad i = 1, \ldots, M \\ & \hat{V}_0 = \hat{V}_M = \hat{V}, \quad P_i \in \mathbf{S}_+, \end{array} \tag{B.7}$$

which is both easier to solve than the original problem (3.10) and provides a tighter underapproximation because some of the elements of z are fixed to their only acceptable values.

B.6 Integer quadratic program reformulation

By considering the input sequence (4.20) and the state sequence over the horizon denoted as

$$x = (x(0), x(1), \ldots, x(N)),$$

the system dynamics (4.15) can be written as

$$x = \mathcal{A}x_0 + \mathcal{B}u, \tag{B.8}$$

where $\mathcal{A}$ and $\mathcal{B}$ are

$$\mathcal{A} \coloneqq \begin{bmatrix} I \\ A \\ \vdots \\ A^N \end{bmatrix}, \quad \mathcal{B} \coloneqq \begin{bmatrix} 0 & \cdots & 0 \\ B & & \vdots \\ AB & B & \\ \vdots & \ddots & \ddots & 0 \\ A^{N-1}B & \cdots & AB & B \end{bmatrix},$$

and the initial state $x(0) = x_0$.

Cost function. Let us separate the cost function in (4.16) in two parts: the cost from stage 0 to $N-1$ and the tail cost. The former can be rewritten as

$$\begin{aligned} &\sum_{k=0}^{N-1} \gamma^k \|Cx(k)\|_2^2 = x^T \mathcal{H} x \\ &= u^T \mathcal{B}^T \mathcal{H} \mathcal{B} u + 2 \left(\mathcal{B}^T \mathcal{H} \mathcal{A} x_0\right)^T u + \text{const}(x_0), \end{aligned} \tag{B.9}$$

where the last equality is obtained using (B.8) and the term $\text{const}(x_0)$ is a constant depending on the initial state. We define matrix $\mathcal{H}$ is as

$$\mathcal{H} = \mathbf{diag}(\gamma^0, \gamma^1, \ldots, \gamma^{N-1}, 0) \otimes C^T C. \tag{B.10}$$

In order derive the tail cost, let us write the last stage as

$$x(N) = A^N x_0 + \mathcal{B}_{\text{end}} u, \tag{B.11}$$

where $\mathcal{B}_{\text{end}}$ is the last row of $\mathcal{B}$ used to compute the last state. Using (B.11) and the tail cost formulation (3.11), the tail cost can be rewritten as

$$\begin{aligned} \hat{V}(x(N)) &= x(N)^T P_0 x(N) + 2 q_0^T x(N) + r_0 \\ &= u^T \left(\mathcal{B}_{\text{end}}^T P_0 B_{\text{end}}\right) u + 2 \left(\mathcal{B}_{\text{end}}^T P_0 A^N x_0 + \mathcal{B}_{\text{end}}^T q_0\right)^T u \\ &\quad + \text{const}(x_0). \end{aligned} \tag{B.12}$$

By combining (B.9) and (B.12) according to (4.16), we obtain the full cost function

$$J = u^T Q u + 2f\left(x_0\right)^T u + \text{const}(x_0),$$

with

$$\begin{aligned} Q &= \mathcal{B}^T \mathcal{H} \mathcal{B} + \gamma^N \mathcal{B}_{\text{end}}^T P_0 \mathcal{B}_{\text{end}} \\ f\left(x_0\right) &= \left(\mathcal{B}^T \mathcal{H} \mathcal{A} + \gamma^N \mathcal{B}_{\text{end}} P_0 \mathcal{A}^N\right) x_0 + \gamma^N \mathcal{B}_{\text{end}}^T q_0. \end{aligned} \tag{B.13}$$

Constraints. We now rewrite the constraints of problem (4.16) in vector form with $k = 0, \ldots, N-1$. Constraint (4.17) can be written as

$$\mathcal{F} u \leq \mathbf{1}. \tag{B.14}$$

Similarly, inequalities (4.18) can be written as

$$\mathcal{R} u \leq \mathcal{S} x \iff (\mathcal{R} - \mathcal{S}\mathcal{B}) u \leq \mathcal{S}\mathcal{A} x_0, \tag{B.15}$$

where in the term on the right we substituted (B.8). Finally constraint (4.19) enforcing the semiconductor devices positions to be integer and between -1 and 1 can be written as

$$\mathcal{G} u \leq \begin{bmatrix} \mathbf{1} \\ -\mathbf{1} \end{bmatrix}, \tag{B.16}$$

together with $u \in \mathbf{Z}^{6N}$. Matrices $\mathcal{F}$, $\mathcal{R}, \mathcal{S}$ and $\mathcal{G}$ are

$$\mathcal{R} = \begin{bmatrix} I \otimes (I - T) \\ I \otimes (-I - T) \end{bmatrix}, \qquad \mathcal{S} = \begin{bmatrix} I \otimes W & 0 \\ I \otimes -W & 0 \end{bmatrix},$$
$$\mathcal{F} = \begin{bmatrix} I \otimes T \\ I \otimes -T \end{bmatrix}, \qquad \mathcal{G} = \begin{bmatrix} I \otimes G \\ I \otimes -G \end{bmatrix}.$$

We can now combine (B.14), (B.15), and (B.16) into a single inequality $Fu \leq g(x_0)$ and rewrite problem (4.16) neglecting the constant terms in the cost function obtaining (4.21).

C

OSQP Benchmark Problem Classes

In this Appendix we collect the problem descriptions used for benchmarking the OSQP solver in Chapter 5.

C.1 Random QP

Consider the following quadratic program (QP)

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u. \end{array}$$

Problem instances. The number of variables and constraints in our problem instances are n and $m = 10n$. We generated random matrix $P = MM^T$ where $M \in \mathbf{R}^{n \times n}$ and 50% nonzero elements $M_{ij} \sim \mathcal{N}(0, 1)$. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0, 1)$ with only 50% being nonzero. The linear part of the cost is normally distributed, *i.e.*,

$q_i \sim \mathcal{N}(0,1)$. We generated the constraint bounds as $u_i \sim \mathcal{U}(0,1)$, $l_i \sim -\mathcal{U}(0,1)$.

C.2 Equality constrained QP

Consider the following equality constrained QP

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & Ax = b. \end{array}$$

This problem can be rewritten as (5.1) by setting $l = u = b$.

Problem instances. The number of variables and constraints in our problem instances are n and $m = \lfloor n/2 \rfloor$. We generated random matrix $P = MM^T$ where $M \in \mathbf{R}^{n \times n}$ and 50% nonzero elements $M_{ij} \sim \mathcal{N}(0,1)$. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0,1)$ with only 50% being nonzero. The vectors are all normally distributed, *i.e.*, $q_i, l_i, u_i \sim \mathcal{N}(0,1)$.

Iterative refinement interpretation. Solution of the above problem can be found directly by solving the following linear system

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}. \tag{C.1}$$

If we apply the ADMM iterations (5.13)–(5.17) for solving the above problem, and by setting $\alpha = 1$ and $y^0 = b$, the algorithm boils down to the following iteration

$$\begin{bmatrix} x^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \nu^k \end{bmatrix} + \begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix}^{-1} \left(\begin{bmatrix} -q \\ b \end{bmatrix} - \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^k \\ \nu^k \end{bmatrix} \right),$$

which is equivalent to (5.42) with $g = (-q, b)$ and $\hat{t}^k = (x^k, \nu^k)$. This means that Algorithm 5 applied to solve an equality constrained QP is equivalent to applying iterative refinement [61] to solve the KKT system (C.1).

C.3 Portfolio optimization

Portfolio optimization is a problem arising in finance that seeks to allocate assets in a way that maximizes the risk adjusted return [122, 31, 28], [33, §4.4.1],

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma(x^T \Sigma x) \\ \text{subject to} & \mathbf{1}^T x = 1 \\ & x \geq 0, \end{array}$$

where the variable $x \in \mathbf{R}^n$ represents the portfolio, $\mu \in \mathbf{R}^n$ the vector of expected returns, $\gamma > 0$ the risk aversion parameter, and $\Sigma \in \mathbf{S}^{n \times n}$ the risk model covariance matrix. The risk model is usually assumed to be the sum of a diagonal and a rank $k < n$ matrix

$$\Sigma = FF^T + D,$$

where $F \in \mathbf{R}^{n \times k}$ is the factor loading matrix and $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix describing the asset-specific risk.

We introduce a new variable $y = F^T x$ and solve the resulting problem in variables x and y

$$\begin{array}{ll} \text{minimize} & x^T D x + y^T y - \frac{1}{\gamma} \mu^T x \\ \text{subject to} & y = F^T x \\ & \mathbf{1}^T x = 1 \\ & x \geq 0, \end{array} \tag{C.2}$$

Note that the Hessian of the objective in (C.2) is a diagonal matrix. Also, observe that FF^T does not appear in problem (C.2).

Problem instances. We generated portfolio problems for increasing number of factors k and number of assets $100k$. The elements of matrix F were chosen as $F_{ij} \sim \mathcal{N}(0, 1)$ with 50% nonzero elements. The diagonal matrix D is chosen as $D_{ii} \in \mathcal{U}[0, \sqrt{k}]$. The mean return was generated as $\mu_i \in \mathcal{N}(0, 1)$. We set $\gamma = 1$.

C.4 Lasso

The *least absolute shrinkage and selection operator (lasso)* is a well known linear regression technique obtained by adding an ℓ_1 regularization term in the objective [168, 38]. It can be formulated as

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda\|x\|_1,$$

where $x \in \mathbf{R}^n$ is the vector of parameters and $A \in \mathbf{R}^{m\times n}$ is the data matrix and λ is the weighting parameter.

We convert this problem to the following QP

$$\begin{array}{ll} \text{minimize} & y^T y + \lambda \mathbf{1}^T t \\ \text{subject to} & y = Ax - b \\ & -t \leq x \leq t, \end{array}$$

where $y \in \mathbf{R}^m$ and $t \in \mathbf{R}^n$ are two newly introduced variables.

Problem instances. The elements of matrix A are generated as $A_{ij} \sim \mathcal{N}(0,1)$ with 50% nonzero elements. To construct the vector b, we generated the true sparse vector $v \in \mathbf{R}^n$ to be learned

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 0.5 \\ \mathcal{N}(0, 1/n) & \text{otherwise.} \end{cases}$$

Then, we let $b = Av + \varepsilon$ where ε is the noise generated as $\varepsilon_i \sim \mathcal{N}(0,1)$. We generated the instances with varying n features and $m = 100n$ data points. The parameter λ is chosen as $(1/5)\|A^T b\|_\infty$ since $\|A^T b\|_\infty$ is the critical value above which the solution of the problem is $x = 0$.

C.5 Huber fitting

Huber fitting or the *robust least-squares problem* performs linear regression under the assumption that there are outliers in the data [98, 99]. The

fitting problem is written as

$$\text{minimize} \quad \sum_{i=1}^{m} \phi_{\text{hub}}(a_i^T x - b_i), \tag{C.3}$$

with the Huber penalty function $\phi_{\text{hub}} : \mathbf{R} \to \mathbf{R}$ defined as

$$\phi_{\text{hub}}(u) = \begin{cases} u^2 & |u| \leq M \\ M(2|u| - M) & |u| > M. \end{cases}$$

Problem (C.3) is equivalent to the following QP [33, p.190]

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}u^T u + M\mathbf{1}^T v \\ \text{subject to} & -u - v \leq Ax - b \leq u + v \\ & 0 \leq u \leq M\mathbf{1} \\ & v \geq 0. \end{array}$$

Problem instances. We generate the elements of A as $A_{ij} \sim \mathcal{N}(0, 1)$. To construct $b \in \mathbf{R}^m$ we first generate a vector $v \in \mathbf{R}^n$ as $v_i \sim \mathcal{N}(0, 1/n)$ and a noise vector $\varepsilon \in \mathbf{R}^m$ with elements

$$\varepsilon_i \sim \begin{cases} \mathcal{N}(0, 1/4) & \text{with probability } p = 0.95 \\ \mathcal{U}[0, 10] & \text{otherwise.} \end{cases}$$

We then set $b = Av + \varepsilon$. For each instance we choose $m = 10n$ and $M = 1$, solve it 10 times and take the mean computation time.

C.6 Support vector machine

Support vector machine problem seeks an affine function that approximately classifies the two sets of points [44]. The problem can be stated as

$$\text{minimize} \quad x^T x + \lambda \sum_{i=1}^{m} \max(0, b_i a_i^T x + 1),$$

where $b_i \in \{-1, +1\}$ is a set label, and a_i is a vector of features for the i-th point. The problem can be equivalently represented as the following QP

$$\begin{array}{ll} \text{minimize} & x^T x + \lambda \mathbf{1}^T t \\ \text{subject to} & t \geq \mathbf{diag}(b) A x + \mathbf{1} \\ & t \geq 0, \end{array}$$

where $\mathbf{diag}(b)$ denotes the diagonal matrix with elements of b on its diagonal.

Problem instances. We choose the vector b so that

$$b_i = \begin{cases} +1 & i \leq m/2 \\ -1 & \text{otherwise,} \end{cases}$$

and the elements of A as

$$A_{ij} \sim \begin{cases} \mathcal{N}(+1/n, 1/n) & i \leq m/2 \\ \mathcal{N}(-1/n, 1/n) & \text{otherwise.} \end{cases}$$

References

[1] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[2] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision (To appear)*, January 2018.

[3] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright. *Nonlinear Predictive Control and Moving Horizon Estimation – An Introductory Overview*, pages 391–449. Springer London, London, 1999.

[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):381–388, 2004.

[5] P. R. Amestoy, I.S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[6] Avnet, Inc. *Zedboard Hardware User's Guide 2.2*, 2014.

[7] D. Axehill and A. Hansson. A mixed integer dual quadratic programming algorithm tailored for MPC. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 5693–5698, December 2006.

[8] H. Axelsson, M. Egerstedt, Y. Wardi, and G. Vachtsevanos. Algorithm for switching-time optimization in hybrid dynamical systems. In *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control*, pages 256–261, June 2005.

[9] H. Balakrishnan, I. Hwang, and C. J. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *IEEE Conference on Decision and Control (CDC)*, volume 5, pages 4874–4879 Vol.5, December 2004.

[10] G. Banjac and P. Goulart. Tight global linear convergence rate bounds for operator splitting methods. *www.optimization-online.org*, October 2016.

[11] G. Banjac, P. Goulart, B. Stellato, and S. Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. *SIAM Journal on Optimization (Submitted)*, June 2017.

[12] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd. Embedded code generation using the OSQP solver. In *IEEE Conference on Decision and Control (CDC) (To appear)*, 2017.

[13] H. H. Bauschke and J. M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.

[14] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 1st edition, 2011.

[15] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[16] A. Bemporad. Solving mixed-integer quadratic programs via nonnegative least squares. *IFAC-PapersOnLine*, 48(23):73–79, 2015. 5th IFAC Conference on Nonlinear Model Predictive Control NMPC.

[17] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[18] A Bemporad, M Morari, V Dua, and E N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[19] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418 – 477, 2002.

[20] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, MA, 3rd edition, 2005.

[21] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, MA, 3rd edition, 2005.

[22] D. Bertsimas, A. King, and R. Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, April 2016.

[23] D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, Belmont, MA, 2005.

[24] P. Beuchat, A. Georghiou, and J. Lygeros. Approximate dynamic programming: a $\mathcal{Q}$-function approach. *ArXiv e-prints*, February 2016, 1602.07273.

[25] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[26] R. E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2010.

[27] P. Bonami and M. A. Lejeune. An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, 57(3):650–670, 2009.

[28] S. Boyd, E. Busseti, S. Diamond, R. N. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.

[29] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, 1994.

[30] S. Boyd and J. Mattingley. Branch and bound methods. *Lecture notes*, 2010.

[31] S. Boyd, M. T. Mueller, B. O'Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment. *Foundations and Trends in Optimization*, 1(1):1–72, 2014.

[32] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[33] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[34] A. Bradley. *Algorithms for the equilibration of matrices and their application to limited-memory Quasi-Newton methods*. PhD thesis, Stanford University, 2010.

[35] R. Breu and C.-A. Burdet. *Branch and bound experiments in zero-one programming*, pages 1–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1974.

[36] R. H. Byrd, J. Nocedal, and R. A. Waltz. *KNITRO: An Integrated Package for Nonlinear Optimization*, pages 35–59. Springer US, Boston, MA, 2006.

[37] T. M. Caldwell and T. D. Murphey. Single integration optimization of linear time-varying switched systems. *IEEE Transactions on Automatic Control*, 57(6):1592–1597, May 2012.

[38] E. J. Candés, M. B. Wakin, and S. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, 2008.

[39] E. Chu, N. Parikh, A. Domahidi, and S. Boyd. Code generation for embedded second-order cone programming. In *European Control Conference (ECC)*, pages 1547–1552, 2013.

[40] V. Chvàtal, W. Cook, and M. Hartmann. On cutting-plane proofs in combinatorial optimization. *Linear Algebra and its Applications*, 114:455–499, 1989.

[41] M. Claeys, J. Daafouz, and D. Henrion. Modal occupation measures and LMI relaxations for nonlinear switched systems control. *Automatica*, 64:143 – 154, 2016.

[42] G. Cornuejols and R. Tütüncü. *Optimization Methods in Finance*. Mathematics, Finance and Risk. Cambridge University Press, 2006.

[43] Intel Corporation. Intel Math Kernel Library. User's Guide, 2017.

[44] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[45] P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez. Predictive control in power electronics and drives. *IEEE Transactions on Industrial Electronics*, 55(12):4312–4324, December 2008.

[46] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[47] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press Princeton, Princeton, N. J., 1963.

[48] T. A. Davis. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Transactions on Mathematical Software*, 31(4):587–591, 2005.

[49] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[50] D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, November 2003.

[51] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[52] S. Diamond and S. Boyd. Stochastic matrix-free equilibration. *Journal of Optimization Theory and Applications*, 172(2):436–454, February 2017.

[53] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*, chapter Lecture Notes in Control and Information Sciences, pages 391–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[54] X. C. Ding, Y. Wardi, and M. Egerstedt. On-line optimization of switched-mode dynamical systems. *IEEE Transactions on Automatic Control*, 54(9):2266–2271, August 2009.

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

[56] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

[57] A. Domahidi and J. Jerez. FORCES Professional. embotech GmbH (`http://embotech.com/FORCES-Pro`), July 2014.

[58] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

[59] D. Dueri, J. Zhang, and B. Açikmeşe. Automated custom code generation for embedded, real-time second order cone programming. In *IFAC World Congress*, volume 47, pages 1605–1612, 2014.

[60] W. C. Duesterhoeft, M. W. Schulz, and E. Clarke. Determination of instantaneous currents and voltages by means of alpha, beta, and zero components. *Transactions of the American Institute of Electrical Engineers*, 70(2):1248–1255, July 1951.

[61] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, London, 1989.

[62] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[63] J. Eckstein. *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, MIT, 1989.

[64] M. Egerstedt, Y. Wardi, and H. Axelsson. Transition-time optimization for switched-mode dynamical systems. *IEEE Transactions on Automatic Control*, 51(1):110–115, January 2006.

[65] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[66] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[67] K. Flaßkamp, T. Murphey, and S. Ober-Blöbaum. Discretized switching time optimization problems. In *2013 European Control Conference (ECC)*, pages 3179–3184, July 2013.

[68] R. Fletcher and S. Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal on Optimization*, 8(2):604–616, 1998.

[69] C. Fougner and S. Boyd. Parameter selection and pre-conditioning for a graph form solver. In R. Tempo, S. Yurkovich, and P. Misra, editors, *Emerging Applications of Control and System Theory (To appear)*. Springer, September 2017.

[70] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[71] D. Frick, A. Domahidi, and M. Morari. Embedded optimization for mixed logical dynamical systems. *Computers & Chemical Engineering*, 72:21–33, 2015.

[72] D. Frick, J. L. Jerez, A. Domahidi, A. Georghiou, and M. Morari. Low-complexity iterative method for hybrid MPC. *ArXiv e-prints*, 2016, 1609.02819.

[73] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen. High-performance small-scale solvers for linear model predictive control. In *2014 European Control Conference (ECC)*, pages 128–133, June 2014.

[74] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17 – 40, 1976.

[75] C. E. García, D. M. Prett, and M. Morari. Model predictive control: Theory and practice–A survey. *Automatica*, 25(3):335 – 348, 1989.

[76] M. Gerdts. A variable time transformation method for mixed-integer optimal control problems. *Optimal Control Applications and Methods*, 27(3):169–182, 2006.

[77] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Trans. Math. Softw.*, 29(1):58–81, March 2003.

[78] T. Geyer. *Low complexity model predictive control in power electronics and power systems*. Cuvillier Verlag, 2005.

[79] T. Geyer, G. Papafotiou, and M. Morari. Model predictive direct torque control – part I: Concept, algorithm, and analysis. *IEEE Transactions on Industrial Electronics*, 56(6):1894–1905, June 2009.

[80] T. Geyer and D. E. Quevedo. Multistep finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, 29(12):6836–6846, 2014.

[81] T. Geyer and D. E. Quevedo. Performance of multistep finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, 30(3):1633–1644, 2015.

[82] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson. Optimal parameter selection for the alternating direction method of multipliers (ADMM): Quadratic problems. *IEEE Transactions on Automatic Control*, 60(3):644–658, 2015.

[83] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On projected newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36(2):183–209, 1986.

[84] P. Giselsson and S. Boyd. Metric selection in fast dual forward–backward splitting. *Automatica*, 62:1–10, 2015.

[85] P. Giselsson and S. Boyd. Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. *IEEE Transactions on Automatic Control*, 62(2):532–544, February 2017.

[86] P. Giselsson, M. Fält, and S. Boyd. Line search for averaged operator iteration. *ArXiv e-prints*, March 2016, 1603.06772.

[87] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76, 1975.

[88] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4rd edition, 1996.

[89] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, September 1958.

[90] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997.

[91] O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[92] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016.

[93] B. Hassibi and H. Vikalo. On the sphere-decoding algorithm I. expected complexity. *IEEE Transactions on Signal Processing*, 53(8):2806–2818, July 2005.

[94] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Review*, 51(4):747–764, November 2009.

[95] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[96] C. Hoffmann, C. Kirches, A. Potschka, S. Sager, and L. Wirsching. *MUSCOD-II Users Manual*. IWR Universität Heidelberg, Germany, 2011.

[97] J. Holtz. The representation of AC machine dynamics by complex signal flow graphs. *IEEE Transactions on Industrial Electronics*, 42(3):263–271, June 1995.

[98] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[99] P. J. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[100] J. L. Jerez, P Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251, December 2014.

[101] E. R. Johnson and T. D. Murphey. Second-order switching time optimization for nonlinear time-varying dynamic systems. *IEEE Transactions on Automatic Control*, 56(8):1953–1957, July 2011.

[102] I. Kale, J. Gryka, G. D. Cain, and B. Beliczynski. FIR filter order reduction: balanced model truncation and hankel-norm optimal approximation. *IEEE Proceedings - Vision, Image and Signal Processing*, 141(3):168–174, June 1994.

[103] R. E. Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):51–60, 1964.

[104] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. English translation.

[105] P. Karamanakos, T. Geyer, and R. Kennel. Reformulation of the long-horizon direct model predictive control problem to reduce the computational effort. In *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 3512–3519, September 2014.

[106] P. Karamanakos, T. Geyer, and R. Kennel. Suboptimal search strategies with bounded computational complexity to solve long-horizon direct model predictive control problems. In *2015 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 334–341, September 2015.

[107] P. Karamanakos, T. Geyer, N. Oikonomou, F. D. Kieferndorf, and S. Manias. Direct model predictive control: A review of strategies that achieve long prediction intervals for power electronics. *IEEE Industrial Electronics Magazine*, 8(1):32–43, March 2014.

[108] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[109] C. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.

[110] B. Khusainov, E. C. Kerrigan, A. Suardi, and G. A. Constantinides. Nonlinear predictive control on a heterogeneous computing platform. In *IFAC World Congress 2017*, July 2017.

[111] Kitware, Inc. CMake, 2012.

[112] V. Klee and G. Minty. How good is the simplex algorithm. Technical report, Department of Mathematics, University of Washington, 1970.

[113] P. A. Knight, D. Ruiz, and B. Uçar. A symmetry preserving algorithm for matrix scaling. *SIAM Journal on Matrix Analysis and Applications*, 35(3):931–955, 2014.

[114] S. Kouro, P. Cortes, R. Vargas, U. Ammann, and J. Rodriguez. Model predictive control - a simple and powerful method to control power converters. *IEEE Transactions on Industrial Electronics*, 56(6):1826–1838, June 2009.

[115] P. Krause, O. Wasynczuk, S. D. Sudhoff, and S. Pekarek. *Analysis of Electric Machinery and Drive Systems*, volume 75. John Wiley & Sons, 3rd edition, 2013.

[116] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, October 1980.

[117] A. H. Land and A. G. Doig. *An Automatic Method for Solving Discrete Programming Problems*, pages 105–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[118] J. Lee and S. Leyffer. *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*. Springer New York, New York, NY, 2012.

[119] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[120] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[121] J. Malmborg and J. Eker. Hybrid control of a double tank system. In *Proceedings of the 1997 IEEE International Conference on Control Applications*, pages 133–138, October 1997.

[122] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[123] J. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 27(3):50–61, May 2010.

[124] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[125] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[126] H. Mittelmann. Benchmarks for optimization software. `http://plato.asu.edu/bench.html`. Accessed: 2017-11-23.

[127] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20:801–836, 1978.

[128] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[129] M. Morari and J. H. Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4):667 – 682, 1999.

[130] MOSEK ApS. *The MOSEK optimization toolbox manual. Version 7.1 (Revision 35)*, 2015.

[131] N. Murgovski, L. Johannesson, J. Sjöberg, and B. Egardt. Component sizing of a plug-in hybrid electric powertrain via convex optimization. *Mechatronics*, 22(1):106–120, 2012.

[132] V. V. Naik and A. Bemporad. Embedded mixed-integer quadratic optimization using accelerated dual gradient projection. In *20th IFAC World Congress*, Toulouse, France, 2017.

[133] G. Nannicini and P. Belotti. Rounding-based heuristics for nonconvex MINLPs. *Mathematical Programming Computation*, 4(1):1–31, 2012.

[134] G. Nemhauser and L. Wolsey. *Computational Complexity*, pages 114–145. John Wiley & Sons, Inc., 1988.

[135] G. L. Nemhauser. Integer programming: a global impact. In *EURO INFORMS, Rome, Italy*, 2013.

[136] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[137] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Berlin, 2006.

[138] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.

[139] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, November 2013.

[140] G. Papafotiou, J. Kley, K. G. Papadopoulos, P. Bohren, and M. Morari. Model predictive direct torque control – part II: Implementation and experimental evaluation. *IEEE Transactions on Industrial Electronics*, 56(6):1906–1915, June 2009.

[141] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *2011 International Conference on Computer Vision*, pages 1762–1769, November 2011.

[142] D. E. Quevedo, G. C. Goodwin, and J. A. De Doná. Finite constraint set receding horizon quadratic control. *International Journal of Robust and Nonlinear Control*, 14(4):355–377, 2004.

[143] J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, LLC, 2015.

[144] B. S. Riar, T. Geyer, and U. K. Madawala. Model predictive direct current control of modular multilevel converters: Modeling, analysis, and experimental evaluation. *IEEE Transactions on Power Electronics*, 30(1):431–439, January 2015.

[145] R. T. Rockafellar and R. J.-B Wets. *Variational analysis*. Grundlehren der mathematischen Wissenschaften. Springer, 1998.

[146] M. Rubagotti, P. Patrinos, A. Guiggiani, and A. Bemporad. Real-time model predictive control based on dual gradient projection: Theory and fixed-point FPGA implementation. *International Journal of Robust and Nonlinear Control*, 26(15):3292–3310, 2016.

[147] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Oxon, UL, 2001.

[148] S. Sager. On the integration of optimization approaches for mixed-integer nonlinear optimal control. Habilitationsschrift eingereicht bei der fakultät für mathematik und informatik, Universität Heidelberg, 2011.

[149] S. Sager. *A Benchmark Library of Mixed-Integer Optimal Control Problems*, pages 631–670. Springer New York, New York, NY, 2012.

[150] S. Sager, H. G. Bock, M. Diehl, G. Reinelt, and J. P. Schlöder. Numerical methods for optimal control with binary control functions applied to a lotka-volterra type fishing problem. In *Lecture Notes in Economics and Mathematical Systems*, pages 269–290, Berlin/Heidelberg, 2006. IWR Universität Heidelberg, Germany, Springer-Verlag.

[151] S. Sager, M. Jung, and C. Kirches. Combinatorial integral approximation. *Mathematical Methods of Operations Research*, 73(3):363, 2011.

[152] J. Scoltock, T. Geyer, and U. K. Madawala. A model predictive direct current control strategy with predictive references for MV grid-connected converters with *LCL*-filters. *IEEE Transactions on Power Electronics*, 30(10):5926–5937, October 2015.

[153] C. Seatzu, D. Corona, A. Giua, and A. Bemporad. Optimal control of continuous-time switched affine systems. *IEEE Transactions on Automatic Control*, 51(5):726–741, May 2006.

[154] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[155] G. Stathopoulos, H. Shukla, A. Szucs, Y. Pu, and C. N. Jones. Operator splitting methods in control. *Foundations and Trends in Systems and Control*, 3(3):249–362, 2016.

[156] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An Operator Splitting Solver for Quadratic Programs. *ArXiv e-prints*, November 2017, 1711.08013.

[157] B. Stellato, T. Geyer, and P. Goulart. High-speed finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, 32(5):4007–4020, May 2017.

[158] B. Stellato and P. Goulart. High-speed direct model predictive control for power electronics. In *European Control Conference (ECC)*, pages 129–134, July 2016.

[159] B. Stellato and P. Goulart. Real-time FPGA implementation of direct MPC for power electronics. In *IEEE Conference on Decision and Control (CDC)*, pages 1471–1476, December 2016.

[160] B. Stellato, V. Naik, A. Bemporad, P. Goulart, and S. Boyd. Embedded mixed-integer quadratic optimization using the OSQP solver. In *European Control Conference (ECC) (Submitted)*, 2018.

[161] B. Stellato, S. Ober-Blöbaum, and P. Goulart. Optimal control of switching times in switched linear systems. In *IEEE Conference on Decision and Control (CDC)*, pages 7228–7233, December 2016.

[162] B. Stellato, S. Ober-Blöbaum, and P. Goulart. Second-order switching time optimization for switched dynamical systems. *IEEE Transactions on Automatic Control*, 62(10):5407–5414, October 2017.

[163] R. A. Stubbs and S. Mehrotra. A branch-and-cut method for 0–1 mixed convex programming. *Mathematical Programming*, 86(3):515–532, 1999.

[164] R. Takapoui and H. Javadi. Preconditioning via diagonal scaling. *EE364b: Convex Optimization II Class Project*, 2014.

[165] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, pages 1–11, 2017.

[166] P. J. G. Teunissen. The least-squares ambiguity decorrelation adjustment: a method for fast GPS integer ambiguity estimation. *Journal of Geodesy*, 70(1–2):65–82, 1995.

[167] A. Themelis and P. Patrinos. SuperMann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators. *ArXiv e-prints*, September 2016, 1609.06955.

[168] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.

[169] P. Tøndel, T. A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489–497, 2003.

[170] F. Ullmann. FiOrdOs: A matlab toolbox for C-code generation for first order methods. Master's thesis, ETH Zürich, 2011.

[171] C. Van Loan. Computing integrals involving the matrix exponential. *IEEE Transactions on Automatic Control*, 23(3):395–404, 1978.

[172] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[173] R. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.

[174] R. Vasudevan, H. Gonzalez, R. Bajcsy, and S. S. Sastry. Consistent approximations for the optimal control of constrained switched systems—part 2: An implementable algorithm. *SIAM Journal on Control and Optimization*, 51(6):4484–4503, January 2013.

[175] J. P. Vielma, S. Ahmed, and G. L. Nemhauser. A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. *INFORMS Journal on Computing*, 20(3):438–450, 2008.

[176] V. Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Memoria della Reale Accademia Nazionale dei Lincei*, VI(2), 1926.

[177] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[178] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, March 2010.

[179] Y. Wang, B. O'Donoghue, and S. Boyd. Approximate dynamic programming via iterated bellman inequalities. *International Journal of Robust and Nonlinear Control*, 25(10):1472–1496, 2015.

[180] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959.

[181] S. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[182] Xilinx, Inc. *Vivado Design Suite User Guide - High-Level Synthesis*, 2014.

[183] Xilinx, Inc. *Zynq-7000 All Programmable SoC Technical Reference Manual*, 2016.

[184] W. Zhang, J. Hu, and A. Abate. On the value functions of the discrete-time switched LQR problem. *IEEE Transactions on Automatic Control*, 54(11):2669–2674, November 2009.

[185] F. Zhu and P. J. Antsaklis. Optimal control of hybrid switched systems: A brief survey. *Discrete Event Dynamic Systems*, 25(3):345–364, May 2014.