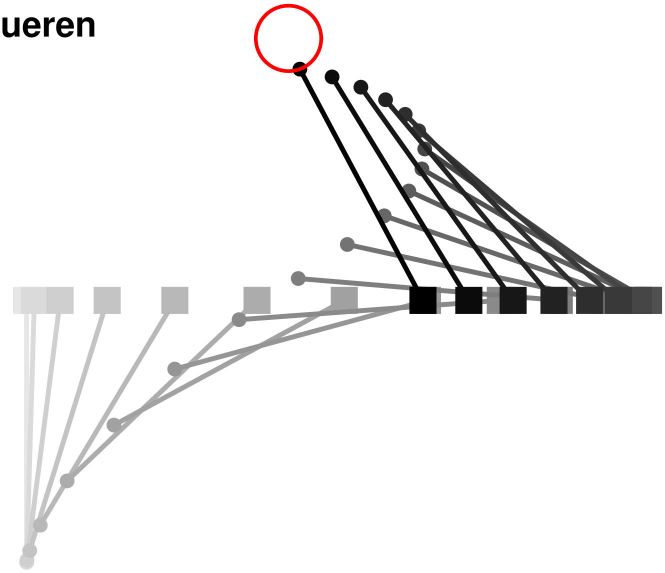
Albert-Ludwigs-Universität Freiburg
Technische Fakultät

Convex approximation methods for nonlinear model predictive control

Robin Verschueren



University of Freiburg
Faculty of Engineering
Department of Microsystems Engineering
Systems Control and Optimization Laboratory

December 2018

Convex approximation methods for nonlinear model predictive control

Robin VERSCHUEREN

Dissertation zur Erlangung des Doktorgrades der
Technischen Fakultät der Albert-Ludwigs-Universität
Freiburg im Breisgau

Dean: Prof. Dr. Oliver Paul

Examination committee:

Supervisor/First referee: Prof. Dr. Moritz Diehl
Second referee: Prof. Dr. Eric Kerrigan
Observer: Prof. Dr. Joschka Bödecker
Chair of committee: Prof. Dr. Lars Pastewka

December 2018

© 2018 Robin Verschueren
Self-published, Georges-Köhler-Allee 102, 79110 Freiburg im Breisgau (Germany)

Alle Rechte vorbehalten. Alle Inhalte dieses Werkes, insbesondere Texte, Fotografien und Grafiken, sind urheberrechtlich geschützt.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission.

Acknowledgments

A few days from now will be the fourth anniversary of my arrival in Freiburg. What a journey it has been! It feels only natural to thank some of my companions here. First of all, I would like to thank my supervisor and mentor Moritz. Thank you for the continuous stream of marvelous scientific ideas and for the guidance through the wondrous world of academia in the last five years, including the master's thesis. Furthermore, I would like to thank all my friends in our lab, being there previously and currently: Mario, Rien, Greg, Savannah, Attila, Joel, Joris, Janick, Slava, Kurt, Milan, Sébastien, Gianni, Dimitris, Adrian, Gianluca, Rachel, Andrea, Matilde, Niels, Jochem, Dang, Per, Jörg, Mischa, Thilo, Fabian, Jonas, Christoph, Tommaso, Yuning, Branimir, Armin, Tobi, Mara, Jonathan, Kathrin, Christine, Kerstin and Gaby. Please keep the office cool and weird! A big thank you also goes out to all the TEMPO fellows: Bartolomeo, Bulat, Deepak, Gianni, Goran, Harsh, Iris, Mohammad, Mohammed, Niels, Parisa, Rajesh, and Sarmad. I had a great time with you, wherever we were on the planet, so let's keep in touch.

Away from work, I was lucky enough to meet a horde of interesting people in Freiburg: a big thank you to the musicians from the ORSO, Aka, Kammerphil and KHG orchestras, the Lindy Hop and Salsa party-goers, the happy hikers group, the badmintoners, the SC-fans, the beer-gardeners and the board gamers, for filling my nights with laughter instead of sleep. Of the people I met here, I want to thank three people in particular: Elena, for being the best housemate I could ever dream of, and Nathi and Kade: I never knew one could be so close to one other, I'm looking forward to living with you (and the turtle). A big shout out as well to all my supporting friends from Belgium who made the long trip southwards to sunny Baden: Christophe, Stefanie, Ivana, Valerie, Hadrien, Naim, Frederik, Frederic, Wim, Tom, Laure, Annelies and Johannes – thanks for staying awesome. Last but not least, I want to thank my wonderful family: mum and dad, my silly brothers Hans and Jonas and my caring sisters Natalie and Sofie, my brother-in-law Bert and my amazing nephews Joren, Siemen and Marijn. To the reader of this thesis: sit back, relax and I hope you enjoy it.

Robin Verschueren

Abstract

In this thesis, we discuss several techniques for solving nonlinear optimization problems arising in nonlinear model predictive control (NMPC). They share two things in common: they all approximate some non-convex functions with convex ones, and they are all useful in a real-time, embedded context. First, a convexification method for structured indefinite quadratic programming problems is presented. Such problems are found in sequential quadratic programming (SQP) methods for NMPC problems. Its main advantage is a convergence speedup, i.e. local quadratic convergence, under some assumptions compared to a quasi-Newton or a generalized Gauss-Newton approach.

A second new technique, called sequential convex quadratic programming (SCQP), is useful in the presence of constraint and objective functions that are 'convex-over-nonlinear'. SCQP is a generalization of the GGN method, similar to sequential convex programming (SCP) but at a lower computational cost. For example, it exploits the convexity in ellipsoidal constraints, often encountered in practice. Two additional convex approximation methods are presented for time-optimal problems. The first one is an approximation of the time-optimal problem with convex l_1 penalties. The second method reformulates nonlinear, non-convex path constraints as simple bounds, under some conditions.

As a final contribution, we present a novel software framework, `acados`, in which the above techniques are implemented. It is a modular framework for embedded optimization, meant to facilitate rapid prototyping of new algorithms. New features with respect to existing software like `ACADO` are that it is built on the optimized linear algebra library `BLASFEO` and on the automatic differentiation library `CasADi`. Interfaces to higher-level languages such as `Python` and MATLAB are available. Some numerical examples show the ease-of-use and a significant speedup with respect to `ACADO`.

Kurze Zusammenfassung

Diese Dissertation behandelt mehrere Methoden zur Lösung von Optimierungsproblemen die in der nichtlinearen Modellprädiktiven Regelung (NMPC) auftreten. Sie haben alle zwei Dinge gemein: alle Methoden approximieren nicht-konvexe Funktionen mit konvexen Funktionen, und sind praktisch einsetzbar für Echtzeitanwendungen auf eingebetteten Systemen. Ein erster Beitrag ist ein Konvexifizierungsverfahren für strukturierte, indefinite quadratische Optimierungsprobleme, wie sie in sequentiellen quadratischen Optimierungsmethoden (SQP) für NMPC vorkommen. Der Hauptvorteil ist eine Konvergenzbeschleunigung, d.h. lokale quadratische Konvergenz, im Vergleich zum quasi-Newton oder dem generalisierten Gauss-Newton (GGN) Verfahren.

Eine zweite neue Methode, genannt sequentielle konvex-quadratische Optimierung (SCQP), ist hilfreich für ‘konvex-nichtlineare’, verkettete Funktionen in den Nebenbedingungen und in der Zielfunktion. SCQP ist eine Verallgemeinerung des GGN Verfahren, ähnlich der sequentiellen konvexen Optimierung (SCP) jedoch zu niedrigeren Rechenkosten pro Iteration. Zum Beispiel kann die Konvexität aus ellipsenförmiger Nebenbedingungen, die oft in der Praxis vorkommen, ausgenutzt werden. Außerdem werden zwei weitere Approximationsmethoden für zeitoptimale Probleme eingeführt. Die erste Methode ist eine konvexe Approximation zeitoptimaler Optimierungsproblem mit l_1 Funktionen. Die zweite Methode ist eine Reformulierung von nichtlinearen, nicht-konvexen Ungleichheitsnebenbedingungen als einfache Ober- und Untergrenzen.

Als letzten Beitrag stellen wir ein neues Softwarepaket vor, `acados`, in dem u.a. die obengenannten Techniken implementiert sind. Es ist ein modulares Paket für eingebettete Optimierung, das eine schnelle Entwicklung von neuen Algorithmen ermöglicht. Eine Neuheit im Vergleich zu schon bestehenden Softwarepaketen wie `ACADO` ist, dass `acados` auf der Sofrware für lineare Algebra `BLASFEO` und der automatischen Differentiationssoftware `CasADi` aufbaut. Interfaces mit höheren Programmiersprachen wie `Python` und MATLAB stehen ebenfalls zur

Verfügung. Numerische Beispiele zeigen eine hohe Benutzerfreundlichkeit und eine signifikante Beschleunigung der Rechenzeiten im Vergleich zu `ACADO`.

Contents

Abstract iii

Kurze Zusammenfassung v

Contents vii

List of Figures xiii

List of Tables xvii

1 Motivation & Background 1

1.1 Motivation 1

1.2 Numerical optimization 4

1.2.1 Convexity 6

1.2.2 A taxonomy of optimization problems 7

1.2.3 Conditions for optimality 10

1.2.4 Newton-type optimization 13

1.3 Dynamics 19

1.3.1 Linear dynamics 20

1.3.2 Nonlinear dynamics 21

1.3.3 Numerical simulation 21

1.4 Optimal control 23
1.4.1 Solution method classes for optimal control 25
1.4.2 Single shooting 26
1.4.3 Multiple shooting 27
1.4.4 Direct collocation 27
1.5 Model predictive control 29
1.5.1 Linear-quadratic MPC 29
1.5.2 Nonlinear MPC 33
1.6 Outline & Contributions 35

2 Convexification for optimal control 38

2.1 Introduction 39
2.2 Equality constrained problems in optimal control 41
2.2.1 A characterization of convexity 43
2.2.2 Structure-preserving convexification 45
2.2.3 Transfer of cost between stages 45
2.2.4 The structure-preserving convexification algorithm . . . 48
2.2.5 Connection with the discrete Riccati equation 51
2.2.6 Recovering the dual solution of the original QP 52
2.2.7 A tutorial example 53
2.3 Inequality constrained optimization 55
2.3.1 Revealing convexity under active inequalities 55
2.3.2 Preserving the OCP structure 57
2.3.3 Recovering the dual solution 60
2.3.4 Local convergence of SQP with structure-preserving convexification 61
2.4 Dealing with indefinite reduced problems 62
2.4.1 Regularization of the Hessian 62

2.4.2 Recovering Lagrange multipliers 64
2.5 Numerical example 64
2.6 Summary 68

3 Sequential convex quadratic programming 70
3.1 Introduction 70
3.2 Problem formulation 72
3.3 Asymptotic stability of a local minimizer 73
3.3.1 Unconstrained optimization problem 74
3.3.2 Equality and inequality constraints 76
3.4 Sequential convex quadratic programming 79
3.5 Numerical example 81
3.5.1 Implementation and software 81
3.5.2 Inverted pendulum with terminal region 81
3.6 Summary 83

4 Time-optimal nonlinear model predictive control for point-to-point motions 86
4.1 Introduction 87
4.2 Problem formulation 88
4.3 Time-scaling approach 88
4.4 Exponentially increasing penalty 90
4.5 NMPC formulation 94
4.6 Simulation results 96
4.6.1 Toy example 96
4.6.2 Application: hanging pendulum 97
4.6.3 Application: permanent magnet synchronous machines . 99
4.7 Summary 101

5 Time-optimal path following for robotic manipulators 102

5.1 Introduction 102
5.2 Spatial reformulation of robot dynamics 104
5.2.1 Path representation 105
5.2.2 Spatial reformulation 106
5.3 Optimal control problem formulation 107
5.3.1 Time-scaled time-optimal control problem 108
5.3.2 Spatial time-optimal control problem 109
5.4 Simulation results 109
5.4.1 Numerical algorithms 109
5.4.2 Time-optimal point-to-point motion along a path 110
5.4.3 Time-optimal obstacle avoidance 112
5.5 Summary 117

6 The `acados` software package 118

6.1 Introduction 119
6.2 The `acados` software package 121
6.2.1 The `acados` core library 122
6.2.2 The C interface 125
6.2.3 The C++ interface 128
6.2.4 The `Python` and MATLAB interfaces 129
6.3 Numerical results 129
6.3.1 Case study 1: Chain of Masses 131
6.3.2 Case study 2: Regularization 137
6.3.3 Case study 3: SCQP 139
6.3.4 Case study 4: Processor-in-the-loop experiments 140
6.4 Summary 145

7 Conclusions & Outlook **147**

Bibliography **151**

Curriculum Vitae **165**

List of publications **167**

List of Figures

1.1 Some simple convex and non-convex sets. *Left.* The hexagon, which includes its boundary (shown darker), is convex. *Middle.* The kidney shaped set is not convex, since the line segment between the two points in the set shown as dots is not contained in the set. *Right.* The square contains some boundary points but not others, and is not convex. Text and image from [Boyd and Vandenberghe, 2004]. 6

1.2 The relations between some important optimization problem classes. The framed classes depict the ones treated in this thesis. 8

1.3 Parameter estimation example. (a) Objective and gradient, with (local) solution found by each method, starting from the same initial guess. (b) Resulting parameter fit. A curve corresponding to the 'true' solution ($w^\star = 1$) is depicted in black. 18

2.1 Convergence for SQP algorithm with three different regularization methods. On the vertical axis we plot the distance to the global solution $w^\star$. The Hessian obtained by Algorithm 2.2 with $\delta = 10^{-4}$ enables quadratic convergence of the exact Newton method, in contrast to the alternative regularization methods with $\epsilon = 10^{-4}$. 54

2.2 Schematic illustrating the inverted pendulum on top of a cart. . 65

2.3 Optimal trajectories for OCP (2.65). Note that the bounds on F_k are active at the solution. 67

2.4 Comparison of the convergence of an SQP-type method applied to obtain the solution to NLP (2.65), with three different regularization methods. We compare two OCP instances, on the left $N = 100$, on the right $N = 200$. 69

3.1 For a solution $w^\star$ to be stable, the Hessian approximation $H(w^\star)$ has to lie inside the shaded region (3.10) for any $w \in \mathbb{R}^n \backslash \{0\}$. . 76

3.2 Comparison of the convergence of exact Hessian SQP, GGN and SCQP to a local minimum of the pendulum OCP. The measure of convergence is the norm of the residual of the KKT system. $R_e = 0.05\,\mathrm{m}$, corresponding to "A" in Figure 3.3. 83

3.3 Greatest eigenvalue $\beta_{\max} = \rho((\widetilde{H}^\star)^{-1}(\widetilde{H}^\star - \Lambda^\star))$ for GGN and SCQP, for different values of R_e. For a solution to OCP (3.24) to be a stable fixed point, $\beta_{\max}$ has to lie below the horizontal line $\beta_{\max} = 1$. The lines "A" and "B" correspond to solutions in Figure 3.4. 84

3.4 Solution of OCP (3.24) for different values of R_e. Note that for these radii, the solution is a stable fixed point for SCQP, and an unstable one for GGN (Figure 3.3). 85

4.1 Closed-loop trajectories for the system of the toy example under NMPC method (4.7). 97

4.2 Solution to the optimal control problem of pendulum with varying length. In black, the exponential weighting method is plotted, in gray, the time scaling method. 100

4.3 CPU times and number of QP iterations against number of steps of the NMPC. 101

5.1 Illustration of the position of the end-effector with respect to the closest position of the path. 106

5.2 Time-optimal solution (red) of optimal control problems (5.13) and (5.14) with cylindrical path constraints. The constraint space is delimited with dark blue faces, the centerline is shown in black. 112

5.3 The lateral and vertical deviation are taken with respect to the centerline between the two cylinders. 113

5.4 Comparison of the solution of the two OCP formulations (5.13) and (5.14). The torques in the robot joints are shown. The joint torque bounds are taken to be $-50\,\mathrm{Nm} \leqslant \tau \leqslant 50\,\mathrm{Nm}$ for both formulations. 114

5.5 Sketch of the ABB IRB120 industrial robot. During the simulation, the end-effector points down (as shown). 115

5.6 Time-optimal optimal control problem with obstacle avoidance around an arbitrary central path with piecewise constant curvature. 116

6.1 Example of the relation between modules and submodules in `acados` for one specific algorithm. 125

6.2 The interplay between the `acados` dependencies, the 'core' `C` library and its interfaces. 126

6.3 Listing of a simple linear MPC example using the `Python` interface to `acados`. 130

6.4 Forces in the springs between the masses in the example of Case Study 1. Replicated from [Wirsching et al., 2006]. 131

6.5 Relative cumulative suboptimality (RCSO) in each step of the simulation of the hanging chain with $M = 5$ and $N = 40$, for the different solvers. The baseline for comparison is the `IPOPT` solution. 135

6.6 Computational time for each iteration of the closed loop simulation, averaged over 10 runs. 136

6.7 Trade-off between sub-optimality (Figure 6.5) and computation time (Figure 6.6). We see that `acados` and `GRAMPC` lie on the Pareto-optimal front. 137

6.8 Convergence comparison of exact Hessian based SQP with three different regularization strategies. 139

6.9 SCQP on the pendulum example (cf. OCP (3.24)). 140

6.10 dSPACE MicroAutoboxII [dSPACE, 2006] embedded PC with a 900 MHz PowerPC processor. Courtesy of dSPACE GmbH. . . . 141

6.11 Schematic drawing of the two-stage turbocharging concept. From [Albin et al., 2017]. 142

6.12 Closed-loop simulation of the engine model with steps in the reference boost pressure. Simulations are carried out on the dSPACE MicroAutoboxII platform at a clock speed of 900 MHz. 145

List of Tables

1.1 Collocation points 28

2.1 Parameters for the inverted pendulum example 64

2.2 Exact-Hessian SQP iterations for the pendulum example (N=100), using the structure-preserving convexification method from Algorithm 2.6. 66

5.1 Denavit-Hartenberg parameters for the three-link robot used in simulation 110

6.1 Software packages for embedded optimization. 119

6.2 Overview of the software modules present in acados. 123

6.3 Design parameters for the chain of masses case study 133

6.4 Tuning parameters for the different solvers in Case Study 1. 134

6.5 Computation times for a closed-loop experiments on a chain of masses (cf. Figure 6.6) 136

6.6 Exact-Hessian based SQP: computation times 138

6.7 Parameter values for the two-stage turbocharged engine model 143

Chapter 1

Motivation & Background

All models are wrong, but some are useful.

George E. P. Box, statistician

1.1 Motivation

Control algorithms operate all around us. From the autonomous subway carriages in cities like Copenhagen and Tokyo, the cooling system in the laptop I'm writing on, to the large chemical plants seen at major ports around the globe. In some cases control algorithms replace human labor, in other cases they augment our human capacity. Often, they appear in places where no human can go, or under conditions that humans would not be willing to accept, or at millisecond and microsecond timescales that are beyond human capabilities. Let us call these last kind of systems *autonomous*: systems that operate in circumstances where man, for some reason, declines or is not able to take control.

Examples of autonomous systems are autonomous submarines looking for oil (a human would be crushed under the pressure), lithography systems producing computer chips (no human could reach the same speed or precision), and robot-taxis already on the road in some cities around the world.

It is safe to say that many autonomous systems rely on *optimization* of some sort. One reason for this is that it offers a meaningful way of explaining our wishes to a machine, e.g. “minimize the time to get to a certain goal while staying within the road bounds and speed limits”. Such a wish could be cast as an optimization problem as follows:

$$\begin{aligned} \underset{\substack{\mathit{throttle,}\\ \mathit{steering}}}{\text{minimize}} \quad & \mathit{arrival\ time} \\ \text{subject to} \quad & \mathit{stay\ on\ road} \\ & \mathit{don't\ violate\ speed\ limits} \\ & \mathit{avoid\ obstacles} \end{aligned}$$

The ‘minimize’ keyword declares an optimization problem. Beneath it, the decision variables are listed. An optimization algorithm searches for values of the decision variables such that the quantity to the right is as small as possible, while respecting the constraints listed in the ‘subject to’ clause. Humans are famously bad at solving optimization problems, except for the most trivial ones. Algorithms can do it much faster and much more accurately[1].

In autonomous systems, the kind of optimization that appears is often called ‘optimization-based control’. An extra ingredient, as opposed to classical optimization problems, is the notion of a dynamic system that evolves over time. It needs to be included in the optimization formulation to be able to meaningfully steer it autonomously. In this context, the notion of *feedback* is important.

There is quite some difference in how optimization-based control methods are used. *Offline* methods solve dynamic optimization problems with no timing constraints - they may take minutes or hours to come up with an answer. These kind of problems arise in e.g. parameter estimation, training of deep neural networks and production planning.

Online or real-time optimization methods, however, are expected to yield results before a certain deadline, as the result is useful only within this time window. Outdated results are meaningless or might even be dangerous. Sometimes, online optimization algorithms operate under very strict timing constraints, often with sampling times of a few milliseconds or less. The study of online methods for dynamic optimization problems within strict computational constraints is called *embedded* optimization, according to the definition from [Ferreau et al., 2017]:

---

[1] On the other hand, humans are far ahead of machines still when it comes to cognitively ‘easy’ problems such as perception and mobility. This is called Moravec’s paradox [Moravec, 1988]

> “An optimization method is called embedded if and only if it a) can run autonomously, i.e., if it delivers a sufficiently accurate solution within a real-time limit given a priori, and b) can be deployed on a computing hardware with limited resources.”

This type of optimization methods forms the focus of this thesis. More specifically, it centers on one optimization-based control method in particular: model predictive control (MPC). From a high-level perspective, it can be conceived as an algorithm repeating the following three steps:

1. Estimate the current state of the system under control,
2. Calculate an optimal response, given a prediction of the future behavior of the system obtained through a model,
3. Feed the optimal input to the system. Go to 1.

The model predictive nature (in step 2) of this technique makes for an attractive alternative to classical control methods, such as proportional-integral-derivative (PID) control. One could liken PID to driving a car by looking out the side window. With MPC, we look through the front windshield. Moreover, the optimization problem to be solved in step 2 can incorporate constraints, such as operating limits and safety margins, directly. For classical state-space control methods, which do feature a (linear) predictive model, constraints are often an afterthought.

Given the tight timing constraints for the calculation of an optimal response, it is typically necessary to make approximations. Approximations can happen in various places, e.g. in the modeling of the system, in the formulation of the optimization problem and in the optimization algorithms themselves. Concretely, we look at convex approximations for non-convex problems. Convexity, defined later in this chapter, is a mathematical concept that, if fulfilled by an optimization problem, allows for efficient algorithms. Non-convex problems, on the other hand, are much harder to solve in general. By making these approximations, we are thus able to calculate accurate solutions efficiently.

Throughout this thesis, we look at various convex approximations with applications in different contexts - this forms the bulk of the text. Another important part is software. Having efficient algorithms is less useful if it is not available as code for academics and practitioners. Therefore, we will introduce a new software package for embedded optimization and nonlinear model predictive control. It includes efficient implementations of the convex approximation methods discussed in the remainder of the thesis. But first, let us lay the mathematical foundations.

The remainder of this chapter forms the groundwork upon which the remaining chapters are built. It follows a bottom-up approach: we introduce the concepts of mathematical optimization and numerical simulation before tackling optimal control and model predictive control. Along the way, we will build up a framework and a notation, in order to better develop the work discussed in the subsequent chapters.

1.2 Numerical optimization

Numerical optimization, sometimes also called mathematical programming, is a field in applied mathematics in which one looks at solving optimization problems using some (approximate) numerical method. Optimization problems exist in a wide variety. In this thesis, we restrict ourselves to nonlinear optimization problems in Euclidean space.

At its core, each optimization problem consists of a set of n decision variables $w \in \mathbb{R}^n$ and an objective function $f : \mathbb{R}^n \to \mathbb{R}$. When we say that we are 'optimizing', we mean trying to find a point $w^\star \in \mathbb{R}^n$ for which the value of f is the lowest. In short, an (unconstrained) optimization problem reads as

$$\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad f(w). \tag{1.1}$$

By convention, we only regard minimization problems, instead of maximization problems, since minimizing $f(w)$ is equivalent with maximizing $-f(w)$.

Some optimization problems come with additional requirements, or constraints, on the decision variables. Such constraints are either equality constraints $g(w) = 0$, with $g : \mathbb{R}^n \to \mathbb{R}^{m_{\text{eq}}}$ or inequality constraints $c(w) \leqslant 0$, where $c : \mathbb{R}^n \to \mathbb{R}^{m_{\text{ineq}}}$. In short, many optimization problems can be cast as

$$\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad f(w) \tag{1.2a}$$

$$\text{subject to} \quad g(w) = 0 \tag{1.2b}$$

$$c(w) \leqslant 0. \tag{1.2c}$$

Remark 1.1. *Please note that this is not at all the most general optimization problem formulation. For instance, instead of vector inequalities with respect to $\mathbb{R}_+^{m_{\text{ineq}}}$, we could have a generalized inequality $\preceq_{\mathcal{K}}$ with respect to any cone $\mathcal{K}$. However, we consider formulations more general than* (1.2) *to be out of the scope of this thesis.*

At the dawn of numerical optimization as a field, quite a bit of attention went to linear programming – problems in which constraint and objective functions are affine. It is maybe for this reason that for a long time, a sharp divide existed between linear and nonlinear programming [Forsgren et al., 2002]. Nowadays, however, there seems to be a large consensus that, at least conceptually, this divide should lie between convex and non-convex optimization. What does it mean for an optimization problem to be convex? Before we introduce the concept of convexity, we discuss a few preliminaries from mathematical optimization.

A candidate point $\overline{w} \in \mathbb{R}^n$ is called *feasible* if it satisfies the constraints (1.2b)-(1.2c). The set of all feasible points is called the feasible set – we define it as follows.

Definition 1.1. *The feasible set* Ω_f *of an optimization problem of the form* (1.2) *is*

$$\Omega_f := \{w \in \mathbb{R}^n \mid g(w) = 0,\, c(w) \leqslant 0\}. \tag{feasible set}$$

In absence of constraints, we call the optimization problem *unconstrained*, and the feasible set is $\mathbb{R}^n$. A problem is called infeasible if $\Omega_f = \emptyset$. An inequality constraint is called *active* at some point $\overline{w} \in \mathbb{R}^n$ if equality holds in (1.2c), i.e. if $c_i(\overline{w}) = 0$ for some integer $i \in \mathcal{I}$ where we define the index set $\mathcal{I} := \{1, \ldots, m_{\text{ineq}}\}$. The active set is a subset of $\mathcal{I}$, namely

$$\mathcal{A}(\overline{w}) := \{i \in \mathcal{I} \mid c_i(\overline{w}) = 0\}. \tag{active set}$$

Next, let us look at what a ‘solution’ means in the context of an optimization problem. A point $w^\star \in \Omega_f$ is called a (global) minimizer if it holds that

$$f(w^\star) \leqslant f(w), \quad \forall w \in \Omega_f. \tag{1.3}$$

Generally speaking, an optimization problem can have any number of minimizers:

- $\min_w e^{-w}$: no minimizer,
- $\min_w w^2$: one minimizer,
- $\min_w w^4 - w^2$: two minimizers,
- $\min_w \sin(w)$: infinitely many minimizers.

A point $w^\star \in \Omega_f$ is called a local minimizer if (1.3) holds in a neighborhood of $w^\star$. More precisely,

$$f(w^\star) \leqslant f(w), \quad \forall w \in \Omega_f \cap \mathcal{B}_\beta^\star, \tag{1.4}$$

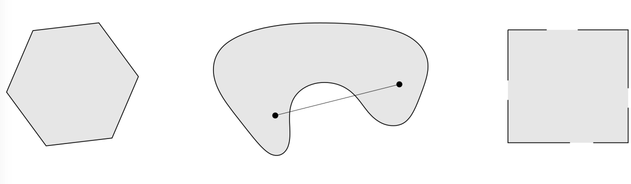


Figure 1.1: Some simple convex and non-convex sets. *Left.* The hexagon, which includes its boundary (shown darker), is convex. *Middle.* The kidney shaped set is not convex, since the line segment between the two points in the set shown as dots is not contained in the set. *Right.* The square contains some boundary points but not others, and is not convex. Text and image from [Boyd and Vandenberghe, 2004].

for some β, where we define $\mathcal{B}_\beta^\star := \{w \in \mathbb{R}^n \mid \|w - w^\star\|_2 \leqslant \beta\}$ to be a Euclidean ball with radius β around $w^\star$. Note that the intersection in (1.4) is important, as the minimizer might lie on the boundary of the constraints.

1.2.1 Convexity

Convexity is a concept that has been extensively studied in theoretical mathematical fields such as convex analysis. For our purposes, we can restrict ourselves to two distinct, but related facets of it: convex functions and convex sets.

Definition 1.2 (Convex set)**.** *A set $\Omega \subset \mathbb{R}^n$ is a convex set if for any two points $w_1, w_2 \in \Omega$ and for all $\theta \in [0,1]$ it holds that*

$$\theta w_1 + (1-\theta) w_2 \in \Omega. \qquad \text{(convex combination lies in } \Omega)$$

In plain language, a convex set is a set in which we can draw a straight line between any two points in the set, that lies itself in the set. We can see examples of convex and non-convex sets in Figure 1.1. Other important examples of convex sets include

- A line : $\{w \in \mathbb{R} \mid aw + b = 0\}$, with $a, b \in \mathbb{R}$,
- An affine set : $\{w \in \mathbb{R}^n \mid Aw + b = 0\}$, with $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$,
- A Euclidean ball : $\{w \in \mathbb{R}^n \mid \|w - w^\star\|_2 \leqslant \beta\}$, with $w^\star \in \mathbb{R}^n$ and $\beta > 0$.

Next, we define what it means for a function to be convex:

Definition 1.3 (Convex function)**.** *A function $f : \Omega \to \mathbb{R}$ is a convex function if Ω is convex and for any two points $w_1, w_2 \in \Omega$ and for all $\theta \in [0, 1]$ it holds that*

$$f(\theta w_1 + (1 - \theta)w_2) \leqslant \theta w_1 + (1 - \theta)w_2. \qquad \text{(Jensen's inequality)}$$

In other words, all lines between two points on the graph $f(w)$ should lie above the graph. If Jensen's inequality is strict, we call the function *strictly* convex. A function f is concave if $-f$ is convex. The following definition is useful in the context of convex functions:

Definition 1.4 (Positive definite matrix)**.** *A symmetric matrix $H \in \mathbb{R}^{n\times n}$ is positive definite, i.e. $H \succ 0$, if*

$$w^\top H w > 0, \quad \forall w \in \mathbb{R}^n \backslash \{0\}.$$

If for all $w \in \mathbb{R}^n$ it holds that $w^\top H w \geqslant 0$, the matrix is called positive semidefinite, denoted by $H \succeq 0$.

Simple examples of convex functions of $w \in \mathbb{R}^n$ are

- An affine function : $a^\top w + b$ with $a \in \mathbb{R}^n, b \in \mathbb{R}$,
- A quadratic function : $w^\top H\, w + b^\top w$, with $H \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$ and $H \succeq 0$.
- An exponential function : $e^{(b^\top w)}$, with $b \in \mathbb{R}^n$.

Note the elegant similarity between Definition 1.2 and 1.3 – indeed, there is a connection between these two concepts, namely that any sub-level set of a convex function f, i.e.

$$\Omega_\alpha := \{w \in \mathbb{R}^n \mid f(w) \leqslant \alpha\}, \qquad (\alpha\text{-sublevel set})$$

is convex.

Armed with the notion of convexity, we can now discuss different types of optimization problems that arise in various fields of engineering.

1.2.2 A taxonomy of optimization problems

Convex optimization problems (i.e. optimization problems with convex objective and convex constraints) have a beneficial property: each local minimizer, if there are any, is a global one. We will make this the main distinction in the zoo of optimization problems.

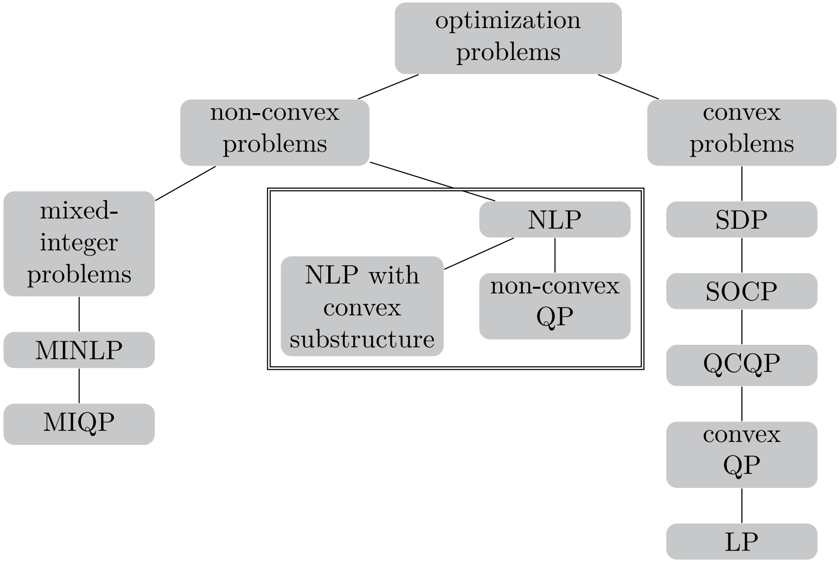


Figure 1.2: The relations between some important optimization problem classes. The framed classes depict the ones treated in this thesis.

Affine problems Linear programming (LP) problems were the first type of optimization problems solved with computers. Indeed, the first computers were running optimization solvers for linear programming problems used to deal with transportation, scheduling and allocation of resources during World War II [Britannica, 2018]. Linear programming is characterized by problems with an affine objective and affine (in)equality constraints. Problems of this kind arise in planning and resource management. Linear programming is still a vibrant topic in e.g. operations research.

Quadratic problems Quadratic programming (QP) problems have a quadratic cost function and affine constraints:

$$\begin{aligned} \underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2} w^\top H w + b^\top w \\ \text{subject to} \quad & Gw + g = 0 \\ & Cw + c \leqslant 0 \end{aligned} \tag{1.5}$$

We call it a *convex QP problem* if $H \succeq 0$.

QP problems arise naturally in statistical regression and parameter estimation. For instance, linear least squares problems can be cast as an unconstrained QP problem. Also in control we often encounter QP problems. Usually, a steady state is tracked and deviations from this steady state are penalized quadratically. The control practitioner will typically choose the cost function such that the QP problem is convex, e.g. a diagonal matrix H with $H \succeq 0$. However, this is not always how QPs emerge.

Non-convex quadratic problems Sometimes, QPs arise as subproblems in algorithms for more general problems. For instance, sequential quadratic programming (SQP) is an iterative method for solving nonlinear optimization problems. It linearizes the nonlinear constraints and builds a second-order approximation of the objective. The resulting quadratic function may be non-convex. In fact, some of the main results of this thesis (cf. Chapter 2) focus on non-convex quadratics. We note that because of their non-convexity, algorithms for global solution of non-convex QP problems have a worst-case exponential complexity.

Mixed-integer problems There is a large body of applications where the decision variables of optimization problem (1.2) are *discrete*: they take on integer values $0, 1, -1, 2$, etc. Examples of this are switches in electronic circuits [Stellato et al., 2017b], gear shifts in a car [Kehrle et al., 2011], or the well-known 3SAT boolean satisfiability problem. Algorithms for global solution of this kind of problems also have an exponential worst-case complexity.

Nonlinear programming In nonlinear programming (NLP) problems, the cost function and constraints in (1.2) are general nonlinear functions. Often, we make the additional assumptions that f, g and c are twice continuously differentiable. A special class of NLP, namely 'NLP with convex substructure', introduced in (1.10), is relevant for Chapter 3.

Nonlinear convex problems In the realm of nonlinear (and non-quadratic) functions, we can find quite a bit more classes of convex problems. The easiest extension to a QP is a quadratically constrained quadratic programming (QCQP) problem, with affine equalities but convex quadratic inequalities. Going further along this route, we have second-order cone programming (SOCP) problems, which in standard form look like:

$$\begin{aligned} \underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & b^\top w \\ \text{subject to} \quad & Gw + g = 0 \\ & \|C_i w + c_i\|_2 \leqslant a_i^\top w + b_i, \quad i = 1, \ldots, m_{\text{ineq}}. \end{aligned} \tag{SOCP}$$

SOCP derives its name from the second-order cone with respect to which we consider the inequalities. A more general cone is the positive semi-definite cone, which gives rise to semi-definite programming (SDP) problems:

$$\begin{aligned} \underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & b^\top w \\ \text{subject to} \quad & Gw + g = 0 \\ & G_0 + G_1 w_1 + \ldots + G_n w_n \preceq 0. \end{aligned} \tag{SDP}$$

It can be shown that the class of SDP problems contains the class of SOCP problems, and more general, for convex problems we have the following hierarchy:

$$\text{LP} \subset \text{convex QP} \subset \text{QCQP} \subset \text{SOCP} \subset \text{SDP},$$

also depicted in Figure 1.2. In this thesis, we will mainly focus on algorithms and methods concerned with NLPs and non-convex QPs.

1.2.3 Conditions for optimality

Conditions for optimality are a central theme in numerical optimization. They not only give us mathematical guarantees, they are often also a starting point for new algorithms. We discuss three optimality conditions in this context: the first-order necessary conditions (better known as the Karush-Kuhn-Tucker or KKT conditions), the second-order necessary conditions and the second-order sufficient conditions. The following technical condition will be useful in the remainder of this section.

Definition 1.5 (LICQ). *The linear independence constraint qualification is said to hold at some point* $\overline{w} \in \mathbb{R}^n$ *if*

$$\left\{ \frac{\partial g_i}{\partial w}(\overline{w}) \;\middle|\; i = 1, \ldots, m_{\text{eq}} \right\} \cup \left\{ \frac{\partial c_i}{\partial w}(\overline{w}) \;\middle|\; i \in \mathcal{A}(\overline{w}) \right\}$$

form a linearly independent set of vectors.

KKT conditions for optimality

First, we define the Lagrangian function corresponding to NLP (1.2). To this end, we introduce Lagrange multipliers (sometimes called 'shadow costs') $\lambda \in \mathbb{R}^{m_{\text{eq}}}$ and $\mu \in \mathbb{R}^{m_{\text{ineq}}}$ associated with the equality and inequality constraints, respectively. The Lagrangian is then defined as

$$\mathcal{L}(w, \lambda, \mu) := f(w) + \lambda^\top g(w) + \mu^\top c(w). \tag{Lagrangian}$$

We can interpret the Lagrangian as an extension to the cost function where we linearly penalize constraint deviations.

The necessary first-order conditions for optimality give us a set of equations. Pedagogically, methods for nonlinear optimization like SQP and interior point (IP) methods can be motivated from these conditions. We state them in the following theorem – for a proof, we refer the reader to [Nocedal and Wright, 2006].

The KKT conditions are then defined as follows.

Theorem 1.1 (KKT conditions)**.** *Assume that there exists a local minimizer* $w^\star$ *of problem* (1.2)*. Furthermore, assume that* f, g *and* c *are continuously differentiable, and that LICQ holds, at* $w^\star$*. Then, there exists a triple* $v^\star := (w^\star, \lambda^\star, \mu^\star)$*, called a KKT point, such that the following equations hold:*

$$\nabla_w \mathcal{L}(w^\star, \lambda^\star, \mu^\star) = 0, \tag{1.6a}$$

$$g(w^\star) = 0, \tag{1.6b}$$

$$c(w^\star) \leqslant 0, \tag{1.6c}$$

$$\mu^\star \geqslant 0, \tag{1.6d}$$

$$\mu_i^\star c_i(w^\star) = 0, \quad i = 1, \ldots, m_{\text{ineq}}. \tag{1.6e}$$

The last three conditions are called the *complementarity* conditions. We say that *strict* complementarity holds at a KKT point $v^\star$ if either

$$\mu_i^\star = 0, c_i(w^\star) \neq 0$$

or

$$\mu_i^\star \neq 0, c_i(w^\star) = 0$$

holds for all $i \in \{1, \ldots, m_{\text{ineq}}\}$, and there exists no $i \in \{1, \ldots, m_{\text{ineq}}\}$ for which $\mu_i^\star = 0, c_i(w^\star) = 0$.

What is the use of strict complementarity? Let us look at an example where strict complementarity does not hold.

Example 1.1. *Consider the following QP, with* $\underline{c} \in [0, \infty]$*:*

$$\begin{aligned} \underset{w \in \mathbb{R}^2}{\text{minimize}} \quad & \frac{1}{2}\left(w_1^2 + w_2^2\right) \\ \text{subject to} \quad & \underline{c} \leqslant w_1. \end{aligned} \tag{1.7}$$

We find that $w^\star = (\underline{c}, 0)$ *is the unique global solution. LICQ holds at that point, so we can compute the optimal Lagrange multiplier from the first KKT condition:*

$$\nabla_w \mathcal{L}(w, \mu) = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \mu \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

By putting $\nabla_w \mathcal{L}(w^\star, \mu^\star) = 0$*, we quickly see that* $\mu^\star = \underline{c}$*. Now, fix* $\underline{c} = 1$*. The optimal value of the multiplier can be interpreted as the 'cost' that the constraint incurs. Thus, we can estimate the optimal value (at least locally) from this multiplier, if the constraint were to move. If we move the constraint bound to* $\underline{c} = 1.1$*, the estimated optimal value would be* $f(w^\star) + \Delta\underline{c} \cdot \mu^\star = 0.5 + 0.1 \cdot 1 = 0.6$ *which is close to the actual new optimal value* 0.605*.*

Now consider $\underline{c} = 0$*. This implies that* $\mu^\star = 0$*, i.e. we have no information on how much the constraint is costing us – in principle, it could be removed from the problem without changing the optimal value, at least in a first-order sense. For theoretical investigations, we often assume strict complementarity to hold at KKT points.*

Second-order conditions for optimality

The KKT conditions only take into account first-order derivative information. If second-order derivatives are available – i.e. f, g and c are twice continuously differentiable – we can write down another set of necessary conditions for optimality of a point $v^\star$.

First, for a fixed point $\overline{w}$, let us introduce a shorthand notation for the equality constraints together with the active inequalities (with a slight abuse of terminology, we will call this the vector of 'active constraints'):

$$\tilde{g}(\overline{w}) := \begin{bmatrix} g(\overline{w}) \\ c_i(\overline{w}) \end{bmatrix}, \quad i \in \mathcal{A}(\overline{w}),$$

with length $m = m_{\text{eq}} + \text{card}(\mathcal{A}(\overline{w}))$.

Now let us look at two of the four fundamental subspaces of $\frac{\partial \tilde{g}}{\partial w}(\overline{w})$.

Definition 1.6 (Range space and null space basis)**.** *For some feasible w, we introduce the shorthand*

$$\widetilde{G} := \frac{\partial \tilde{g}}{\partial w}(w).$$

We define an orthonormal basis for the range space of $\widetilde{G}^\top$ and the null space of $\widetilde{G}$ as follows:

$$\widetilde{G}\widetilde{Z} = 0,$$

$$\left[\widetilde{Y}\widetilde{Z}\right]^\top \left[\widetilde{Y}\widetilde{Z}\right] = I,$$

with $\widetilde{Z} \in \mathbb{R}^{n\times(n-m)}$ and $\widetilde{Y} \in \mathbb{R}^{n\times m}$.

Employing this definition we can now state the second-order necessary conditions for optimality.

Theorem 1.2 (SONC)**.** *Let $v^\star = (w^\star, \lambda^\star, \mu^\star)$ be a KKT point, assume strict complementarity holds and define $\widetilde{Z}$ as in Definition 1.6. If $w^\star$ is a local minimizer, it holds that*

$$\widetilde{Z}^\top \nabla_w^2 \mathcal{L}(v^\star)\, \widetilde{Z} \succeq 0.$$

In Chapter 2, we will see an equivalent statement to SONC, motivating an efficient method for recovering convexity in non-convex QPs. For completeness, we now also state the second-order *sufficient* conditions for optimality, under the same assumptions as Theorem 1.2:

Theorem 1.3 (SOSC)**.** *Let $v^\star = (w^\star, \lambda^\star, \mu^\star)$ be a KKT point, assume strict complementarity holds and define $\widetilde{Z}$ as in Definition 1.6. If it holds that*

$$\widetilde{Z}^\top \nabla_w^2 \mathcal{L}(v^\star)\, \widetilde{Z} \succ 0,$$

then $w^\star$ is a unique local minimizer.

The following definition is useful in the rest of the text.

Definition 1.7. *A KKT-point $v^\star := (w^\star, \lambda^\star, \mu^\star)$ at which LICQ, SOSC and strict complementarity hold, is called a* regular solution *of NLP* (1.2)*.*

1.2.4 Newton-type optimization

The conditions for optimality discussed in the previous section inspired different optimization methods. In this thesis, we solve optimization problems such as

NLP (1.2) with Newton-type methods. A Newton-type algorithm proceeds to find solutions to the KKT complementarity system (1.6) by iterating on the triple

$$v^j := (w^j, \lambda^j, \mu^j) \quad \text{(at iteration } j\text{)}$$

until a solution is reached, up to some accuracy. Two major approaches exist – sequential quadratic programming (SQP) and interior point methods. They differ in the way they treat the inequalities.

Interior point methods

Instead of solving the non-smooth complementarity system formed by the KKT conditions (1.6), we can approximate the system by decreasingly perturbing it with a barrier parameter. The resulting (ordinary) nonlinear system of equations is then solved with Newton's method. This is the fundamental idea behind interior point methods. Pedagogically, there are two equivalent ways of describing interior point methods – as a barrier problem or as a perturbation of the KKT conditions. We will opt for the latter here.

When looking more closely at the KKT conditions, we see that all equations are smooth, except for (1.6e). In fact, there is a kink at (w, λ, μ) for which $c_i(w) = 0, \mu_i = 0, i \in \mathcal{I}$. In order to apply Newton's method, we require a smooth set of nonlinear equations, so we introduce an approximation as follows:

$$\nabla_w \mathcal{L}(w, \lambda, \mu) = 0 \tag{1.8a}$$

$$g(w) = 0 \tag{1.8b}$$

$$\mu_i c_i(w) = \eta_j, \quad i = 1, \ldots, m_{\text{ineq}} \tag{1.8c}$$

where η_j is called the *barrier* parameter at iteration j. For fixed η_j, solving the perturbed KKT conditions with a Newton-type method gives rise to structured linear systems that can be solved by any sparsity-exploiting linear algebra solver, e.g. one from the HSL suite [HSL, 2011]. By adapting the barrier parameter in each iteration such that

$$\lim_{j \to \infty} \eta_j = 0,$$

the method ultimately converges (under some conditions) to $(w^\star, \lambda^\star, \mu^\star)$. Since the focus of this thesis does not lie on interior point methods, we refer the reader to [Bertsekas, 1999] and the survey paper [Forsgren et al., 2002] for a much deeper treatise on the subject.

We remark that the biggest contrast between interior point methods for solving QPs and NLPs is the barrier update strategy. An important ingredient of interior point methods is how to update the barrier parameter from one iteration to the next. The Mehrotra predictor-corrector [Mehrotra, 1992] is by far the most popular one for interior point methods for linear and quadratic programming. It is an adaptive technique, meaning that the value of the barrier parameter can move up and down, before going to zero in the limit. Another barrier update strategy, sometimes encountered in nonlinear programming, e.g. in `IPOPT` [Wächter and Biegler, 2006] or `KNITRO` [Byrd et al., 2006], is the monotone Fiacco-McCormick approach.

Compared to sequential quadratic programming methods, treated in the following section, it is more difficult to warm-start interior point methods, although recent advances have been made in this direction [Shahzad and Goulart, 2011; Gondzio and Grothey, 2006; Zanelli et al., 2017b].

Sequential quadratic programming

Just like interior point methods, sequential quadratic programming (SQP) methods make an approximation to the KKT conditions (1.6). However, instead of perturbing the nonlinear conditions directly, it constructs successive quadratic approximations of the KKT conditions around the current iterate $v^j = (w^j, \lambda^j, \mu^j)$. This gives rise to inequality-constrained QP problems, of the form:

$$\begin{aligned} \underset{\Delta w \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\Delta w^\top H \, \Delta w + \nabla f(w^j)\Delta w \\ \text{subject to} \quad & \frac{\partial g}{\partial w}\left(w^j\right) \Delta w + g(w^j) = 0 \\ & \frac{\partial c}{\partial w}\left(w^j\right) \Delta w + c(w^j) \leqslant 0. \end{aligned} \tag{1.9}$$

We denote the primal solution Δw_{QP} and dual solution $\lambda_{\mathrm{QP}}, \mu_{\mathrm{QP}}$, corresponding to the equalities and inequalities, respectively. We introduced the Hessian matrix $H \in \mathbb{R}^{n \times n}$. For now, we assume an exact Hessian based SQP method, so $H = \nabla_w^2 \mathcal{L}(v^j)$.

At its core, an SQP algorithm successively linearizes the KKT conditions around the current iterate v^j, solves QP subproblem (1.9) and updates the current iterate by

$$\begin{aligned} w^{j+1} &= w^j + \Delta w_{\mathrm{QP}} \\ \lambda^{j+1} &= \lambda_{\mathrm{QP}} \\ \mu^{j+1} &= \mu_{\mathrm{QP}}. \end{aligned}$$

There are many extensions to this basic SQP algorithm. For instance, a *globalized* SQP method employs line search or a trust region to guarantee convergence to a local optimum. In *embedded* optimization, typically globalization is not pursued as it incurs some overhead and real-time constraints might already be stringent. We refer to the monograph [Conn et al., 2000] for a detailed description of trust region methods and to [Nocedal and Wright, 2006] for line search methods.

Another extension is formed by the quasi-Newton SQP methods. These do not compute the Hessian of the Lagrangian ($H = \nabla_w^2 \mathcal{L}(v^j)$) exactly, but instead rely an approximations of it, by an update rule. Among the quasi-Newton methods we have the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm, the symmetric rank 1 (SR1) update, and Broyden's method. Some of these are widely used in nonlinear optimization methods. However, the fact that the number of Hessian updates per iteration might become large renders these methods less attractive for embedded optimization.

One Hessian approximation that is popular in embedded optimization is the Gauss-Newton Hessian approximation. Given a nonlinear least squares objective,

$$f(w) = \frac{1}{2}\|r(w)\|_2^2,$$

we can derive the Gauss-Newton Hessian approximation as follows: as opposed to an exact Hessian approach, we linearize 'inside the norm':

$$f_{\mathrm{GN}}^j(w) = \frac{1}{2}\|r(w^j) + \frac{\partial r}{\partial w}(w^j)\left(w - w^j\right)\|_2^2.$$

This is a quadratic function of w, with Hessian

$$H_{\mathrm{GN}} = \frac{\partial r}{\partial w}(w^j)^\top \frac{\partial r}{\partial w}(w^j).$$

Advantages of using a Gauss-Newton Hessian in an SQP-type context are the following: first, it is always positive definite. Second, it is cheap to compute compared to an exact Hessian, and it does not require an estimate of the Lagrange multipliers. Lastly, though convergence of an SQP method using a Gauss-Newton Hessian approximation is 'only' linear, if the residuals $r(w^\star)$ are small at the solution, the linear convergence rate is fast.

One lesser known property of the Gauss-Newton method, in the context of parameter estimation, is that it avoids convergence to 'statistically unstable' solutions [Bock et al., 2007]. We will show this behavior in an example.

Example 1.2. *Consider the following parameter estimation problem:*

$$\underset{w \in \mathbb{R}}{\text{minimize}} \quad f(w) = \frac{1}{2} \sum_{i=1}^{M} \left(y_i - \frac{1}{1 + (w + x_i)^2} \right)^2 ,$$

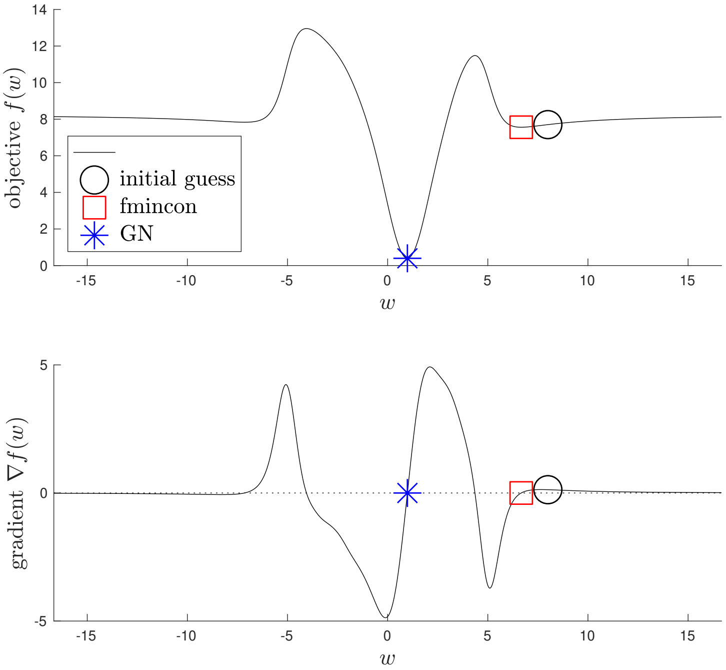
with $M = 100$ *the number of measurements* (x_i, y_i)*, and* w *the parameter with true value* $w^\star = 1$*. Measurements are generated from the 'true' model* $1/(1 + (w^\star + x_i)^2) + \varepsilon_i$ *with* $x_i, i = 1, \ldots, M$*, being* M *equidistant points between* -5 *and* 5*, and* $\varepsilon_i \sim \mathcal{N}(0, 0.1)$ *being normally distributed.*

We solve this optimization problem once with a globalized SQP solver (the SQP algorithm from `fmincon` *in* MATLAB*) and once with a full-step Gauss-Newton method. Interestingly, as can be seen from Figure 1.3, when starting both methods from the same point, the globalized* `fmincon` *routine finds a solution at* $w \approx 6.67$*, as the gradient is zero there. However, it is not a solution in the statistical sense, i.e. it is not a "continuous deformation of the true parameter value due to perturbations by measurement noise" [Bock et al., 2007]. If we look at the Gauss-Newton method, it seems to be able to 'jump over' the hill in the objective value to reach the correct solution* $w^\star = 1$*, see Figure 1.3 for the fit. Moreover, even if we start really close to* $w \approx 6.67$*, the Gauss-Newton method will move away and either converge to the correct solution, or diverge. This behavior shows that even if a full-step Gauss-Newton method does not always converge, it is not a sign of a bad optimization method, but of a wrong problem formulation. To quote [Bock et al., 2007]:*

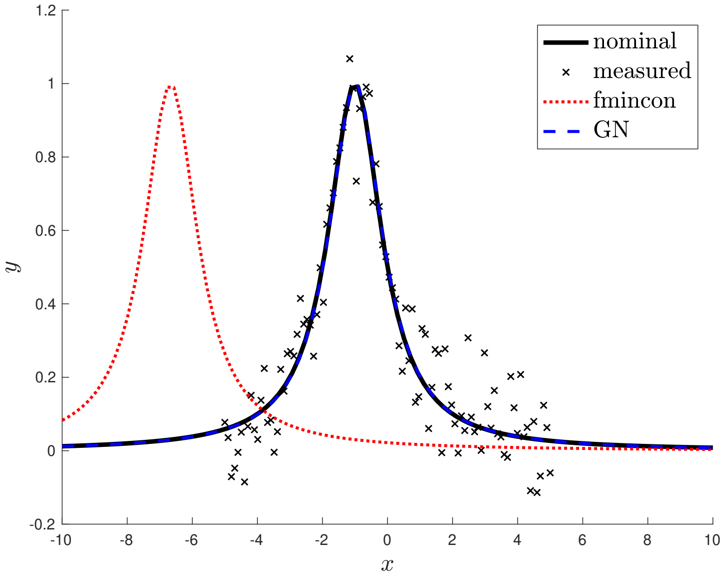
> *"From this point of view quasi-Newton methods have only limited use for parameter estimation, since they do not deliver estimates for the true parameter values! Thus, slow local convergence or no convergence of the full-step Gauss-Newton method, seen often as a disadvantage of the method, indicates deficiencies in the model or lack of data and can be considered as its advantage!"*

Sequential convex programming

In the last section we tackled SQP, in which we solve a QP problem in each iteration. We can take this idea one step further and solve a more general convex problem in each iteration. Such an approach is called sequential convex programming (SCP) [Tran-Dinh and Diehl, 2010]. We will discuss SCP briefly for the following NLP with convex substructure in objective and constraints:



(a) Objective and gradient value.



(b) Resulting fit for both methods.

Figure 1.3: Parameter estimation example. (a) Objective and gradient, with (local) solution found by each method, starting from the same initial guess. (b) Resulting parameter fit. A curve corresponding to the ‘true’ solution ($w^\star = 1$) is depicted in black.

$$\begin{aligned} \underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & \phi_0(r_0(w)) \\ \text{subject to} \quad & g(w) = 0 \\ & r_i(w) \in \Omega_i, \quad i = 1, \ldots, m_{\text{ineq}}, \end{aligned} \tag{1.10}$$

where Ω_i are convex sets and ϕ_0 is a convex function. Note that a constrained nonlinear least squares problem is a special case of above problem with $\phi_0 \equiv \frac{1}{2}\|\cdot\|_2^2$.

For problem (1.10), full-step SCP amounts to solving convex subproblems of the form

$$\begin{aligned} \underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & \phi_0\left(r_0(w^j) + \frac{\partial r_0}{\partial w}(w^j)\left(w - w^j\right)\right) \\ \text{subject to} \quad & g(w^j) + \frac{\partial g}{\partial w}(w^j)\left(w - w^j\right) = 0 \\ & r_i(w^j) + \frac{\partial r_i}{\partial w}(w^j)\left(w - w^j\right) \in \Omega_i, \quad i = 1, \ldots, m_{\text{ineq}}. \end{aligned}$$

Note that the convex substructure is kept in the subproblems. Typically, off-the-shelf nonlinear convex optimization solvers are then used to solve these possibly nonlinear convex subproblems. In Chapter 3, we will see an algorithm called ‘sequential convex quadratic programming’ (SCQP) that finds the middle ground between SQP and SCP: in SCQP we do not neglect convex substructure in objective and/or constraints, but we are still able to use QP solvers, which have been proven to work efficiently and robustly.

1.3 Dynamics

In this work, our goal is to control systems in some optimized way. In this context, ‘systems’ can be regarded as mathematical objects with inputs, some internal state, and outputs. Systems as encountered in the real world are always time-varying, nonlinear and stochastic. However, for pedagogical reasons, and for reason of scope, we can study idealized systems that do not exhibit all of these properties. In this thesis, we focus on deterministic nonlinear dynamical systems. If the system varies over time, it will be stated explicitly. Furthermore, we look at lumped parameter systems from a *state-space* point of view. We will not discuss any distributed parameter systems (e.g. systems governed by partial differential equations) or non-parametric systems (e.g. frequency-domain models).

The state $x(t)$ of a system is denoted by n_x quantities, varying over time:

$$x : \mathbb{R} \rightarrow \mathbb{R}^{n_x}$$

We influence the state of the system by applying controls (sometimes called inputs), defined as a function of time,

$$u : \mathbb{R} \rightarrow \mathbb{R}^{n_u}$$

to the system. The equations that govern this interaction are called the *dynamics* of the system.

1.3.1 Linear dynamics

In the history of control theory, most research effort is spent on the study of linear time-invariant systems. These are systems of the form

$$\dot{x}(t) = Ax(t) + Bu(t), \quad \forall t \in [0, \infty). \tag{linear dynamics}$$

To study and simulate these kind of systems in a digital computer, we need some means of evaluating the evolution of the state at discrete points in time $t_k = k \cdot \Delta t$, for $k = 0, 1, \ldots$, with Δt the sampling time. Thus, we need a discrete-time formulation of the continuous-time dynamics. In fact, for linear systems, if we assume the input to be piecewise constant the matrix exponential lets us bridge the gap between continuous-time and discrete-time dynamics:

$$x_{k+1} = A_{\mathrm{d}} x_k + B_{\mathrm{d}} u_k, \quad k = 0, 1, \ldots,$$

with

$$\begin{aligned} A_{\mathrm{d}} &= e^{A\Delta t}, \qquad \text{(matrix exponential)} \\ B_{\mathrm{d}} &= \int_0^{\Delta t} e^{A(\Delta t - \tau)} B \, \mathrm{d}\tau. \end{aligned}$$

There are only few systems that can be realistically modeled by linear equations. Indeed, linear dynamical systems often arise as a linearization of some nonlinear system, which we look at next.

1.3.2 Nonlinear dynamics

We start with a basic class of nonlinear dynamic systems. An initial value problem (IVP) of an ordinary differential equation (ODE) is characterized as follows:

$$\dot{x}(t) = \psi(x(t), u(t)), \quad \forall t \in [0, T], \quad x(0) = \overline{x}_0, \tag{1.11}$$

with $\psi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_x}$ and initial value $\overline{x}_0 \in \mathbb{R}^{n_x}$. It is well-established that existence and uniqueness of a solution holds when ψ is continuously differentiable and T small enough, which we will assume for the remainder of the text. Note that a solution to an initial value problem can still 'explode' in finite time, e.g. $\dot{x} = 1 + x^2, x(0) = x_0$ [Strogatz, 2001].

Throughout the thesis, we will see various examples of different ODEs.

Differential-algebraic equations

A differential-algebraic equations (DAE) is more general than an ODE. It includes additional algebraic functions $\nu : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_a}$. The algebraic variables $z(t)$, with $z : \mathbb{R} \rightarrow \mathbb{R}^{n_z}$, along with the states and controls, are governed by the DAE as follows:

$$\begin{aligned} \dot{x}(t) &= \psi(x(t), u(t), z(t)), \\ 0 &= \nu(x(t), u(t), z(t)), \quad \forall t \in [0, T]. \end{aligned} \tag{semi-explicit DAE}$$

These kind of equations arise naturally from conservation laws or from additional constraints on the system, often induced by some physical law, for example mechanical constraints or chemical reaction rates. Often, DAEs arise in the Euler-Lagrange modeling formalism. Please refer to [Brenan et al., 1987] for an in-depth discussion.

An even more general formulation is a so-called fully implicit set of DAEs:

$$0 = \Psi(\dot{x}(t), x(t), u(t), z(t)).$$

If the Jacobian $\frac{\partial \Psi}{\partial(\dot{x}, z)}$ is invertible, we say that the DAE is of index 1.

1.3.3 Numerical simulation

For nonlinear dynamics, there is no general closed-form expression to obtain discrete-time dynamics from continuous-time dynamics, as was the case for linear systems. Instead, we rely on numerical integration routines that compute simulations and corresponding sensitivities with respect to the input variables.

Given an initial state x_0 and a control u (assumed to be constant on the interval $[0, T]$), we want to obtain the following quantities from our simulation routines:

$$x(T; x_0, u), \quad \frac{\partial x(T; x_0, u)}{\partial x_0}, \quad \frac{\partial x(T; x_0, u)}{\partial u},$$

that is, the simulated value of the state at time T given an initial value for the state and a value for the control vector, and its derivatives with respect to the parameters x_0 and u. There are many ways of computing these derivatives. For an in-depth discussion, we refer the reader to [Quirynen, 2017]. We discuss one particular class of methods next.

Runge-Kutta integration methods

Among the most well-known integration methods are the Runge-Kutta (RK) methods. They form a family of integrators of different orders and come in both explicit and implicit forms [Hairer et al., 1993]. 'Order' in this context means that, for example, if an integrator has order p and the step size Δt shrinks by a factor 2, the integration error shrinks by 2^p. The difference between implicit and explicit methods is that explicit methods do not have to solve a nonlinear set of equations to obtain the simulation result, due to the fact that the Butcher tableau is lower triangular. Implicit methods exhibit better integration stability properties in the case of stiff equations compared to explicit RK methods.

In the following, we denote an integrator by ψ_{d}, a 'discrete' version of ODE ψ. One of the two most famous Runge-Kutta integrators is the explicit Runge-Kutta integrator of order 1 (also called 'explicit Euler method'):

$$x_{k+1} = \psi_{\mathrm{d}}(x_k, u_k) = x_k + \Delta t \cdot \psi(x_k, u_k). \tag{explicit Euler}$$

It borrows its popularity from its simplicity. However, not much more difficult to implement, but significantly more accurate at lower computational cost, is the explicit Runge-Kutta integrator of order 4, also called ‘RK4’:

$$\begin{aligned} k_1 &= \Delta t \cdot \psi(x_k, u_k), \\ k_2 &= \Delta t \cdot \psi(x_k + \frac{k_1}{2}, u_k), \\ k_3 &= \Delta t \cdot \psi(x_k + \frac{k_2}{2}, u_k), \\ k_4 &= \Delta t \cdot \psi(x_k + k_3, u_k), \\ x_{k+1} &= \psi_{\mathrm{d}}(x_k, u_k) = x_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{RK4}$$

DAEs are often simulated with implicit Runge-Kutta integration methods, such like collocation integrators. We refer to [Quirynen, 2017] for an in-depth discussion on the implementation of efficient implicit integrators in the context of embedded optimization.

1.4 Optimal control

The previous two topics of this chapter, namely numerical optimization and dynamical systems, form the two crucial ingredients for optimal control. An optimal control problem in continuous time, being an optimization problem where some constraints include continuous-time dynamics, reads as follows:

$$\begin{aligned} \underset{x(\cdot),\, u(\cdot)}{\text{minimize}} \quad & \int_0^T l(x(t), u(t))\,\mathrm{d}t + l_{\mathrm{f}}(x(T)) \\ \text{subject to} \quad & x(0) = \overline{x}_0, \\ & \dot{x}(t) = \psi(x(t), u(t)) \quad \forall t \in [0, T], \\ & c(x(t), u(t)) \leqslant 0 \qquad \forall t \in [0, T], \\ & c_{\mathrm{f}}(x(T)) \leqslant 0. \end{aligned} \tag{1.12}$$

In contrast to an NLP – for example, (1.2) – the solution is not a vector of numbers but rather a function (for simplicity, we do not specify in which function space[2]). The equality constraints arise from the dynamics, in this

---

[2] We refer to [Luenberger, 1969] for a treatise on optimization in more general spaces than Euclidean space.

case modeled by an explicit ODE as in (1.11). The inequality constraints $c : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{m_{\mathrm{ineq}}}$ are called path constraints and $c_{\mathrm{f}} : \mathbb{R}^{n_x} \to \mathbb{R}^{m_{\mathrm{ineq},T}}$ is called the terminal constraint. The initial value for the state is $\overline{x}_0$. Function $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is called the running cost or stage cost, $l_{\mathrm{f}} : \mathbb{R}^{n_x} \to \mathbb{R}$ is called the terminal cost. These are typically chosen by the control designer to obtain the desired control performance. Before we discuss different methods for solving OCP (1.12), we discuss two different choices for l, l_{f}, arising in practice.

Reference tracking In optimization-based control, the most widely studied form of cost function is a reference tracking cost. This means that we penalize (often quadratically) the deviation from the state to some steady state $(x_{\mathrm{SS}}, u_{\mathrm{SS}})$, for which it holds that

$$\dot{x} = \psi(x_{\mathrm{SS}}, u_{\mathrm{SS}}) = 0.$$

This is often exactly the behavior that you want in practice: some steady state of the system is known exactly or approximately, and the system behavior is beneficial at that steady state. An example might be a chemical reactor producing a certain product of a required quality at a fixed rate. The stage cost and terminal cost functions then read as

$$\begin{aligned} l(x(t), u(t)) &= \|x(t) - x_{\mathrm{SS}}\|_Q^2 + \|u(t) - u_{\mathrm{SS}}\|_R^2, \qquad \text{(steady state tracking)} \\ l_{\mathrm{f}}(x(T)) &= \|x(T) - x_{\mathrm{SS}}\|_{Q_{\mathrm{f}}}^2. \end{aligned}$$

Here, Q, Q_{f} and R are weighting matrices to be chosen by the control designer. Often, they are chosen diagonal and positive definite.

Sometimes, we do not track a reference state/control pair but an output:

$$\begin{aligned} l(x(t), u(t)) &= \|y(x(t), u(t)) - y_{\mathrm{ref}}\|_W^2, \qquad \text{(output tracking)} \\ l_{\mathrm{f}}(x(T)) &= \|y_{\mathrm{f}}(x(T)) - y_{\mathrm{ref}}\|_{W_{\mathrm{f}}}^2, \end{aligned}$$

where $y : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ and $y_{\mathrm{f}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ form the (nonlinear) output functions and $y_{\mathrm{ref}} \in \mathbb{R}^{n_y}$ are the reference values for those outputs, respectively, weighted by matrices W and W_{f}.

Time-optimal control When we optimize for time, there is nothing to track: we want to minimize the time spent for our system to move from some initial state to some terminal state. Without loss of generality, we assume the terminal state is the origin. This implies that the total moving time T becomes a decision variable:

$$
\begin{aligned}
\underset{x(\cdot),u(\cdot),T}{\text{minimize}} \quad & T \\
\text{subject to} \quad & x(0) = \overline{x}_0, \\
& \dot{x}(t) = \psi(x(t), u(t)) \quad \forall t \in [0, T], \\
& c(x(t), u(t)) \leqslant 0 \qquad \forall t \in [0, T], \\
& x(T) = 0, \\
& 0 \leqslant T.
\end{aligned}
\tag{1.13}
$$

As such, $l \equiv 1$ and $l_{\mathrm{f}} \equiv 0$. A standard technique (see e.g. [Rösmann et al., 2015]) to work around the fact that the time interval $[0, T]$ is not known a priori, is to scale the time variable and consequently, the dynamics, by a factor T.

$$\tau := \frac{t}{T} \tag{pseudo-time}$$

which implies

$$
\begin{aligned}
\frac{\mathrm{d}x}{\mathrm{d}\tau}(\tau) = \frac{\mathrm{d}x}{\mathrm{d}t}(t) \cdot \underbrace{\frac{\mathrm{d}t}{\mathrm{d}\tau}(\tau)}_{=T}, & \\
= \psi(x(t), u(t)) \cdot T, \qquad & \forall t \in [0, T], \qquad \text{(time transformation)} \\
= \widetilde{\psi}(x(\tau), u(\tau), T), \qquad & \forall \tau \in [0, 1].
\end{aligned}
\tag{1.14}
$$

We will call this technique *time scaling*. A different, more useful alternative to formulation (1.13) in the context of embedded optimization will be presented in Chapter 4.

1.4.1 Solution method classes for optimal control

Indirect methods for optimal control formulate the necessary conditions for optimality for an infinite-dimensional solution to optimal control problem (1.12), sometimes based on the Pontryagin maximum principle, and as such result in a boundary-value problem (BVP) that is solved numerically. Once functions $x(\cdot)$ and $u(\cdot)$ are found, we need to sample them at the rate of our system under control. These methods are often described to 'first optimize, then discretize'. Historically, the methods we now call indirect have taken up most of the attention, most notably in aerospace research. For an introductory work on indirect methods, we refer to [Athans and Falb, 1966]. Many different types of optimal control problems can be solved using this technique. One difficulty with indirect methods is the handling of arbitrary path constraints.

Direct methods for optimal control form another class of methods. They, by contrast, operate under the motto ‘first discretize, then optimize’. The more recently developed direct methods offer more flexibility and are easier to implement. We will focus on the latter for the rest of this thesis.

The main drawback of indirect methods is that they require the solution of a BVP, which is difficult for the general case. Direct methods, by contrast, approximate the solution of this BVP by introducing a grid on the time interval $[0, T]$ into N intervals. On each interval we simulate the continuous-time dynamics. By doing so, we obtain a finite-dimensional problem (i.e. with decision variables in $\mathbb{R}^n$ instead of some function space), which we can solve with the optimization methods discussed in Section 1.2. Three often used direct methods are single shooting, multiple shooting and direct collocation. Each of them will be discussed subsequently.

1.4.2 Single shooting

The term ‘shooting’ refers in this context to the simulation of the dynamical system. By single shooting, we mean the simulation of the state trajectories on all the grid intervals, given a control trajectory, at once. The controls are usually assumed to be piecewise constant (this corresponds to practice, where zero-order hold signals are used for analog-to-digital conversion). Doing so, the decision variables in single shooting are the controls $u_0, \ldots, u_{N-1}$, which are used to simulate the state trajectory, as follows:

$$x(k\Delta t; \bar{x}_0, \mathfrak{u}), \quad k = 0, \ldots, N,$$

where we concatenate all control variables in $\mathfrak{u}^\top := [u_0^\top, u_1^\top, \ldots, u_{N-1}^\top]$.

Optimization problems obtained from single shooting read as

$$\begin{aligned} \underset{\mathfrak{u}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} l(x(k\Delta t; \bar{x}_0, \mathfrak{u}), u_k) + l_T(x(T; \bar{x}_0, \mathfrak{u})) \\ \text{subject to} \quad & c(x(k\Delta t; \bar{x}_0, \mathfrak{u}), u_k) \leqslant 0, \quad k = 0, \ldots, N-1 \\ & c_T(x(T; \bar{x}_0, \mathfrak{u})) \leqslant 0. \end{aligned} \tag{1.15}$$

We can see why single shooting is sometimes called the ‘sequential approach’: in order to formulate the optimization problem fully, we need to have simulated through all shooting intervals. This means that optimization and simulation are performed in lockstep, hence the adjective ‘sequential’.

1.4.3 Multiple shooting

Multiple shooting takes it one step further: it not only introduces a grid of control values, but also introduces additional decision variables for the states, on the same grid. A typical multiple shooting problem has the following form:

Definition 1.8 (NLP with OCP structure)**.**

$$\underset{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad \sum_{k=0}^{N-1} l_k(x_k,u_k) + l_\mathrm{f}(x_N) \tag{1.16a}$$

$$\text{subject to} \quad x_0 = \overline{x}_0, \tag{1.16b}$$

$$x_{k+1} = \psi_{\mathrm{d},k}(x_k,u_k), \quad k = 0,\dots,N{-}1, \tag{1.16c}$$

$$c_k(x_k,u_k) \leqslant 0, \quad k = 0,\dots,N{-}1, \tag{1.16d}$$

$$c_\mathrm{f}(x_N) \leqslant 0. \tag{1.16e}$$

Here, the simulation and optimization is interwoven, so this approach is sometimes called ‘simultaneous’. For instance, the integration could be readily computed in parallel. Although multiple shooting introduces extra optimization variables, it does not necessarily come at higher computational cost to solve, by employing structure-exploiting solver techniques, as discussed in Section 1.5. Moreover, multiple shooting has better convergence properties than single shooting, as discussed in [Bock, 1987].

1.4.4 Direct collocation

Instead of employing an integrator to simulate the dynamics, we could also make use of a polynomial function that interpolates the continuous-time dynamics at well-chosen points. This is the main idea behind direct collocation. Again, we choose a time grid, indexed by $k = 0, 1, \dots, N$ with piecewise constant controls. On one grid interval, we choose a smaller grid of *collocation* points indexed by $i = 0, 1, \dots, K$. On all N grid intervals we define polynomials of degree K, which interpolate the continuous-time dynamics at the collocation points [Lynn and Zahradnik, 1970; Stryk, 1993].

Table 1.1: Collocation points

K	Gauss-Legendre	Radau IIA
1	0.5	1.0
2	0.211325 0.788675	0.333333 1.000000
3	0.112702 0.500000 0.887298	0.155051 0.644949 1.000000
4	0.069432 0.330009 0.669991 0.930568	0.088588 0.409467 0.787659 1.000000
5	0.046910 0.230765 0.500000 0.769235 0.953090	0.057104 0.276843 0.583590 0.860240 1.000000

Direct collocation results in an NLP that is even more structured, when comparing it to the NLP arising in multiple shooting. The NLP can directly be solved with a large-scale sparse optimization solver like `IPOPT` [Wächter and Biegler, 2006] in combination with a sparse linear algebra solver, for example `ma86` from HSL [HSL, 2011].

It remains to be chosen where the collocation points lie; two often-used collocation methods are 'Gauss-Legendre' and 'Radau IIA', which use collocation points at the roots of Legendre polynomial $L_K : \mathbb{R} \rightarrow \mathbb{R}$ of degree K and at the roots of $L_K - L_{K-1}$, respectively. The Radau IIA collocation method (integration order $2K - 1$, i.e. polynomial dynamics of order $2K - 1$ are represented *exactly*) has better stability properties, the Gauss-Legendre method is of one order higher. For low-order schemes, we list the collocation points for both methods in Table 1.1. From the table, we can see one additional advantage of Radau IIA methods: in a given stage, the last node lies at the first time point of the next stage. This simplifies implementation a little. We note that many other collocation schemes exist, for example the 'Gauss-Lobatto' or 'Chebyshev' methods, for which we refer the reader to [Hairer and Wanner, 1991].

1.5 Model predictive control

In the preceding sections, we discussed optimization methods for solving optimal control problems in order to find an optimal state and control trajectory. Having an optimal control trajectory at hand, it can then be fed (or 'replayed') to the system. In an isolated setting, this might perform satisfactorily. In a dynamically changing environment, however, we would like to adjust our control solutions as the system under control progresses. Arguably the most popular online optimization-based control technique is model predictive control (MPC). The basic idea is to carry out the following steps repeatedly:

1. Measure or estimate the system state $\overline{x}_0$ at the current time.
2. Calculate an (approximately) optimal control trajectory by using a prediction of the future behavior of the system.
3. Feed the first optimal control $u_0^\star$ to the system until the next measurement arrives.

In step 2 above, we typically solve an OCP of type (1.12) with any of the direct methods that were discussed in Section 1.4. MPC gives rise to problems that we can categorize in two broad categories: linear-quadratic MPC (LQMPC) and nonlinear MPC.

1.5.1 Linear-quadratic MPC

LQMPC is characterized by affine (in)equality constraints and a quadratic cost function. LQMPC gives rise to structured QPs, as follows:

Definition 1.9 (QP with OCP structure).

$$\mathrm{QP}(H): \quad \underset{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \overbrace{\begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix}}^{H_k} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \frac{1}{2} x_N^\top Q_N x_N \tag{1.17a}$$

$$\text{subject to} \quad x_0 = \overline{x}_0, \tag{1.17b}$$

$$x_{k+1} = A_k x_k + B_k u_k, \quad k = 0,\dots,N-1, \tag{1.17c}$$

$$C_{k,x} x_k + C_{k,u} u_k \leqslant 0, \quad k = 0,\dots,N-1, \tag{1.17d}$$

$$C_N x_N \leqslant 0, \tag{1.17e}$$

where we define the state vectors as $x_k \in \mathbb{R}^{n_x}$, the controls as $u_k \in \mathbb{R}^{n_u}$, and the cost matrices as $Q_k \in \mathbb{R}^{n_x \times n_x}, S_k \in \mathbb{R}^{n_u \times n_x}, R_k \in \mathbb{R}^{n_u \times n_u}$ and the Hessian matrix $H := \mathrm{diag}(H_0, \ldots, H_{N-1}, Q_N)$. In LQMPC, the cost matrices $H_k, k = 0, \ldots, N-1$ and Q_N are to be chosen by the control designer, often as positive-definite diagonal matrices, such that QP (1.17) is convex. The constraints denote, respectively, dynamic constraints with matrices $A_k \in \mathbb{R}^{n_x \times n_x}, B_k \in \mathbb{R}^{n_x \times n_u}$, inequality constraints with $C_{k,x} \in \mathbb{R}^{m_{\mathrm{ineq},k} \times n_x}, C_{k,u} \in \mathbb{R}^{m_{\mathrm{ineq},k} \times n_u}, C_N \in \mathbb{R}^{m_{\mathrm{ineq},N} \times n_x}$ and an initial constraint with $\bar{x}_0 \in \mathbb{R}^{n_x}$.

Remark 1.2. *Note that this compact notation, as proposed in e.g. [Frison et al., 2014], allows for a more general OCP formulation including linear cost terms and constant terms in the constraints, coming from e.g. a tracking objective, if the state is augmented by one constant entry '1'.*

Solving QP problems arising in LQMPC in an embedded context with tight timing constraints calls for fast tailored methods. We discuss several strategies shortly.

Explicit MPC

Consider QP (1.17) as a *parametric* optimization problem, with parameter $\overline{x}_0$. The optimal control solution map as a function of the initial state, $u_0^\star(\overline{x}_0)$, can be shown to be continuous and piecewise affine. Each 'piece' of the solution map lives in a so-called critical region. This insight gave rise to the idea of *explicit* MPC [Bemporad et al., 1999]. It computes and enumerates all possible critical regions and corresponding control laws. These computations can be done offline, i.e. ahead of the runtime of the process. During the runtime of the process, the optimal control can be found by a simple query in a lookup table. A drawback of explicit MPC is the high memory requirements for storing the different regions due to the combinatorial explosion of the number of so-called 'critical regions'. In [Pannocchia et al., 2007], it is mentioned that "Practically speaking, the offline computation - identification of all regions [...] may be tractable for single-input-single-output problems with relatively short control horizon N, but it quickly becomes intractable as the dimensions [...] and control horizon N grows." Since the problems we target in this thesis potentially have a long horizon and more than a few states, we focus on other methods.

Condensing

One other idea for a tailored QP solution method was presented in [Bock and Plitt, 1984] and is called *condensing*. The dynamic equality constraints in QP (1.17) are used to eliminate the state variables x_k, $k = 0, \ldots, N$. The resulting QP has $N \cdot n_u$ optimization variables, collected in $\mathfrak{u}$, and reads as

$$\begin{aligned} \underset{\mathfrak{u}}{\text{minimize}} \quad & \mathfrak{u}^\top H_{\text{cd}}\, \mathfrak{u} + b_{\text{cd}}^\top \mathfrak{u} \\ \text{subject to} \quad & C_{\text{cd}} \mathfrak{u} + c_{\text{cd}} \leqslant 0, \end{aligned} \tag{condensed QP}$$

where the inequalities consist of the eliminated dynamic equality constraints and the original inequalities. The constraints are defined by $C_{\text{cd}} \in \mathbb{R}^{m_{\text{cd}} \times n_{\text{cd}}}$ and $c_{\text{cd}} \in \mathbb{R}^{m_{\text{cd}}}$, with $n_{\text{cd}} = N n_u$ and $m_{\text{cd}} = N(n_x + m_{\text{ineq}}) + m_{\text{ineq},N}$. The condensed 'Hessian' matrix and 'gradient' are defined by $H_{\text{cd}} \in \mathbb{R}^{n_{\text{cd}} \times n_{\text{cd}}}$ and $b_{\text{cd}} \in \mathbb{R}^{n_{\text{cd}}}$. From [Frison, 2015] we know that an efficiently implemented condensing has *quadratic* worst-case complexity as a function of the horizon length N. For details on the algorithm and notes on how to efficiently implement such a condensing method, please refer to that work.

After condensing, the dense subproblem can be passed to a generic QP solver, e.g. `QPKWIK` [Schmid and Biegler, 1994], `QPOPT` [Gill et al., 1995], `qpOASES` [Ferreau et al., 2014], to obtain the solution and afterwards recover all of the state variables and Lagrange multipliers by a so-called expansion step [Bock and Plitt, 1984].

Partial condensing A slightly different method for solving (1.17) can be obtained by not eliminating all state variables, but only per block of N/N_2 stages (if N is an integer multiple of N_2), where N_2 is the 'new' horizon length of the partially condensed problem. By this additional degree of freedom, partial condensing enables us to find a better trade-off between horizon length and number of optimization variables, for a given problem. For more details on partial condensing, please refer to [Axehill, 2015].

Active-set methods

Active-set methods are based on the principle that if one happens to know the set of active constraints at the solution, the optimization amounts to nothing more than the solution of a single set of linear equations, which is substantively simpler than solving an inequality constrained QP. Thus, active-set methods possess in each iteration a current guess of the set of active constraints (also called the working set), and, when the optimal active set is not found yet, updates this guess accordingly.

An active-set QP solver that has shown to be practical in the context of (N)MPC is `qpOASES` [Ferreau et al., 2014]. Its active set updates are based on the fact that subsequent problems in real-time MPC lie close to each other. `qpOASES` is best suited for small and dense problems, and it is typically used in combination with the condensing approach (discussed above). The fundamental idea behind `qpOASES` is based on a homotopy on the initial state. Starting at the optimal solution of a QP, the algorithm proceeds as follows. Given a new parameter $\overline{x}_0$, we can construct a homotopy with homotopy parameter $\overline{p} \in [0, 1]$, where $\overline{p} = 0$ is associated with the 'old' solution and $\overline{p} = 1$ with the new one. Three different possibilities arise, when moving along the homotopy path: 1) an active constraint becomes inactive, therefore we remove it from the working set, 2) an inactive constraint becomes active, such that we add it to the working set, or 3) we reach $\overline{p} = 1$, which means we found the optimal solution with respect to the new parameter $\overline{x}_0$. Since `qpOASES` is able to reuse information (i.e. *warm-starting*) from one problem to the next, it is particularly well-suited for (N)MPC.

Interior point methods

In principle, problem (1.17) could be solved by any general-purpose interior point QP solver, like `OOQP` [Gertz and Wright, 2003]. Often, such solvers are based on direct sparse linear algebra solvers. However, given the special 'optimal control' structure of the problem, we can do even better.

In [Steinbach, 1994; Rao et al., 1998], a structure-exploiting strategy for linear MPC problems is presented, which inspired others to write efficient interior point solvers. One such solver is `FORCES` [Domahidi et al., 2012]. It is a code-generation tool for linear MPC problems (recently, `Forces Pro` [Domahidi and Perez, 2013; Zanelli et al., 2017a] also offers nonlinear MPC solvers).

Another example of structure-exploiting interior point solver based on a Riccati recursion for the linear system solution is `HPMPC` [Frison et al., 2014], and its successor `HPIPM` [Frison, 2017]. They use a similar strategy as presented in [Rao et al., 1998], but additionally offer linear algebra kernels for small-to-midsize matrices (implemented in `BLASFEO` [Frison et al., 2018]), optimized for different computer architectures. This enabled an additional speedup compared to existing methods.

Dual-Newton strategy

The dual-Newton strategy is an alternative active set method tailored to QP (1.17), with an open-source implementation called `qpDUNES` [Frasch et al., 2015].

By looking at QP (1.17) in detail, we see that all constraints except for the dynamic constraints are *local* to a stage, i.e. they only refer to variables of that stage. Thus, we can dualize the dynamic constraints in (1.17) and solve the (unconstrained) dual problem, e.g. by Newton's method with line search. This dual problem is concave and piecewise quadratic, given that it is a strictly convex problem. The Newton system is constructed by solving $N + 1$ decoupled QP problems, localized to each stage. To this end, we can pick any QP solver, `qpDUNES` is implemented with `qpOASES`. An additional ingredient of the dual-Newton strategy is that it features an efficient factorization of the dual Hessian going from one problem to the next.

1.5.2 Nonlinear MPC

Many physical systems can not accurately be represented by linear dynamics. This motivates a generalization to allow for nonlinear dynamic equations and nonlinear path constraints. In this thesis, we will mainly look at nonlinear MPC (NMPC) problems in a multiple shooting context, giving rise to problems like NLP (1.16).

A popular technique in embedded optimization to solve NLP (1.16), if it has a least-squares objective, is the generalized Gauss-Newton (GGN) method proposed by [Bock, 1983]. It is an SQP method that employs a Gauss-Newton Hessian. In each iteration, the QP subproblem to be solved is of the form (1.17), similar to LQMPC. Thus, the same tailored solution strategies can be used to solve the QP subproblems in the SQP algorithm.

Advantages of the GGN method include that no second order derivatives need to be evaluated, that it is a multiplier-free algorithm, and that the Hessian approximation is always positive (semi-) definite. Therefore each QP subproblem is convex and can be solved efficiently and reliably. The GGN algorithm has been shown to be a reliable approach for real-time NMPC [Diehl et al., 2002].

The GGN method makes no distinction between convex or arbitrary nonlinear inequality constraints. This is undesirable: the linearization of e.g. an elliptical constraint might give rise to a bad approximation. A generalization of GGN, as one of the contributions in this thesis, is presented in Chapter 3.

One drawback to GGN is that its local convergence rate is only linear, in general. A remedy would be to use a similar SQP method, but with exact Hessians instead. This, however, can result in indefinite QP problems. One solution to this problem is presented in Chapter 2.

Online Methods

For NMPC problems, in contrast to linear-quadratic MPC, we have no guarantees on the solution manifold $u_0^\star(\bar{x}_0)$. In general, it is nonlinear and non-continuous, as shown by [Rawlings et al., 2017].

Suppose we have a good approximation of the solution to some problem instance. How do we prepare for the problem that arrives at the next sampling time? In general, there are two actions that can be taken. First, we realize that consecutive problems are not wildly different from each other. Thus, we can shift our 'old' solution forwards in time to obtain a reasonable approximation to be used with the next problem instance. Secondly, we can make use of a tangential predictor, which amounts to a local approximation to the solution manifold $u_0^\star(\bar{x}_0)$. Combining these two techniques, we have an adequate initial guess which we use to jump-start our optimization.

However, we face the *real-time* dilemma [Diehl et al., 2009]:

> "Either the nonlinear iteration procedure is performed until a pre-specified convergence criterion is met, which might introduce considerable feedback delays, or the procedure is stopped prematurely with only an approximate solution, so that a pre-specified computation time limit can be met."

One example of a method that tries to overcome this is the *advanced step controller* by [Zavala and Biegler, 2009]. It compensates for the computational delay introduced by the method. It is based on a nonlinear interior point algorithm and solves a full NLP in each sampling time.

An alternative is to not fully solve the optimization problem to convergence. This is opted for in e.g. the Newton-type controller by [Li and Biegler, 1989], which only performs one full Newton SQP-step per sampling time. This method is based on a *single shooting* discretization and does not make use of a tangential predictor. The Continuation/GMRES method by [Ohtsuka, 2004] also takes just one Newton step, but is based on a variant of an interior point algorithm, and it does make use of a tangential predictor.

Another approach for online NMPC and NMHE that has been proven to be efficient in practice is the real-time iteration (RTI) method by [Diehl et al., 2002]. This SQP-type method only solves one QP like (1.17) per sampling instant. The underlying idea is to approach a local solution *while* controlling the plant. The RTI method has some attractive properties for practical application. First, it makes use of a *generalized* tangential predictor, which can be shown to work well across active set changes, even when started from an approximate solution (we refer to [Diehl et al., 2009] for a more in-depth comparison between the different types of tangential predictors). Secondly, we can save time by splitting our computations in a *feedback* and *preparation* phase. The preparation phase consists of simulation of the nonlinear system, as well as generating first- and second-order derivative information. Also condensing is part of this phase. The feedback phase starts when the measurement of the initial state becomes available, and consists just of the solution of one QP, thus minimizing feedback delay.

A real-time implementation of the RTI method can be found in the `ACADO` Toolkit by [Houska et al., 2011], more specifically its Code Generation Tool. Many examples of experimental application of the RTI method exist, e.g. on electrical drives in the Megawatt range in [Besselmann et al., 2015], on small-scale race cars in [Verschueren et al., 2014], on autonomous agricultural vehicles in [Kayacan et al., 2014] and on a 6-DOF robotic manipulator in [van Duijkeren et al., 2016].

1.6 Outline & Contributions

After this introductory chapter we are ready to tackle the body of the thesis. Chapters 2-5 each present one theoretical result in the context of nonlinear model predictive control and show its usefulness with simulation studies. Chapter 6 presents a new software approach aspiring to be the successor of the `ACADO` Code Generation Tool. We shortly discuss the main contributions of each chapter.

Chapter 2 : Convexification for optimal control. This chapter presents a convexification method for indefinite QP problems as encountered in SQP-type methods for direct optimal control. The three main contributions are:

- An alternative, more general, necessary and sufficient condition for a QP problem to be convex (Theorem 2.6),

- A proof that the proposed method always finds a positive definite Hessian with OCP structure, if the reduced Hessian of the QP problem is positive definite (Theorem 2.7),
- A result that states that the iterations of a convexified SQP method are equal to the non-convexified SQP iterations, close to the NLP solution, resulting in local quadratic convergence (Theorem 2.9).

A simulation study on a nonlinear optimal control problem illustrates the local quadratic convergence and shows a significant convergence speedup compared to state-of-the-art regularization methods.

Chapter 3 : Sequential convex quadratic programming. The generalized Gauss-Newton (GGN) algorithm is a popular and successful algorithm for constrained nonlinear least squares problems. However, in the presence of e.g. ellipsoidal constraints, GGN sometimes perform badly. We introduce a generalization of GGN, similar to sequential convex programming (SCP), called sequential convex quadratic programming (SCQP). The contributions in this chapter are as follows:

- A condition on the Hessian for local linear convergence (Theorem 3.2),
- A new Hessian approximation readily usable in SQP-type methods, called SCQP,
- An illustration on a nonlinear optimal control problem where Gauss-Newton does not converge and SCQP converges fast.

Chapter 4 : Time-optimal point-to-point motions. A classical optimal control problem is the time-optimal control problem: bring a system from some initial state to some final state, in as little time as possible. However, the classical 'time-scaling' technique creates difficulties in obtaining Lyapunov-based stability results. We contribute the following:

- A time-optimal formulation with l_1 penalty functions and exponentially increasing weights,
- A result stating that this formulation recovers the time-optimal solution (Theorem 4.1),
- A stability proof in a nonlinear model predictive control context.

The practical usefulness of the time-optimal NMPC method is illustrated with hardware-in-the-loop experiments on an embedded platform.

Chapter 5 : Time-optimal path following for robotic manipulators. Sometimes, a system needs to move time-optimally along a path, as opposed to between two points. We propose a reformulation of the dynamics that achieves the following:

- It decouples the timing information from the path geometry,
- As an added benefit, some path constraints are reformulated as simple bounds.

Chapter 6 : The `acados` software framework. As a last contribution of this thesis, we introduce a novel software framework called `acados`. It is novel in the sense that it combines the numerical performance of the optimized linear algebra package `BLASFEO` while staying modular, which enables rapid prototyping. Moreover, it offers integration with `CasADi` as a modeling language and automatic differentiation tool. Moreover, it comes with interfaces to high-level languages `Python` and `Matlab`. Numerical experiments on an industrial embedded platform show the practical applicability of the software.

We conclude the thesis text in Chapter 7, and present an outlook to future research directions.

Chapter 2

Convexification methods for exact Hessian sequential quadratic programming problems arising in direct optimal control

> The great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity.
>
> R. Tyrrell Rockafellar, mathematician

This chapter offers one solution to a problem arising in SQP-type methods for direct optimal control: the indefiniteness of the Hessian of the Lagrangian. This is a problem, because an indefinite Hessian is not guaranteed to produce a descent direction. Furthermore, many QP solvers do not support indefinite Hessian matrices.

We present and discuss a method that convexifies the indefinite Hessian. The convexified Hessian has the same OCP structure as the original Hessian. Far from the solution, the convexified Hessian is an approximation. Close to the solution of the NLP (i.e. when the correct active set is identified), however, we can show that our convexification, combined with SQP, gives rise to exactly the same iterations as an SQP method with the original indefinite Hessian, under some regularity conditions. This is important, as we could then apply any QP solver that expects (strictly) positive-definite Hessian matrices to the convexified problem. In a sense, we make a 'jump' over the great watershed mentioned in the quote above.

An edited version of the work in this chapter has been published as an article in the SIAM Journal on Optimization, as [Verschueren et al., 2017b].

2.1 Introduction

As we saw in Chapter 1, there exist several approaches to solve the structured QP subproblem arising in SQP-type methods for optimal control. One method is to use condensing in combination with a dense QP solver. Alternatively, one can solve the QP directly using a structure-exploiting QP solver, of which the computational complexity typically scales linearly with the horizon length. This becomes advantageous in OCPs with long horizon lengths, in which case the condensing approach is less competitive [Vukov et al., 2013]. Examples of structure-exploiting QP solvers tailored to optimal control are `qpDUNES` [Frasch et al., 2015], `FORCES` [Domahidi et al., 2012] and `HPMPC` [Frison et al., 2014]. The software package `HQP` [Franke and Arnold, 1997] is a general-purpose sparse QP solver that can readily be used to solve large-scale OCPs. Although the structure-exploiting QP solvers `FORCES`, `HPMPC` and `qpDUNES` require a positive definite Hessian matrix, the second order sufficient conditions (SOSC) for optimality require positive definiteness only of the reduced Hessian, which is defined to be the Hessian matrix projected onto the null space of the active constraints [Nocedal and Wright, 2006]. We point out that `HPMPC` and `FORCES` would be in principle compatible with problems with an indefinite Hessian with positive definite reduced Hessian, however, this would not allow part of the code optimization for the Cholesky factorization and, therefore, indefinite Hessians are not supported. Moreover, `qpDUNES` does not allow indefinite Hessians, not even ones that are positive definite in the reduced space, as it is based on dual decomposition, which requires strictly convex Hessian matrices.

In NMPC, it often happens that the Hessian of the QP subproblem with OCP structure is indefinite, and therefore should be approximated with a positive definite Hessian in order to make sure that the calculated step is a descent direction; we refer to such a procedure as *regularization*. The Hessian regularization is performed right before solving the QP. More specifically, for the condensing approach, we could either regularize the full Hessian before performing the condensing step, or we could regularize the condensed Hessian afterwards. A numerical case study comparing these two alternatives is presented in [Verschueren et al., 2016c]. On the other hand, for the case of structure-exploiting QP solvers, the solvers mentioned in the previous paragraph all require the full Hessian to be positive definite. The proposed convexification method is therefore particularly suited for structured QP subproblems.

There are several ways of performing regularization. Levenberg-Marquardt regularization consists in adding a multiple of the identity matrix to the Hessian [Nocedal and Wright, 2006]. In [Murray and Yakowitz, 1984], a way of ensuring a positive definite Hessian without checking its eigenvalues, based on differential dynamic programming, is presented. One other method adapts the positive definiteness of the Hessian by directly modifying the factors of the Cholesky factorization or the symmetric indefinite factorization of the Hessian [Nocedal and Wright, 2006]. Quasi-Newton methods can generally be modified to directly provide a positive definite Hessian approximation, see e.g. [Gill et al., 2005; Gould and Robinson, 2010].

Regularization is also an important algorithmic component for interior point methods. One could for example look at the KKT matrix and ensure that it has the correct inertia, as e.g. in [Forsgren and Gill, 1998] by using an inertia-controlling factorization. A similar idea is used in IPOPT [Wächter and Biegler, 2006], that performs an inertia correction step on the KKT matrix whenever necessary. The inertia information comes from the indefinite symmetric linear system solvers used in that code. An improvement of the IPOPT regularization in case of redundant constraints is presented in [Wan and Biegler, 2016]. In [Morales et al., 2011], the non-convexity is overcome by solving an additional equality constrained QP. One interesting alternative method of dealing with indefiniteness is presented in [Gill and Robinson, 2013], which consists of solving QP subproblems with indefinite Hessians that can be proven to be equivalent to strictly convex QPs. Straightforward regularization methods typically modify the reduced Hessian of an optimization problem, and therefore also its corresponding optimal solution. Replacing the Hessian with a positive definite one without altering the reduced Hessian is called *convexification*.

As the main contribution of this chapter, we propose a structure-preserving convexification method for indefinite QPs with positive definite reduced Hessian. In case the Hessian is indefinite but the reduced Hessian is positive definite, we prove that the underlying convexity can be recovered by applying a modification to the original Hessian without altering the reduced Hessian and at the same time preserving the sparsity structure of the problem. The proposed algorithm can readily be extended to the case of indefinite reduced Hessians, resulting in a heuristic regularization approach which will be shown to perform well in practice.

Our convexification approach therefore (a) provides a fully positive definite Hessian; (b) can be applied as a separate routine, independent of the QP solver used; (c) preserves the optimal control sparsity structure and has a computational complexity that is linear in the horizon length.

Our convexification algorithm, which fulfills the above criteria, is a recursive procedure which exploits the block-diagonal structure of the Hessian and the stage-by-stage structure which is typical for direct optimal control. The resulting *convexified* Hessian can then be fed directly to a structure-exploiting QP solver. Note that by doing so, we avoid the potentially costly step of condensing. Instead, we directly solve the structured QP with the additional computational cost resulting from the above convexification procedure.

Our approach is motivated by the convergence of a Newton-type SQP method to a local solution for a nonlinear OCP. When employing the exact Hessian in such a method, convergence to a nearby local minimum is quadratic under mild assumptions [Nocedal and Wright, 2006]. When starting close enough to a local minimum, the convergence of the Newton-type method with convexified Hessian remains quadratic under the same assumptions. This will additionally be illustrated further in a numerical case study. We would like to point out that e.g. the method in [Gill and Robinson, 2013] will ultimately also recover quadratic convergence, but indefinite QP subproblems are solved instead of positive definite ones.

The structure of this chapter is as follows. In Section 2.2 we show how to recover convexity from general QPs and QPs with optimal control structure with positive definite reduced Hessians. This section also presents the structure preserving convexification algorithm. Section 2.3 shows how to handle inequality constraints and Section 2.4 deals with problems with indefinite reduced Hessian. An illustration of our regularization method is given, based on a nonlinear OCP example, in Section 2.5. We summarize the chapter in Section 2.6.

2.2 Equality constrained problems in optimal control

We focus now on QPs with an optimal control problem structure, as in Definition 1.9. They typically arise as a subproblem in an SQP-type method to solve the structured NLP (1.16). In addition to the notation in (1.17), we define $\widehat{Q}_N := Q_N$. We repeat the definition of a QP with optimal control problem structure, for the purpose of readability:

$$\text{QP}(H): \quad \underset{\substack{x_0,\ldots,x_N,\\ u_0,\ldots,u_{N-1}}}{\text{minimize}} \quad \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \overbrace{\begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix}}^{H_k} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \frac{1}{2} x_N^\top \widehat{Q}_N x_N \tag{2.1a}$$

$$\text{subject to} \quad x_0 = \overline{x}_0, \tag{2.1b}$$

$$x_{k+1} = A_k x_k + B_k u_k, \quad k = 0, \ldots, N{-}1, \tag{2.1c}$$

$$C_{k,x} x_k + C_{k,u} u_k \leqslant 0, \quad k = 0, \ldots, N{-}1, \tag{2.1d}$$

$$C_N x_N \leqslant 0, \tag{2.1e}$$

In general, the Hessian H of QP (2.1) might be indefinite, for example in case of an exact Hessian based SQP method to solve NLP (1.16). However, this does not necessarily prevent the problem from being convex and therefore the solution of QP (2.1) to be global and unique. First, we present a general framework to recover this underlying convexity.

For the sake of clarity of the exposition, we omit the inequality constraints from QP (2.1) and refer to Section 2.3 for a discussion on how to deal with them within the proposed convexification approach. Without inequality constraints, we can write QP (2.1a)-(2.1b) in a more compact form, as follows.

Definition 2.1 (Equality constrained QP)**.**

$$\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2} w^\top H w \tag{2.2a}$$

$$\text{subject to} \quad Gw + g = 0, \tag{2.2b}$$

with constraint matrix $G \in \mathbb{R}^{m_{\text{eq}} \times n}$ of full rank m_{eq} with $m_{\text{eq}} \leqslant n$. Note that the linear independence constraint qualification (LICQ) requires full rank of G [Nocedal and Wright, 2006]. Furthermore, we have a symmetric but possibly indefinite Hessian matrix $H \in \mathbb{S}^n$, where we define the space of symmetric matrices of size n as follows:

$$\mathbb{S}^n := \{X \in \mathbb{R}^{n \times n} \mid X = X^\top\}. \tag{2.3}$$

In case of a structured equality constrained QP (2.1a)-(2.1b), we have that $n = (N+1) \cdot n_x + N \cdot n_u$, $m_{\text{eq}} = (N+1) \cdot n_x$ and we choose the following ordering of the optimization variables: $w^\top = [x_0^\top, u_0^\top, \ldots, x_N^\top]$. Let us define the following, similar to Definition 1.6.

Definition 2.2 (Range space and null space basis of G)**.** *Consider QP* (2.2) *and assume LICQ holds. We let* $Z \in \mathbb{R}^{n\times(n-m_{\text{eq}})}$ *denote a null space basis with corresponding range space basis* $Y \in \mathbb{R}^{n\times m_{\text{eq}}}$ *of* G *that satisfy the following:*

$$GZ = 0, \tag{2.4a}$$

$$(Y|Z)^\top (Y|Z) = I. \tag{2.4b}$$

Note that for any such Z*, it holds that* $\text{span}(Z) = \text{null}(G)$*.*

2.2.1 A characterization of convexity

Using Definitions 2.1-2.2, we can state some interesting properties of QP (2.2) with regard to convexity of the reduced Hessian. The following theorem is a well-known result, of which a proof is presented in [Nocedal and Wright, 2006].

Theorem 2.1. *Consider QP* (2.2) *and Definition 2.2, assuming that LICQ holds. Then QP* (2.2) *has a unique global optimum if and only if the reduced Hessian is positive definite, i.e.*

$$Z^\top H Z \succ 0. \tag{2.5}$$

We note that the reduced Hessian being positive definite corresponds to the second order sufficient condition (SOSC) for optimality, as defined in [Nocedal and Wright, 2006]. A well-known fact related to this SOSC is stated in the following theorem (for a proof, see e.g. [Nocedal and Wright, 2006]).

Theorem 2.2. *Consider QP* (2.2) *with arbitrary* $H \in \mathbb{S}^n$*, Definition 2.2 and assume that LICQ holds. Then*

$$Z^\top H Z \succ 0 \Longleftrightarrow \exists \gamma \in \mathbb{R} : H + \gamma G^\top G \succ 0. \tag{2.6}$$

Theorem 2.2 is at the basis of the augmented Lagrangian method for optimization, where one typically calls $\gamma > 0$ the *quadratic penalty parameter*. By choosing γ large enough one can always create a positive definite Hessian at points satisfying SOSC. Note that $H + \gamma G^\top G$ destroys the sparsity pattern present in the original Hessian H, but an actual implementation would solve an augmented primal-dual system with the correct sparsity in the Hessian, e.g. as in [Gill and Robinson, 2013]. Theorem 2.2 can be generalized as follows:

Theorem 2.3 (Revealing Convexity)**.** *Consider QP* (2.2)*, Definition 2.2, and assume LICQ holds. The reduced Hessian satisfies*

$$Z^\top H Z \succ 0 \tag{2.7a}$$

if and only if there exists a symmetric matrix $U \in \mathbb{S}^n$ *with*

$$Z^\top U Z = 0 \tag{2.7b}$$

such that

$$H + U \succ 0. \tag{2.7c}$$

Proof. From (2.7b), (2.7c) and the fact that Z is of full rank, (2.7a) directly follows. In order to prove the converse, let us introduce a change of basis, where the new basis is formed by $(Y|Z)$. Doing so, matrix inequality (2.7c) is equivalent to $(Y|Z)^\top (H+U)(Y|Z) \succ 0$. This again, due to (2.7b) and the Schur complement lemma [Haynsworth, 1968], is equivalent to

$$Z^\top H Z \succ 0, \tag{2.8}$$

$$Y^\top (H+U) Y \succ Y^\top (H+U) Z \cdot (Z^\top H Z)^{-1} \cdot Z^\top (H+U) Y. \tag{2.9}$$

Thus, we need to show that there always exists a matrix U satisfying (2.7b) and (2.9).

Using the same change of basis as above for U and using (2.7b), it holds that

$$U = YKY^\top + YMZ^\top + ZM^\top Y^\top, \tag{2.10}$$

with $K \in \mathbb{S}^{m_{\text{eq}}}$ and $M \in \mathbb{R}^{m_{\text{eq}} \times (n - m_{\text{eq}})}$. It directly follows that $Z^\top U Z = 0$. Statement (2.9) can then be written as

$$K \succ -Y^\top H Y + (Y^\top H Z + M) \cdot (Z^\top H Z)^{-1} \cdot (Y^\top H Z + M)^\top. \tag{2.11}$$

As K appears individually on the left hand side, for given H, M, there always exists a matrix K such that (2.11) holds. Thus, there always exists a matrix U satisfying (2.9). □

The proof of Theorem 2.2 is obtained by choosing $M = 0$, i.e. $U = YKY^\top$, and observing that Y is a basis of the range space of $G^\top$. It is interesting to remark that the introduction of matrix U in order to obtain a certificate for second order optimality bears some similarity in spirit to the introduction of Lagrange multipliers in order to obtain a certificate of first order optimality.

The next section regards a different special case of Theorem 2.3, where we impose an OCP structure on U, which cannot be obtained by $U = \gamma G^\top G$ or its generalization $U = G^\top \Gamma G$. *Convexification* is our name for the process of finding such a matrix U, which we call *structure-preserving convexification* if U has the same sparsity structure as the original Hessian matrix H.

2.2.2 Structure-preserving convexification

The convexification algorithm presented in this section exploits the convexity of the reduced Hessian in order to compute a modified quadratic cost matrix $\widetilde{H} \in \mathbb{S}_{\text{OCP}}^{N,n_x,n_u}$ which is positive definite and has the same sparsity pattern as H; here, we use the following definition for the space of symmetric block-diagonal matrices with OCP structure:

$$\mathbb{S}_{\text{OCP}}^{N,n_x,n_u} := \{X \in \mathbb{S}^{N(n_x+n_u)+n_x} \mid X = \operatorname{diag}(X_0, \dots, X_N),$$

$$X_k \in \mathbb{S}^{n_x+n_u}, k = 0, \dots, N-1, X_N \in \mathbb{S}^{n_x}\}.$$

The convexification can be performed by using the structure of the equality constraints. The resulting modified QP($\widetilde{H}$) has two important properties: (a) the primal solutions of QP(H) and QP($\widetilde{H}$) are equal; (b) $\widetilde{H}$ is positive definite if and only if the reduced Hessian of QP(H) is positive definite.

In the following, we establish property (a) in Theorem 2.4. Afterwards, we present the convexification procedure in detail and state (b) in Theorem 2.5, which is the main result of this section. Furthermore, we propose a procedure to recover the dual solution of QP(H) from the solution of QP($\widetilde{H}$) in Section 2.2.6. To conclude this section, we present a tutorial example.

Throughout this section, we use the following definitions (see the equality constrained QP (2.1a)-(2.1b)):

$$G := \begin{bmatrix} -I & & & & & \\ A_0 & B_0 & -I & & & \\ & & \ddots & \ddots & \ddots & \\ & & & A_{N-1} & B_{N-1} & -I \end{bmatrix}, \quad g := \begin{bmatrix} \overline{x}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.12}$$

such that the dynamic equalities can be written as $G\,w + g = 0$, where $w^\top = [x_0^\top, u_0^\top, \dots, x_N^\top]$. Furthermore, we will use matrix Z as in Definition 2.2.

2.2.3 Transfer of cost between stages

The N stages of QP (2.1a)-(2.1b) are coupled by the dynamic constraints $x_{k+1} = A_k x_k + B_k u_k$. These constraints are used to transfer cost between consecutive stages of the problem without changing its primal solution. We introduce the *transferred cost* as follows:

$$\overline{q}_k(x) := x^\top \overline{Q}_k x, \qquad k = 0, \dots, N, \tag{2.13}$$

for given matrices $\overline{Q}_k, \in \mathbb{S}^{n_x}, k = 0, \dots, N$. Then, the stage cost l_k and the modified cost L_k are defined as follows (see (2.1a)-(2.1b)):

$$l_k(x_k, u_k) := \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top H_k \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \quad k = 0, \dots, N-1, \tag{2.14a}$$

$$L_k(x_k, u_k) := l_k(x_k, u_k) - \overline{q}_k(x_k) + \overline{q}_{k+1}(A_k x_k + B_k u_k), \tag{2.14b}$$

$$= \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \widetilde{H}_k \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \quad k = 0, \dots, N-1, \tag{2.14c}$$

in which the modified cost L_k is calculated by adding the transferred cost from the next stage $\overline{q}_{k+1}$ to the stage cost l_k and subtracting the cost $\overline{q}_k$, which is the cost to transfer to the previous stage. This yields a Hessian $\widetilde{H} = \operatorname{diag}(\widetilde{H}_0, \dots, \widetilde{H}_{N-1}, \widetilde{Q}_N)$, where $\widetilde{Q}_N := \widehat{Q}_N - \overline{Q}_N$, and we recall that $\widehat{Q} := Q_N$, which allows us to state the following theorem.

Theorem 2.4 (Equality of primal QP solutions)**.** *Consider the equality constrained QP* (2.1a)-(2.1b) *and assume that* $Z^\top H Z \succ 0$. *Then, the primal solutions of* $\mathrm{QP}(H)$ *and* $\mathrm{QP}(\widetilde{H})$ *are equal.*

Proof. By assumption $Z^\top H Z \succ 0$, therefore $\mathrm{QP}(H)$ has a unique global minimum, by Theorem 2.1. The cost function of $\mathrm{QP}(\widetilde{H})$ satisfies

$$\begin{aligned}
&\sum_{k=0}^{N-1} L_k(x_k, u_k) + x_N^\top \widetilde{Q}_N x_N \\
&\qquad = \sum_{k=0}^{N-1} l_k(x_k, u_k) - \overline{q}_k(x_k) + \overline{q}_{k+1}(A_k x_k + B_k u_k) + x_N^\top \widetilde{Q}_N x_N, \\
&\qquad = \sum_{k=0}^{N-1} l_k(x_k, u_k) - \overline{q}_0(x_0) + \overline{q}_N(x_N) + x_N^\top \widetilde{Q}_N x_N \\
&\qquad = \sum_{k=0}^{N-1} l_k(x_k, u_k) + x_N^\top \widehat{Q}_N x_N - \overline{q}_0(\overline{x}_0), \\
&\qquad = \sum_{k=0}^{N-1} l_k(x_k, u_k) + x_N^\top Q_N x_N - \overline{q}_0(\overline{x}_0),
\end{aligned}$$

which is equal to the cost function of $\mathrm{QP}(H)$, up to the constant term $-\overline{q}_0(\overline{x}_0)$. It follows, because the constraints of $\mathrm{QP}(H)$ and $\mathrm{QP}(\widetilde{H})$ are identical, that the primal solutions of both problems coincide. $\square$

Note that we can write $\widetilde{H} = H + U(\overline{Q})$ where we let matrix $U(\overline{Q}) := \operatorname{diag}(U_0, \ldots, U_N)$, with $\overline{Q} := \operatorname{diag}(\overline{Q}_0, \ldots, \overline{Q}_N)$ and the quantities $U_k, k = 0, \ldots, N$ are defined as follows:

$$U_k := \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k - \overline{Q}_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix}, \quad U_N := -\overline{Q}_N. \tag{2.15}$$

Matrix U can be computed by Algorithm 2.1. Note that $H, U \in \mathbb{S}_{\mathrm{OCP}}^{N,n_x,n_u}$ so that $\widetilde{H} \in \mathbb{S}_{\mathrm{OCP}}^{N,n_x,n_u}$ also.

Algorithm 2.1 $U(\overline{Q})$

```
Require: $\overline{Q}$
Ensure: $U$
$U_N := -\overline{Q}_N$
for $k = N-1, \ldots, 0$ do
    $U_k := \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k - \overline{Q}_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix}$
end for
$U := \operatorname{diag}(U_0, \ldots, U_N)$
```

Theorem 2.4 states that transferring cost as in (2.14) does not alter the primal solution. In the following lemma we prove that also the reduced Hessian is invariant under cost transfer (2.14).

Lemma 2.1. *Consider the equality constrained QP* (2.1a)-(2.1b)*, Z from Definition 2.2 and U from* (2.15) *and assume LICQ is satisfied. Then, for any $\overline{Q}$, it holds that $Z^\top U(\overline{Q}) Z = 0$.*

Proof. Using Equations (2.15) and (2.12), we can rewrite U as

$$\begin{aligned} U(\overline{Q}) &= (G + \Sigma)^\top \overline{Q} (G + \Sigma) - \Sigma^\top \overline{Q} \Sigma, \\ &= G^\top \overline{Q} G + G^\top \overline{Q} \Sigma + \Sigma^\top \overline{Q} G, \end{aligned} \tag{2.16}$$

where we introduced

$$\Sigma := \begin{bmatrix} I & 0 & & & & \\ & & I & 0 & & \\ & & & & \ddots & \\ & & & & & I \end{bmatrix}, \tag{2.17}$$

with $I \in \mathbb{R}^{n_x \times n_x}$ so that $\Sigma \in \mathbb{R}^{m_{\mathrm{eq}} \times n}$, where $n = (N+1) \cdot n_x + N \cdot n_u$, $m_{\mathrm{eq}} = (N+1) \cdot n_x$, as before. By definition $GZ = 0$, therefore it follows directly that $Z^\top U Z = 0$. ☐

To summarize, we give an explicit form for the blocks of the diagonal block matrix $\widetilde{H}$. For the last stage it holds that:

$$\widetilde{Q}_N = \widehat{Q}_N + U_N = \widehat{Q}_N - \overline{Q}_N. \tag{2.18}$$

For the rest of the stages, we go in reverse order from $k = N - 1$ to $k = 0$ and first compute the intermediate quantities

$$\begin{bmatrix} \widehat{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} := \begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix} + \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix}, \tag{2.19}$$

and then set

$$\widetilde{Q}_k = \widehat{Q}_k - \overline{Q}_k, \tag{2.20}$$

$$\widetilde{H}_k = H_k + U_k = \begin{bmatrix} \widetilde{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix}, \qquad k = 0, \dots, N-1. \tag{2.21}$$

Moreover, we remark the resemblance of Equations (2.16) and (2.10), so that we can write the following expressions for K, M:

$$K = Y^\top U Y = Y^\top \big(G^\top \overline{Q} G + G^\top \overline{Q} \Sigma + \Sigma^\top \overline{Q} G \big) Y,$$

$$M = Z^\top U Y = Z^\top \Sigma^\top \overline{Q} G Y.$$

Please note that for arbitrary $\overline{Q}$, the cost transfer operation presented in this section generally results in indefinite $\widetilde{H}$. In the next section, we will establish Theorem 2.5, which states that we can find a $U \in \mathbb{S}_{\mathrm{OCP}}^{N, n_x, n_u}$ with corresponding positive definite $\widetilde{H}$, and we present an algorithm to compute it.

2.2.4 The structure-preserving convexification algorithm

We propose a structure-preserving convexification procedure, which is built on Equations (2.18)-(2.21). It computes $\widetilde{H} = H + U(\overline{Q})$ which can be shown to be positive definite, where $U(\overline{Q})$ is defined as in (2.15) based on a careful choice of $\overline{Q}$.

The procedure, shown in Algorithm 2.2, proceeds as follows: starting from the last stage, we choose a positive definite matrix $\widetilde{Q}_N = \delta I$, with $\delta > 0$ a small constant, such that $\overline{Q}_N = \widehat{Q}_N - \delta I$ (lines 2 and 3), and we use this matrix to transfer the cost $x_N^\top \overline{Q}_N x_N$ to the previous stage. The updated quantities $\widehat{Q}_{N-1}, \widehat{S}_{N-1}, \widehat{R}_{N-1}$ are calculated according to (2.19) in line 5. We use the Schur complement lemma [Haynsworth, 1968] in line 6 of the algorithm to ensure that $\widetilde{H}_{N-1} \succ 0$. Next, we compute $\overline{Q}_{N-1} = \widehat{Q}_{N-1} - \widetilde{Q}_{N-1}$ and we repeat steps 5-8 until we arrive at the first stage of the problem.

By inspection of Algorithm 2.2, we have that the computational complexity scales linearly with the horizon length, i.e. it is $\mathcal{O}(N)$. We include $\overline{Q}(\delta)$ and $\widehat{R}(\delta) := \operatorname{diag}(\widehat{R}_0, \ldots, \widehat{R}_{N-1})$ in the output of the algorithm for convenience.

Algorithm 2.2 Structure-Preserving Convexification: equality constrained case

```
Require: $H, \delta$
Ensure: $\overline{Q}(\delta), \widetilde{H}(\delta), \widehat{R}(\delta)$
$\widehat{Q}_N = Q_N$
$\widetilde{Q}_N = \delta I$
$\overline{Q}_N = \widehat{Q}_N - \widetilde{Q}_N$
for $k = N-1, \ldots, 0$ do
    $\begin{bmatrix} \widehat{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} = \begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix} + \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix}$
    $\widetilde{Q}_k := \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k + \delta I$
    $\widetilde{H}_k := \begin{bmatrix} \widetilde{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix}$
    $\overline{Q}_k = \widehat{Q}_k - \widetilde{Q}_k$
end for
$\widetilde{H} := \operatorname{diag}(\widetilde{H}_0, \ldots, \widetilde{Q}_N)$
```

In Theorem 2.5, we show that Algorithm 2.2 indeed produces a positive definite $\widetilde{H}$, given a sufficiently small value for δ, if and only if the reduced Hessian is positive definite. Lemmas 2.2 and 2.3, presented next, help us to prove this result.

Lemma 2.2. *Consider the equality constrained QP* (2.1a)-(2.1b) *with OCP structure and Definition 2.2, assuming LICQ, $Z^\top H Z \succ 0$ hold. Then, Algorithm* (2.2) *with $\delta = 0$ delivers positive definite $\widehat{R}(0) \succ 0$ and positive semi-definite $\widetilde{H}(0) \succeq 0$.*

Proof. Since $\delta = 0$, Algorithm 2.2 starts with $\overline{Q}_N = \widehat{Q}_N = Q_N$. Following a dynamic programming argument in order to solve QP (2.1a)-(2.1b), we have at each stage the following problem, with x_k fixed:

$$\underset{u_k}{\text{minimize}} \quad \frac{1}{2} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \frac{1}{2} x_{k+1}^\top \overline{Q}_{k+1} x_{k+1} \tag{2.22a}$$

$$\text{subject to} \quad x_{k+1} = A_k x_k + B_k u_k, \tag{2.22b}$$

which is equivalent to

$$\underset{u_k}{\text{minimize}} \quad \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \begin{bmatrix} Q_k + A_k^\top \overline{Q}_{k+1} A_k & S_k^\top + A_k^\top \overline{Q}_{k+1} B_k \\ S_k + B_k^\top \overline{Q}_{k+1} A_k & R_k + B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}. \tag{2.23}$$

By assumption, the reduced Hessian of the full QP (2.1a)-(2.1b) is positive definite, which, by Theorem 2.1, entails that the minimum of the QP is unique. Dynamic programming yields the same *unique* solution for each u_k, which in turn implies that the Hessian of (2.23) must be positive definite as well. This amounts to $(R_k + B_k^\top \overline{Q}_{k+1} B_k) = \widehat{R}_k \succ 0$, for $k = 0, \ldots, N-1$. From the Schur complement lemma, we have that if $\widehat{R}_k \succ 0$,

$$\widetilde{Q}_k - \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k \succeq 0 \iff \begin{bmatrix} \widetilde{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} \succeq 0. \tag{2.24}$$

With $\delta = 0$, from Algorithm 2.2 it follows that $\widetilde{Q}_k = \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k$, such that the left hand side of (2.24) holds. This entails that the Hessian blocks $\widetilde{H}_k \succeq 0$, such that, together with $\widetilde{Q}_N = 0$, it holds that $\widetilde{H} \succeq 0$. ☐

In the following, we regard matrices $\widehat{R}_k$ from Algorithm 2 as the map $\widehat{R}(\delta)$ and we recall that $\widehat{R}(\delta) := \operatorname{diag}(\widehat{R}_0, \ldots, \widehat{R}_{N-1})$. We use this map in Lemma 2.3, which will help us prove Theorem 2.5.

Lemma 2.3. *Consider QP* (2.1a)-(2.1b) *and assume that* $Z^\top H Z \succ 0$ *holds. Then there exists a value* $\delta > 0$ *such that Algorithm 2.2 computes a positive definite matrix* $\widehat{R}(\delta) \succ 0$.

Proof. Consider the map $\widehat{R}(\delta)$, implicitly defined by Algorithm 2.2. By Lemma 2.2 it holds that $\widehat{R}(0) \succ 0$. Furthermore, δ enters linearly in the equations of Algorithm 2.2 and each step of the algorithm is continuous. This includes line 6, where the inverse of $\widehat{R}_k$ appears, which is well-defined and continuous as long as $\widehat{R}_k(\delta)$ remains positive definite, which is true at $\delta = 0$. As a consequence, the map $\widehat{R}(\delta)$ is continuous at the origin. It follows that there exists a value $\delta > 0$ such that $\widehat{R}_k \succ 0, k = 0, \ldots, N-1$. ☐

We conclude this section by establishing our main result.

Theorem 2.5. *Consider QP* (2.1a)-(2.1b)*, Definition 2.2 and assume LICQ holds. Then* $Z^\top H Z \succ 0 \iff \exists \delta > 0,$ *such that* $\widetilde{H}(\delta) \succ 0$ *as defined by Algorithm 2.2.*

Proof. Assume there exists some $\delta > 0$ such that $\widetilde{H} \succ 0$. Then $Z^\top H Z \succ 0$ follows from $\widetilde{H} = H + U$ and Lemma 2.1. The converse is proven as follows. From Algorithm 2.2 we have that $\widetilde{Q}_k - \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k = \delta I \succ 0$. By Lemma 2.3, $\exists \delta > 0$ such that $\widehat{R}_k \succ 0$. Then, from the Schur complement lemma, it follows that

$$\widetilde{H}_k := \begin{bmatrix} \widetilde{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} \succ 0. \tag{2.25}$$

As $\widetilde{Q}_N = \delta I \succ 0$, it follows that $\widetilde{H} = \operatorname{diag}(\widetilde{H}_0, \ldots, \widetilde{H}_{N-1}, \widetilde{Q}_N) \succ 0$. □

2.2.5 Connection with the discrete Riccati equation

We establish next an interesting relation between Algorithm 2.2 and the discrete-time Riccati equation. This result is not needed in the remainder of this chapter, but is given for completeness.

Definition 2.3 (DRE). *For a discrete linear time varying system $x_{k+1} = A_k x_k + B_k u_k$, the discrete-time Riccati equation starting with $X_N := \overline{Q}_N$ iterates backwards from $k = N-1$ to $k = 0$ by computing*

$$\begin{aligned} X_k &= Q_k + A_k^\top X_{k+1} A_k \\ &\quad - (S_k^\top + A_k^\top X_{k+1} B_k)(R_k + B_k^\top X_{k+1} B_k)^{-1}(S_k + B_k^\top X_{k+1} A_k). \end{aligned} \tag{2.26}$$

We call matrices X_k the cost-to-go *matrices for $k = 0, \ldots, N$.*

Lemma 2.4. *Consider QP* (2.1a)-(2.1b)*, Algorithm 2.2 and assume $Z^\top H Z \succ 0$, with Z as in Definition 2.2. If $\delta = 0$, then the matrices $\overline{Q}_0, \ldots, \overline{Q}_N$ in the output of Algorithm 2.2 are equal to the cost-to-go matrices $X_0, \ldots, X_N$ computed with the DRE as defined above.*

Proof. For stage N, $X_N = \overline{Q}_N$ by definition. For $k = 0, \ldots, N-1$, from Definition 2.3, we have that

$$\begin{aligned} X_k &= Q_k + A_k^\top X_{k+1} A_k \\ &\quad - (S_k^\top + A_k^\top X_{k+1} B_k)(R_k + B_k^\top X_{k+1} B_k)^{-1}(S_k + B_k^\top X_{k+1} A_k), \end{aligned} \tag{2.27}$$

and, if we replace X_{k+1} by $\overline{Q}_{k+1}$,

$$X_k = \widehat{Q}_k - \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k, \tag{2.28}$$

$$= \overline{Q}_k, \tag{2.29}$$

where we used $\widehat{Q}_k, \widehat{S}_k, \widehat{R}_k$ as in (2.19) for ease of notation, and (2.29) follows from Algorithm 2.2, where $\overline{Q}_k = \widehat{Q}_k - \widetilde{Q}_k$, which is equivalent to the right hand side of (2.28) for $\delta = 0$. □

2.2.6 Recovering the dual solution of the original QP

We propose a procedure to recover the dual solution of QP(H) from its primal solution. We define the Lagrangian of the equality constrained QP (2.1a)-(2.1b) as follows:

$$\begin{aligned}\mathcal{L}(w,\lambda) = \frac{1}{2}\sum_{k=0}^{N-1} x_k^\top Q_k x_k + 2x_k^\top S_k^\top u_k + u_k^\top R_k u_k + \frac{1}{2}x_N^\top Q_N x_N \\ + \sum_{k=0}^{N-1} \lambda_{k+1}^\top \big(A_k x_k + B_k u_k - x_{k+1}\big) + \lambda_0^\top(\overline{x}_0 - x_0),\end{aligned} \tag{2.30}$$

with $\lambda^\top = [\lambda_0^\top, \dots, \lambda_N^\top], \lambda_k \in \mathbb{R}^{n_x}$. We can obtain the Lagrange multipliers by computing the partial derivatives of the Lagrangian with respect to x_k, which should equal zero by the necessary conditions for optimality [Nocedal and Wright, 2006]. The derivation is shown below, a procedure to compute λ is stated in Algorithm 2.3.

$$0 = \frac{\partial \mathcal{L}(w,\lambda)}{\partial x_k}^\top, \qquad k = 0, \dots, N-1 \tag{2.31}$$

$$= Q_k x_k + S_k^\top u_k + A_k^\top \lambda_{k+1} - \lambda_k, \tag{2.32}$$

where additionally

$$0 = \frac{\partial \mathcal{L}(w,\lambda)}{\partial x_N}^\top = Q_N x_N - \lambda_N. \tag{2.33}$$

Algorithm 2.3 Recovery of Lagrange multipliers: equality constrained case

```
Require: $w$
Ensure: $\lambda$
$\lambda_N = Q_N x_N$
for $k = N-1, \dots, 0$ do
    $\lambda_k \leftarrow Q_k x_k + S_k^\top u_k + A_k^\top \lambda_{k+1}$
end for
```

2.2.7 A tutorial example

To illustrate our convexification method with a simple example, we regard the following one-stage OCP:

$$\underset{x_0,u_0,x_1}{\text{minimize}} \quad x_0^2 - \frac{1}{2}u_0^2 + x_1^2 + x_1^4 \tag{2.34a}$$

$$\text{subject to} \quad x_1 = x_0 + u_0 \tag{2.34b}$$

$$x_0 = \overline{x}_0. \tag{2.34c}$$

As the term x_1^4 makes this problem an NLP, we solve it by using an SQP method with exact Hessian. We set $\overline{x}_0 = 1$.

We define $w := [x_0, u_0, x_1]^\top$. Optimization problem (2.34) has a global minimizer at $w^\star = [1, -3/2, -1/2]^\top$. The Hessian of the Lagrangian at the solution is equal to

$$\nabla_w^2 \mathcal{L}(w^\star, \lambda^\star) = \nabla^2 f(w^\star) = \operatorname{diag}(2, -1, 5) \nsucc 0. \tag{2.35}$$

We define

$$G := \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & -1 \end{bmatrix}, \quad Z := \begin{bmatrix} 0 \\ \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}, \tag{2.36}$$

such that $GZ = 0$ and $Z^\top Z = I$. The reduced Hessian at the solution $w^\star$ then reads as $Z^\top \nabla_w^2 \mathcal{L}(w^\star, \lambda^\star) Z = 2 > 0$.

Applying our convexification with sufficiently small δ therefore results in strictly convex QP subproblems, when the SQP method is sufficiently close to the minimizer.

We compare the convergence of Newton's method which employs either the proposed convexification method or the regularization methods defined in (2.37), called *project* and *mirror*, which do, instead, modify the reduced Hessian. Note that these regularizations fulfill all three properties of the regularizations that we desire, as mentioned in the introduction: they yield a fully positive definite Hessian, they are independent of the QP solver and they preserve the OCP structure.

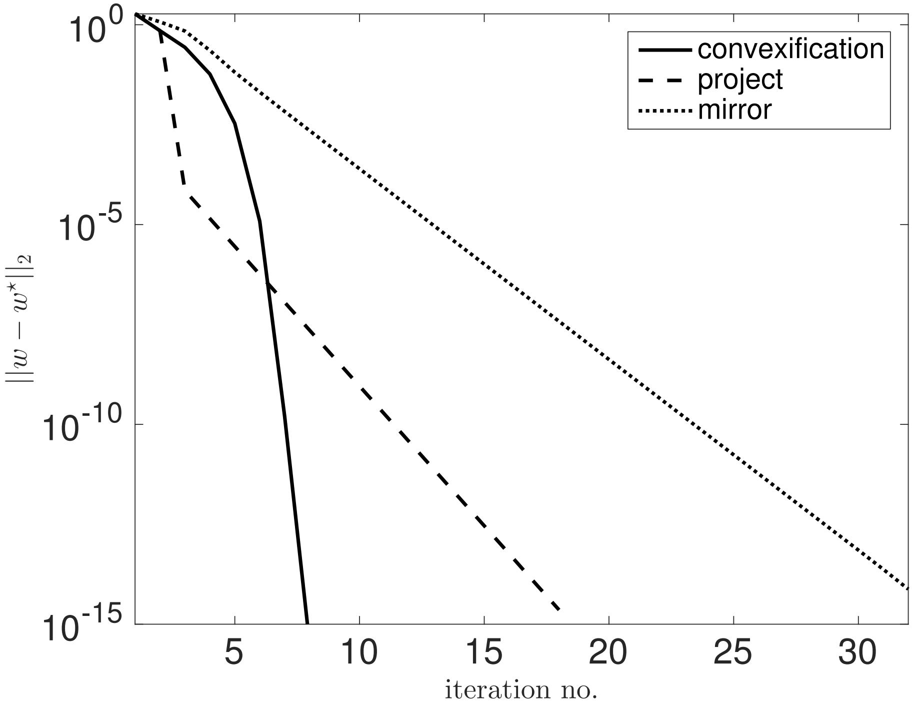


Figure 2.1: Convergence for SQP algorithm with three different regularization methods. On the vertical axis we plot the distance to the global solution $w^\star$. The Hessian obtained by Algorithm 2.2 with $\delta = 10^{-4}$ enables quadratic convergence of the exact Newton method, in contrast to the alternative regularization methods with $\epsilon = 10^{-4}$.

With $V_k D_k V_k^{-1}$ the eigenvalue decomposition of the Hessian block H_k, $k = 0, \ldots, N$, these two regularizations are defined as follows:

$$\text{project}(H_k, \epsilon) := V_k \left[\max(\epsilon, D_k)\right] V_k^{-1}, \tag{2.37a}$$

$$\text{mirror}(H_k, \epsilon) := V_k \left[\max(\epsilon, \text{abs}(D_k))\right] V_k^{-1}, \tag{2.37b}$$

where abs$(\cdot)$ is the operator which takes the element-wise absolute value and $\epsilon > 0$.

In Figure 2.1, we compare convergence of the SQP method for the convexification method and the two alternative regularizations in (2.37). We can observe from the figure that Newton's method which employs the proposed convexification procedure exhibits locally quadratic convergence to the solution, as it finds the correct primal and dual solution of the QP in each iteration. The other regularization methods result in linear convergence.

2.3 Inequality constrained optimization

In the previous section, we analyzed the case without inequality constraints. In this section, we analyze the general case and present a generalized version of Algorithm 2.2.

2.3.1 Revealing convexity under active inequalities

Consider solving NLP (1.2) with an exact Hessian SQP method, starting from some point $\overline{v} = (\overline{w}, \overline{\lambda}, \overline{\mu})$. As discussed in Chapter 1, see (1.9), this gives rise to QP subproblems of the form:

$$\mathrm{QP}_{\mathrm{SQP}}(\overline{w}, H) := \underset{\Delta w \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\Delta w^\top H \, \Delta w + \nabla f(\overline{w})^\top \Delta w \tag{2.38a}$$

$$\text{subject to} \quad 0 = g(\overline{w}) + \frac{\partial g}{\partial w}(\overline{w})\Delta w, \tag{2.38b}$$

$$0 \geqslant c(\overline{w}) + \frac{\partial c}{\partial w}(\overline{w})\Delta w, \tag{2.38c}$$

with primal solution $\Delta w^\star$ and optimal active set $\mathcal{A}^\star_{\mathrm{QP}}(\overline{w}, \Delta w^\star, H)$.

Remark 2.1. *Since $\nabla^2_w \mathcal{L}(\overline{v})$ might be indefinite, there could be multiple solutions to* $\mathrm{QP}_{\mathrm{SQP}}(\overline{w}, \nabla^2_w \mathcal{L}(\overline{v}))$*. In the following, we will assume, as e.g. in [Gould and Robinson, 2010; Robinson, 1974], that $\Delta w^\star$ is the* minimum-norm *solution.*

For a *minimum-norm* solution of $\mathrm{QP}_{\mathrm{SQP}}(\overline{w}, \nabla^2_w \mathcal{L}(\overline{v}))$, the following lemma from [Nocedal and Wright, 2006] holds.

Lemma 2.5. *Suppose that $v^\star$ is a regular solution of NLP* (1.2)*. Then if $\overline{v} := (\overline{w}, \overline{\lambda}, \overline{\mu})$ is sufficiently close to $v^\star$, the minimum-norm solution of* $\mathrm{QP}_{\mathrm{SQP}}(\overline{w}, \nabla^2_w \mathcal{L}(\overline{v}))$ *has an active set* $\mathcal{A}^\star_{\mathrm{QP}}(\overline{w}, \Delta w^\star, \nabla^2_w \mathcal{L}(\overline{v}))$ *that is the same as the active set $\mathcal{A}(w^\star)$ of NLP* (1.2) *at $v^\star$.*

Suppose we are at a point $(\overline{w}, \overline{\lambda}, \overline{\mu})$ sufficiently close to a solution of the NLP. Then Lemma 2.5 serves as a motivation to define a QP, which is the same as QP (2.38), but with the active inequalities replaced by equalities. We will use the following shorthands $G := \frac{\partial g}{\partial w}(\overline{w}), G_{\text{act}} := \frac{\partial c_i}{\partial w}(\overline{w}), i \in \mathcal{A}(w^\star)$, and m_{act} denotes the number of active constraints. For ease of notation, we omit the dependence of G, G_{act} on $\overline{w}$, as it is constant within one QP subproblem.

Definition 2.4 (QP with fixed active set)**.**

$$\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2} w^\top H w \tag{2.39a}$$

$$\text{subject to} \quad \widetilde{G} w + \widetilde{g} = 0, \tag{2.39b}$$

with $\widetilde{G} := \begin{bmatrix} G \\ G_{\text{act}} \end{bmatrix}$, $G \in \mathbb{R}^{m_{\text{eq}} \times n}, G_{\text{act}} \in \mathbb{R}^{m_{\text{act}} \times n}$ and $\widetilde{g} := \begin{bmatrix} g \\ g_{\text{act}} \end{bmatrix}$, $g \in \mathbb{R}^{m_{\text{eq}}}$, $g_{\text{act}} \in \mathbb{R}^{m_{\text{act}}}$. Note that we omitted the gradient term in the objective from (2.38), for ease of notation, by performing the same transformation of variables as in [Frison et al., 2014]. Additionally, we introduce the following matrices:

Definition 2.5 (Null space of equalities and active inequalities)**.** *Consider* $\widetilde{G}$ *as in QP* (2.39)*. We define* $\widetilde{Z} \in \mathbb{R}^{n \times (n - m_{\text{eq}} - m_{\text{act}})}$*, such that the following holds:*

$$\widetilde{G}\widetilde{Z} = 0, \quad \widetilde{Z}^\top \widetilde{Z} = I. \tag{2.40}$$

Matrix $\widetilde{Z}$ *is a basis for the null space of* $\widetilde{G}$*. The null space of* G *comprises* $\widetilde{Z}$ *but is possibly larger. We complement* $\widetilde{Z}$ *with* $Z_c \in \mathbb{R}^{n \times m_{\text{act}}}$*, and introduce* $Z \in \mathbb{R}^{n \times (n - m_{\text{eq}})}$ *as a basis for the null space of* G*, as follows:*

$$Z = (Z_c | \widetilde{Z}), \quad GZ = 0, \quad Z^\top Z = I. \tag{2.41}$$

From now on we call $\widetilde{Z}^\top H \widetilde{Z}$ the reduced Hessian. Furthermore, note that the above definition of Z is compatible with Definition 2.2.

Using Definition 2.5, we establish an extension of Theorem 2.3 for the case of active inequality constraints in Theorem 2.6, after proving the following lemma.

Lemma 2.6. *Consider QP* (2.39)*, Definition 2.5 and assume LICQ holds. We then have the following equivalence:*

$$\widetilde{Z}^\top H \widetilde{Z} \succ 0 \iff \exists \Gamma \in \mathbb{S}^{m_{\text{act}}} : Z^\top (H + G_{\text{act}}^\top \Gamma G_{\text{act}}) Z \succ 0. \tag{2.42}$$

Proof. The proof follows a similar argument as the proof of Theorem 2.3. We consider $H + G_{\text{act}}^\top \Gamma G_{\text{act}}$ in the basis $Z = (Z_c | \widetilde{Z})$, see (2.41). Applying the Schur complement lemma yields the following conditions:

$$\widetilde{Z}^\top H \widetilde{Z} \succ 0, \tag{2.43}$$

$$Z_c^\top (H + G_{\text{act}}^\top \Gamma G_{\text{act}}) Z_c \succ Z_c^\top H \widetilde{Z} \cdot (\widetilde{Z}^\top H \widetilde{Z})^{-1} \cdot \widetilde{Z}^\top H Z_c, \tag{2.44}$$

where we used the fact that $G_{\text{act}}\widetilde{Z} = 0$. The first inequality is the same as the left hand side of (2.42). Using (2.41), and due to LICQ and the fact that Z_c is orthogonal to the null space of $\widetilde{G}$ and part of the null space of G, it holds that $G_{\text{act}}Z_c$ is of full rank and therefore invertible. Thus, (2.44) becomes

$$\Gamma \succ (Z_c^\top G_{\text{act}}^\top)^{-1}(Z_c^\top H\widetilde{Z} \cdot (\widetilde{Z}^\top H\widetilde{Z})^{-1} \cdot \widetilde{Z}^\top HZ_c - Z_c^\top HZ_c)(G_{\text{act}}Z_c)^{-1}. \quad (2.45)$$

As Γ appears solely on the left side of the inequality, there always exists a Γ such that condition (2.44) is met. □

Theorem 2.6. *Consider QP* (2.39)*, Definition 2.5 and assume LICQ holds. It then holds that*

$$\widetilde{Z}^\top H\widetilde{Z} \succ 0 \iff \exists\Gamma \in \mathbb{S}^{m_{\text{act}}}, \exists U \in \mathbb{S}^n : Z^\top UZ = 0, H + G_{\text{act}}^\top \Gamma G_{\text{act}} + U \succ 0. \quad (2.46)$$

Proof The proof of the theorem follows from Theorem 2.3 and Lemma 2.6.

2.3.2 Preserving the OCP structure

Theorem 2.6 can be specialized for the case of an OCP structure. To this end, let us introduce the following notation. We consider again problems with OCP structure as in (1.16). For such a problem we have a stage-wise active set as follows:

$$\mathcal{A}_k(\overline{w}) := \{i \in \mathcal{I}_k \mid \text{row}_i(c_k(\overline{w})) = 0\}, \quad (2.47)$$

with $\mathcal{I}_k$ the index set corresponding to the inequalities in each stage $k = 0, \dots, N$, respectively. We can now define $G_{\text{act},k}$ at some feasible point $\overline{w}$, as follows:

$$G_{\text{act},k} := \frac{\partial \text{row}_i(c_k)}{\partial w}(\overline{w}) \quad : i \in \mathcal{A}_k(\overline{w}). \quad (2.48)$$

for $k = 0, \dots, N$. Again, we omit $\overline{w}$ in the notation for $G_{\text{act},k}$ for improved readability of the equations. Furthermore, we define $G_{\text{act}}^\top := (G_{\text{act},0}^\top \mid \dots \mid G_{\text{act},N}^\top)$. We remark that $G_{\text{act}}^\top G_{\text{act}} \in \mathbb{S}_{\text{OCP}}^{N,n_x,n_u}$. Using these definitions, we can establish the following theorem:

Theorem 2.7. *Consider QP* (2.1)*, Definition 2.5 and assume that LICQ holds. Then, it holds that*

$$\widetilde{Z}^\top H\widetilde{Z} \succ 0$$

$$\iff$$

$$\exists\gamma \in \mathbb{R}, \exists U \in \mathbb{S}_{\text{OCP}}^{N,n_x,n_u} : Z^\top UZ = 0,\ H + \gamma G_{\text{act}}^\top G_{\text{act}} + U \succ 0.$$

Proof. The proof follows from Theorem 2.6 with matrix $\Gamma = \gamma I$ and Theorem 2.5. Note that the sparsity structure of H is preserved in $\widetilde{H} := H + \gamma G_{\text{act}}^\top G_{\text{act}} + U$, as $U \in \mathbb{S}_{\text{OCP}}^{N,n_x,n_u}$ and $G_{\text{act}}^\top G_{\text{act}} \in \mathbb{S}_{\text{OCP}}^{N,n_x,n_u}$. □

We now present the structure-preserving convexification algorithm, for problems with inequalities, in Algorithm 2.4. It works along the same lines as Algorithm 2.2, with the difference that we add $\gamma G_{\text{act},k}^\top G_{\text{act},k}$ to the original Hessian blocks. For clarity, we introduce an operator $\mathcal{H}(H, \delta, \gamma, \mathcal{A})$ that computes the convexified Hessian $\widetilde{H}$.

Algorithm 2.4 Structure-Preserving Convexification: inequality constrained case

```
Require: $H, \delta, \gamma$, current active set $\mathcal{A}$
Ensure: $\widetilde{H} := \mathcal{H}(H, \delta, \gamma, \mathcal{A})$
1: $\widehat{Q}_N = Q_N$
2: $\widetilde{Q}_N = \delta I$
3: $\overline{Q}_N = \widehat{Q}_N + \gamma G_{\text{act},N}^\top G_{\text{act},N} - \widetilde{Q}_N$
4: for $k = N-1, \dots, 0$ do
5:     $\begin{bmatrix} \widehat{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} = \begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix} + \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix} + \gamma G_{\text{act},k}^\top G_{\text{act},k}$
6:     $\widetilde{Q}_k := \widehat{S}_k^\top \widehat{R}_k^{-1} \widehat{S}_k + \delta I$
7:     $\widetilde{H}_k := \begin{bmatrix} \widetilde{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix}$
8:     $\overline{Q}_k = \widehat{Q}_k - \widetilde{Q}_k$
9: end for
10: $\widetilde{H} := \text{diag}(\widetilde{H}_0, \dots, \widetilde{Q}_N)$
```

Moreover, in the OCP case, we can again show that the primal solutions of QP(H) and QP($\widetilde{H}$) are equal.

Theorem 2.8. *Consider a primal solution $\Delta w^\star$ of QP(H) as defined in* (2.1) *with optimal active set $\mathcal{A}_{\text{QP}}^\star$. Furthermore, consider Definition 2.4 and Definition 2.5, and assume that LICQ and $\widetilde{Z}^\top H \widetilde{Z} \succ 0$ hold at $w^\star$. Then there exist δ, γ such that $\Delta w^\star$ is the unique primal solution of the convexified QP($\widetilde{H}$) with $\widetilde{H} = \mathcal{H}(H, \delta, \gamma, \mathcal{A}_{\text{QP}}^\star)$.*

Proof. The proof is based on the null space method for solving equality constrained QPs, as presented in [Nocedal and Wright, 2006]. We decompose the primal solution vector w as follows (dropping the 'QP' subscript):

$$w = \widetilde{Z} w_z + \widetilde{Y} w_y, \tag{2.49}$$

with $\widetilde{Z}$ as in Definition 2.5, and we complement the basis of the null space of $\widetilde{G}$ with a basis of its range space $\widetilde{Y}$, such that $(\widetilde{Z}|\widetilde{Y})^\top(\widetilde{Z}|\widetilde{Y}) = I$. We can compute w_y from the constraints:

$$G\widetilde{Y}w_y = -g, \tag{2.50}$$

$$G_{\text{act}}\widetilde{Y}w_y = -g_{\text{act}}. \tag{2.51}$$

We can obtain w_z as follows. From the first order optimality conditions for QP (2.1) we have that

$$H\widetilde{Z}w_z + H\widetilde{Y}w_y + G^\top\lambda + G_{\text{act}}^\top\mu = 0. \tag{2.52}$$

Multiplying from the left with $\widetilde{Z}^\top$ gives

$$\widetilde{Z}^\top H\widetilde{Z}w_z = -\widetilde{Z}^\top H\widetilde{Y}w_y, \tag{2.53}$$

where we used the fact that $\widetilde{Z}$ forms a basis for the null space of $\widetilde{G}$. Since $\widetilde{Z}^\top H\widetilde{Z} \succ 0$ by assumption, the solution is well-defined. Substituting H by $\widetilde{H} = H + U + \gamma G_{\text{act}}^\top G_{\text{act}}$, we have that

$$\begin{aligned}\left(\widetilde{Z}^\top H\widetilde{Z} + \underbrace{\widetilde{Z}^\top U\widetilde{Z}}_{=0} + \gamma\underbrace{\widetilde{Z}^\top G_{\text{act}}^\top}_{=0} G_{\text{act}}\widetilde{Z}\right) w_z \\ = -\widetilde{Z}^\top H\widetilde{Y}w_y - \widetilde{Z}^\top U\widetilde{Y}w_y - \underbrace{\widetilde{Z}^\top G_{\text{act}}^\top}_{=0} G_{\text{act}}\widetilde{Y}w_y.\end{aligned} \tag{2.54}$$

The left hand side of (2.54) is equal to the left hand side of (2.53), due to Theorem 2.7 and Definition 2.5. In order to prove equality of the right hand side, we use Definition 2.5 and (2.16) to obtain

$$\widetilde{Z}^\top U\widetilde{Y}w_y = \underbrace{\widetilde{Z}^\top G^\top}_{=0}\overline{Q}(G+\Sigma)\widetilde{Y}w_y + \widetilde{Z}^\top\Sigma^\top\overline{Q}G\widetilde{Y}w_y, \tag{2.55}$$

and we can show that $\widetilde{Z}^\top\Sigma^\top\overline{Q}G\widetilde{Y}w_y = 0$, as follows:

$$\begin{aligned}\widetilde{Z}^\top\Sigma^\top\overline{Q}G\widetilde{Y}w_y &= \widetilde{Z}^\top\Sigma^\top\overline{Q}(-g), \quad \text{from (2.50)} \\ &= 0,\end{aligned}$$

where the last step follows from the fact that only the first n_x rows of g are non-zero (see (2.12)), and the first n_x columns of $Z_c^\top\Sigma^\top\overline{Q}$ are zero (see (2.17) and (2.41)). Thus, (2.53) and (2.54) are identical and, together with (2.50)-(2.51), yield the same solution for w_y, w_z, from which we can compute the primal solution w of the QP. □

2.3.3 Recovering the dual solution

Recovering the Lagrange multipliers of the original problem is possible also for the case of active inequalities. Suppose we are sufficiently close to a regular solution of NLP (1.2), such that QP (2.38) has a regular solution whose active set is the same as the one from the solution $w^\star$ of the NLP, as in Lemma 2.5. Supposing we have identified the correct active set, we can write the QP as in (2.39). The corresponding Lagrangian function and its gradient are

$$\mathcal{L}(w,\lambda,\mu) := \frac{1}{2} w^\top H w + \lambda^\top (Gw + g) + \mu_{\text{act}}^\top (G_{\text{act}} w + g_{\text{act}}), \tag{2.56}$$

$$\nabla_w \mathcal{L}(w,\lambda,\mu) = Hw + G^\top \lambda + G_{\text{act}}^\top \mu_{\text{act}}, \tag{2.57}$$

where we define $g_{\text{act}} := c_i(\overline{w}), i \in \mathcal{A}(w^\star)$, and μ_{act} are the multipliers corresponding to the active inequalities.

Multipliers of active inequality constraints Using the definitions of $\widetilde{G}, \widetilde{Z}$ and Z_c as in (2.41), and stating $\nabla_w \mathcal{L}(w^\star, \lambda^\star, \mu^\star) = 0$, we can write

$$(G_{\text{act}} Z_c)^\top \mu_{\text{act}}^\star = -Z_c^\top (H w^\star), \tag{2.58}$$

where we multiplied $\nabla_w \mathcal{L}(w^\star, \lambda^\star, \mu^\star) = 0$ from the left with $Z_c^\top$. Note that the matrix $G_{\text{act}} Z_c$ is invertible, for the same reasons as given in the proof of Lemma 2.6. Substituting the original Hessian H by the convexified Hessian $\widetilde{H} = H + U + \gamma G_{\text{act}}^\top G_{\text{act}}$ gives us an expression for the multipliers corresponding to the active inequalities of the convexified problem:

$$(G_{\text{act}} Z_c)^\top \mu_{\text{conv,act}}^\star = -Z_c^\top (H w^\star + U w^\star + \gamma G_{\text{act}}^\top G_{\text{act}} w^\star). \tag{2.59}$$

For QPs with OCP structure, as in (2.1), it holds that

$$\begin{aligned} Z_c^\top U w^\star &= \underbrace{Z_c^\top G^\top}_{=0} \overline{Q}(G + \Sigma) w^\star + Z_c^\top \Sigma^\top \overline{Q} G w^\star \\ &= Z_c^\top \Sigma^\top \overline{Q}(-g), \\ &= 0, \end{aligned}$$

where we used a similar argument as in the proof of Theorem 2.8.

Comparing (2.58) and (2.59), we can recover the correct multipliers of the original problem as

$$\mu_{\text{act}}^\star = \mu_{\text{conv,act}}^\star + \gamma G_{\text{act}} w^\star. \tag{2.60}$$

We remark that the multipliers of the inequalities can be recovered in a stage-wise fashion, as with the multipliers corresponding to the equality constraints, as shown in Algorithm 2.5.

Multipliers of equality constraints We need to make a small modification to Algorithm 2.3 in order to recover the correct multipliers of the equality constraints, because they depend on the multipliers of the active inequality constraints. With the notation of QP (2.1), the procedure is shown in Algorithm 2.5.

Algorithm 2.5 Recovery of Lagrange multipliers: inequality constrained case

```
Require: $w$, $\mu_{\text{conv,act}}$
Ensure: $\lambda$, $\mu_{\text{act}}$
$\mu_{\text{act},N} \leftarrow \mu_{\text{conv,act},N} + \gamma G_{\text{act},N}\, x_N$
$\lambda_N \leftarrow Q_N x_N + C_N^\top \mu_{\text{act},N}$
for $k = N-1, \dots, 0$ do
    $\mu_{\text{act},k} \leftarrow \mu_{\text{conv,act},k} + \gamma G_{\text{act},k}\, w_k$
    $\lambda_k \leftarrow Q_k x_k + S_k^\top u_k + A_k^\top \lambda_{k+1} + C_{k,x}^\top \mu_{\text{act},k}$
end for
```

To summarize, we first compute the primal solution, the QP solver provides us with $\mu_{\text{conv,act}}$, with which we compute the multipliers corresponding to the active inequalities and the equality constraints.

2.3.4 Local convergence of SQP with structure-preserving convexification

Since our convexification method does not alter the primal solution locally, and we can correctly recover the dual solution, a full step SQP algorithm that employs our structure-preserving convexification algorithm converges quadratically under some assumptions, as is established in the next theorem. Note that this is a local convergence result in a neighborhood of a minimizer, while global convergence results require additional globalization strategies as discussed in [Nocedal and Wright, 2006].

Theorem 2.9. *Regard NLP* (1.16) *with a regular solution* $v^\star = (w^\star, \lambda^\star, \mu^\star)$*. Then there exist* $\delta > 0$*,* $\gamma > 0$*, and* $\varepsilon > 0$ *so that for all* $v_0 = (w_0, \lambda_0, \mu_0)$ *with* $\|v_0 - v^\star\| < \varepsilon$*, a full step SQP algorithm with Hessian approximation* $\widetilde{H} = \mathcal{H}(\nabla_w^2 \mathcal{L}(v), \delta, \gamma, \mathcal{A}(w^\star))$ *converges, the QP subproblems are convex, and the convergence rate is quadratic.*

Proof. We start the iterations with the optimal active set $\mathcal{A}(w^\star)$ and the exact Hessian at v_0. By convexifying the exact Hessian, we do not alter the primal solution, as established in Theorem 2.8. Moreover, by using Algorithm 2.5

we also recover the correct dual step. Therefore, by using our convexification approach we take the same primal-dual steps as the exact Hessian SQP method while solving convex QP subproblems. The quadratic convergence then follows from a standard convergence proof of Newton's method, e.g. Theorem 3.5 in [Nocedal and Wright, 2006]. □

Remark 2.2. *In Theorem 2.9 we assume that the SQP iterations start by using the optimal active set of the NLP solution. We remark that this assumption is not very restrictive: it is a standard property (see e.g. [Boggs and Tolle, 1995]) that if* v_0 *is close enough to* $v^\star$*, the QP subproblem even with inexact positive definite Hessian matrix will identify the correct active set. This is further illustrated by the numerical experiments in Section 2.5.*

2.4 Dealing with indefinite reduced problems

In the previous sections, we assumed a positive reduced Hessian. We call problems with an indefinite reduced Hessian $\widetilde{Z}^\top H \widetilde{Z} \nsucc 0$ *indefinite reduced* problems. In order to compute a positive definite Hessian approximation, we need to introduce some regularization. There are different heuristics of doing this which modify the problem in different ways to allow a unique global solution. We present one variant here.

2.4.1 Regularization of the Hessian

Consider Algorithm 2.4. For an indefinite reduced problem, it is possible that $\widehat{R}_k \nsucc 0$ holds in line 6 of the algorithm, such that the Schur complement lemma no longer holds. Instead, we still compute $\widehat{H}_k$ but regularize this Hessian block by removing all negative eigenvalues and replacing them with slightly positive eigenvalues. We call this action the 'projection' of the eigenvalues as in (2.37), where $\epsilon > 0$ is some small positive number. Applying this regularization results in Algorithm 2.6, which is based on Algorithm 2.4 but includes an if-clause checking for positive definiteness of $\widehat{R}_k$.

Remark 2.3. *In Theorem 2.7, we establish that there exists some* $\gamma > 0$*, such that there exists a positive definite convexified Hessian. In an SQP setting, when we are close to a local solution of NLP* (1.2)*, the active set of NLP* (1.2) *and QP* (2.38) *are identical and in principle, we can choose* γ *arbitrarily big. However, when we are at a point that is far from a local solution and the active set is not stable yet, large values for* γ *might result in poor convergence. A possible heuristic as an alternative to a fixed value* γ *is motivated by referring*

Algorithm 2.6 Structure-Preserving Convexification for inequality constrained optimization, including the regularization of Hessian blocks

```
Require: $H, \delta, \gamma, \epsilon$, current active set $\mathcal{A}$
Ensure: $\widetilde{H}$
 1: $\widehat{Q}_N = Q_N$
 2: $\widetilde{Q}_N = \delta I$
 3: $\overline{Q}_N = \widehat{Q}_N + \gamma G_{\text{act},N}^\top G_{\text{act},N} - \widetilde{Q}_N$
 4: for $k = N-1, \dots, 0$ do
 5:     $\begin{bmatrix} \widehat{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix} := \begin{bmatrix} Q_k & S_k^\top \\ S_k & R_k \end{bmatrix} + \begin{bmatrix} A_k^\top \overline{Q}_{k+1} A_k & A_k^\top \overline{Q}_{k+1} B_k \\ B_k^\top \overline{Q}_{k+1} A_k & B_k^\top \overline{Q}_{k+1} B_k \end{bmatrix} + \gamma G_{\text{act},k}^\top G_{\text{act},k}$
 6:     if $\widehat{R}_k \nsucc 0$ then
 7:         $\check{H}_k := \begin{bmatrix} \check{Q}_k & \check{S}_k^\top \\ \check{S}_k & \check{R}_k \end{bmatrix} = \text{project}(\widehat{H}_k, \epsilon)$
 8:     else
 9:         $\check{H}_k := \begin{bmatrix} \check{Q}_k & \check{S}_k^\top \\ \check{S}_k & \check{R}_k \end{bmatrix} = \begin{bmatrix} \widehat{Q}_k & \widehat{S}_k^\top \\ \widehat{S}_k & \widehat{R}_k \end{bmatrix}$
10:     end if
11:     $\widetilde{Q}_k := \check{S}_k^\top \check{R}_k^{-1} \check{S}_k + \delta I$
12:     $\widetilde{H}_k := \begin{bmatrix} \widetilde{Q}_k & \check{S}_k^\top \\ \check{S}_k & \check{R}_k \end{bmatrix}$
13:     $\overline{Q}_k = \check{Q}_k - \widetilde{Q}_k$
14: end for
15: $\widetilde{H} := \text{diag}(\widetilde{H}_0, \dots, \widetilde{Q}_N)$
```

to Algorithm 2.6. In line 6 of the algorithm, we check for positive definiteness of $\widehat{R}_k$. In line 5, we will try to make this matrix positive definite, by adding a matrix $G_{\text{act},k}^\top \Gamma G_{\text{act},k}$, where we compute Γ as follows. Consider a decomposition of matrix $R_k + B_k^\top \overline{Q}_{k+1} B_k$ in a basis for the range space $Y_{\text{act},k}$ and null space $Z_{\text{act},k}$ of $G_{\text{act},k}$. A necessary condition for the positive definiteness of $\widehat{R}_k$, by the Schur complement lemma, is then

$$Y_{\text{act},k}^\top (R_k + B_k^\top \overline{Q}_{k+1} B_k + G_{\text{act},k}^\top \Gamma G_{\text{act},k}) Y_{\text{act},k} \succ 0, \tag{2.61}$$

such that we could propose the following expression for Γ:

$$\Gamma = -(Y_{\text{act},k}^\top G_{\text{act},k}^\top)^{-1} Y_{\text{act},k}^\top (R_k + B_k^\top \overline{Q}_{k+1} B_k) Y_{\text{act},k} (G_{\text{act},k} Y_{\text{act},k})^{-1} + \gamma I, \tag{2.62}$$

for some $\gamma > 0$, where we used the fact that $G_{\text{act},k} Y_{\text{act},k}$ is invertible by construction. Note that (2.62) *is a heuristic choice for Γ in the sense that the* *above condition is only necessary, i.e. you still might have to apply regularization on $\widehat{H}_k$, as in line 7 of Algorithm 2.6.*

2.4.2 Recovering Lagrange multipliers

By applying our structure-preserving regularization method that is based on the convexification method of Section 2.2.2, we have $\mathrm{QP}(\widetilde{H})$ resulting in a different primal and dual solution than the original problem. We need to recover the dual solution with respect to the modified problem, i.e. without the backward transfer of cost but including the extra convexity introduced by the 'project' operation in line 7 of Algorithm 2.6. In order to do so, we do not start from the original Hessian as in Algorithm 2.5. Instead, we make use of the Hessian with the regularization terms added, but without the cost transfer terms. In other words, we keep a separate *modified* Hessian, which consists of the following blocks:

$$H_{\mathrm{mod}} = H + \Delta H = H + \mathrm{diag}(\Delta H_0, \ldots, \Delta H_{N-1}, 0), \tag{2.63}$$

where $\Delta H_k = 0$ when there was no regularization and $\Delta H_k = \breve{H}_k - \widehat{H}_k$ otherwise. We then apply Algorithm 2.5 to H_{mod} instead of H.

2.5 Numerical example

In this section, we offer a numerical example as an illustration of the practical use of our convexification method. We solve a nonlinear optimal control problem on an inverted pendulum. We will encounter this example a couple of times in this thesis, as it is simple to understand but gives rise to non-trivial optimization problems. This system, depicted in Figure 2.2, consists of a rod of length l making an angle θ with the vertical axis, attached to a cart with mass M that can move horizontally only, driven by a force F. At the end of the rod is a ball of mass m. The values of the parameters are listed in Table 2.1.

Table 2.1: PARAMETERS FOR THE INVERTED PENDULUM EXAMPLE

Parameter	Description	Value
M	mass of cart	1 kg
m	mass of pendulum	0.1 kg
l	length of rod	0.8 m
g	gravitational acceleration	9.81 kg m/s^2

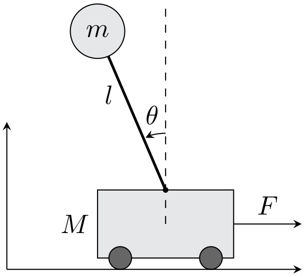


Figure 2.2: Schematic illustrating the inverted pendulum on top of a cart.

The dynamics of the inverted pendulum are described by the following ODE, where p, v are the horizontal displacement and horizontal velocity, respectively, θ is the angle with the vertical (see Figure 2.2) and ω the corresponding angular velocity:

$$\dot{p} = v, \tag{2.64a}$$

$$\dot{\theta} = \omega, \tag{2.64b}$$

$$\dot{v} = \frac{-ml\sin(\theta)\dot{\theta}^2 + mg\cos(\theta)\sin(\theta) + F}{M + m - m(\cos(\theta))^2}, \tag{2.64c}$$

$$\dot{\omega} = \frac{-ml\cos(\theta)\sin(\theta)\dot{\theta}^2 + F\cos(\theta) + (M+m)g\sin(\theta)}{l(M + m - m(\cos(\theta))^2)}. \tag{2.64d}$$

The control objective is to swing up the ball ($\theta = 0$), starting with the rod hanging vertically down, $\theta = \pi$. We collect the states in the state vector $x := [p, \theta, v, \omega]^\top$. A multiple-shooting discretization of the control problem corresponds to the following OCP formulation:

$$\underset{\substack{x_0,\ldots,x_N,\\F_0,\ldots,F_{N-1}}}{\text{minimize}} \quad \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ F_k \end{bmatrix}^\top \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_k \\ F_k \end{bmatrix} + x_N^\top Q x_N \tag{2.65a}$$

$$\text{subject to} \quad x_{k+1} = \psi_{\mathrm{d},k}(x_k, F_k), \qquad k = 0, \ldots, N{-}1, \tag{2.65b}$$

$$-80 \leqslant F_k \leqslant 80, \qquad k = 0, \ldots, N{-}1, \tag{2.65c}$$

$$x_0 = \overline{x}_0, \tag{2.65d}$$

Table 2.2: Exact-Hessian SQP iterations for the pendulum example (N=100), using the structure-preserving convexification method from Algorithm 2.6.

it.	**KKT norm**	**step size**	$H \succ 0$	$\tilde{Z}^\top H \tilde{Z} \succ 0$	$Z^\top H Z \succ 0$	**regs.**	**act. set chgs.**
1	1.52e+02	7.39e+02	True	True	True	0	74
2	5.33e+06	9.63e+02				9	10
3	2.02e+06	6.01e+02				9	1
4	1.47e+06	3.27e+02				5	4
5	9.44e+05	3.22e+02				4	0
6	3.75e+05	4.87e+02				3	7
7	1.04e+05	4.93e+02				1	22
8	9.13e+03	3.23e+02				2	14
9	3.90e+02	4.14e+02		True		0	2
10	1.25e+02	9.53e+01		True		0	5
11	6.66e+00	3.21e+00		True		0	0
12	1.38e-03	2.88e-03		True		0	0
13	2.02e-08	5.33e-09		True		0	0
14	1.42e-10	6.99e-11		True		0	0

where $\psi_{\mathrm{d},k}$ denotes a numerical integration method (explicit Runge-Kutta method of order 4) to simulate the continuous-time dynamics in (2.64) over one shooting interval, the weight matrices are chosen as

$$Q = \mathrm{diag}([10^3, 10^3, 10^{-2}, 10^{-2}]),$$

$$R = 10^{-2}.$$

Because our aim is to swing up the pendulum, we selected strong weights on the position and angle. The other states and the control are assigned a weak penalty in order to avoid too abrupt swing-ups and to favor smooth trajectories. Note that the weighting matrices Q and R are tuning parameters used by the control engineer in the design process in order to obtain a desired behavior. Different choices are therefore equally valid. The initial value is $\bar{x}_0 = [0, \pi, 0, 0]^\top$. We choose $N = 100$ shooting intervals of length 0.01 s.

We solve NLP (2.65) with a full-step SQP method, starting from $v_0 = (0, 0, 0)$, i.e. we assume all inequality constraints *inactive*. In each iteration, we apply our convexification method. We choose the following values for the parameters: $\delta = 1 \cdot 10^{-4}, \gamma = 1$. In Table 2.2, the iterations are given. The SQP method converges in 14 steps, given a tolerance of 10^{-8}. The solution is plotted in Figure 2.3. As can be seen, some inequality constraints are active at this (local) minimum.

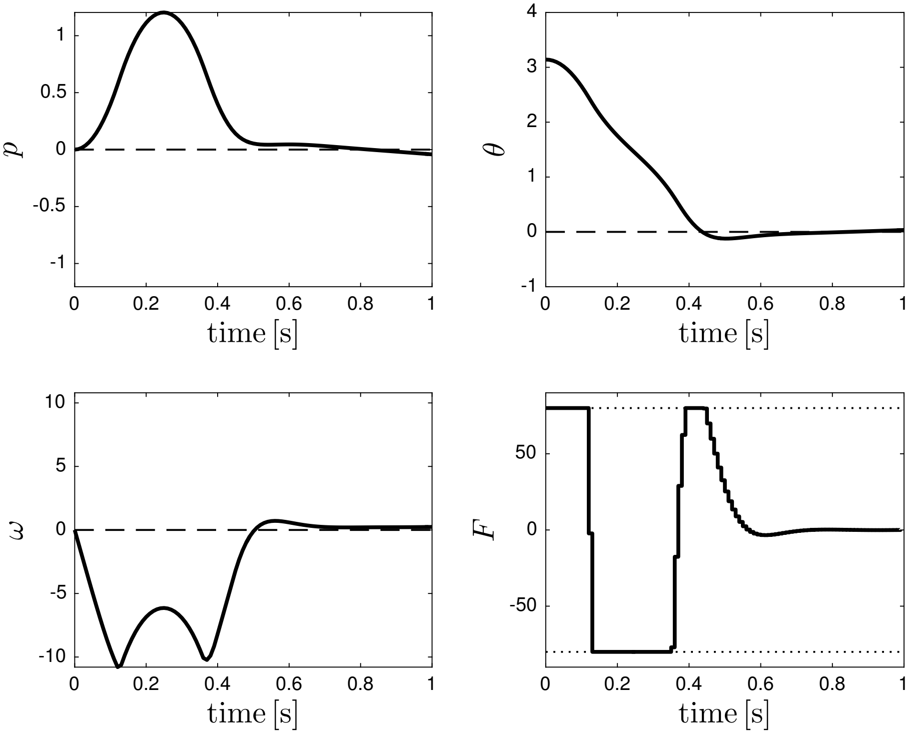


Figure 2.3: Optimal trajectories for OCP (2.65). Note that the bounds on F_k are active at the solution.

Only in the first iteration the Hessian matrix is positive definite. At the solution, only the reduced Hessian is positive definite. Whenever the reduced Hessian is not positive definite, we need to apply regularization as in Algorithm 2.6. This is denoted in Table 2.2 with the amount of shooting intervals in which we needed to regularize in the next to last column. The number of active set changes in each iteration is listed in the rightmost column.

It is interesting to remark that whenever the reduced Hessian $\widetilde{Z}^\top H \widetilde{Z}$ is positive definite, but $Z^\top H Z \nsucc 0$, we do not need to regularize thanks to the terms $\gamma G_{\mathrm{act},k}^\top G_{\mathrm{act},k}$ coming from the active inequality constraints in each stage k. However, please note that adding this term when we are still far from the NLP solution adds extra regularization, as the correct active set has not been identified yet. In the case the reduced Hessian is not positive definite and as a consequence we have to regularize, we only need to do so at maximum 9 intervals of the 100 control intervals (in iteration 2 and 3 , see Table 2.2).

We compare these results obtained with the structure-preserving convexification against two other regularization methods, namely *project* and *mirror* as described in (2.37), applied directly to each Hessian block in order to preserve the OCP structure, where we choose $\epsilon = \delta = 1 \cdot 10^{-4}$. The comparison in convergence is made in Figure 2.4. As can be seen, using the convexification as regularization method yields faster convergence, namely convergence in less than half the number of iterations of regularization by projection, using a tolerance of 10^{-8}. Moreover, we obtain quadratic convergence, as we established in Theorem 2.9, when using the convexification method once the optimal active set is found (see Table 2.2). By contrast, the other regularization methods result in linear convergence. For different horizon lengths, e.g. $N = 50, 150, 200$, the convergence behavior of the structure-preserving convexification method is similar to the one reported in Table 2.2 and we obtain similar convergence profiles for the other methods (see Figure 2.4, right side).

We remark that more advanced regularization schemes than the two that we are comparing to would yield similar convergence rates, e.g. the methods in [Gill and Robinson, 2013] or in [Wächter and Biegler, 2006]. Those methods, however, do not fulfill the desired properties (as mentioned in the introduction and Section 2.2.7) of the regularization schemes. The possibility of combining our approach with the one of [Gill and Robinson, 2013] is the subject of ongoing research.

2.6 Summary

In this chapter, we presented a structure-preserving convexification procedure for indefinite QPs arising from solving nonlinear optimal control problems using SQP. We proved that there is an equivalence between the existence of a convexified Hessian and the reduced Hessian being positive definite, which result in equal primal solutions. Furthermore, we offered an algorithm that constructs such a convexified Hessian with the same structure as the original Hessian and recovers the dual solution of the original problem. Doing so, we retain a locally quadratic rate of convergence using full steps in the SQP algorithm.

In the case the reduced Hessian is not positive definite, we proposed a regularization method based on the convexification. We illustrated our findings with a numerical example, which consists of solving a nonlinear OCP with an SQP-type method. Possible regularization methods were compared for the indefinite reduced case.

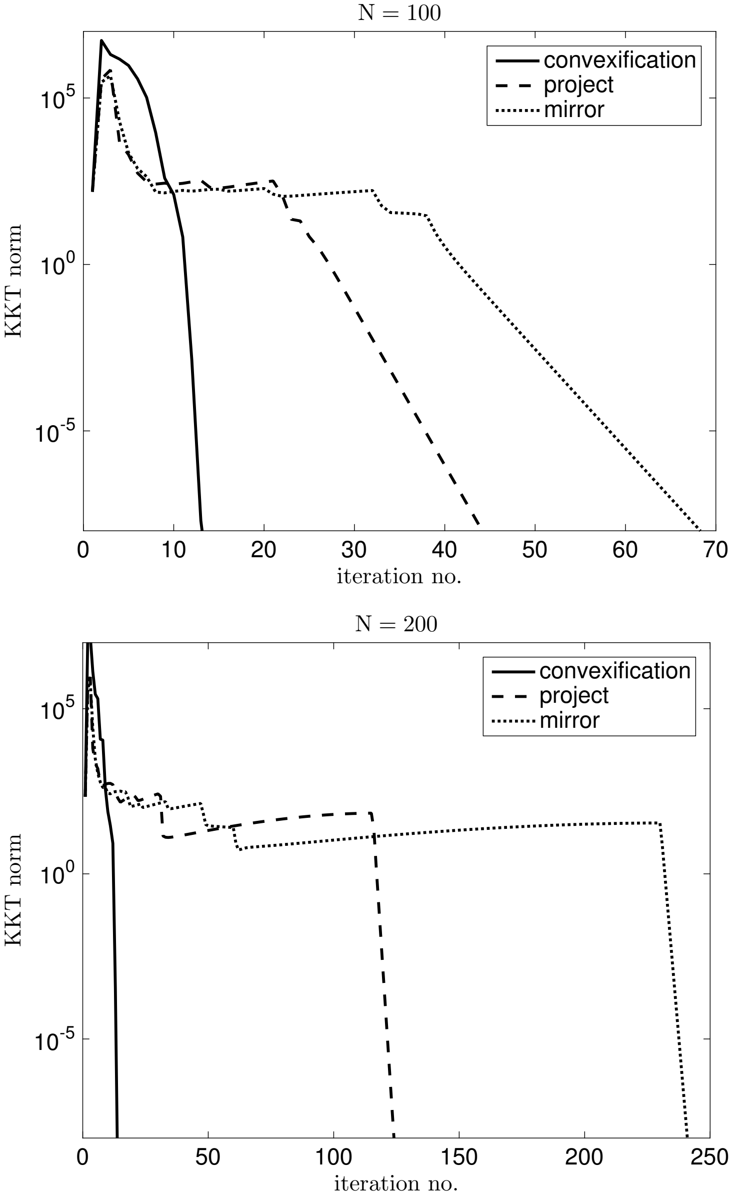


Figure 2.4: Comparison of the convergence of an SQP-type method applied to obtain the solution to NLP (2.65), with three different regularization methods. We compare two OCP instances, on the left $N = 100$, on the right $N = 200$.

Chapter 3

Sequential convex quadratic programming

> Do not disturb my circles!
>
> ---
>
> Archimedes, scientist

In the former chapter, we looked at how to approximate a non-convex structured QP with a convex approximation (under some conditions, the 'approximation' is exact), by modifying the Hessian of the QP. In this chapter, we take a different approach: we construct positive-definite Hessians much like the generalized Gauss-Newton algorithm does, but our approach allows more general convex functions than quadratics in the cost function, and it exploits convexity in the constraints as well.

The work presented in this chapter was submitted and accepted to the Conference on Decision and Control as [Verschueren et al., 2016a].

3.1 Introduction

In nonlinear model predictive control (NMPC), we typically encounter structured NLPs arising from a multiple shooting discretization of a continuous-time OCP, as in (1.16). Often, the stage cost and terminal cost are quadratic, or quadratic-over-nonlinear:

$$\underset{\substack{x_0,\ldots,x_N\\u_0,\ldots,u_{N-1}}}{\text{minimize}} \quad \frac{1}{2}\sum_{k=0}^{N-1} \|r_k(x_k,u_k)\|_2^2 \;+\; \frac{1}{2}\|r_N(x_N)\|_2^2 \tag{3.1a}$$

$$\text{subject to} \quad x_0 = \bar{x}_0, \tag{3.1b}$$

$$x_{k+1} = \psi_{\mathrm{d},k}(x_k,u_k), \quad k = 0,\ldots,N-1, \tag{3.1c}$$

$$c_k(x_k,u_k) \leqslant 0, \qquad k = 0,\ldots,N-1, \tag{3.1d}$$

$$c_N(x_N) \leqslant 0. \tag{3.1e}$$

A popular method for solving problems of the above type is the generalized Gauss-Newton (GGN) algorithm. It is well-known that the GGN algorithm has good local convergence properties when the residual functions $r_k(x_k^\star, u_k^\star)$ are small at the solution [Diehl, 2001]. Whenever this is not the case, the second order derivatives for either the residual or for the constraint functions in (3.1c)-(3.1e) need to be evaluated and included in the Hessian to obtain convergence even when initializing the algorithm arbitrarily close to a local minimizer. In case of an exact Hessian based method, one always obtains contraction close to a minimizer and this convergence speed is quadratic [Nocedal and Wright, 2000]. However, the Hessian of the Lagrangian can be indefinite such that the corresponding QP subproblem is non-convex and therefore generally not easy to solve, as we saw in the last chapter.

Instead of approximating an indefinite QP, we aim at directly formulating convex subproblems. Similar to the family of sequential convex programming (SCP) methods as discussed in [Tran-Dinh and Diehl, 2010], one can often exploit some convexity in either the objective or the constraint functions. The idea of an SCP method is to linearize all non-convex functions and solve the resulting convex subproblem in each iteration. Even though this approach can indeed locally result in good linear convergence [Tran-Dinh and Diehl, 2010], one needs to rely on a general convex solver. In this chapter, we instead propose an SQP method in which we use the convexity available from objective and constraint functions to obtain a more accurate Hessian approximation. This Hessian is based on the second order derivatives of convex functions and is therefore always positive semidefinite, similar to the case for the GGN method. A similar method which only exploits convexity in the objective is presented in [Martens and Sutskever, 2011] in the context of training deep artificial neural networks.

As a main contribution of this chapter, we propose the sequential convex quadratic programming (SCQP) method, which is presented as a generalization of the classical GGN method. The advantages of this algorithm are motivated from the computational point of view, since the second order derivatives needed for each Hessian approximation are relatively easy to evaluate unlike the propagation of second order derivatives for the dynamics (3.1c). In addition, each subproblem is a convex QP which is typically easier to solve than a more general convex subproblem, as in SCP methods. Note that because of these reasons, SCQP is suited for real-time optimization, e.g. in an NMPC setting.

Furthermore, it will be shown that SCQP can indeed improve the local convergence properties as compared to the GGN method, possibly resulting in a stronger contraction rate. The performance of this method is illustrated by a numerical example.

3.2 Problem formulation

In order to present our method in a more compact way, we introduce a slightly more general NLP formulation than NLP (3.1). As in Chapter 1, we will call it an 'NLP with convex substructure'. It should be noted that a non-convex function can always be rewritten as $\phi(r(w)) \leqslant 0$, with ϕ convex, such that the optimization problem reads as:

$$\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad \phi_0(r_0(w)) \tag{3.2a}$$

$$\text{subject to} \quad g_i(w) = 0 \qquad i = 1, \dots, m_{\text{eq}}, \tag{3.2b}$$

$$\phi_i(r_i(w)) \leqslant 0, \quad i = 1, \dots, m_{\text{ineq}}, \tag{3.2c}$$

with convex output functions $\phi_i : \mathbb{R}^{n_r^i} \rightarrow \mathbb{R}$ and possibly nonlinear 'residual' functions $r_i : \mathbb{R}^n \rightarrow \mathbb{R}^{n_r^i}$ for $i = 0, \dots, m_{\text{ineq}}$. GGN is well suited for the case $\phi_0(r_0(w)) = \frac{1}{2}\|r_0(w)\|_2^2$ as it exploits the least-squares nature of the objective function. SCQP extends this idea to exploit general convex output functions in the objective as well as in the inequality constraints. We call a function of the form $\phi(r(w))$ 'convex-over-nonlinear'.

To arrive at an even more compact notation for NLP (3.2), we will refer to the functions $\phi_i(r_i(w))$ as $\varphi_i(w)$. These functions are generally non-convex and further assumed to be three times continuously differentiable.

We define the Lagrangian of NLP (3.2) with objective function $\varphi_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ as

$$\mathcal{L}(w, \lambda, \mu) := \varphi_0(w) + \lambda^\top g(w) + \mu^\top \varphi(w), \tag{3.3}$$

with the Lagrange multipliers $\lambda \in \mathbb{R}^{m_{\text{eq}}}, \mu \in \mathbb{R}^{m_{\text{ineq}}}$, and $g : \mathbb{R}^n \rightarrow \mathbb{R}^{m_{\text{eq}}}, \varphi : \mathbb{R}^n \rightarrow \mathbb{R}^{m_{\text{ineq}}}$, respectively, denote the concatenation of equality and inequality constraints. Furthermore, as before in Chapter 1, we define

$$\widetilde{g}(w) = \begin{bmatrix} g(w) \\ \varphi_i(w) \end{bmatrix}, \quad i \in \mathcal{A}(w),$$

as the vector of equality constraints and active inequality constraints, with $\mathcal{A}(w)$ the active set of in w. Similarly, the Lagrange multipliers corresponding to constraints $\widetilde{g}(w)$ are denoted by

$$\widetilde{\mu} = \begin{bmatrix} \lambda \\ \mu_i \end{bmatrix}, \quad i \in \mathcal{A}(w).$$

When NLP (3.2) is solved with a full-step SQP method, the primal iterates evolve according to $w^{j+1} = w^j + \Delta w$, where Δw is the solution (see (1.9)) of

$$\begin{aligned} \underset{\Delta w \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\Delta w^\top H \, \Delta w + \nabla\varphi_0(w^j)\Delta w \\ \text{subject to} \quad & \frac{\partial g}{\partial w}\left(w^j\right)\Delta w + g(w^j) = 0 \\ & \frac{\partial \varphi}{\partial w}\left(w^j\right)\Delta w + \varphi(w^j) \leqslant 0. \end{aligned} \tag{3.4}$$

As mentioned in Section 1.2.4, in the case of a least squares cost function $\varphi_0(w) = \frac{1}{2}\|r_0(w)\|_2^2$ one often uses the Gauss-Newton Hessian approximation [Bock, 1983]:

$$H_k^{\mathrm{GN}} = J(w_k)^\top J(w_k), \text{ where } J(w_k) = \frac{\partial r_0(w_k)}{\partial w}. \tag{3.5}$$

However, a local minimizer $(w^\star, \lambda^\star, \mu^\star)$ of NLP (3.2) might become an unstable point for the SQP iterations with a Gauss-Newton Hessian approximation, i.e. the SQP method in some cases may not converge to $(w^\star, \lambda^\star, \mu^\star)$ even when initialized arbitrarily close to this minimizer [Bock, 1987; Diehl et al., 2010].

Motivated by this observation, the next section presents a necessary and sufficient condition on the Hessian approximation for asymptotic stability of a local minimizer $w^\star$.

3.3 Asymptotic stability of a local minimizer

First, we consider the case of unconstrained optimization. Afterwards, we show that the same result is applicable to constrained optimization problems. We assume that the Hessian approximation $H(w_k)$ is continuously differentiable in w_k. Furthermore, unless it is specified, we will assume that $H(w_k) \succ 0$.

3.3.1 Unconstrained optimization problem

Consider the simplified version of the NLP in (3.2) without any constraints:

$$\min_{w \in \mathbb{R}^n} \quad \varphi_0(w). \tag{3.6}$$

Assume that the second order sufficient conditions (SOSC) for optimality hold at a local minimizer $w^\star$, i.e. $\nabla^2 \varphi_0(w^\star) \succ 0$. We solve the first order necessary optimality condition $\nabla \varphi_0(w) = 0$ with a Newton-type SQP method with a Hessian only depending on the current linearization point w_k, resulting in iterations

$$\begin{aligned} w_{k+1} &= F(w_k) \\ &= w_k + \arg \min_{\Delta w_k \in \mathbb{R}^n} \frac{1}{2} \Delta w_k^\top H(w_k) \Delta w_k + \nabla \varphi_0(w_k)^\top \Delta w_k \\ &= w_k - H(w_k)^{-1} \nabla \varphi_0(w_k). \end{aligned} \tag{3.7}$$

A standard result from linear stability analysis is stated without proof in the following lemma.

Lemma 3.1 (Linear Stability Analysis)**.** *Regard an iteration of the form* $w_{k+1} = F(w_k)$ *with* F *a continuously differentiable function in a neighborhood of a fixed point* $F(w^*) = w^*$*. If all eigenvalues of the Jacobian* $\frac{\partial F}{\partial w}(w^*)$ *have a modulus smaller than one, i.e. if the spectral radius is smaller than one* $\rho\left(\frac{\partial F}{\partial w}(w^*)\right) < 1$*, then the fixed point* w^* *is asymptotically stable. In that case, when started in a neighborhood of the fixed point, the iterates converge to* w^* *with a Q-linear convergence rate with asymptotic contraction rate* $\rho\left(\frac{\partial F}{\partial w}(w^*)\right)$*. On the other hand, if one of the eigenvalues has a modulus larger than one, i.e. if* $\rho\left(\frac{\partial F}{\partial w}(w^*)\right) > 1$*, then the fixed point is unstable.*

The Taylor expansion of $F(w_k)$ in (3.7) around $w^\star$ reads as

$$\begin{aligned} w_{k+1} = w_k &- H(w^\star)^{-1} \nabla^2 \varphi_0(w^\star)(w_k - w^\star) \\ &+ \mathcal{O}(\|w_k - w^\star\|^2), \end{aligned} \tag{3.8}$$

where we used the fact that $\nabla \varphi_0(w^\star) = 0$.

Neglecting higher order terms, we can rewrite (3.8) as

$$\Delta w_{k+1}^\star = \underbrace{H(w^\star)^{-1}(H(w^\star) - \nabla^2 \varphi_0(w^\star))}_{M^\star} \Delta w_k^\star, \tag{3.9}$$

with $\Delta w_k^\star = w_k - w^\star$.

Note that from (3.9) it follows that $\frac{\partial F}{\partial w}(w^\star) = M^\star$. A different characterization of the necessary and sufficient condition on the spectral radius in Lemma 3.1 is stated in the following lemma.

Lemma 3.2. *Define $M^\star$ as in (3.9). Then the two following statements are equivalent.*

1. *The spectral radius $\rho(M^\star) \leqslant \alpha$,*
2. *$-\alpha H(w^\star) \preceq H(w^\star) - \nabla^2\varphi_0(w^\star) \preceq \alpha H(w^\star)$.*

Proof. By assumption, $H(w^\star) \succ 0$, so $H(w^\star)^{-\frac{1}{2}}$ exists. It follows that the eigenvalues of $M^\star$ and $\Sigma := H(w^\star)^{-\frac{1}{2}}(H(w^\star) - \nabla^2\varphi_0(w^\star))H(w^\star)^{-\frac{1}{2}}$ are the same. Assume now that $\rho(M^\star) \leqslant \alpha$. As Σ is symmetric, we can write that $-\alpha I \preceq \Sigma \preceq \alpha I$ and thus $-\alpha H(w^\star) \preceq H(w^\star) - \nabla^2\varphi_0(w^\star) \preceq \alpha H(w^\star)$. The converse follows from the definition of spectral radius. ☐

Motivated by Lemmas 3.1 and 3.2, we can now state a necessary and sufficient condition on $H(w^\star)$ in order for $w^\star$ to be asymptotically stable with contraction rate α.

Theorem 3.1. *A solution $w^\star$ of NLP (3.6) is an asymptotically stable fixed point for the Newton-type iteration in (3.7) with asymptotic contraction rate $0 \leqslant \alpha < 1$, if and only if the following conditions hold:*

$$H(w^\star) \succeq \frac{\nabla^2\varphi_0(w^\star)}{1+\alpha}, \tag{3.10a}$$

$$H(w^\star) \preceq \frac{\nabla^2\varphi_0(w^\star)}{1-\alpha}. \tag{3.10b}$$

Proof. These two conditions are equivalent to

$$-\alpha H(w^\star) \preceq H(w^\star) - \nabla^2\varphi_0(w^\star) \preceq \alpha H(w^\star). \tag{3.11}$$

This condition is equivalent to $\rho(\frac{\partial F}{\partial w}(w^\star)) \leqslant \alpha < 1$ because of Lemma 3.2, which is in turn the necessary and sufficient condition for asymptotic stability as defined by Lemma 3.1. ☐

The bounds in (3.10) are illustrated in Figure 3.1.

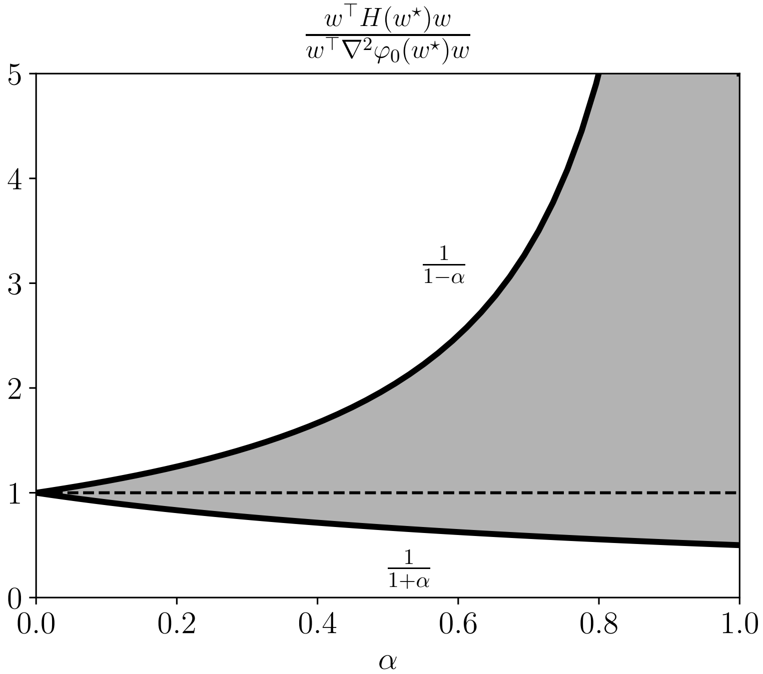


Figure 3.1: For a solution $w^\star$ to be stable, the Hessian approximation $H(w^\star)$ has to lie inside the shaded region (3.10) for any $w \in \mathbb{R}^n \backslash \{0\}$.

3.3.2 Equality and inequality constraints

Let us return to the Newton-type SQP method from (3.4), applied to the original NLP (3.2). Assume that the linear independence constraint qualification (LICQ), SOSC and strict complementarity hold at the local minimizer $(w^\star, \lambda^\star, \mu^\star)$. It follows that the active set for the QP solution is stable close to a local minimizer of the NLP, i.e. the active set for the QP in (3.4) is also the active set of the original NLP [Nocedal and Wright, 2006].

The KKT system corresponding to the QP subproblem (3.4) after fixing the active inequality constraints and omitting the inactive ones, reads as

$$\begin{bmatrix} H(w,\nu) & G(w)^\top \\ G(w) & 0 \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta \widetilde{\mu} \end{bmatrix} = - \begin{bmatrix} \nabla_w \mathcal{L}(w, \widetilde{\mu}) \\ \widetilde{g}(w) \end{bmatrix}, \tag{3.12}$$

where $G(w) = \frac{\partial \widetilde{g}}{\partial w}(w)$ and we recall that the multipliers $\widetilde{\mu}$ correspond to the active constraints $\widetilde{g}(w) := [g(w)^\top, \varphi_{i \in \mathcal{A}}(w)^\top]^\top$ only.

Let us denote the constraint matrix at the local solution by $G := G(w^\star)$, such that its QR factorization reads:

$$G^\top = Q_G \overline{R}_G = \begin{bmatrix} Y & Z \end{bmatrix} \begin{bmatrix} R_G \\ 0 \end{bmatrix}, \tag{3.13}$$

with orthogonal matrix $\begin{bmatrix} Y & Z \end{bmatrix}$. It follows that the matrix Z is a null space basis for the constraint matrix G at the solution, i.e. $GZ = 0$. Let us define the reduced Hessian matrix and its corresponding approximation at a local solution $(w^\star, \widetilde{\mu}^\star)$:

$$\Lambda^\star := Z^\top \nabla_w^2 \mathcal{L}^\star \, Z,$$

$$\widetilde{H}^\star := Z^\top H^\star \, Z,$$

with shorthands $\nabla_w^2 \mathcal{L}^\star := \nabla_w^2 \mathcal{L}(w^\star, \widetilde{\mu}^\star)$ and $H^\star := B(w^\star, \widetilde{\mu}^\star)$. By assumption, SOSC holds at the local minimizer $(w^\star, \widetilde{\mu}^\star)$, which implies that the reduced Hessian $\Lambda^\star \succ 0$.

Introducing the compact notation $v = [w^\top, \widetilde{\mu}^\top]^\top$, one step of the Newton-type SQP method (3.12) reads as

$$\begin{aligned} v_{k+1} &= v_k - \begin{bmatrix} H(v_k) & G(w_k)^\top \\ G(w_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla_w \mathcal{L}(v_k) \\ \widetilde{g}(w_k) \end{bmatrix} \\ &= v_k - \begin{bmatrix} H^\star & G^\top \\ G & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla_w^2 \mathcal{L}^\star & G^\top \\ G & 0 \end{bmatrix} (v_k - v^\star), \end{aligned} \tag{3.14}$$

after neglecting higher order terms. Here, we used a Taylor expansion of $\begin{bmatrix} \nabla_w \mathcal{L}(v_k) \\ \widetilde{g}(w_k) \end{bmatrix}$ around $v^\star$ and the fact that this quantity vanishes at $v^\star$. We can now state a version of Theorem 3.1, including equality and inequality constraints.

Theorem 3.2. *Assume we are close to a local minimizer* $(w^\star, \lambda^\star, \mu^\star)$ *of NLP problem* (3.2) *where LICQ, SOSC and strict complementarity hold, such that the current active set is equal to the active set at the local minimizer. Then* $(w^\star, \widetilde{\mu}^\star)$ *is asymptotically stable for the Newton-type iteration* (3.14) *with asymptotic contraction rate* $0 \leqslant \alpha < 1$*, if and only if the following conditions hold:*

$$\widetilde{H}^\star \succeq \frac{\Lambda^\star}{1+\alpha}, \tag{3.15a}$$

$$\widetilde{H}^\star \preceq \frac{\Lambda^\star}{1-\alpha}. \tag{3.15b}$$

Proof. Let us define a similar change of variables as in (3.9), $\Delta v_k^\star = U^\top(v_k - v^\star)$, with orthogonal matrix

$$U = \begin{bmatrix} Z & Y & \\ & & I \end{bmatrix}, \tag{3.16}$$

where Z, Y are defined by the QR factorization in (3.13). Equation (3.14) then reads as:

$$\begin{aligned} \Delta v_{k+1}^\star &= U^\top \begin{bmatrix} H^\star & G^\top \\ G & 0 \end{bmatrix}^{-1} U U^\top \begin{bmatrix} H^\star - \nabla_w^2 \mathcal{L}^\star & 0 \\ 0 & 0 \end{bmatrix} U \Delta v_k^\star, \\ &= L^{-1} P \, \Delta v_k^\star, \end{aligned}$$

where we defined the matrices

$$\begin{aligned} L &= U^\top \begin{bmatrix} H^\star & G^\top \\ G & 0 \end{bmatrix} U, \\ P &= U^\top \begin{bmatrix} H^\star - \nabla_w^2 \mathcal{L}^\star & 0 \\ 0 & 0 \end{bmatrix} U. \end{aligned}$$

For each eigenvalue β of matrix $L^{-1}P$ there exists a $v \neq 0$ satisfying $Pv = \beta Lu$. Expanding the matrix products yields

$$\begin{aligned} &\beta \begin{bmatrix} Z^\top H^\star Z & Z^\top H^\star Y & 0 \\ Y^\top H^\star Z & Y^\top H^\star Y & R_G \\ 0 & R_G^\top & 0 \end{bmatrix} \begin{bmatrix} v_z \\ v_y \\ v_r \end{bmatrix} = \\ &\begin{bmatrix} Z^\top (H^\star - \nabla_w^2 \mathcal{L}^\star) Z & Z^\top (H^\star - \nabla_w^2 \mathcal{L}^\star) Y & 0 \\ Y^\top (H^\star - \nabla_w^2 \mathcal{L}^\star) Z & Y^\top (H^\star - \nabla_w^2 \mathcal{L}^\star) Y & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_z \\ v_y \\ v_r \end{bmatrix}, \end{aligned} \tag{3.17}$$

where $GZ = 0$ and $GY = R_G^\top$ have been used.

For $\beta \neq 0$, from the bottom row of the equation in (3.17) we have that $\beta R_G^\top v_y = 0$, which implies $v_y = 0$ since R_G is invertible. From the top row it then follows that $\beta Z^\top H^\star Z v_z = Z^\top (H^\star - \nabla_w^2 \mathcal{L}^\star) Z v_z$, and consequently that

$$\beta v_z = (\underbrace{Z^\top H^\star Z}_{\widetilde{H}^\star})^{-1} (\underbrace{Z^\top H^\star Z}_{\widetilde{H}^\star} - \underbrace{Z^\top \nabla_w^2 \mathcal{L}^\star Z}_{\Lambda^\star}) v_z. \tag{3.18}$$

Thus, the nonzero eigenvalues of matrix $L^{-1}P$ are equal to the eigenvalues of $(\widetilde{H}^\star)^{-1}(\widetilde{H}^\star - \Lambda^\star)$. We have now recovered the same form of (3.9) for each Newton-type iteration. Consequently, the rest of the proof follows that of Theorem 3.1 for the unconstrained case. $\square$

As a corollary of Theorem 3.2 it holds that $\widetilde{H}^\star \succ \frac{1}{2}\Lambda^\star$ implies asymptotic stability for the local minimizer $v^\star$. This can be seen from taking the limit of (3.15a) for $\alpha \to 1$ (see Figure 3.1). Motivated by these results, we introduce a novel Hessian approximation in the next section.

3.4 Sequential convex quadratic programming

Consider the NLP (3.2), which is the more general form of OCP (3.1), and we recall that ϕ_i for $i = 0, \ldots, m_{\text{ineq}}$ are convex. We construct a Hessian approximation using the contributions from the functions we know to be convex. This is motivated by the fact that SQP might yield large steps Δw_k due to the linearization of the inequalities, whose convexity is ignored. We propose to use a modification of the generalized Gauss-Newton method, where we linearize the convex inequalities, but add their positive definite second order derivative to the Hessian. This Hessian approximation then reads

$$\begin{aligned} H^{\text{SCQP}}(w,\mu) := {} & \frac{\partial r_0}{\partial w}(w)^\top \nabla_r^2 \phi_0(r_0(w)) \frac{\partial r_0}{\partial w}(w) \\ & + \sum_{i=1}^{m_{\text{ineq}}} \mu_i \frac{\partial r_i}{\partial w}(w)^\top \nabla_r^2 \phi_i(r_i(w)) \frac{\partial r_i}{\partial w}(w). \end{aligned} \tag{3.19}$$

We will refer to the SQP method using H^{SCQP} as a Hessian approximation in its QP (3.4) subproblem, as *sequential convex quadratic programming* (SCQP). The corresponding Hessian approximation error reads as:

$$\begin{aligned} E^{\text{SCQP}}(w,\lambda,\mu) = {} & \sum_{i=1}^{m_{\text{eq}}} \lambda_i \nabla^2 g_i(w) + \sum_{j=1}^{n_r^i} \frac{\partial \phi_0}{\partial r_{0,j}}(w) \nabla^2 r_{0,j}(w) \\ & + \sum_{i=1}^{m_{\text{ineq}}} \mu_i \sum_{j=1}^{n_r^i} \frac{\partial \phi_i}{\partial r_{i,j}}(w) \nabla^2 r_{i,j}(w). \end{aligned} \tag{3.20}$$

Note that the GGN method is a special case of this class of methods. In comparison to the Gauss-Newton Hessian, H^{SCQP} has the benefit that we are closer to a Hessian approximation $\widetilde{H}^\star \succ \frac{1}{2}\Lambda^\star$ that implies asymptotic stability of a local minimizer $w^\star$. This fact is a corollary from the following lemma.

Lemma 3.3. *For a local minimum* $(w^\star, \lambda^\star, \mu^\star)$ *of NLP* (3.2) *with least squares cost function* $\phi_0(r_0(w)) = \frac{1}{2}\|r_0(w)\|_2^2$, *it holds that*

$$H^{\text{SCQP}}(w^\star, \mu^\star) \succeq H^{\text{GN}}(w^\star). \tag{3.21}$$

Proof. For a least squares cost function, (3.21) follows directly from

$$\begin{aligned} H^{\mathrm{SCQP}}(w,\mu) &= \frac{\partial r_0}{\partial w}(w)^\top \frac{\partial r_0}{\partial w}(w) + \sum_{i=1}^{m_{\mathrm{ineq}}} \mu_i \frac{\partial r_i}{\partial w}(w)^\top \nabla_r^2 \phi_i(r_i(w)) \frac{\partial r_i}{\partial w}(w), \\ &= H^{\mathrm{GN}}(w) + \sum_{i=1}^{m_{\mathrm{ineq}}} \mu_i \frac{\partial r_i}{\partial w}(w)^\top \nabla_r^2 \phi_i(r_i(w)) \frac{\partial r_i}{\partial w}(w), \end{aligned}$$

and the fact that inequalities ϕ_i are convex and multipliers μ_i are nonnegative at a local solution. ☐

Remark 3.1. *SCQP is motivated by the results of Theorem 3.2. However, we do not guarantee that the SCQP Hessian satisfies the bounds* (3.15) *in general. As such, local convergence, just as in the case of GGN, is not guaranteed. In practice, the SCQP Hessian is often 'closer' to the exact Hessian, which might result in better convergence properties, as shown in Section 2.5.*

In the case of constant second order derivatives for the objective and inequality constraint functions, SCQP is specifically easy to implement and computationally cheap. But also in general, the second order derivatives for the Hessian in (3.19) can be efficiently evaluated using Algorithmic Differentiation (AD) [Griewank, 2000].

An alternative point of view on SCQP is the following. Applying sequential convex programming (SCP) in order to solve NLP (3.2) results in the subproblems:

$$\min_{w\in\mathbb{R}^n} \quad \phi_0\left(\frac{\partial r_0}{\partial w}(w_k)(w-w_k) + r_0(w_k)\right) \tag{3.22a}$$

$$\text{s.t.} \quad \frac{\partial g_i}{\partial w}(w_k)(w-w_k) + g_i(w_k) = 0, \tag{3.22b}$$

$$\phi_j\left(\frac{\partial r_j}{\partial w}(w_k)(w-w_k) + r_j(w_k)\right) \leqslant 0, \tag{3.22c}$$

with $i = 1,\dots,m_{\mathrm{eq}}$, $j = 1,\dots,m_{\mathrm{ineq}}$ and the current linearization point w_k. One iteration of the SCQP method is then equivalent to an exact Hessian based SQP iteration for the latter convex subproblem (3.22).

We illustrate the benefits of the SCQP algorithm using a numerical case study in the next section.

3.5 Numerical example

In this section, we solve an OCP of the form in (3.1) using sequential quadratic programming. More specifically, we compare different techniques of Hessian approximation and their resulting local convergence properties, including the proposed SCQP method and the more classical Gauss-Newton (GGN) and exact Hessian (EH) based SQP method.

3.5.1 Implementation and software

To numerically solve the optimal control problems, we adopt the open-source CasADi software framework, which has been proven to solve OCPs reliably and efficiently [Andersson et al., 2018]. More specifically, we use the Python front-end to formulate the OCP as a nonlinear program (NLP) using direct multiple shooting [Bock and Plitt, 1984] as a discretization method.

The resulting NLP is passed to the open-source solver IPOPT [Wächter and Biegler, 2006]. It implements a primal-dual interior point method suited for solving large-scale NLPs. The linear algebra subroutine calls were passed to the sparse solver `ma86` from the HSL library [HSL, 2011; Duff, 2004]. For SCQP, GGN and exact Hessian SQP, we use the open-source QP solver qpOASES [Ferreau et al., 2014]. Note that the software packages mentioned above can be conveniently called from within the CasADi framework.

3.5.2 Inverted pendulum with terminal region

As an example, we regard a pendulum on a cart, as in Section 2.5. The $X - Y$ position of the pendulum is given by the equations

$$r(x) := \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} p - l\sin(\theta) \\ l\cos(\theta) \end{bmatrix}, \tag{3.23}$$

with p the horizontal position of the cart, θ the angle with the vertical and l the length of the rod (see Section 2.5).

We solve the following OCP:

$$\min_{\substack{x_0,\ldots,x_N\\u_0,\ldots,u_{N-1}}} \sum_{k=0}^{N-1} u_k^2, \tag{3.24a}$$

$$\text{s.t.} \qquad x_0 = \overline{x}_0, \tag{3.24b}$$

$$x_{k+1} = \psi_{\mathrm{d},k}(x_k, u_k), \quad k = 0, \ldots, N-1, \tag{3.24c}$$

$$\|[X_N - l, Y_N - l]^\top\|_2^2 - R_e^2 \leqslant 0, \tag{3.24d}$$

with $x_k = [p_k, \theta_k, v_k, \omega_k]^\top$, $u_k = F_k$, initial value $\overline{x}_0 = [0, \pi, 0, 0]^\top$ and X_N, Y_N the position of the pendulum at the end of the control horizon. The discrete dynamics $\psi_{\mathrm{d},k}$ are obtained by simulating the continuous-time dynamics with an RK4 integrator with 20 integration stages and a sampling time of 0.05 s. Note that the terminal constraint is of the convex-over-nonlinear form $\phi(r(x_N)) \leqslant 0$, with

$$\phi(r(x_N)) = \left\| r(x_N) - \begin{bmatrix} l \\ l \end{bmatrix} \right\|_2^2 - R_e^2,$$

and $r(x_N)$ as in (3.23). We use $N = 20$ control intervals.

We compare the convergence of SCQP with that of GGN and exact Hessian SQP, in Figure 3.2. We start each method close to a local solution with $R_e = 0.05\,\mathrm{m}$, plotted in Figure (3.4a). Exact Hessian SQP converges quadratically, as expected. SCQP converges linearly, in contrast to GGN, for which the local solution is unstable and thus does not converge at all to the local minimum. In the sense of Figure 3.1, the GGN Hessian approximation falls 'under' the shaded region.

In fact, for decreasing radius R_e, the solution stays an unstable fixed point for GGN. This is shown in Fig 3.3, where we plot the largest eigenvalue of $(\widetilde{H}^\star)^{-1}(\widetilde{H}^\star - \Lambda^\star)$. Even for the terminal region quite large, as in Figure 3.4b, GGN does not converge. GGN needs $R_e \approx 2\,\mathrm{m}$ for the solution to become stable; in this case, there is no swing-up. SCQP converges to a nearby local solution for all radii.

Remark 3.2. *In this example, the extra computational cost of using SCQP instead of GGN is almost negligible. More specifically, we need: (1)* $\nabla_r^2 \phi(r(x_N))$*, which is constant and can be computed offline, (2) evaluation of* $\frac{\partial r}{\partial x}(x_N)$*, and (3) the Lagrange multiplier, which we can directly get from the QP solver.*

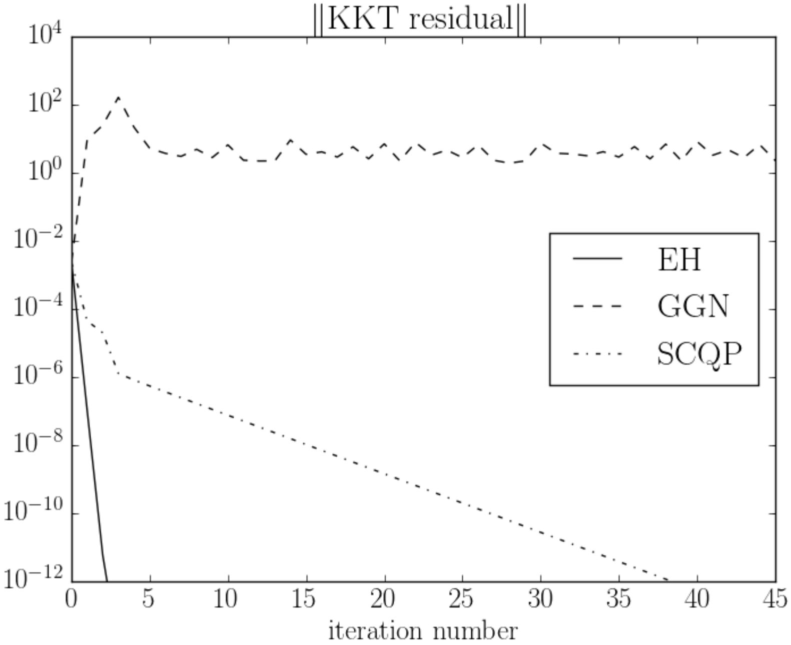


Figure 3.2: Comparison of the convergence of exact Hessian SQP, GGN and SCQP to a local minimum of the pendulum OCP. The measure of convergence is the norm of the residual of the KKT system. $R_e = 0.05\,\mathrm{m}$, corresponding to "A" in Figure 3.3.

3.6 Summary

This chapter proposed a generalization of the classical GGN method, referred to as the sequential convex quadratic programming (SCQP) method. Similar to Gauss-Newton, the novel Hessian approximation always results in a convex QP subproblem by including the second order derivatives of only convex objective and inequality constraint functions. It is shown that the SCQP approach has better local convergence properties compared to GGN, as illustrated with a numerical example.

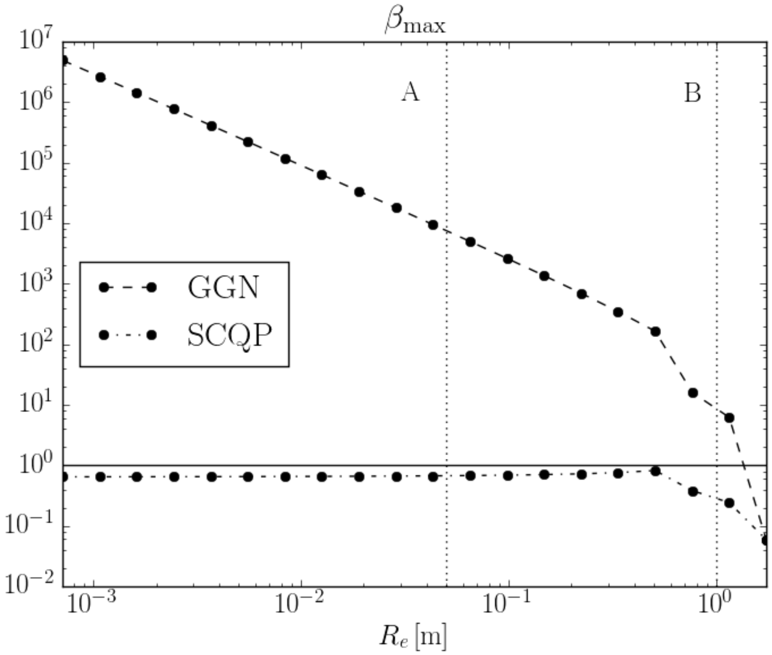
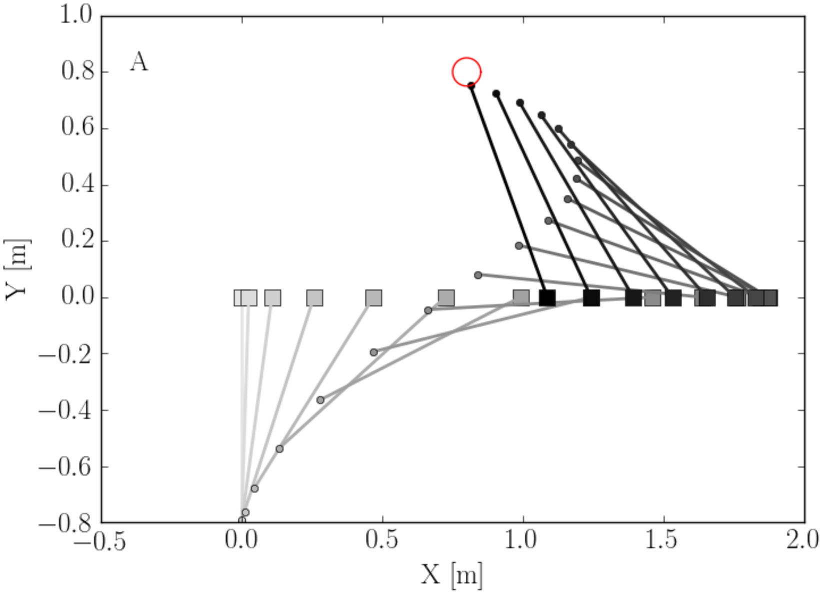
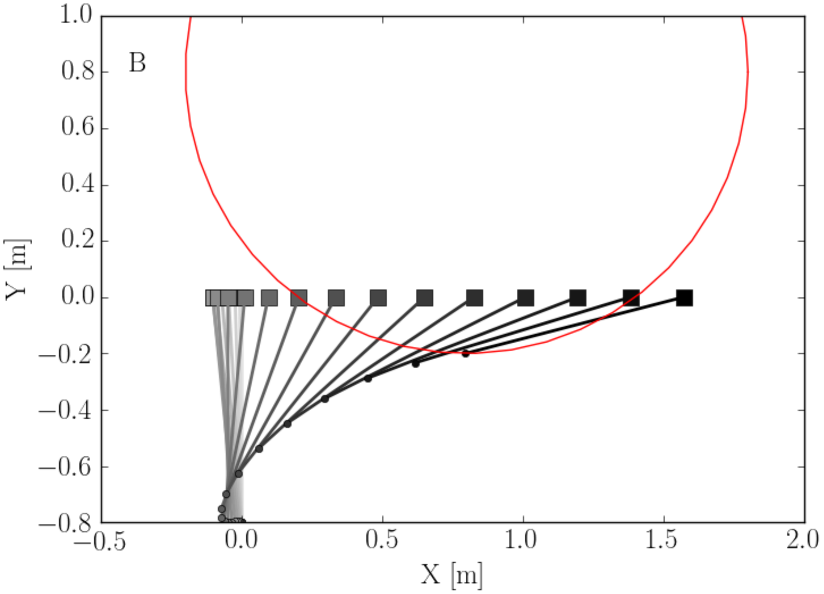


Figure 3.3: Greatest eigenvalue $\beta_{\max} = \rho((\widetilde{H}^\star)^{-1}(\widetilde{H}^\star - \Lambda^\star))$ for GGN and SCQP, for different values of R_e. For a solution to OCP (3.24) to be a stable fixed point, $\beta_{\max}$ has to lie below the horizontal line $\beta_{\max} = 1$. The lines "A" and "B" correspond to solutions in Figure 3.4.



(a) $R_e = 0.05\,\mathrm{m}$



(b) $R_e = 1.0\,\mathrm{m}$

Figure 3.4: Solution of OCP (3.24) for different values of R_e. Note that for these radii, the solution is a stable fixed point for SCQP, and an unstable one for GGN (Figure 3.3).

Chapter 4

Time-optimal nonlinear model predictive control for point-to-point motions

> But meanwhile time flies; it flies never to be regained.
>
> Virgil, poet

Time-optimal control is one of the oldest optimal control problem classes, both in continuous time and in discrete time, and an ample literature on this topic exists. Often the solution techniques are based on indirect methods (see e.g. [Athans and Falb, 1966]), and assume some properties on the optimal control a priori (for example, a bang-bang structure, i.e. the controls are almost always at their limits). For nonlinear systems, it is even more challenging to arrive at a time-optimal control law in closed form; for some special classes of nonlinear systems, this has been achieved, see e.g. [Nešić et al., 1998] and the references therein.

For problems in discrete time, it was Kalman who first derived a time-optimal controller for linear time-invariant (LTI) systems, also sometimes called deadbeat control, for an excellent overview of such controllers, see [O'Reilly, 1981]. Although time-optimal control in both continuous and discrete time are similar in nature, an important difference is that for discrete time systems, the time-optimal control is generally non-unique and not always bang-bang, see [Desoer and Wing, 1961].

In this chapter and the following one, we tackle two different discrete time-optimal NMPC techniques: one for point-to-point motions and one for path following. The one of this chapter focuses on point-to-point motions and has been published as a separate conference paper in the Conference on Decision and Control [Verschueren et al., 2017a].

4.1 Introduction

In this chapter, we focus on point-to-point time-optimal control of discretized nonlinear dynamic systems, using direct methods. Point-to-point motions are a sizable application field, for example in robotic manipulators [Geering et al., 1986], wafer steppers [Lau and Pao, 2003] and cranes [Vukov et al., 2012].

Some applications use a receding horizon technique, e.g. model predictive control (MPC). In [Van den Broeck et al., 2011], a two-level time-optimal MPC method for linear systems is presented and experimentally validated, where the upper level determines the optimal horizon length, and the lower level is a tracking MPC with fixed horizon length. One drawback of this method is that the horizon length can change from one problem to the next. In [Zhao et al., 2004], a nonlinear MPC (NMPC) method for time-optimal control has been proposed, based on a varying time scaling (the horizon length is a decision variable). Close to the desired end point, the horizon is kept fixed and a standard regulator NMPC is used to keep the system there. A different but related idea is proposed in [Rösmann et al., 2015], where the continuous dynamics are scaled with a constant factor, such that the horizon length becomes fixed. This scaling factor then enters the objective function linearly, such that time-optimality is recovered. One disadvantage of this method is that the underlying optimization problem becomes nonlinear, even for linear systems. Furthermore, a stability proof of an NMPC method using such time-scaled dynamics is not known to the authors.

The main focus of this chapter is a novel NMPC method for time-optimal point-to-point motions. We make two theoretical contributions: we show that nominal stability of the end point holds (for simplicity of the exposition, the end point is assumed to be the origin), and we prove that it is time-optimal, if a certain tuning parameter is chosen sufficiently high. The method is based on an optimal control problem (OCP) formulation employing an exponential increase of the stage costs along the horizon, based on an idea as presented (but not further developed) in [Van den Broeck et al., 2010]. The stage costs are chosen to be the l_1-norm of the difference between the end point and the state at each stage. As such, exponentially increasing weights encourage the motion to arrive at the end point “as early as possible”, thus recovering time-optimality. The resulting NMPC method inherits this property of time-optimality. The practical usefulness of the method is illustrated with numerical experiments.

4.2 Problem formulation

We consider continuous-time time-optimal control problems as in (1.13), which we briefly repeat here:

$$
\begin{aligned}
\underset{x(\cdot),u(\cdot),\,T}{\text{minimize}} \quad & T = \int_0^T 1 \,\mathrm{d}t && \text{(4.1a)}\\
\text{subject to} \quad & x(0) = \overline{x} && \text{(4.1b)}\\
& \dot{x}(t) = \psi(x(t), u(t)), \quad \forall t \in [0, T] && \text{(4.1c)}\\
& c(x(t), u(t)) \leqslant 0, \qquad \forall t \in [0, T] && \text{(4.1d)}\\
& x(T) = 0 && \text{(4.1e)}\\
& 0 \leqslant T, && \text{(4.1f)}
\end{aligned}
$$

The final state is the origin, without loss of generality, as we can introduce a different final steady state by a simple state transformation. It is well-established in the literature that the solution of (4.1) for linear systems with linear constraints gives rise to bang-bang solutions [Athans and Falb, 1966], and uniqueness of the optimal solution, under some conditions, can be established by the uniqueness of a boundary-value problem (BVP). In the case of nonlinear systems, these properties cannot be guaranteed, but the solution is often still found to be of bang-bang type.

In this chapter, we use a receding horizon formulation, more specifically time-optimal nonlinear model predictive control (TONMPC), which consists in repeatedly solving discretized versions of OCP (4.1). We treat two different approaches; the first can be found in the literature [Rösmann et al., 2015], the second one (presented in Section 4.4) is a new method.

4.3 Time-scaling approach

One straightforward approach to solve OCP (4.1) is to introduce a time transformation as in (1.14). One advantage of doing so is that the horizon length T becomes independent of the pseudo-time τ over which we integrate the continuous-time system. To discretize the continuous-time system, we choose a multiple shooting discretization [Bock and Plitt, 1984]. Since τ is now the independent variable, we choose a shooting interval length of $\Delta\tau = 1/N$, with N the number of shooting intervals. Discretizing OCP (4.1) becomes

$$\begin{aligned}
&\underset{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1},\\ T_0,\dots,T_{N-1}}}{\text{minimize}} && T = \sum_{k=0}^{N-1} \frac{T_k}{N} && \text{(4.2a)}\\
&\text{subject to} && x_{k+1} = \widetilde{\psi}_{\mathrm{d},k}(x_k, u_k, T_k),\ \ k = 0,\dots,N-1 && \text{(4.2b)}\\
& && c(x_k,u_k) \leqslant 0, \qquad k = 0,\dots,N-1 && \text{(4.2c)}\\
& && x_0 = \overline{x} && \text{(4.2d)}\\
& && x_N = 0 && \text{(4.2e)}\\
& && T_k = T_{k+1}, \qquad k = 0,\dots,N-2 && \text{(4.2f)}\\
& && 0 \leqslant T_k, \qquad k = 0,\dots,N-1, && \text{(4.2g)}
\end{aligned}$$

where $x_k \in \mathbb{R}^{n_x}$, $u_k \in \mathbb{R}^{n_u}$ are the vectors of states and controls, and we introduce decision variables T_k on each shooting interval in order to preserve the problem structure. Function $\widetilde{\psi}_{\mathrm{d},k} : \mathbb{R}^{n_x+n_u+1} \to \mathbb{R}^{n_x}$ is the discrete-time representation of the time-scaled dynamic system (1.14), e.g. obtained by numerical integration. The inequality constraints are denoted by $c : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{m_{\mathrm{ineq},k}}$. We would like to point out that equality constraint (4.2f) is not strictly necessary, but it is advisable to include it, as the optimizer might exploit the time warping to "cut corners" as to accommodate the path constraints (4.2c).

In an NMPC setting, we repeatedly solve OCP (4.2), feeding back the optimal control $u_0^\star$ to the system. However, we need to include one additional constraint, such that the sampling time Δt of the real control system matches that of the first time interval in OCP (4.2), namely $T_0 = N \cdot \Delta t$, while keeping all other T_k, $k = 1,\dots,N-1$ free but equal.

Note that such a fixed first interval makes a possible proof of nominal stability for the NMPC method more intricate, in particular because the property of recursive feasibility is difficult to establish. Indeed, such a stability proof does not yet exist, to the author's knowledge.

4.4 Exponentially increasing penalty

The second formulation, by contrast, does not employ a time-scaling and uses the discrete-time dynamics

$$x_{k+1} = \psi_{\mathrm{d}}(x_k, u_k), \quad k = 0, 1, \ldots,$$

obtained by e.g. numerically simulating the continuous-time dynamics $\psi(x(t), u(t))$ over one sampling interval Δt. We make the following standard assumptions on the system.

Assumption 4.1. *We assume that* ψ_{d} *is continuous, and that* $\psi_{\mathrm{d}}(0, 0) = 0$.

The new time-optimal method is based on the following discrete time minimum-time problem which is another discretization of continuous-time problem (4.1):

$$N^\star(\overline{x}) = \begin{aligned} \min_{\substack{N, x_0, \ldots, x_N, \\ u_0, \ldots, u_{N-1}}} \quad & N \\ \text{s.t.} \quad & x_0 = \overline{x} \\ & x_{k+1} = \psi_{\mathrm{d}}(x_k, u_k), \; k = 0, \ldots, N-1 \\ & c(x_k, u_k) \leqslant 0, \qquad k = 0, \ldots, N-1 \\ & x_N = 0, \end{aligned} \tag{4.3}$$

where $N \in \mathbb{N}_0$ (the set of nonnegative integer numbers), and $x_k, u_k, \overline{x}$ as before.

If we found a solution such that the state arrives at the origin at stage $N^\star$ (we will sometimes use $N^\star$ as a shorthand for $N^\star(\overline{x})$ in the following), we want to keep it there, such that we make the following assumption:

Assumption 4.2. *For the inequality constraints* $c(x, u)$ *it holds that* $c(0, 0) \leqslant 0$.

The following definition will prove to be useful in the subsequent discussion.

Definition 4.1. *We define a time-optimal solution subject to discrete dynamical system* $x_{k+1} = \psi_{\mathrm{d}}(x_k, u_k)$ *as any solution to* (4.3) *that brings the system from* $\overline{x}$ *to the origin in* $N^\star(\overline{x})$ *steps, where* $N^\star(\overline{x})$ *is the solution to* (4.3). *Furthermore, let* $\mathcal{X}_{N^\star}$ *denote the set of states* $\overline{x}$ *such that the optimal value of* (4.3) *is smaller than or equal to* $N^\star(\overline{x})$.

The fact that the horizon length is not fixed in OCP (4.3) is cumbersome in an algorithmic setting, because the problem dimensions will change in each iteration of the solution method. By contrast, we introduce our time-optimal formulation with fixed horizon length N (possibly much larger than $N^\star$) as follows:

$V_N^\star(\overline{x}) =$

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}} \quad \sum_{k=0}^{N-1} \theta^k \|x_k\|_1 \tag{4.4a}$$

$$\text{s.t.} \quad x_0 = \overline{x} \tag{4.4b}$$

$$x_{k+1} = \psi_\mathrm{d}(x_k, u_k),\ k = 0, \dots, N-1 \tag{4.4c}$$

$$c(x_k, u_k) \leqslant 0, \qquad k = 0, \dots, N-1 \tag{4.4d}$$

$$x_N = 0, \tag{4.4e}$$

where $\theta \in \mathbb{R}$ is a fixed parameter. Note that we fix x to zero at a later stage than $N^\star$. With that regard, an interesting connection between problems (4.3) and (4.4) is stated in the following theorem.

Theorem 4.1. *Assume OCP* (4.3) *is feasible and choose $N \geqslant N^\star(\overline{x})$. There exists a number θ_1 such that, for all $\theta \geqslant \theta_1$, the solution of* (4.4) *satisfies $x_k^\star = 0, \quad k = N^\star, \dots, N$, i.e. the solution is time-optimal with respect to Definition 4.1.*

Naturally, this discrete-time time-optimal solution is a mere approximation of the continuous-time problem (4.1) but the approximation gets better as the fixed sampling time Δt gets smaller and simultaneously N gets bigger. In order to prove Theorem 4.1, we will first present a result for which we consider the following related time-optimal problem with horizon length $N^\star(\overline{x})$, where $N^\star(\overline{x})$ is the optimal horizon length, obtained from (4.3). Furthermore, we introduce the objective function

$$f(x_0, \dots, x_{N^\star-1}) := \sum_{k=0}^{N^\star-1} \theta^k \|x_k\|_1,$$

and we relax the terminal constraint to be $x_{N^\star} = \varepsilon$, such that the OCP becomes:

$$\underset{\substack{x_0,\dots,x_{N^\star},\\ u_0,\dots,u_{N^\star-1}}}{\text{minimize}} \quad \sum_{k=0}^{N^\star-1} \theta^k \|x_k\|_1 \tag{4.5a}$$

$$\text{subject to} \quad x_0 = \overline{x} \tag{4.5b}$$

$$x_{k+1} = \psi_{\mathrm{d}}(x_k, u_k), \; k = 0, \dots, N^\star - 1 \tag{4.5c}$$

$$c(x_k, u_k) \leqslant 0, \qquad k = 0, \dots, N^\star - 1 \tag{4.5d}$$

$$x_{N^\star} = \varepsilon, \tag{4.5e}$$

with $\varepsilon \in \mathbb{R}^{n_x}$. The optimal solution depends parametrically on ε, and we remark that ε enters the OCP formulation linearly. With the terminal constraint $x_{N^\star} = 0$ we associate the Lagrange multiplier $\lambda_{N^\star}(\theta) \in \mathbb{R}^{n_x}$, of which the value depends on θ. In fact, the value of the Lagrange multipliers is nothing else than (see [Nocedal and Wright, 2006]):

$$\lambda_{N^\star,i}(\theta) = \left.\frac{\mathrm{d}f(x^\star(\varepsilon_i,\theta))}{\mathrm{d}\varepsilon_i}\right|_{\varepsilon_i=0}$$

with $i \in \{1, \dots, n_x\}$ and the ε-perturbed trajectory

$$x^\star(\varepsilon_i,\theta) := [x_0^\star(\varepsilon_i,\theta), \dots, x_{N^\star}^\star(\varepsilon_i,\theta)],$$

and $\varepsilon_j = 0, \text{for} j \neq i$. We make the following technical assumption on these perturbed trajectories:

Assumption 4.3. *The perturbation on the optimal trajectory with respect to $\varepsilon_i, i = 1, \dots, n_x$ is bounded, i.e. there exists a value $L \in \mathbb{R}$ such that*

$$\left|\frac{\partial x_k^\star(0,\theta)}{\partial \varepsilon_i}\right| \leqslant L, \quad k = 0, \dots, N^\star,$$

for all $i \in \{1, \dots, n_x\}$, for $k = 0, \dots, N^\star$, for all $\overline{x} \in \mathcal{X}_{N^\star}$ and all $\theta \geqslant \theta_1$.

We establish the following result with regard to the value of the Lagrange multiplier $\lambda_{N^\star}(\theta)$.

Lemma 4.1. *For each $\overline{x}$, there exists a number θ_1 such that $\theta^{N^\star} \geqslant \|\lambda_{N^\star}(\theta)\|_\infty$, for all $\theta \geqslant \theta_1$.*

Proof. Consider OCP (4.5) with relaxed constraint (4.5e). For the Lagrange multipliers $\lambda_{N^\star,i}(\theta), i = 1, \ldots, n_x$, it holds that:

$$\begin{aligned}
|\lambda_{N^\star,i}(\theta)| &= \left| \left. \frac{\mathrm{d}f(x^\star(\varepsilon_i,\theta))}{\mathrm{d}\varepsilon_i} \right|_{\varepsilon_i=0} \right| \\
&= \left| \frac{\partial f(x^\star(0,\theta))}{\partial x^\star(0,\theta)} \cdot \frac{\partial x^\star(0,\theta)}{\partial \varepsilon_i} \right| \\
&\leqslant \theta^0 \left\| \frac{\partial x_0^\star(0,\theta)}{\partial \varepsilon_i} \right\|_1 + \ldots \\
&\qquad + \theta^{N^\star-1} \left\| \frac{\partial x_{N^\star-1}^\star(0,\theta)}{\partial \varepsilon_i} \right\|_1 \\
&= \mathcal{O}(\theta^{N^\star-1}), \qquad \theta \to \infty,
\end{aligned} \tag{4.6}$$

where the last step holds because of Assumption 4.3.

From (4.6) it follows that

$$\begin{aligned}
\|\lambda_{N^\star}(\theta)\|_\infty &= \max[|\lambda_{N^\star,1}(\theta)|, \ldots, |\lambda_{N^\star,n_x}(\theta)|] \\
&= \mathcal{O}(\theta^{N^\star-1}), \qquad \theta \to \infty.
\end{aligned}$$

The fact that $\|\lambda_{N^\star}(\theta)\|_\infty = \mathcal{O}(\theta^{N^\star-1})$ and $\theta^{N^\star} = \mathcal{O}(\theta^{N^\star})$, directly leads to the existence of a θ_1 that satisfies $\theta^{N^\star} \geqslant \|\lambda_{N^\star}\|_\infty$, for all $\theta \geqslant \theta_1$, which proves the lemma. ☐

We are now ready to prove Theorem 4.1.

Proof. We can rewrite OCP (4.4) in terms of OCP (4.5) as follows:

$$\begin{aligned}
\underset{\substack{x_0,\ldots,x_N,\\ u_0,\ldots,u_{N-1}}}{\text{minimize}} \quad & f(x_0, \ldots, x_{N^\star-1}) + \sum_{k=N^\star}^{N} \theta^k \|x_k\|_1 \\
\text{subject to} \quad & x_0 = \overline{x} \\
& x_{k+1} = \psi_\mathrm{d}(x_k, u_k), \; k = 0, \ldots, N-1 \\
& c(x_k, u_k) \leqslant 0, \qquad k = 0, \ldots, N-1 \\
& x_N = 0.
\end{aligned}$$

Take first $N = N^\star$; then, we have a terminal constraint $x_{N^\star} = 0$ such that the theorem holds. For $N > N^\star$, for all $\theta \geqslant \theta_1$, the term $\theta^{N^\star} \|x_{N^\star}\|_1$ represents an exact penalty term for the constraint $x_{N^\star} = \varepsilon = 0$ in (4.5), by Lemma (4.1) and the fact that the ∞-norm is the dual of the 1-norm, see [Nocedal and Wright, 2006]. As such, we have that $x_{N^\star} = 0$. For stages $k = N^\star + 1, \ldots, N$, the solution will stay at the origin, as it is feasible (by Assumption 4.2) and optimal (because of the 1-norm in the objective). This proves the theorem. □

4.5 NMPC formulation

The time-optimality of OCP (4.4) has been established in Theorem 4.1. Next, we will use this in a receding horizon fashion, i.e. we formulate the NMPC problem. The NMPC method consists of the following steps:

$$\text{1) Estimate the state } \overline{x} \text{ at the current time } t_k, \tag{4.7a}$$

$$\text{2) Solve OCP (4.4) for } u_0^\star, \tag{4.7b}$$

$$\text{3) Apply control } \kappa_N(\overline{x}) := u_0^\star \text{ to } \psi_\mathrm{d}, \tag{4.7c}$$

$$\text{4) Proceed to the next time point } t_{k+1}. \tag{4.7d}$$

Let $\mathcal{X}_N$ be the set of states $\overline{x}$ for which (4.4) has a solution. We make the following (standard) assumption, referring to [Rawlings and Mayne, 2009]:

Assumption 4.4. *There exists a $\mathcal{K}_\infty$ function $\alpha(\cdot)$ such that $V_N^\star(\overline{x}) \leqslant \alpha(|\overline{x}|)$.*

The stability of the NMPC method is established in the next theorem.

Theorem 4.2. *Take $\theta > 1$. Then, the origin is asymptotically stable for the system $\overline{x}_\mathrm{next} = \psi_\mathrm{d}(\overline{x}, \kappa_N(\overline{x}))$.*

Proof. We follow a standard Lyapunov argument, using the notation as in [Rawlings and Mayne, 2009]. We need to show that the optimal value function in (4.4) satisfies:

$$V_N^\star(\overline{x}) \geqslant \alpha_1(|\overline{x}|) \qquad \forall \overline{x} \in \mathcal{X}_N \tag{4.8a}$$

$$V_N^\star(\overline{x}) \leqslant \alpha_2(|\overline{x}|) \qquad \forall \overline{x} \in \mathcal{X}_N \tag{4.8b}$$

$$V_N^\star(\psi_\mathrm{d}(\overline{x}, \kappa_N(\overline{x}))) \leqslant V_N^\star(\overline{x}) - \alpha_1(|\overline{x}|), \quad \forall \overline{x} \in \mathcal{X}_N, \tag{4.8c}$$

for $\mathcal{K}_\infty$ functions $\alpha_1(\cdot), \alpha_2(\cdot)$. The optimal cost decrease (4.8c) is proven as follows: consider the optimal value function

$$V_N(\overline{x}, \mathbf{u}^\star) := V_N^\star(\overline{x}) = \sum_{k=0}^{N} \theta^k \|x_k^\star\|_1,$$

with $\mathbf{u}^\star := [u_0^\star, \ldots, u_{N-1}^\star]$. Going to the next time step, with feasible but suboptimal control sequence $\tilde{\mathbf{u}} := [u_1^\star, \ldots, u_{N-1}^\star, \tilde{u}]$, with $\tilde{u}$ to be determined, it holds that

$$\begin{aligned} V_N^\star(\psi_\mathrm{d}(\overline{x}, \kappa_N(\overline{x}))) &\leqslant V_N(\psi_\mathrm{d}(\overline{x}, \kappa_N(\overline{x})), \tilde{\mathbf{u}}) \\ &= \sum_{k=1}^{N} \theta^{k-1} \|x_k^\star\|_1 + \theta^N \|\psi_\mathrm{d}(x_N^\star, \tilde{u})\|_1 \\ &\leqslant V_N^\star(\overline{x}) - \alpha_1(|\overline{x}|), \end{aligned}$$

provided that there exists a $\mathcal{K}_\infty$ function $\alpha_1(\cdot)$, such that

$$\begin{aligned} \alpha_1(|\overline{x}|) &\leqslant \|x_0^\star\|_1 - \theta^N \|\psi_\mathrm{d}(x_N^\star, 0)\|_1 \\ &\leqslant \|\overline{x}\|_1, \end{aligned} \tag{4.9}$$

because from constraint (4.4e) we have that $\psi_\mathrm{d}(x_N^\star, 0) = 0$, as $\tilde{u} = 0$ is the optimal control at the origin. It follows that a function α_1 satisfying (4.9) always exists and (4.8c) holds.

Inequality (4.8a) follows from (4.9) and $\|\overline{x}\|_1 = \theta^0 \|x_0^\star\|_1 \leqslant V^\star(\overline{x})$, and inequality (4.8b) holds because of Assumption 4.4.

Thus, the optimal value function $V_N^\star(\overline{x})$ is a Lyapunov function and the theorem is proven. □

Note that recursive feasibility is implied by the terminal constraint $x_N = 0$, such that we can make the following statement with respect to time-optimality for the nominal NMPC method.

Corollary 4.1. *For $\overline{x} \in \mathcal{X}_{N^\star}$ (see Definition 4.1) and $\theta \geqslant \theta_1$, the NMPC method (4.7), started from $\overline{x}$ at t_0, satisfies $\overline{x} = 0$ for $t_{N^\star}, t_{N^\star+1}, \ldots$, i.e. the closed-loop trajectory is time-optimal with regard to Definition 4.1.*

Proof. The proof follows from Theorem 4.1 (with $\theta \geqslant \theta_1$) and nominal stability as established in Theorem 4.2. □

Apart from enabling stability, the terminal constraint (4.4e) yields another advantage, namely that θ does not need to be as big as θ_1. This can be seen by considering the terminal constraint $x_N = 0$ as an implicit weighting on state $x_{N^\star}$, such that θ can be smaller than θ_1, while still yielding time-optimal trajectories.

4.6 Simulation results

We present simulation results on two dynamical systems; the first is a toy example, the other one is a mechanical system, more specifically a hanging pendulum with varying length in the presence of obstacles.

In all of the following, the non-smooth 1-norm is implemented by employing a smooth standard reformulation with slack variables.

4.6.1 Toy example

As a first example to test our time-optimal NMPC method on, we take the system from [Chen and Allgöwer, 1998]:

$$\dot{p} = q + u(\mu + (1-\mu)p),$$

$$\dot{q} = p + u(\mu - 4(1-\mu)q).$$

We define the state vector $x := [p, q]^\top$, and we set $\mu = 0.5$. It is reported in [Chen and Allgöwer, 1998] that for $\mu \in (0, 1)$ the system is unstable and its linearization is stabilizable. To obtain a discrete-time system $\psi_\mathrm{d}(x, u)$, we integrate above system with numerical code `cvodes` [Hindmarsh et al., 2005] over $\Delta t = 0.01\,\mathrm{s}$. Furthermore, the calculation in each time step of the NMPC is carried out in the CasADi [Andersson et al., 2018] framework for algorithmic differentiation and dynamic optimization, using IPOPT [Wächter and Biegler, 2006] as optimization solver.

We then run NMPC method (4.7), with parameters $N = 50$, $\theta = 1.6$, starting from three different points at $t_0 = 0\,\mathrm{s}$. The inequality constraints are

$$c(x, u) = \begin{bmatrix} u - 10 \\ -u - 10 \end{bmatrix} \leqslant 0.$$

For each of the three scenarios, we also compute $N^\star$, the minimum time. The closed-loop trajectories are shown in Figure 4.1. Notice how all the state trajectories go to the origin (and not earlier) at $k = N^\star$ and remain there,

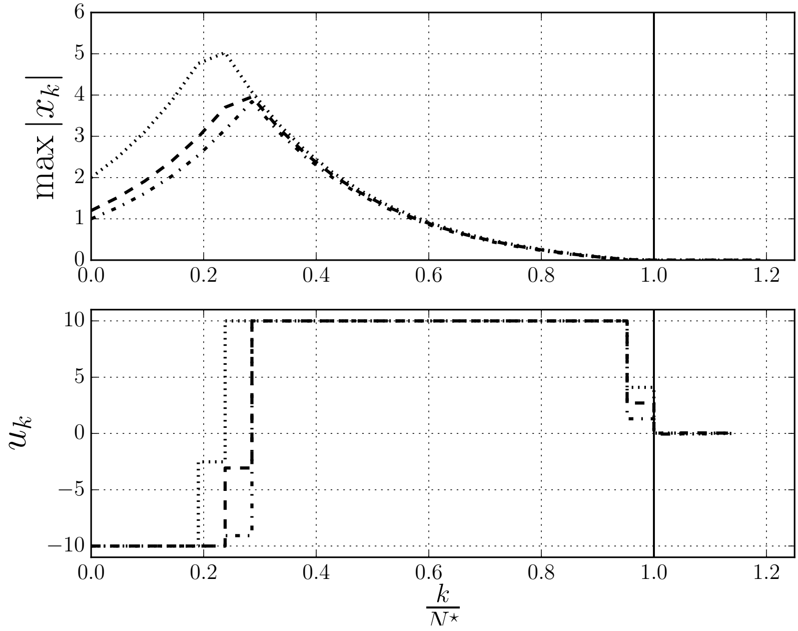


Figure 4.1: Closed-loop trajectories for the system of the toy example under NMPC method (4.7).

which illustrates Corollary 4.1. Furthermore, it is interesting to look at the control trajectories: the closed-loop control lies against its limits $[-10, 10]$ except for some points, which usually holds for discrete-time time-optimal control (see [Desoer and Wing, 1961]).

4.6.2 Application: hanging pendulum

A second system we apply our time-optimal NMPC method on is a hanging pendulum with varying length, that moves from start to end point, which are both equilibria for the system. The continuous-time equations of motion are

$$\dot{x}_t = v_t, \quad \dot{v}_t = a_t, \tag{4.10a}$$

$$\dot{x}_c = v_c, \quad \dot{v}_c = a_c, \tag{4.10b}$$

$$\dot{\varphi} = \omega, \tag{4.10c}$$

$$\dot{\omega} = \frac{-(2\omega v_c + a_t \cos(\varphi) + g\sin(\varphi))}{x_c}. \tag{4.10d}$$

with $x_t[\mathrm{m}], v_t[\mathrm{m/s}], a_t[\mathrm{m/s^2}]$ the horizontal displacement, velocity and acceleration, respectively. We assume that we can control the acceleration directly, as in [Auernig and Troger, 1987]. The same holds for the cable length $x_c[\mathrm{m}]$, with corresponding cable (un)rolling velocity $v_c[\mathrm{m/s}]$ and acceleration $a_c[\mathrm{m/s^2}]$. The angle that the pendulum makes with the vertical is denoted by $\varphi[\mathrm{rad}]$, its angular velocity is $\omega[\mathrm{rad/s}]$. We define the state vector $x = [x_t, v_t, x_c, v_c, \varphi, \omega]^\top$ and controls $u = [a_t, a_c]^\top$.

The height of the pendulum is $H = 0.5\,\mathrm{m}$, and the following bounds apply on states and controls:

$$\begin{gathered} -0.8\,\mathrm{m/s} \leqslant v_t \leqslant 0.8\,\mathrm{m/s} \\ 0\,\mathrm{m} \leqslant x_c \leqslant 0.5\,\mathrm{m} \\ -0.5\,\mathrm{m/s} \leqslant v_c \leqslant 0.5\,\mathrm{m/s} \\ -1\,\mathrm{m/s^2} \leqslant a_t \leqslant 1\,\mathrm{m/s^2} \\ -1\,\mathrm{m/s^2} \leqslant a_c \leqslant 1\,\mathrm{m/s^2}. \end{gathered} \tag{4.11}$$

Furthermore, we introduce a static obstacle that the pendulum should avoid touching, of the form

$$\begin{aligned} H - x_c\cos(\varphi) \geqslant & \\ & \frac{0.4}{3}(\arctan(200 * (x_t + x_c\sin(\varphi) - 0.5) \\ & - \arctan(200 * (x_t + x_c\sin(\varphi) - 0.7))). \end{aligned} \tag{4.12}$$

Optimal control problem We compare optimal control formulations (4.2) and (4.4), with the following parameters: $N = 25$, starting point is $\overline{x} = [0, 0, 0.4, 0, 0, 0]$, end point $x_{\text{end}} = [1, 0, 0.4, 0, 0, 0]$, and for the exponential formulation we take $\theta = 1.5$, $\Delta t = 0.1\,\mathrm{s}$. Constraints (4.11) apply to both formulations.

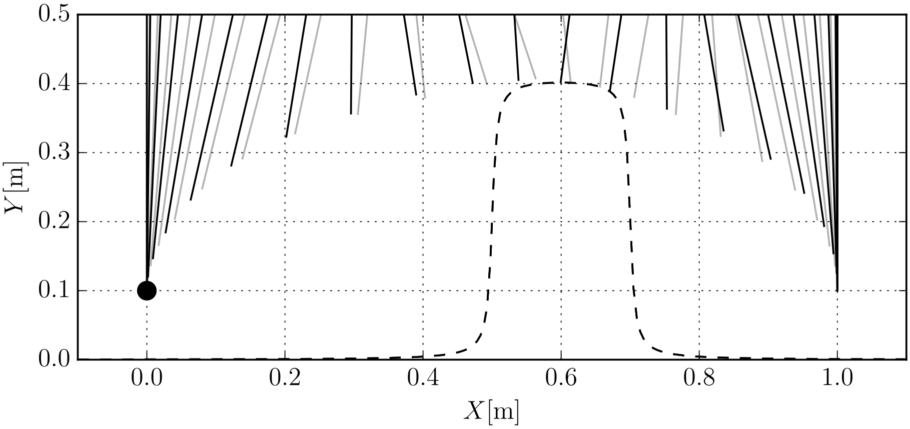
The results can be seen in Figure 4.2a. We note that both trajectories are almost equal, as they should be (the difference arises from the general non-uniqueness of time-optimal control in discrete time). The optimal time for the time scaling method is $T^\star = 2.219\,\mathrm{s}$, which compares with the optimal time of the exponential method, $N^\star \cdot \Delta t = 2.3\,\mathrm{s}$: the difference in optimal value for the time is smaller than one sampling time interval. Also the optimal state and control trajectories look very similar, the only difference being the horizontal acceleration a_t: the exponential weighting method decides to swing more when going over the obstacle. Although the trajectories could be brought closer to each other by additional control penalties, we show this result to illustrate the non-uniqueness in the time-optimal controller.

Processor-in-the-loop simulations We now show hardware in the loop (HIL) simulations of an efficient NMPC implementation. For this, we use the ACADO Code Generation Tool [Houska et al., 2011] with qpOASES [Ferreau et al., 2014] as QP solver. We then run this code on ABB's high performance controller AC 800PEC, which is typically used for time- and safety-critical applications in the power electronics domain. It features a dual-core CPU running at a clock speed of up to 1200 MHz as well as a field-programmable gate array. In our simulations, both the RTI method as well as the simulation model are running on a single CPU core at a clock speed of 800 MHz. Simulation results are obtained via Ethernet using a dedicated communication protocol.

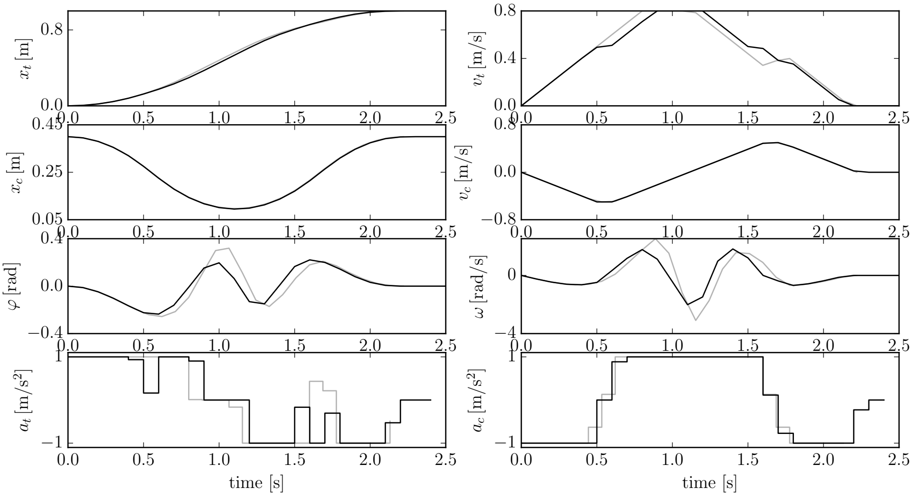
For the HIL-simulation, we show one run on the pendulum without constraint (4.12). We show the CPU times of the closed loop controller in Figure 4.3. As we can see, in the beginning there are a lot of active set changes in the active set QP solver (right hand axis), such that the CPU time taken for one RTI step are relatively high (more than 700 ms). However, there is a very steep drop afterwards. Ultimately, the number of QP iterations goes to zero as the pendulum arrives at its end point. It should be noted that the number of active set changes (and thus the CPU time spent in the QP solver) at the beginning of the run could be drastically reduced by first executing a few “warm-starting” NMPC runs, where the initial value is not updated.

4.6.3 Application: permanent magnet synchronous machines

As a second application, a very similar method as the one presented in this chapter has been applied to permanent magnet synchronous machines in a



(a) Optimal control trajectories in the $X-Y$ plane, with obstacle as in (4.12).



(b) Optimal state and control trajectories.

Figure 4.2: Solution to the optimal control problem of pendulum with varying length. In black, the exponential weighting method is plotted, in gray, the time scaling method.

master thesis by [Graber, 2018]. There, the state of the system needs to move time-optimally from one working point to another. Furthermore, the voltage and current constraints are elliptical in nature. As such, this is an interesting application, as it could combine two techniques presented in this

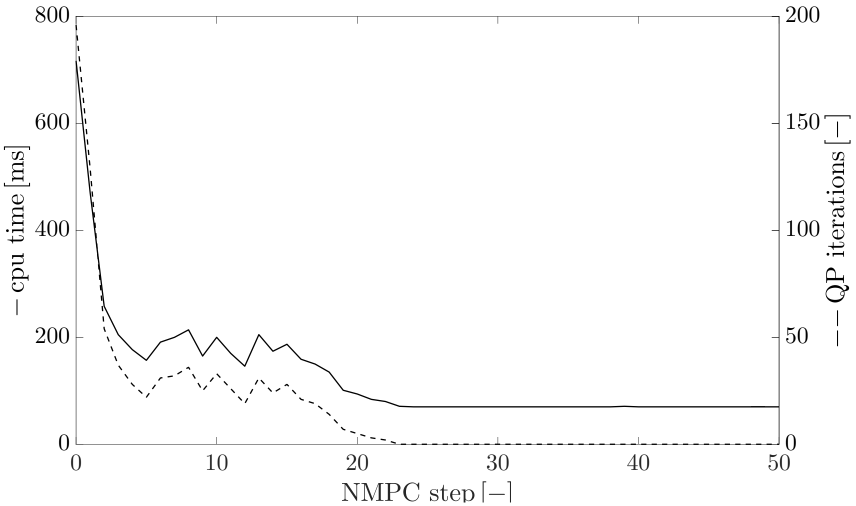


Figure 4.3: CPU times and number of QP iterations against number of steps of the NMPC.

thesis: the time-optimal NMPC technique from this chapter and the SCQP Hessian approximation of the last chapter. The preliminary results are promising, unfortunately no experimental results are available yet.

4.7 Summary

This chapter is the first of two on time-optimal NMPC. We introduced a stable time-optimal NMPC scheme for point-to-point motions based on an increasing exponential penalty. As an added benefit, the resulting cost function is convex and is easily implemented with a slack reformulation. We showed the usefulness of the method with an HIL simulation on an embedded platform.

Chapter 5

Time-optimal path following for robotic manipulators

> If you find a path with no obstacles, it probably doesn't lead anywhere.
>
> ---
>
> Frank A. Clarke, politician

In the last chapter, we saw a time-optimal NMPC method for point-to-point motions. Sometimes, a time-optimal control task is not about moving from point to point but about following or staying near a path [Verscheure et al., 2009], [Lipp and Boyd, 2014]. Examples include control of computer numerical control (CNC) machines in manufacturing, welding robots, but also control of autonomous aerial, land, underwater vehicles. The method presented in this chapter makes use of a prescribed path that needs to be followed, with some accuracy. Using a reformulation of the dynamics, our method is especially useful in the presence of obstacles on the path. The work in this chapter was presented at the Americal Control Conference [Verschueren et al., 2016b].

5.1 Introduction

We focus on time-optimal motion of robotic arms along predefined trajectories, in order to maximize productivity. From previous work we learn that generating time-optimal motion starting from a given geometric path can be beneficial.

The operator can choose a path that ensures collision avoidance and satisfies other geometric constraints in advance. Given a fixed path in the configuration space of the robot (which may require an inverse kinematics step), the remainder of the trajectory generation problem consists of the significantly simpler step of choosing the optimal timing *where* to be *when* on the given path [Bobrow et al., 1985; Shin and McKay, 1985]. We will call this the *decoupled* approach to time-optimal motion generation. An example of this is [Dahl and Nielsen, 1990], an online two-level method, where the secondary level modifies a nominal trajectory by performing a time scaling during motion. This idea is elaborated in [Olofsson, 2015] with a decoupling between lateral and longitudinal control. Other online approaches to path following go one step further and consider a geometric path solely as reference and generate an optimal feedback law that decides on both the timing and the distance to the reference using a nonlinear model predictive control (NMPC) approach [Faulwasser et al., 2009; Lam et al., 2010].

The geometric path in the workspace of the robot may not be completely strict and, in fact, limited deviations may be perfectly acceptable, for instance in tasks with a certain machining tolerance, or in orientation invariant tasks. The work in [Debrouwere et al., 2014] presents a time-optimal approach which allows deviation from the reference path, by projecting the robot dynamics on the reference path and introducing constraints on the forward position kinematics. As in [Debrouwere et al., 2014], we propose a trajectory generation methodology closely related to the decoupled approach. The main difference is that we do not project the robot dynamics onto a predefined path. Rather, we reformulate the dynamics *around* it; by doing so, we allow the end-effector of the robot to deviate from the path. This method relies on a so-called *spatial reformulation* (or *time transformation*) previously proposed in [Gao et al., 2012], [Frasch et al., 2013] for planar motions. The concept of this reformulation is to introduce a variable s that measures the progress of the motion in the workspace of our system and to transform the system dynamics to evolve with this progress instead of time. This approach is appealing, since it allows all static geometric aspects in the optimization problem to appear explicitly in the horizon of the optimal control problem. The price to pay is a nonlinear transformation of the system dynamics and the loss of the ability to explicitly define changing geometric properties in time. Examples of implementations of the spatial reformulation include research on driver assistance functions for high way driving of long heavy vehicle combinations (LHVCs) [van Duijkeren et al., 2015], and time-optimal nonlinear model predictive control (NMPC) on small-scale race cars [Verschueren et al., 2014].

In this chapter, we present a generalization of the time-transformation for paths in three-dimensional Euclidean space, applied to a reference path in the workspace of a serial-link robotic arm. The spatial reformulation enables us to express a natural formulation of an optimal control problem (OCP) for time and energy optimal motion, since travel time becomes a state variable of the system equations.

This chapter is organized in the following way. First we introduce and derive the so-called spatial reformulation for motion of the end-effector of a robotic arm with respect to a reference curve in the Euclidean workspace of the robot. Secondly, an OCP is introduced that is solved with two equivalent formulations; the first approach uses the novel technique presented in this section, the second takes the traditional approach of scaling the horizon and system dynamics linearly by an optimization variable. We illustrate the efficacy of the method to describe geometric constraints to facilitate e.g., collision avoidance. Thereafter we briefly elaborate on the implementation and the tools that were employed. The section is concluded with a discussion on the simulation results and a brief preview of future work.

In the remainder of this chapter, we will make use of the notation $(\cdot)' = \dfrac{\mathrm{d}(\cdot)}{\mathrm{d}s}$ and $\dot{(\cdot)} = \dfrac{\mathrm{d}(\cdot)}{\mathrm{d}t}$.

5.2 Spatial reformulation of robot dynamics

To illustrate the spatial reformulation, we apply it to an optimal motion planning task for a robotic arm. Let us consider a rigid-body n-DOF serial-link robotic manipulator. Recall that the motion equations of this kind of systems can be written in the form, cf. [Spong et al., 2006]:

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \dot{q} \tag{5.1a}$$

$$M(q)\frac{\mathrm{d}\dot{q}}{\mathrm{d}t} = u - C_o(q,\dot{q})\dot{q} - G_r(q), \tag{5.1b}$$

where $q, \dot{q}, u \in \mathbb{R}^n$ are the joint angles, joint velocities and actuation torques in the joints, respectively. $M(q)$, $C_o(q,\dot{q})$ denote the mass matrix and a matrix accounting for Coriolis and centrifugal effects, $G_r(q)$ is a vector of torques due to gravitation. Note that Coulomb friction and viscous friction are neglected as they are not readily taken into account in applying the spatial reformulation.

In this section, we will first present the way in which we represent the path and establish a formula for the progress along the path. We then use this relation to apply the spatial reformulation of the dynamics.

5.2.1 Path representation

Let $\zeta(t)$ be a continuous, sufficiently often differentiable curve in three-dimensional Euclidean space, assuming that the velocity vector $\dot{\zeta}(t) \neq 0$. We introduce the arc length $s(t)$ as the distance traveled along the path. The path $\Theta = \{\zeta(s) \in \mathbb{R}^3 : s \in [0, l] \rightarrow \zeta(s)\}$ is parametrized by its arc length

$$s(t) = \int_0^t \|\dot{\zeta}(x)\|_2 \, \mathrm{d}x. \tag{5.2}$$

Local properties of the curve are characterized by the curvature κ and the torsion σ. At each point s on Θ we define an orthonormal basis frame of three vectors $\mathcal{T}$, $\mathcal{N}$ and $\mathcal{B}$, referred to as the tangent, normal and binormal unit vectors. These unit vectors are defined by $\mathcal{T}(s) := \zeta'(s)$, $\mathcal{N}(s) := \mathcal{T}'(s)/\|\kappa(s)\|_2$ and $\mathcal{B}(s) := \mathcal{T}(s) \times \mathcal{N}(s)$ and satisfy the Frenet-Serret formulas, cf. [Guggenheimer, 1977]:

$$\mathcal{T}' = \kappa\mathcal{N}, \quad \mathcal{N}' = -\kappa\mathcal{T} + \sigma\mathcal{B}, \quad \mathcal{B}' = -\sigma\mathcal{N}. \tag{5.3}$$

Furthermore, let $p(t)$ be the vector of positional coordinates at fixed time t in the inertial world frame (forward position kinematics of the robotic manipulator), then the point on the path ζ closest to $p(t)$ is $\zeta(s^\star)$, where

$$e(s,t) = p(t) - \zeta(s) \tag{5.4}$$

$$s^\star = \arg\min_s \frac{1}{2}\|e(s,t)\|_2^2. \tag{5.5}$$

See Figure 5.1 for an illustration of the concept.

As is clear from (5.4)-(5.5), finding $s^\star(t)$ involves an optimization problem. Since it is undesired to embed this into a higher-level optimization problem, we attempt to find the temporal evolution of $s^\star(t)$ by looking at the optimality conditions. Recall that for unconstrained optimization, the first order necessary condition for optimality is:

$$0 = \frac{\mathrm{d}}{\mathrm{d}s}\left(\frac{1}{2}\|e(s,t)\|_2^2\right) \tag{5.6a}$$

$$= -e(s,t)^\top \zeta'(s). \tag{5.6b}$$

Consider that the position $s^\star$ is known at an initial time-point, we can enforce the solution to be optimal in time by setting the time derivative of the necessary

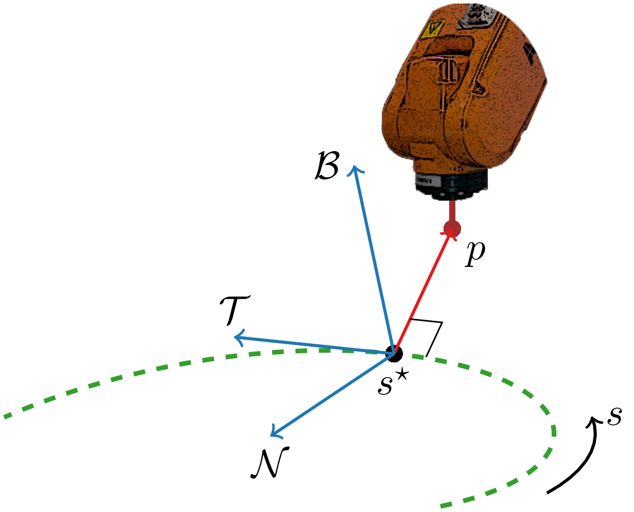


Figure 5.1: Illustration of the position of the end-effector with respect to the closest position of the path.

first order optimality condition (5.6) to zero, i.e.,

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}\left(e(s,t)^\top \zeta'(s)\right) \tag{5.7a}$$

$$= \left(v(t) - \zeta'(s)\dot{s}(t)\right)^\top \zeta'(s) + e(s,t)^\top \zeta''(s)\dot{s}(t), \tag{5.7b}$$

where $v(t) = \frac{\mathrm{d}p(t)}{\mathrm{d}t}$. This ultimately gives us a closed formula for the velocity of the point on the path closest to $p(t)$,

$$\dot{s}(t) = \frac{v(t)^\top \mathcal{T}(s)}{1 - \kappa(s)e(s,t)^\top \mathcal{N}(s)}. \tag{5.8}$$

For $e(s,t)$ sufficiently small, the denominator is positive and this expression is well-defined.

5.2.2 Spatial reformulation

We augment the state vector with $e = [e_x, e_y, e_z]^\top$. Additionally, the state $t(s)$ is included to keep track of the evolution of time. The state vector then reads $x = \left[q^\top, \dot{q}^\top, e^\top, t\right]^\top \in \mathbb{R}^{n_x}$. Using the established representation for the dynamics of the position of the end-effector $p(t)$ with respect to the path, we perform a spatial transformation of the equations of motion:

$$x' := \frac{\mathrm{d}x}{\mathrm{d}s} = \frac{\mathrm{d}x}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}s}, \tag{5.9}$$

with the state vector x. For $\dot{s}(t) \neq 0$, we have that $\frac{\mathrm{d}t}{\mathrm{d}s} = \frac{1}{\dot{s}(t)}$, and therefore

$$x' = \frac{1}{\dot{s}(t)}\dot{x}. \tag{5.10}$$

The resulting equations of motion are:

$$\frac{\mathrm{d}q}{\mathrm{d}s} = \frac{\dot{q}}{\dot{s}} \tag{5.11a}$$

$$M(q)\frac{\mathrm{d}\dot{q}}{\mathrm{d}s} = \frac{u - C_o(q,\dot{q})\dot{q} - G_r(q)}{\dot{s}} \tag{5.11b}$$

$$\frac{\mathrm{d}e}{\mathrm{d}s} = \frac{\dot{p}}{\dot{s}} - \mathcal{T}(s) \tag{5.11c}$$

$$\frac{\mathrm{d}t}{\mathrm{d}s} = \frac{1}{\dot{s}}, \tag{5.11d}$$

$$\tag{5.11e}$$

where the velocity of the end-effector can be written as $\dot{p} = J(q)\dot{q}$, with $J(q)$ the robot Jacobian, and $\dot{s}$ is obtained from (5.8).

The time transformation applied above is nonlinear, but is nevertheless appealing for two reasons. First, with the newly obtained state variable $t(s)$ and the corresponding first-order differential equation $\frac{\mathrm{d}t}{\mathrm{d}s}$, time-optimal motion is equivalent to minimizing t over the motion along the path. Secondly, the required knowledge about the temporal evolution of the $\mathcal{T}$, $\mathcal{N}$ and $\mathcal{B}$ vectors describing the local Frenet-Serret frame and many other geometric properties (such as obstacles) at time t become explicitly available in the integration method for x'.

5.3 Optimal control problem formulation

In order to illustrate the benefits of the spatial reformulation of the robot dynamics, we formulate a time-optimal trajectory generation problem. The OCP we intend to solve is stated as

$$\begin{aligned}
\underset{\substack{x(\cdot)\in\mathbb{R}^{n_x},\\ u(\cdot)\in\mathbb{R}^{n_u}}}{\text{minimize}} \quad & T = \int_{t=0}^{T} \mathrm{d}t && && \text{(5.12a)}\\
\text{subject to} \quad & \dot{x}(t) = \psi(x(t), u(t)) && \forall t \in [0, T] && \text{(5.12b)}\\
& g(p(t)) \leqslant 0 && \forall t \in [0, T] && \text{(5.12c)}\\
& \underline{u}(t) \leqslant u(t) \leqslant \overline{u}(t) && \forall t \in [0, T], && \text{(5.12d)}
\end{aligned}$$

with system dynamics, path constraints and torque bounds as constraints, respectively. This formulation is not readily passed to an optimization routine, as the time interval $[0, T]$ on which we solve the OCP is not independent of the optimization variables. Therefore, we proceed to pose two reformulations of the above OCP, which we will compare in the subsequent sections. The first is a time-optimal formulation using a rescaling of the time variable, with geometric constraints as nonlinear constraints. The second one makes use of our proposed spatial reformulation.

For both formulations, $x = [q^\top, \dot{q}^\top, e^\top, t]^\top$ is the state vector and u are the controls.

5.3.1 Time-scaled time-optimal control problem

We introduce $\tau = t/T$ as a scaling of the time variable, as in (1.14). Furthermore, for the formulation in time, we do not make use of a predefined path, so we take $\zeta(s) = 0$; this results in $e = p$. In this way, notation is consistent with the spatial formulation. Using the scaled time τ, the OCP in (5.12) becomes

$$\begin{aligned}
\underset{x(\cdot), u(\cdot)}{\text{minimize}} \quad & T = \int_{\tau=0}^{1} T \, \mathrm{d}\tau && && \text{(5.13a)}\\
\text{subject to} \quad & \frac{\mathrm{d}x(\tau)}{\mathrm{d}\tau} = T \cdot \psi(x(\tau), u(\tau)) && \forall \tau \in [0, 1] && \text{(5.13b)}\\
& g_\mathcal{T}(e(\tau)) \leqslant 0 && \forall \tau \in [0, 1] && \text{(5.13c)}\\
& \underline{u}(\tau) \leqslant u(\tau) \leqslant \overline{u}(\tau) && \forall \tau \in [0, 1]. && \text{(5.13d)}
\end{aligned}$$

Using the above linear rescaling, we are indeed capable of making the integration interval $[0, 1]$ independent of the decision variables x, u.

5.3.2 Spatial time-optimal control problem

From (5.10), we get the spatial dynamics as $\psi_s(x(s), u(s))/\dot{s}$. The optimal control formulation then reads as

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad T = \int_{s=0}^{s_\mathrm{f}} \frac{\mathrm{d}t}{\mathrm{d}s}\,\mathrm{d}s \tag{5.14a}$$

$$\text{subject to} \quad x'(s) = \frac{\psi_s(x(s), u(s))}{\dot{s}} \qquad \forall s \in [0, s_\mathrm{f}] \tag{5.14b}$$

$$g_s(e(s)) \leqslant 0 \qquad \forall s \in [0, s_\mathrm{f}] \tag{5.14c}$$

$$\underline{u}(s) \leqslant u(s) \leqslant \overline{u}(s) \qquad \forall s \in [0, s_\mathrm{f}], \tag{5.14d}$$

where we obtain $\dot{s}$ from (5.8).

Note that both OCPs (5.13) and (5.14) are equivalent to (5.12), as the objective function and the constraints remain equivalent after transformation of variables.

5.4 Simulation results

We show the performance of our time-optimal OCP with spatial reformulation of the dynamics on a simple three link robot manipulator of which the Denavit-Hartenberg parameters are shown in Table 5.1. We consider two different motion experiments: the first is a time-optimal point-to-point motion, where we compare the results of both the spatial reformulation and the linear time scaling. The second is a problem where no end effector position is specified, and we introduce a static obstacle.

Note that in the following, a small control regularization of the form $10^{-4}\sum_i u_i^2$ is added to the objective function in order to obtain smooth control trajectories.

5.4.1 Numerical algorithms

To numerically solve the optimal control problems of the previous section, we adopt the open-source CasADi [Andersson et al., 2018] framework. More specifically, we use the Python front-end to formulate the OCP as a nonlinear program (NLP) using Bock's multiple shooting method [Bock and Plitt, 1984] as a discretization method. To this end, we relied on the integrators `cvodes` and `idas` inside the SUNDIALS suite [Hindmarsh et al., 2005].The resulting NLP is passed to the open-source solver IPOPT [Wächter and Biegler, 2006].

Table 5.1: DENAVIT-HARTENBERG PARAMETERS FOR THE THREE-LINK ROBOT USED IN SIMULATION

	d [m]	θ [rad]	a [m]	α [rad]
Link 1	0.1	0	0	$\pi/2$
Link 2	0	$\pi/2$	1	0
Link 3	0	$-\pi/2$	0.7	0

The linear algebra subroutine calls were passed to the sparse solver `ma57` from the HSL library [HSL, 2011].

For the second part of the simulations, we use a 6-DOF robot model, which is constructed with the Python-based toolbox SympyBotics [Sousa, 2014].

5.4.2 Time-optimal point-to-point motion along a path

To illustrate the equivalence of both methods discussed in Section 5.3, we solve the OCPs (5.13)-(5.14) for a simple time-optimal motion planning task. The robot end-effector travels from a starting point p_0 to a fixed final position p_f, staying inside a certain space, defined as the space between two cylinders, see Figure 5.2.

This constraint might be posed for the time-dependent formulation as

$$\underline{E} \leqslant \|[e_x(\theta), e_y(\theta)]^\top\|_2 \leqslant \overline{E}, \quad \forall\tau \in [0,1], \tag{5.15a}$$

$$\underline{Z} \leqslant e_z(\theta) \leqslant \overline{Z}, \qquad \forall\tau \in [0,1]. \tag{5.15b}$$

Recall that, for the time-domain formulation, e denotes the vector from the origin to the end-effector. For the spatial formulation the constraints read as

$$\underline{E_s} \leqslant e^\top \mathcal{N}(s) \leqslant \overline{E_s}, \quad \forall s \in [0,\pi/2], \tag{5.16a}$$

$$\underline{Z} \leqslant e_z(\theta) \leqslant \overline{Z}, \qquad \forall s \in [0,\pi/2], \tag{5.16b}$$

with

$$\begin{aligned}
\mathcal{T}(s) &= [-\sin(s), \cos(s), 0]^\top, \\
\mathcal{N}(s) &= [-\cos(s), -\sin(s), 0]^\top, \\
\mathcal{B}(s) &= [0, 0, 1]^\top, \\
\kappa &= 1.0\,\mathrm{m}^{-1}, \\
\sigma &= 0.0\,\mathrm{m}^{-1}.
\end{aligned}$$

The graph in Figure 5.2 corresponds with the values $\underline{E} = 0.8\,\mathrm{m}$, $\overline{E} = 1.2\,\mathrm{m}$, $\underline{E_s} = 0.2\,\mathrm{m}$, $\overline{E_s} = 0.2\,\mathrm{m}$, $\underline{Z} = -0.1\,\mathrm{m}$, $\overline{Z} = 0.1\,\mathrm{m}$.

One advantage of the spatial reformulation follows from the comparison of (5.16a) and (5.15a): in the latter case, the path constraint is convex, in the first it is not. If convexity holds for the constraint functions, convergence of the NLP solver is often accelerated.

We compare the solution of the two different formulations in Figure 5.3-5.4. The solution trajectories of both methods are shown as a function of time. The torque trajectories clearly show that the methods result in different discretizations. The spatial formulation yields a solution that takes longer time-steps in the beginning of the interval than the time-based formulation, which takes equidistant steps. It is clear from the figure that the resulting trajectories for the lateral and vertical deviation from the centerline of the cylindrical path are the same, up to the different discretizations. For a finer discretization grid, these differences vanish.

Another interesting remark is that the constraint on the first joint torque is active throughout the entire interval, as we would expect in a time-optimal motion. In addition, the path constraints are seen to be active most of the times as well, except to meet the initial and end point constraints. To conclude the comparison, we examine the NLP solver convergence. In this particular example, the time-optimal OCP was solved faster with the spatial formulation than the time formulation: the interior point solver needed 36 iterations (0.094 s) instead of 57 (0.191 s). More experiments need to be carried out to confirm this as a general rule.

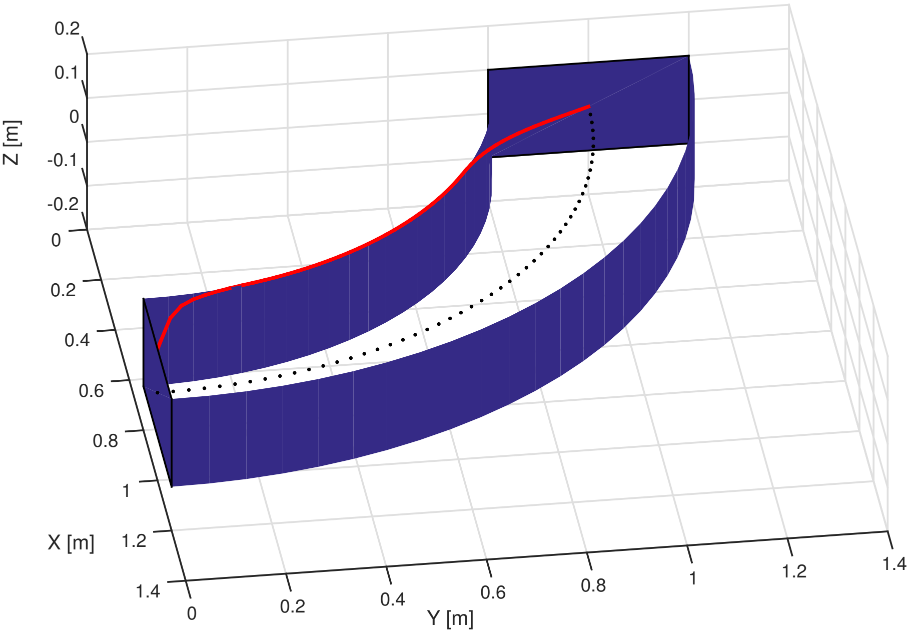


Figure 5.2: Time-optimal solution (red) of optimal control problems (5.13) and (5.14) with cylindrical path constraints. The constraint space is delimited with dark blue faces, the centerline is shown in black.

5.4.3 Time-optimal obstacle avoidance

A major advantage of employing the time transformation instead of the conventional formulation, is the natural way to insert geometric constraints. Also, this method is suited for arbitrary paths in space, not just analytical ones as in the above example. Both properties are made apparent in the following simulation result, where we consider a 6-DOF ABB IRB120 industrial robot (Figure 5.5) [ABB, 2015]. The simulation experiments are based on a dynamic model of this robot that was made available by ABB, albeit without taking friction into account.

In the following example, we perform a time-optimal motion planning task. The robot has to follow a given path within given tolerances as fast as possible, avoiding a static cylindrical obstacle at the end of the path. Additionally, we constrain the end-effector to point vertically down by imposing additional constraints on the forward orientation kinematics. The path specified (cf. dotted line in Figure 5.6a) is a planar path with piecewise constant curvature and

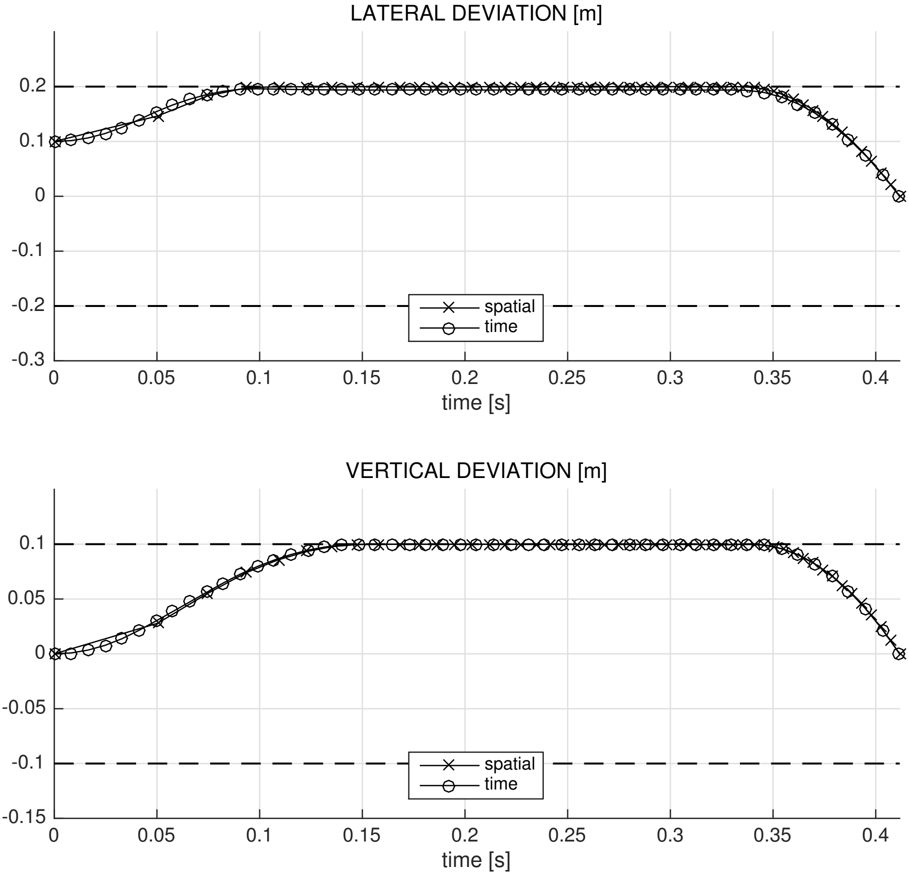


Figure 5.3: The lateral and vertical deviation are taken with respect to the centerline between the two cylinders.

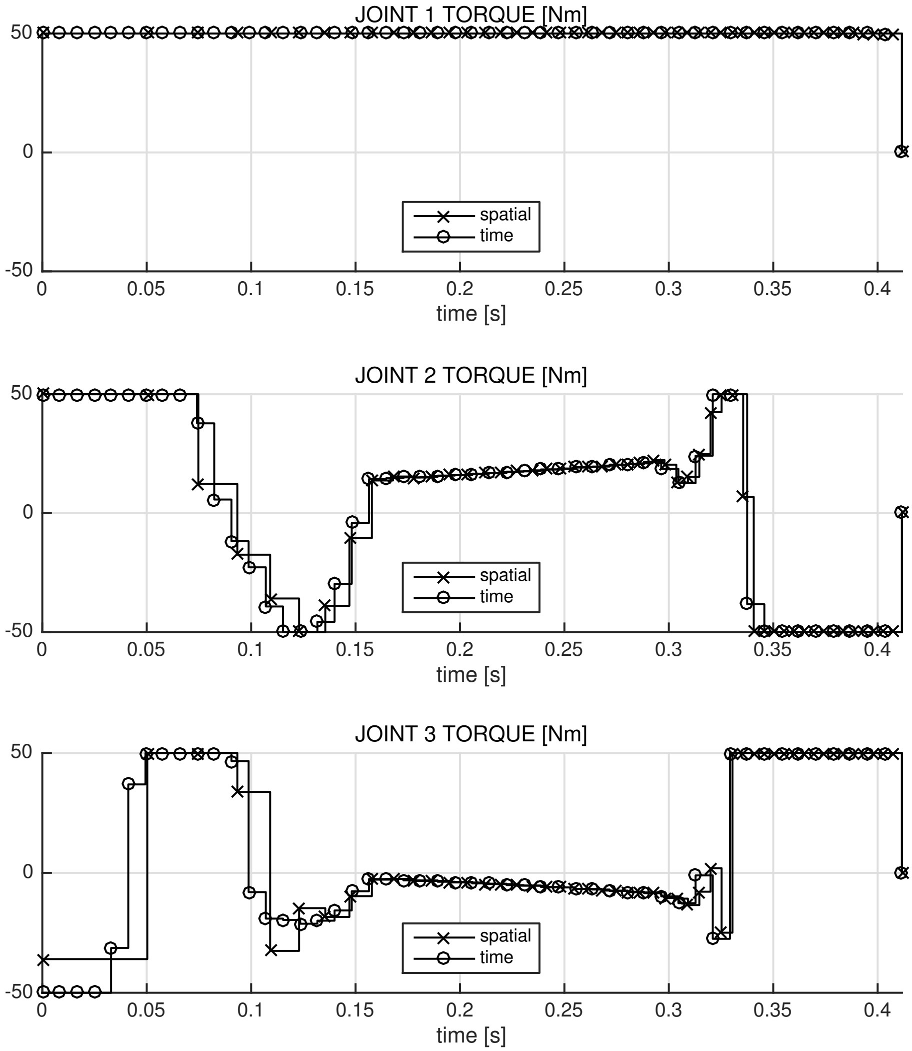


Figure 5.4: Comparison of the solution of the two OCP formulations (5.13) and (5.14). The torques in the robot joints are shown. The joint torque bounds are taken to be $-50\,\mathrm{Nm} \leqslant \tau \leqslant 50\,\mathrm{Nm}$ for both formulations.

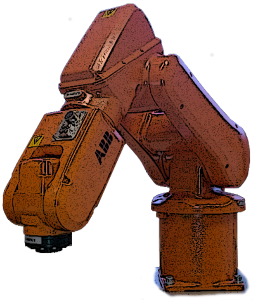
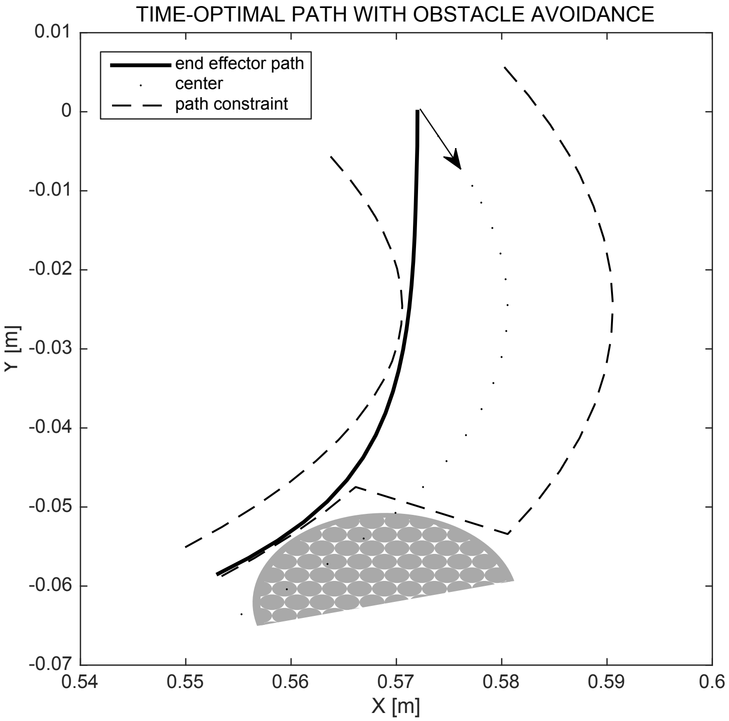


Figure 5.5: Sketch of the ABB IRB120 industrial robot. During the simulation, the end-effector points down (as shown).

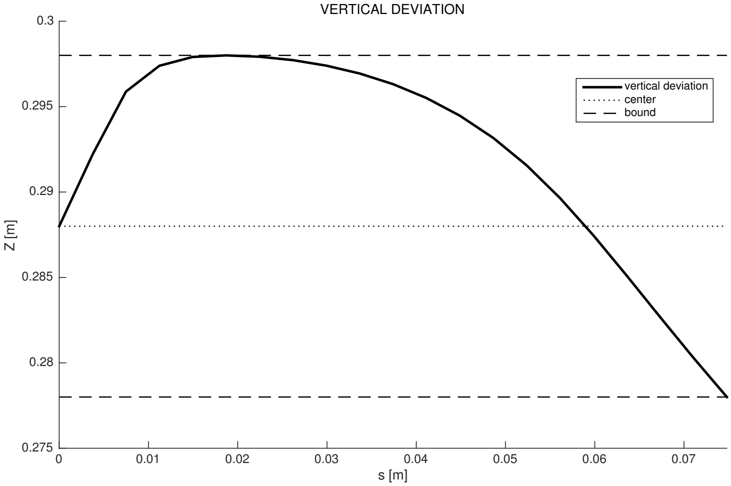
given initial tangent and normal vectors, both normalized. The binormal, which is vertical at all points, is taken arbitrarily as $\mathcal{B}_0 = [0, 0, 1]$. The cartesian coordinates of the path can be retrieved by integrating the Frenet-Serret formulas (5.3) forward in the path coordinate s, starting at $s_0 = 0\,\mathrm{m}$. Note that this integration can be incorporated in the state equations (which are already in function of s) by augmenting the dynamics with (5.3).

At the end of the pre-specified path, there is a static cylindrical obstacle narrowing down the maneuvering possibilities of the end-effector. In the spatial formulation, this constraint will amount to increasing the lower bound of the constraint (5.16a); it remains a simple bound. The resulting time-optimal path can be seen in Figure 5.6a. Again, we see that the path touches the inner path constraint as we expect for time-optimal trajectories. Note that in this case, the final position of the end effector is not completely specified, only constrained to lie in the plane spanned by $\mathcal{N}$ and $\mathcal{B}$. The vertical deviation is depicted in Figure 5.6b. Also here the constraints on the vertical deviation become active at some point in the motion.

Note that the time-domain formulation (5.13) is not readily applicable in this example, because the path constraints are not straightforward to compute in the time domain for arbitrary paths, in contrast with paths with an analytical description, as in the first example. This shortcoming originates from the fact that we do not know beforehand where in the configuration space the end effector will be at which point in time, i.e. the relation between t and s is not stated explicitly. An expression of the path constraints in the time domain results in possibly very nonlinear constraints; this holds even more so for static obstacles as in the above example.



(a) Top view of the time-optimal path. Near the end of the path, there is a static obstacle narrowing the available space of the robot end-effector.



(b) Vertical deviation from the $z = 0.288\,\mathrm{m}$ horizontal plane.

Figure 5.6: Time-optimal optimal control problem with obstacle avoidance around an arbitrary central path with piecewise constant curvature.

5.5 Summary

The main contribution of this chapter is a path-parametric system reformulation for robotic manipulators. It is aimed at convenience for path specification, and exhibits a natural way to impose geometric constraints. These advantages have been shown in simulations in relation to time-optimal control using a time-scaling approach, which does not possess these properties.

Chapter 6

`acados` – a modular open-source framework for fast embedded optimal control problem solvers

> Premature optimization is the root of all evil.
>
> Donald Knuth, computer scientist

Software development and numerical optimization share a long and interesting history. In fact, the first electronic computers were used to solve LP problems. Throughout the years, software for a wide variety of optimization problems has been produced, both in industry and academia. A general overview of software for solving optimization problems would be extensive, and calls for its own monograph.

Also in embedded optimization, software plays a central role of making algorithms usable in the real world. In this chapter, we present a new software package, called `acados`, which has been in development for approximately two years. People that played a role in its conception and realization are, apart from the author and his supervisor, Rien Quirynen, Dimitris Kouzoupis, Gianluca Frison, Niels van Duijkeren, Andrea Zanelli, Dang Doan, Jonathan Frey and Branimir Novoselnik. Throughout the chapter, we will mention some important (embedded) optimization packages. A brief, non-exhaustive list can be found in Table 6.1. Please note that not all of these packages are still under active development.

Table 6.1: SOFTWARE PACKAGES FOR EMBEDDED OPTIMIZATION.

Software	Year	Latest Reference	Targets	License
MUSCOD-II	1997	[Leineweber et al., 2003]	NMPC	proprietary
MPC Toolbox	1998	[MathWorks, 2005]	LQMPC	proprietary
AutoGenU	2000	[Ohtsuka, 2004]	NMPC	propietary
OOQP	2001	[Gertz and Wright, 2003]	QP	propietary
MPT3	2003	[Herceg et al., 2013]	expl. MPC	GPL
Hybrid toolbox	2003	[Bemporad, 2003]	expl. MPC	proprietary
qpOASES	2006	[Ferreau et al., 2014]	QP	LGPL v2.1
PQP	2008	[Di Cairano et al., 2013]	QP	proprietary
CVXGEN	2009	[Mattingley and Boyd, 2012]	LP, QP	proprietary
ACADO Codegen	2009	[Houska et al., 2011]	NMPC	LGPL v3
FiOrdOs	2011	[Ullmann, 2011]	QP	GPL v3
FORCES	2011	[Domahidi et al., 2012]	QP, QCQP	proprietary
ECOS	2013	[Domahidi et al., 2013]	SOCP	GPL v3
GRAMPC 1.0	2014	[Käpernick and Graichen, 2014]	NMPC	LGPL v3
qpDUNES	2014	[Frasch et al., 2015]	LQMPC	LGPL v3
DuQuad	2014	[Kvamme, 2014]	QP	unknown
HPMPC	2014	[Frison et al., 2014]	LQMPC	LGPL v2.1
VIATOC	2015	[Kalmari et al., 2015]	NMPC	GPL v3 and LGPL v3
PIPS-NLP	2016	[Chiang et al., 2017]	NLP	3-clause BSD
Forces NLP	2017	[Zanelli et al., 2017a]	NMPC	proprietary
OSQP	2017	[Stellato et al., 2017a]	QP	Apache v2.0
FalcOpt	2017	[Torrisi et al., 2017]	NMPC	MIT
HPIPM	2017	[Frison, 2017]	LQMPC, QP	GPL v3 (CE)
ODYS	2017	[Cimini and Bemporad, 2017]	QP	proprietary
protoip	2017	[Khusainov et al., 2017]	NMPC	unknown
SPLIT toolbox	2017	[Shukla et al., 2017]	LQMPC	unknown
GRAMPC 2.0	2018	[Englert et al., 2018]	NMPC	LGPL v3
PANOC.jl	2018	[Sathya et al., 2018]	NMPC	unknown
nmpc-codegen	2018	[Melis and Patrinos, 2018]	NMPC	LGPL v3
PRESAS	2018	[Quirynen et al., 2018]	QP	proprietary
ParNMPC	2018	[Deng and Ohtsuka, 2018]	NMPC	unknown
qrqp	2018	[Andersson and Rawlings, 2018]	QP	LGPL v3
acados	2018	[Verschueren et al., 2018]	NMPC	LGPL v3

6.1 Introduction

Embedded optimization, according to the definition in [Ferreau et al., 2017], is solving optimization problems *autonomously* and *with limited resources*. A big role in bringing MPC to embedded applications is played by the implementation of efficient embedded optimal control methods, such as MPT [Herceg et al., 2013] (explicit MPC), FORCES [Domahidi and Perez, 2013], qpOASES [Ferreau et al., 2014] and the ACADO Code Generation Tool [Houska et al., 2011]. Software development for embedded optimization is a very active field (see Table 6.1) and many software packages were successfully applied on real-time and embedded systems [Ferreau et al., 2006; Kraus et al., 2013; Liniger et al., 2015; Zanelli et al., 2018]. For an overview please consult [Ferreau et al., 2017].

The challenge in developing software for embedded optimal control lies in the trade-off between flexibility, memory usage and speed. Many of the above software packages are based on *automatic code generation*. They successfully achieve flexibility from a high-level perspective, for the user to design their algorithm in a high-level language like MATLAB or `Python`. If there is a need to run the control algorithm on embedded hardware, it can be readily translated to a more embeddable language, often `C` or `C++`.

Let us focus on a particular tool, namely the `ACADO` Code Generation Tool. One advantage of `ACADO` code generation is the possibility to have *tailored* code: it is written by the software tool for one specific application only. This often incurs lower memory usage and/or lower running times. One disadvantage is the loss of flexibility: the code is generated for the fixed dimensions of the problem only. Any conceptual change requires regeneration and recompilation of the solver. Another disadvantage of code generation is that the resulting code is often hard to read by humans, which makes the process of debugging the embedded code challenging or impossible. Furthermore, it can be hard to predict if an automatic code optimization strategy, like loop unrolling, is beneficial in general or if it is perhaps counterproductive for some types of problems. In some cases, a compiler could do better. Lastly, performing automatic code generation is most effective for the most heavily utilized subroutines, typically a small number of linear algebra operations. The contribution of code generating other parts of the algorithm is less obvious, and is not always justified.

For the above reasons, `acados` does not rely on automatic code generation. Instead, for the linear algebra operations, we make use of the recently developed linear algebra package `BLASFEO` [Frison et al., 2018] which can outperform triple-loop `C` implementations by one order of magnitude. Furthermore, we perform no *a priori* automatic code optimizations, we delegate that task to the optimizing compiler.

Another challenge for embedded optimal control software is related to the process of software development. Often, to not sacrifice speed of execution and/or memory footprint, embedded optimal control software uses global data and suffers from tight coupling between algorithmic components. This might lead to a codebase that is difficult to understand, maintain, and extend. We choose, as opposed to other embedded optimal control software packages, to avoid these pitfalls by not unnecessarily sacrificing maintainability and readability of the codebase for a small gain in efficiency and/or a reduction of memory footprint.

Furthermore, we organize our code in a modular fashion, with formal interfaces between the different algorithmic components. This allows for a straightforward way of interchanging solvers, routines, and libraries needed for the embedded control algorithm.

A final aspect of embedded optimal control software that affects flexibility, memory and runtime is the choice of modeling language and derivative generation tool. A myriad of modeling languages exist, e.g. `Mathematica`, `sympy` or the `MATLAB Symbolic Toolbox`. Many of these languages make use of expression trees to represent mathematical functions, which potentially leads to a large code size, high memory usage and slow evaluation of higher-order derivatives for non-trivial models. On the contrary, the `CasADi` [Andersson et al., 2018] modeling language is based on expression *graphs*. This leads to shorter instruction sequences, and therefore to smaller, typically faster code, and thus is more usable in embedded applications. Also, it is free and open-source software. For these reasons, we choose `CasADi` for modeling nonlinear functions and system models. Additionally, we allow the use of hand-written (or code-generated) models as `C` source files.

The contribution of this chapter is a new software package, called `acados`. It offers the following features:

- efficient optimal control algorithms written in `C`,
- modular architecture enabling rapid prototyping,
- interfaces to `Python` and MATLAB,
- linear algebra based on `BLASFEO` [Frison et al., 2018],
- compatible with `CasADi` [Andersson et al., 2018],
- deployable on a wide variety of embedded devices.

The remainder of this chapter is organized as follows. The software package `acados` is introduced in Section 6.2. Various numerical experiments and comparisons are carried out and presented in Section 6.3.

6.2 The `acados` software package

`acados` implements some of the optimization methods mentioned in the previous sections. It is meant to be user-friendly at a high level, and efficient at a low level. In order to balance these properties, we developed a core library written in `C` which exposes functionality to the `Python` and Matlab interfaces. In this section, we first discuss the inner workings of this core module, we then describe internal and external interfaces that are crucial for usability, and we conclude with some example of the syntax in both `Python` and Matlab.

6.2.1 The `acados` core library

Typically, embedded optimization algorithms (e.g. as presented in Chapter 1) are built up in a modular fashion. For example, there is a clear contract between an NLP solver and an integrator. The integrator expects a linearization point w^i, and returns a simulated trajectory, along with first/second-order sensitivities:

$$\text{NLP solver} \underset{\text{sim.,sens.}}{\overset{\text{lin. point}}{\rightleftarrows}} \text{integrator}$$

Similar diagrams can be drawn for all other algorithmic components, including (partial) condensing, QP solver, function evaluations etc. Each of these algorithmic components are modeled within `acados` as *modules*. Some modules can be used as standalone modules, or in combination with others, for instance, depending on the choice of algorithm, an NLP solver will make use of some or all of the other modules.

In Table 6.2, we see an overview of all modules currently present in `acados`, together with the implemented variants. The underlying design principle is that each module has an interface that is general enough to be extended by implementing a new variant of the module.

Moreover, it is an important design choice that all modules are identical in their signature. That way, all modules look similar to the users of `acados`. For developers, it should be straightforward to extend `acados` with another module. The signature is as follows (in `C` syntax):

```c
int <solver>(void *config,
    <module>_dims *dims,
    <module>_in *in,
    <module>_out *out,
    void *opts,
    void *mem,
    void *work);
```

Table 6.2: Overview of the software modules present in `acados`.

Module	Variants

OCP QP	`qpDUNES`
	`HPMPC`
	`HPIPM`
	`OOQP`

Dense QP	`qpOASES`
	`HPIPM`
	`OOQP`

Condensing	Full condensing Partial condensing
Integrator	Explicit Runge-Kutta Implicit Runge-Kutta (Gauss-Legendre)
OCP NLP	Gauss-Newton SQP SCQP Exact-Hessian based SQP RTI
Nonlinear function	`CasADi` code-generated functions Hand-written `C`-code

Here, `<module>` stands for the name of the module at hand, for example `ocp_qp` for QP problems with optimal control structure or `sim` for integration problems, and `<solver>` is a placeholder for a function implementing the specific solver for problems corresponding to this module, e.g. `ocp_qp_hpipm` (interface to `HPIPM` solver) or `sim_erk` (explicit Runge-Kutta method), etc. Each module returns an `int` which denotes a solver-specific error status – zero means successful completion in this context. All of the input arguments are pointers, where the use of a null pointer denotes that the argument is not used in this module. Each of the arguments comes with a set of helper functions, called `*_calculate_size`, computing the size (in bytes) of the `struct` pointed to, as well as a set of functions, called `*_assign`, to initialize a block of memory. The usual workflow is as follows:

```
int num_bytes = <module>_solver_config_calculate_size();
void *mem_ptr = malloc(num_bytes);
void *<module>_config = <module>_solver_config_assign(mem_ptr);
```

By making this design choice, all memory allocations happen before execution of any of the `acados` solvers and no dynamic memory allocation is needed. This eliminates the risk of leaking memory, as the user of (or the interface to) the core library is fully responsible for allocating and deallocating the memory used by `acados`.

We will now look into the different arguments in a bit more detail.

`config` - a pointer to `struct` containing function pointers to the above-mentioned helper functions `*_calculate_size` and `*_assign`, among others.

`dims` - a pointer to `struct` with the data pertaining to the dimensions of the problem.

`in` - a pointer to `struct` with the input data to the specific solver. For a given module, this `struct` is identical for all variants.

`out` - a pointer to `struct` with the solution data from the solver. Identical for all solvers for this module.

`opts` - a void pointer that can be cast to a pointer to `struct` containing algorithmic options needed by the solver, and thus solver-specific.

`mem` - a void pointer that can be cast to a pointer to `struct` with additional memory used by the solver. This memory is to be preserved between calls to the solver.

`work` - a void pointer that points to a block of memory that functions as 'scrap' space, i.e. this memory does not have to be preserved between calls to the solver.

We remark that the first four arguments are absolutely necessary for all solver variants of a module, the last three are optional, i.e. when they are not needed by a specific solver, they can be `NULL`. Note that the layout of the `mem` and `work` elements may depend on which solver options the user chooses.

Some modules comprise other modules. For example, an SQP solver for optimal control problems might need an integrator, which is on its own a proper `acados` module. In this context, we call the integrator a *submodule*. Each of the arguments above, `dims, in` etc., have fields corresponding to submodules. For an NLP solver, the relation between it and the submodules are depicted in Figure 6.1. We remark that the calculation of the memory size of a module with submodules is done recursively, i.e. calling the `calculate_size` function on the top module returns the required memory size of the top module and all of its submodules, and submodules of submodules, etc. This allows users to allocate all the memory outside of `acados`, by design.

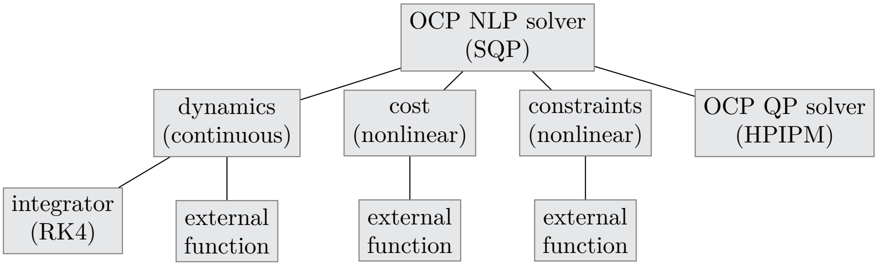


Figure 6.1: Example of the relation between modules and submodules in `acados` for one specific algorithm.

The *core* library of `acados` contains mostly what has been described in this section: a collection of modules, each with corresponding data types and variants of solvers, as well as helper functions for memory management. Using the core library directly can be cumbersome and error-prone, as many details need to be taken into account: it is designed to be as flexible as possible. To cater to the needs of the end user, we offer high-level interfaces, which are described next. The connection between the core library and its interfaces is depicted in Figure 6.1.

6.2.2 The C interface

The `C` interface is mainly responsible for two things: passing options to solvers and memory management.

Choosing solvers When working with the core library, all functions are specific to one variant of a module: when solving a QP with, say, `qpOASES`, the code will refer to `struct`s like `dense_qp_qpoases_memory`, `dense_qp_qpoases_opts`, etc. We want to make abstraction of this to facilitate switching solvers easily. To this end, for each module we define a ‘plan’. A plan is a `struct` that contains a number of fields representing the choice of a particular combination of solvers. For example, the plan for an SQP-type method with Gauss-Newton Hessian approximation, for a problem discretized with the RK4 integrator using `HPIPM` as an underlying QP solver, reads as

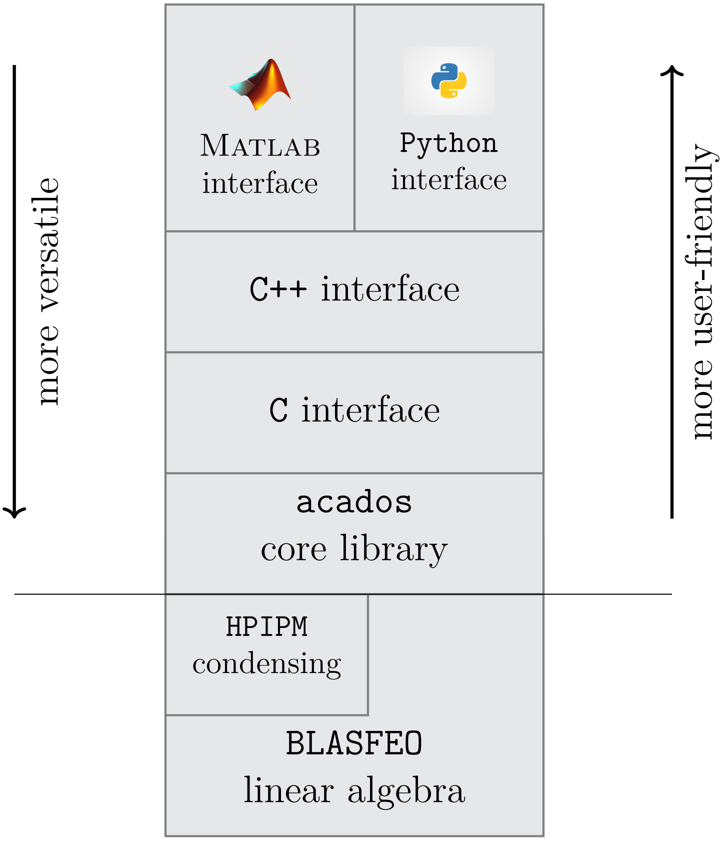


Figure 6.2: The interplay between the `acados` dependencies, the ‘core’ `C` library and its interfaces.

```
ocp_nlp_solver_plan plan = {
    {PARTIAL_CONDENSING_HPIPM},
    {ERK, ERK, ERK, ...},
    SQP_GN,
    {LINEAR_LS, ...},
    {CONTINUOUS_MODEL, ...},
    {BGH, ...},
};
```

Here, the arrays should be of the correct length (omitted for brevity, with a slight abuse of notation). As a general rule, solvers that make use of other modules should include them in their plan.

Passing options Ideally, options are passed as associative arrays (sometimes called dictionaries). Unfortunately, in `C` we do not have those at our disposal. For this reason, we manipulate a specific options `struct` with functions taking a textual representation of the option via a string. The string encodes both the module that the option belongs to, as well as the name of the option. For example,

```
options_set_int(void *opts, "qpoases.warm_start", 1);
```

A similar function `options_set_double` exists for passing options that take a floating-point value, such as solver tolerances.

Memory management Allocating memory 'manually' as described above can quickly become cumbersome. For this reason, we make available a few routines that automate that process. To this end, in the `C` interface each module from the core library is mirrored by an additional function with signature

```
<module>_solve(<module>_solver *solver,
    <module>_in *in,
    <module>_out *out);
```

`solver` is a pointer to an object that encapsulates the data needed other than input and output. By doing so, we reduce the amount of boilerplate code. Pseudocode for a typical workflow could look as follows:

```
<module>_config *config = <module>_config_create(plan);
<module>_dims *dims = <module>_dims_create(N);
<module>_opts *opts = <module>_opts_create();

// ...
// setup options in opts
// ...

<module>_solver *solver = <module>_create(config, dims, opts);

int status = <module>_solve(solver, in, out);
```

6.2.3 The C++ interface

For non-expert users of embedded NMPC software, writing `C` code manually can be error-prone and tedious. Therefore, we offer interfaces to two popular languages for technical computing: `Python` and MATLAB. As such, we created a small domain-specific language within each of these frameworks. To keep maintenance work to a minimum, we automatically generate these interfaces using `SWIG` [Beazley, 2003]. To this end, a relatively small `C++` wrapper around the `C` interface of `acados` has been developed. This `C++` wrapper relies heavily on the `C++` Standard Library, most notably on `std::string`, `std::vector` and `std::map`. These are all readily interfaced with `Python` and MATLAB via `SWIG`. Each module of acados is modeled by its own `C++` class. As is depicted in Figure 6.2, each class makes use of the `C` interface of `acados`.

Passing options In `C++`, solver options are defined to be

```
std::map<std::string, option_t *> solver_options;
```

where `option_t` is a pure virtual base class. It is subclassed by the template class `option<T>` with `T` the underlying type, for example `option<int>`, `option<double>`, or `option<std::map<std::string, option_t *>`. The latter allows us to define options for submodules, for example when passing options to a QP solver used within an SQP-type solver.

Modeling language The `C++` (and MATLAB, `Python`) interfaces of `acados` use `CasADi` as a modeling language, which is itself a `C++` code. As such, it is readily interfaced with `acados`. Moreover, the `C++` interface of `acados` directly calls the code generator of `CasADi` in `C++` and compiles/links the model code into a shared library, ready to be used by the MATLAB or `Python` interface. An additional benefit of using `CasADi` is that the solution behavior of `acados` can be easily compared with the solutions coming from the numerous optimization tools interfaced with `CasADi`.

We remark that model equations and other nonlinear functions are called from `acados` in a completely language-agnostic way: `acados` is at no point aware of which modeling tool is being used. One benefit is that this facilitates self-written models (in `C/C++`), which are also completely compatible with `acados`. However, they should conform to a certain calling convention similar to the ones of `CasADi` functions.

6.2.4 The `Python` and Matlab interfaces

In Figure 6.3, we see an example of the `acados` syntax from `Python` and MATLAB, in which we describe how to formulate a linear MPC problem. We note that the `Python` and MATLAB syntax are almost identical, where the differences arise from formatting and from language-specific constructs like lists and dictionaries in `Python` and cell arrays and structures in MATLAB. In either language, an `acados` module is modeled as an object which you can construct by passing the dimensions. On this object, (e.g. `qp`), different operations can be invoked in order to build up the optimization problem formulation with the correct weighting matrices, dynamics and bounds. After initialization of the solver (by passing the name of the variant – in this case `qpoases`), the solver is ready to be used in an MPC loop. The syntax is meant to be easy to read and remember. One additional benefit of operating on objects (instead of following an approach with functions), is that the user can use tab-completion on the objects to get a quick overview of the available functionality.

6.3 Numerical results

This section consists of a few numerical experiments with `acados` and comparisons to other embedded optimization software packages. We discuss performance on the nonlinear chain-of-masses problem, and show one closed-loop experiment on an embedded platform.

Figure 6.3: Listing of a simple linear MPC example using the `Python` interface to `acados`.

```
------------------------------------------
from numpy import array, diag, inf

from acados import ocp_qp

qp = ocp_qp(N=5, nx=2, nu=1)

# specify OCP
qp.set_field('A', array([[1, 1], [0, 1]]))
qp.set_field('B', array([[0], [1]]))
qp.set_field('Q', diag([1, 1]))
qp.set_field('R', diag([1]))
qp.set_field('Q', N, diag([10, 20]))

# specify bounds
qp.set_field('lbx', array([0.5, -inf]))
qp.set_field('ubx', array([3.0, +inf]))

# specify initial condition
x0 = array([1.1, 1.1])
qp.set_field('lbx', 0, x0)
qp.set_field('ubx', 0, x0)

# initialize solver
qp.initialize_solver('qpoases')

# simulate MPC loop
while True:
    qp.set_field('lbx', 0, x0)
    qp.set_field('ubx', 0, x0)

    output = qp.solve()
    u_opt = output.controls()

    x0 = measurement(u_opt[0])

------------------------------------------
```

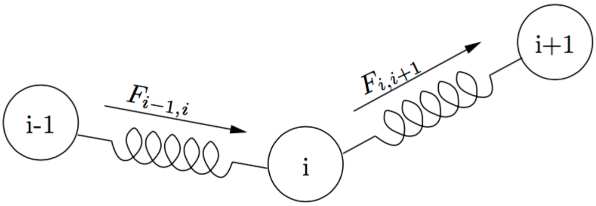


Figure 6.4: Forces in the springs between the masses in the example of Case Study 1. Replicated from [Wirsching et al., 2006].

6.3.1 Case study 1: Chain of Masses

As a benchmarking problem, we take the chain-of-masses problem as presented in [Wirsching et al., 2006]. It is useful in the sense that the problem is simple enough to understand intuitively, yet complicated enough to get non-trivial results from a range of different solvers. Also, by increasing the number of masses, one could easily compare behavior for different numbers of states, without changing much code.

System description

The control objective in this example is to stabilize the motion of a chain of $M = 5$ balls with mass m connected by springs to an equilibrium position. The mass on one end of the string is fixed at $(0, 0, 0)$. We can freely move the spring on the other end.

Let p_i be the position of mass i, for $i = 1, \ldots, M$. The model equations can then be derived as follows. From Hooke's law, we know that (see Figure 6.4)

$$F_{i,i+1} = D\left(1 - \frac{L}{\|p_{i+1} - p_i\|}\right)(p_{i+1} - p_i),$$

with each spring having spring constant D and rest length L.

This allows us to write the equations of motion for the middle balls, which read as

$$\ddot{p}_i = \frac{1}{m}(F_{i,i+1} - F_{i-1,i}) + g_z, \quad i = 2, \ldots, M-1,$$

with g_z the gravitational acceleration vector. For the free ball, we assume we control the velocity directly:

$$\dot{p}_M = u,$$

with $u \in \mathbb{R}^3$.

We now introduce a state space formulation with states

$$x = [p_2^\top, p_3^\top, \dots, p_{M-1}^\top, p_M^\top, v_2^\top, v_3^\top, \dots, v_{M-1}^\top]^\top$$

where $x \in \mathbb{R}^{n_x}$ with $n_x = 3 \cdot (2 \cdot (M-2) + 1)$, which results in the following set of ODEs:

$$\dot{x} = f(x,u) = \begin{bmatrix} v_2 \\ \vdots \\ v_{M-1} \\ u \\ \frac{1}{m}(F_{2,3} - F_{1,2}) + g_z \\ \vdots \\ \frac{1}{m}(F_{M-2,M-1} - F_{M-3,M-2}) + g_z \end{bmatrix}. \tag{6.1}$$

We remark that the only nonlinearity is introduced in the calculation of the forces. The steady state $(x_\text{ss}, u_\text{ss})$ of the system can be found by setting $f(x_\text{ss}, u_\text{ss}) = 0$ for any given $p_{M,\text{ss}}$. In all the following, we take $p_{M,\text{ss}} = [7.5, 0, 0]^\top$.

Optimal control problem formulation

In order to stabilize the motion of the string of balls to the steady state, we propose the following optimal control problem, obtained by performing a multiple shooting discretization of ODE (6.1):

$$\begin{aligned} \underset{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \begin{bmatrix} x_k - x_\text{ref} \\ u_k - u_\text{ref} \end{bmatrix}^\top \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_k - x_\text{ref} \\ u_k - u_\text{ref} \end{bmatrix} \\ & \qquad + (x_N - x_\text{ref})^\top Q_N (x_N - x_\text{ref}) \\ \text{subject to} \quad & x_0 = \overline{x}_0 \\ & x_{k+1} = f_\text{d}(x_k, u_k), \quad k = 0, \dots, N-1 \\ & -1 \leqslant u_k \leqslant 1, \qquad k = 0, \dots, N-1, \end{aligned} \tag{6.2}$$

where the initial state $\overline{x}_0$ is the current estimate of the state vector, $f_\text{d} : \mathbb{R}^{n_x} \times \mathbb{R}^3 \to \mathbb{R}^{n_x}$ is obtained by performing a single RK4 step of length 0.2 s on ODE (6.1). Furthermore, we choose a horizon length $N = 40$ and the weighting

Table 6.3: Design parameters for the chain of masses case study

Quantity	Description	Value
m	mass of one ball	$0.1125\,\mathrm{kg}$
D	spring constant	$0.4\,\mathrm{N/m}$
L	rest length of the springs	$0.1375\,\mathrm{m}$
g_z	gravitational acceleration vector	$[0, 0, -9.81]^\top\ \mathrm{m/s^2}$
N	horizon length	40
Δt	discretization step	$0.2\,\mathrm{s}$
$p_{M,\mathrm{ref}}$	reference position of free ball	$[7.5, 0, 0]^\top\ \mathrm{m}$

matrices

$$\begin{aligned}
Q &= \mathrm{diag}(\underbrace{0, \ldots, 0,}_{p_2,\ldots,p_{M-1}}\ \underbrace{2.5, 2.5, 2.5,}_{p_M}\ \underbrace{25, \ldots, 25}_{v_2,\ldots,v_{M-1}}),\\
Q_N &= \mathrm{diag}(\underbrace{0, \ldots, 0,}_{p_2,\ldots,p_{M-1}}\ \underbrace{10, 10, 10,}_{p_M}\ \underbrace{0, \ldots, 0}_{v_2,\ldots,v_{M-1}}\ ),\\
R &= \mathrm{diag}(0.1, 0.1, 0.1),
\end{aligned}$$

and corresponding reference values

$$\begin{aligned}
x_{\mathrm{ref}} &= [\ \underbrace{0, \ldots, 0}_{p_2,\ldots,p_{M-1}}\ , \underbrace{7.5, 0, 0,}_{p_M}\ \underbrace{0, \ldots, 0}_{v_2,\ldots,v_{M-1}}\ ]^\top,\\
u_{\mathrm{ref}} &= [0, 0, 0]^\top.
\end{aligned}$$

The design parameters are chosen as in [Englert et al., 2018] and are summarized in Table 6.3. Note that we did not introduce path constraints or state bounds, since these are not supported by all solvers that we compare to below.

Closed-loop experiments

In closed-loop, an MPC controller repeatedly (approximately) solves OCP (6.2). The first control u_0 is passed to the dynamic system under control and a new initial state $\bar{x}_0$ is obtained. Here, we simulate the system by using a more accurate integrator than the one in OCP (6.2), namely the Dormand-Prince method, as implemented in the MATLAB routine `ode45`.

We introduce one disturbance into the closed-loop system, similar as in [Wirsching et al., 2006]: in the beginning of the simulation, we start from a horizontal configuration of the chain of masses. Around the midpoint of the simulation, we override the closed loop control with a constant $u_\mathrm{d} = [-1, 1, 1]^\top$. After one second of simulation time, the controller takes over again. We compare the following solvers with each other for this particular closed-loop setup:

`IPOPT`	[Wächter and Biegler, 2006]. As a solver *not* targeting embedded devices specifically, we use it as a baseline to compare against.
`FalcOPT`	[Torrisi et al., 2016]. A projected gradient descent method tailored for NMPC.
`VIATOC`	[Kalmari et al., 2015]. A gradient projection method for MPC that only allows linear inequality constraints.
`ACADO`	[Houska et al., 2011]. Code Generation Tool. Generates SQP-based solvers.
`GRAMPC`	[Englert et al., 2018]. An embedded Augmented Lagrangian-based solver.
`acados`	Framework presented currently.

The tuning parameters for the different solvers are listed in Table 6.4. This benchmarking problem and accompanying MATLAB scripts have been published online, see [Verschueren, 2018].

Table 6.4: TUNING PARAMETERS FOR THE DIFFERENT SOLVERS IN CASE STUDY 1.

Solver	Tuning parameters
`IPOPT`	Called through `CasADi`, default parameters
`FalcOPT`	`eps`: 0.1, `maxIt`: 100
`VIATOC`	Maximum number of iterations: 20
`ACADO`	RTI solver, Full condensing, QP solver `qpOASES`
`GRAMPC`	Parameters chosen as in [Englert et al., 2018]
`acados`	`SQP_RTI` solver, QP solver `HPIPM`, partial condg. horizon of 5

In order to compare the quality of the closed-loop solutions, we use the notion of cumulative sub-optimality (CSO), which is an approximation of the integrated cost along closed-loop trajectories:

$$\mathrm{CSO}_{(\cdot),n} = \sum_{i=0}^{n} \begin{bmatrix} x_{(\cdot),i} - x_\mathrm{ref} \\ u_{(\cdot),i} - u_\mathrm{ref} \end{bmatrix}^\top \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_{(\cdot),i} - x_\mathrm{ref} \\ u_{(\cdot),i} - u_\mathrm{ref} \end{bmatrix}.$$

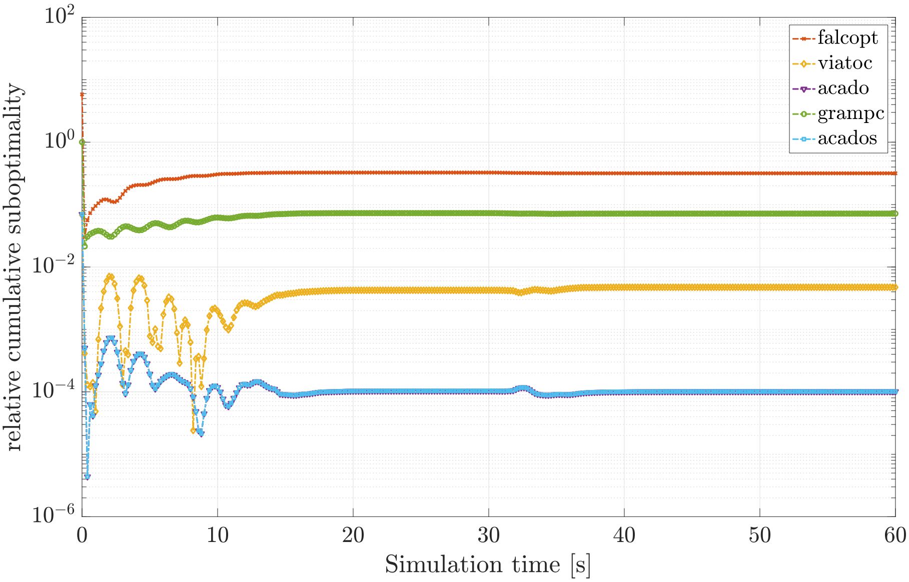


Figure 6.5: Relative cumulative suboptimality (RCSO) in each step of the simulation of the hanging chain with $M = 5$ and $N = 40$, for the different solvers. The baseline for comparison is the `IPOPT` solution.

To compare the different solvers, we plot the relative cumulative sub-optimality (RCSO), relative to a fully converged solution, in this case, the `IPOPT` solution, which reads as

$$\mathrm{RCSO}_{(\cdot),n} = \left| \frac{\mathrm{CSO}_{(\cdot),n} - \mathrm{CSO}_{\texttt{ipopt},n}}{\mathrm{CSO}_{\texttt{ipopt},n}} \right|,$$

for n going from 0 to 300, the number of time steps in our simulation. We plot a comparison in Figure 6.5. The trajectories of `ACADO` and `acados` are *exactly* the same, as they implement the same real-time algorithm, with both being very close to the reference solution from `IPOPT`. The solvers `GRAMPC`, `VIATOC` and `FalcOPT`, being based on first-order methods, are further away from the `IPOPT` solution. These findings are consistent with previously published work by other authors, see [Englert et al., 2018].

We have a look at the numerical performance along the closed-loop trajectories in Figure 6.6. `GRAMPC`, `ACADO`, `VIATOC` and `acados` produce consistent timings throughout the entire experiment, even when the disturbance occurs. This is a beneficial property for embedded solvers, as they often have a fixed time deadline, being part of a larger control application. `GRAMPC` and `acados` produce solutions at almost the same speed, both approximately a factor 2 faster than

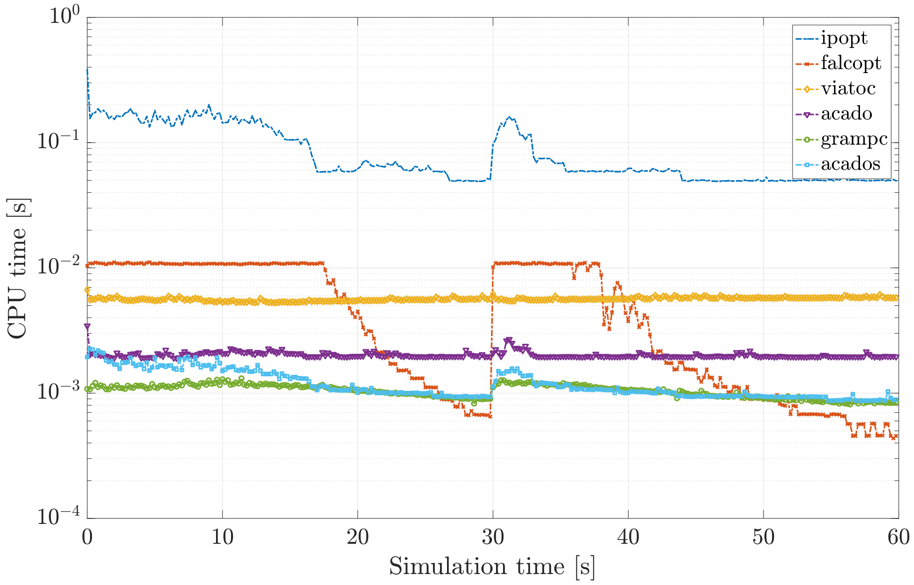


Figure 6.6: Computational time for each iteration of the closed loop simulation, averaged over 10 runs.

Table 6.5: Computation times for a closed-loop experiments on a chain of masses (cf. Figure 6.6)

comp. time per iteration (ms)	median	minimum	maximum
`IPOPT`	59.84	49.06	384.90
`FalcOPT`	4.36	0.44	11.10
`VIATOC`	5.63	5.27	6.68
`ACADO`	1.97	1.90	3.45
`GRAMPC`	1.06	0.81	1.31
`acados`	1.05	0.87	2.23

`ACADO` which is in turn a factor 2-3 faster than `VIATOC`. Near the equilibrium, `FalcOPT` takes the shortest computation time, as it is performing only a few gradient steps per iteration. `IPOPT` is included as a baseline for comparison to non-embedded solvers. The timings are summarized in Table 6.5.

Of course, an optimization solver can always trade off sub-optimality with computation time. To get the full picture, we plot both measures against each other in Figure 6.7: we look at relative cumulative sub-optimality over the entire length of the experiment, versus worst-case computation times. By

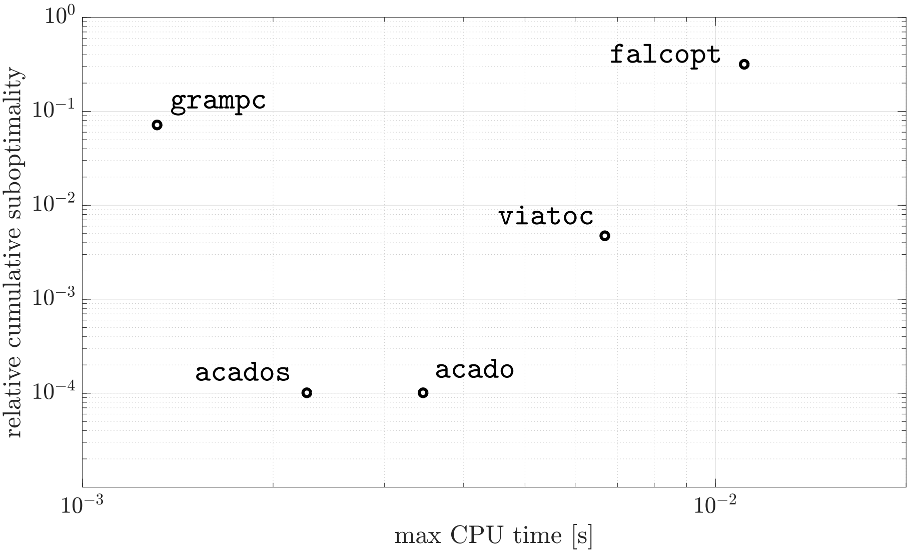


Figure 6.7: Trade-off between sub-optimality (Figure 6.5) and computation time (Figure 6.6). We see that `acados` and `GRAMPC` lie on the Pareto-optimal front.

this comparison, we see that `acados` and `GRAMPC` are on the Pareto-optimal front: although `acados` is a factor 1000 times less suboptimal than `GRAMPC`, the computational cost is higher. By the median computation times, `acados` is faster (see Table 6.5).

6.3.2 Case study 2: Regularization

In Chapter 2, we discussed the impact of Hessian regularization on SQP methods. In this case study, we compare the convergence of exact Hessian based SQP with three different Hessian regularizations, including the convexification method of Chapter 2, on a simple control problem. We control a mass on a rod (a pendulum), balanced on a horizontally moving cart, see Section 2.5.

Exact-Hessian based SQP

We solve OCP (2.65) with SQP, where we use the exact Hessian of the Lagrangian. We choose $N = 100$ shooting intervals of length 0.01 s. The blocks of the Hessian matrix are identified as

Table 6.6: Exact-Hessian based SQP: computation times

regularization	convexification	project	mirror
avg iteration (ms)	2.540	2.303	2.264
total time (ms)	66.034	103.65	174.32

$$H_k = \begin{bmatrix} Q & \\ & R \end{bmatrix} + \sum_{i=0}^{n_x} \lambda_{k,i} \nabla^2_{(x,u)} \psi_{\mathrm{d},i}(x_k, u_k), \quad k = 0, \dots, N-1,$$

$$H_N = Q,$$

where λ_k are the Lagrange multipliers associated with the dynamic equality constraints.

In some cases, the non-convexity of the dynamic equations gives rise to an indefinite Hessian matrix. We apply the project$(\cdot)$ and mirror$(\cdot)$ regularizations, as well as the structure-exploiting regularization from Algorithm 2.2, which we will call 'convexification'. These regularizations are implemented as modules in the `acados` framework.

We now compare the convergence behavior of all three regularization methods. For each SQP variant, we start the SQP iterations from the point $z_0 = (0, 0, 0)$. The result can be seen in Figure 6.8. The structure-exploiting convexification converges almost twice as fast as the projection regularization, and is in turn much faster than the mirroring regularization. Intuitively, this makes sense, as mirroring is 'blocking' directions associated with large negative eigenvalues, by introducing large positive eigenvalues in those directions. This prevents the solver from taking larger steps[1]. In turn, the structure-exploiting regularization is faster than merely projecting the eigenvalues on the positive definite cone, because it is redistributing convexity among all stages, and thus needs less regularization overall.

It must be said that the convexification of Algorithm (2.2) is quite a bit more involved than the other two regularization schemes. However, by using the optimized linear algebra kernels of `BLASFEO`, we implemented the convexification method such that it is only slightly more expensive per iteration than the basic regularization schemes, see Table 6.6, but much more computationally cheap overall.

---

[1] This is also the approach followed by the algorithms obtained with the `ACADO` Code Generation Tool.

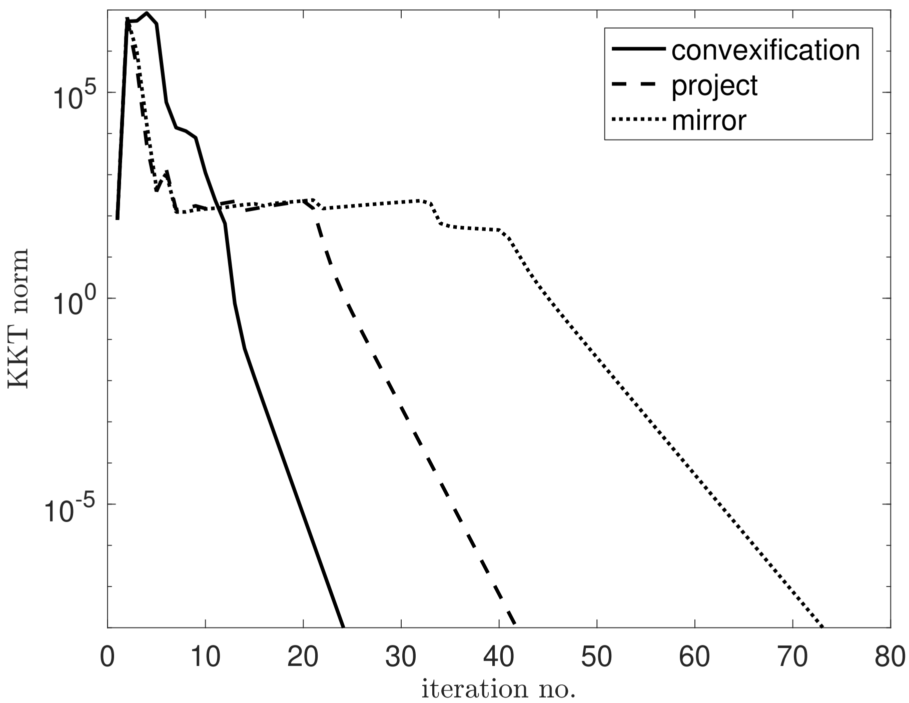


Figure 6.8: Convergence comparison of exact Hessian based SQP with three different regularization strategies.

6.3.3 Case study 3: SCQP

In this case study, we showcase the SCQP method of Chapter 3 with an efficient implementation in `acados`. For this, we use the inverted pendulum model and the problem formulation in (3.24) with $N = 40$.

We briefly study the numerical performance of the `acados` SCQP implementation. No comparison with `ACADO` is possible here, because SCQP did not exist as a feature in `ACADO` and would be cumbersome to implement. For different values of the radius of the circular constraint, R_e in (3.24), we have different Hessian approximations for SCQP, as the optimal Lagrange multiplier $\mu^\star$ varies. By contrast, a Gauss-Newton approximation would be constant for all values of R_e.

For some R_e, the Gauss-Newton Hessian approximation is not sufficiently good for a full-step SQP method to converge, not even when initialized close to a local minimizer, as we saw in Section 3.5. In contrast, SCQP converges

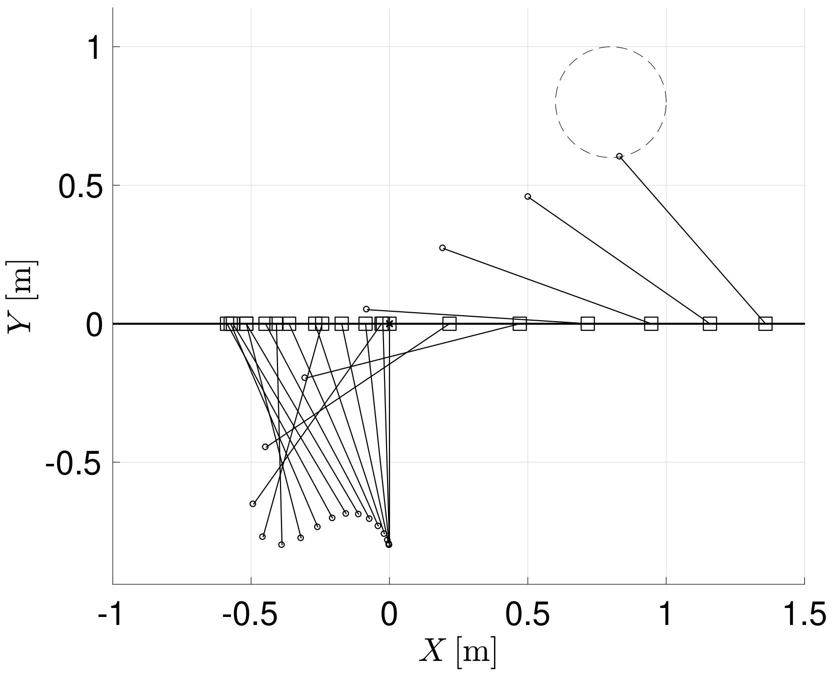


Figure 6.9: SCQP on the pendulum example (cf. OCP (3.24)).

for all R_e, depending where the solver is initialized. As an example, look at $R_e = 0.04\,\text{m}$. We initialize the full-step SCQP method at $x_k^0 = 0, k = 0, \dots, N$ and $u_k^0 = 0, k = 0, \dots, N-1$. SCQP converges in 38 iterations, with a total computation time of 5.5 ms. The optimal control trajectory is shown in Figure 6.9. Note that we used a different radius and initialization point as in Figure 3.4a, and hence the trajectory looks different. Full-step SQP with a Gauss-Newton Hessian approximation, however, would not converge, regardless of where it is initialized (except the solution).

6.3.4 Case study 4: Processor-in-the-loop experiments

As a last case study, we discuss the performance of `acados` on an embedded platform, namely the dSPACE MicroAutoboxII [dSPACE, 2006], depicted in Figure 6.10. It is an industrial computing platform that is used in the car industry. It features a 900 MHz PowerPC processor (IBM PPC 750GL) with

Figure 6.10: dSPACE MicroAutoboxII [dSPACE, 2006] embedded PC with a 900 MHz PowerPC processor. Courtesy of dSPACE GmbH.

16MB of main memory. The control application that we focus on is engine control, with the engine model as presented in [Albin et al., 2017], which we will briefly reproduce here.

Two-stage turbocharging gasoline engines have been introduced as a flexible alternative to conventional turbocharged engines. The main advantage they offer is a better trade-off between short transient times after load changes and a high specific power. However, the two-stage architecture puts a higher burden on the engine controller. NMPC has been proposed as a viable control strategy [Albin et al., 2017].

In Figure 6.11, a sketch of the two-stage turbocharged engine is depicted. The high-pressure (HP) stage is able to realize fast transients, the low-pressure (LP) stage produces a higher specific power, but with slower dynamics. The control challenge lies in accurately tracking the boost pressure p_{boost}, given the highly nonlinear coupling between both stages.

For reasons of brevity, we directly present the engine model of [Albin et al., 2017] and refer the interested reader to that work for a derivation. We model the engine with a set of semi-explicit DAEs. The differential states consist of $\Pi_{c,\mathrm{lp}}, \Pi_{c,\mathrm{hp}}$ the pressure ratios between input and output of the compressor in the low pressure and high pressure stage, respectively. The algebraic states are $\Pi_{t,\mathrm{lp}}, \Pi_{t,\mathrm{hp}}$, the pressure ratios on the turbine. The inputs are the wastegate actuation pulse-width modulated signals $u_{\mathrm{wg,lp}}, u_{\mathrm{wg,hp}}$, which take on values between 0% (fully open) and 100% (fully closed).

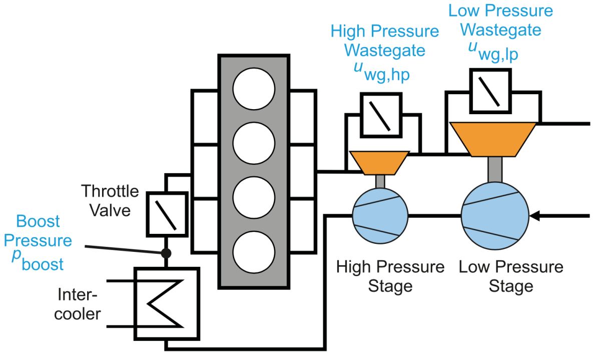


Figure 6.11: Schematic drawing of the two-stage turbocharging concept. From [Albin et al., 2017].

The resulting DAE system reads as

$$\begin{aligned}
\dot{\Pi}_{c,\text{lp}} &= c_1(\Pi_{t,\text{lp}}^{1.5} - \Pi_{t,\text{lp}}^{1.25})\sqrt{\Pi_{t,\text{lp}}^{-1.5} - \Pi_{t,\text{lp}}^{-1.75}} && (6.3)\\
&\quad - c_2 n_{\text{eng}} \Pi_{c,\text{hp}} (\Pi_{c,\text{lp}}^{1.29} - \Pi_{c,\text{lp}}) && (6.4)\\
0 &= \Pi_{c,\text{lp}}\Pi_{c,\text{hp}} && (6.5)\\
&\quad - \frac{c_3}{n_{\text{eng}}}\sqrt{\Pi_{t,\text{lp}}^{0.5} - \Pi_{t,\text{lp}}^{0.25}}\left(\sqrt{\Pi_{t,\text{lp}}} + c_4 A(\Pi_{c,\text{lp}} \cdot \Pi_{c,\text{hp}}, u_{\text{wg,lp}})\right) && (6.6)\\
\dot{\Pi}_{c,\text{hp}} &= c_5(\Pi_{t,\text{hp}}^{1.5} - \Pi_{t,\text{hp}}^{1.25})\sqrt{\Pi_{t,\text{hp}}^{-1.5} - \Pi_{t,\text{hp}}^{-1.75}} && (6.7)\\
&\quad - c_6 n_{\text{eng}} \Pi_{c,\text{lp}} (\Pi_{c,\text{hp}}^{1.29} - \Pi_{c,\text{hp}}) && (6.8)\\
0 &= \Pi_{c,\text{lp}}\Pi_{c,\text{hp}} && (6.9)\\
&\quad - \frac{c_7}{n_{\text{eng}}}\sqrt{\Pi_{t,\text{hp}}^{0.5} - \Pi_{t,\text{hp}}^{0.25}}\left(\sqrt{\Pi_{t,\text{hp}}} + c_8(1 - u_{\text{wg,hp}}/100)\right), && (6.10)
\end{aligned}$$

with, additionally, $n_{\text{eng}} = 2000\,\text{min}^{-1}$ the engine speed and $A : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is defined by

$$A(u, v) = \gamma_1(u) \cdot \gamma_2(v),$$

Table 6.7: PARAMETER VALUES FOR THE TWO-STAGE TURBOCHARGED ENGINE MODEL

Parameter	Unit	Value	Parameter	Unit	Value
c_1	–	25.3	$b_{1,1}$	–	0
c_2	min	0.0034	$b_{2,1}$	–	1
c_3	min^{-1}	7700	$b_{3,1}$	–	1.49
c_4	–	0.6	$b_{4,1}$	–	0.0377
c_5	–	43,6	$b_{1,2}$	–	67.5
c_6	min	0.0092	$b_{2,2}$	–	4.712
c_7	min^{-1}	3600	$b_{3,2}$	–	1
c_8	–	0.9	$b_{4,2}$	–	-1

with $\gamma_i : \mathbb{R} \rightarrow \mathbb{R}$:

$$\gamma_i(u) = b_{1,i} + b_{2,i}\left(1 + e^{\frac{-u+b_{3,i}}{b_{4,i}}}\right)^{-1}.$$

The values of all model parameters can be found in Table 6.7.

In order to obtain a smooth control behavior, we include the time derivative of the controls in the optimization formulation, as follows: $\dot{u}_{\text{wg,lp}} = d_{u,\text{lp}}$, $\dot{u}_{\text{wg,hp}} = d_{u,\text{hp}}$, and we collect these rates in

$$d = \begin{bmatrix} d_{u,\text{lp}} \\ d_{u,\text{hp}} \end{bmatrix}.$$

We then define the vector of differential states, algebraic states and controls, respectively, as follows:

$$x = \begin{bmatrix} \Pi_{c,\text{lp}} \\ \Pi_{c,\text{hp}} \\ u_{\text{wg,lp}} \\ u_{\text{wg,hp}} \end{bmatrix}, \quad z = \begin{bmatrix} \Pi_{t,\text{lp}} \\ \Pi_{t,\text{hp}} \end{bmatrix}, \quad u = \begin{bmatrix} d_{u,\text{lp}} \\ d_{u,\text{hp}} \end{bmatrix}.$$

The control objective is to track a boost pressure signal, where the boost pressure is given by $y_p(x) = \Pi_{c,\text{lp}} \cdot \Pi_{c,\text{hp}}$. To this end, we solve an OCP arising from a multiple shooting formulation with an implicit Runge-Kutta (GL3) integrator with sampling time 0.05 s and $N = 20$ shooting intervals. The DAE simulation functions are denoted by ψ_d (differential part) and ν_d (algebraic part). Let

$$r(x,u) = \begin{bmatrix} y_p(x) \\ x \\ u \end{bmatrix} - \begin{bmatrix} y_{p,\mathrm{ref}} \\ 1.14 \\ 1.54 \\ 50 \\ 50 \\ 0 \\ 0 \end{bmatrix}, \qquad r_N(x) = \begin{bmatrix} y_p(x) \\ x \end{bmatrix} - \begin{bmatrix} y_{p,\mathrm{ref}} \\ 1.14 \\ 1.54 \\ 50 \\ 50 \end{bmatrix}.$$

The OCP then reads as

$$\begin{aligned} \underset{\substack{x_0,\dots,x_N,\\ z_0,\dots,z_{N-1}\\ u_0,\dots,u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \|r(x_k,u_k)\|_W^2 + \|r_N(x_N)\|_{W_N}^2 \\ \text{subject to} \quad & x_0 = \overline{x}_0, \\ & x_{k+1} = \psi_\mathrm{d}(x_k,z_k,u_k), \quad k = 0,\dots,N-1, \\ & 0 = \nu_\mathrm{d}(x_k,z_k,u_k), \quad k = 0,\dots,N-1, \\ & 0 \leqslant u_k \leqslant 100, \quad k = 0,\dots,N-1, \\ & 0.5 \leqslant \Pi_{c,\mathrm{lp},k} \leqslant 1.757, \quad k = 1,\dots,N, \\ & 0.5 \leqslant \Pi_{c,\mathrm{hp},k} \leqslant 2.125, \quad k = 1,\dots,N, \end{aligned} \tag{6.11}$$

with

$$\begin{aligned} W &= \mathrm{diag}([10^3, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-4}, 10^{-4}]), \\ W_N &= \mathrm{diag}([10^3, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}]). \end{aligned}$$

We repeatedly solve this OCP approximately by performing real time iterations. As an underlying QP solver, we use `HPIPM`. When ran in closed loop on the dSPACE MicroAutoboxII, the results can be seen in Figure 6.12. Control bounds and state bounds become active at some point in the simulation, for the high-pressure stage. The reference is tracked closely and without oscillations, which have been observed when linear-quadratic MPC is used [Albin et al., 2017]. As for the computation times, it is interesting to note that there are spikes everywhere where a jump occurs or a constraint becomes (in)active. The computation times close to the solution (i.e. at the beginning of the simulation) drop to almost zero. In any case, the maximum computation times remains under 10 ms, which is 5x faster than the sampling time of the system (50 ms). We

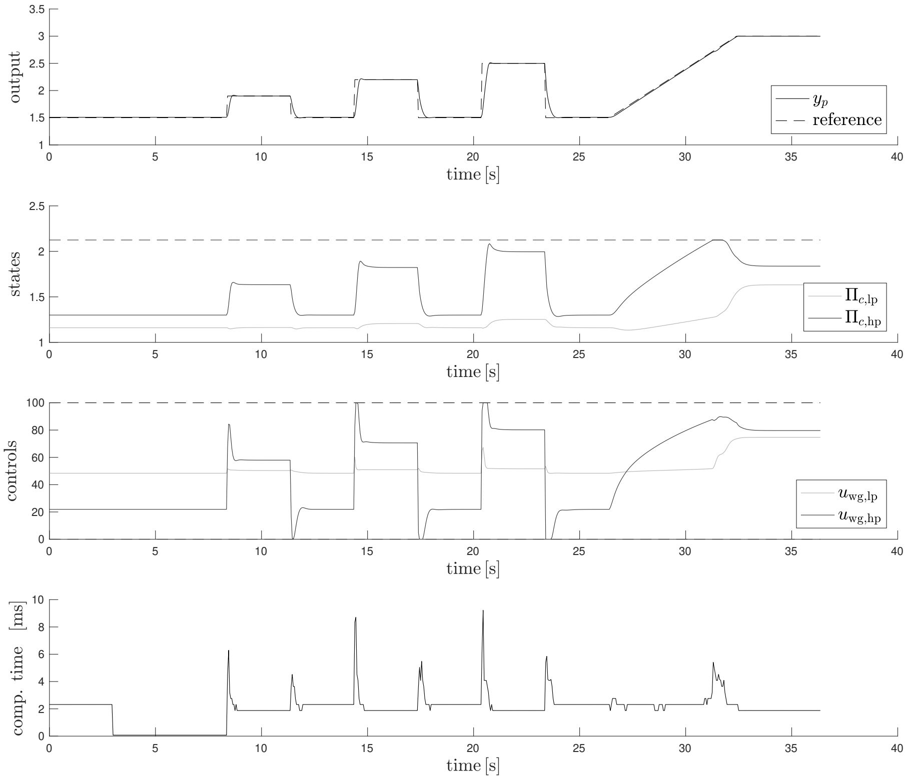


Figure 6.12: Closed-loop simulation of the engine model with steps in the reference boost pressure. Simulations are carried out on the dSPACE MicroAutoboxII platform at a clock speed of 900 MHz.

remark that the computation times obtained with the dSPACE MicroAutoboxII, for this experiment, are more or less 3x slower than a desktop computer with a 2.5GHz Intel Core i7-4870HQ processor.

6.4 Summary

In this chapter, we presented a novel software package, called `acados`. Compared to e.g. the `ACADO` Code Generation Tool, it has the following new features:

- different dynamics for each stage,

- new QP solver interfaces,
- new linear algebra (`BLASFEO`)
- new AD code (`CasADi`),
- SCQP and Hessian convexification,
- `Python` interface (as well as MATLAB),
- modular algorithmic components.

Even though `acados` is a more flexible tool than `ACADO`, it still results in better numerical performance than `ACADO`, as was shown with some numerical and HIL simulations.

Chapter 7

Conclusions & Outlook

> Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
>
> ---
>
> Douglas R. Hofstadter, cognitive scientist

Making smart decisions will keep becoming more important for autonomous systems. Optimization-based control is one way of doing that. In this thesis, we presented a couple of techniques targeted at *embedded* optimization. More specifically, we made various convex approximations to non-convex problems, as there exist reliable and efficient solvers for this type of problems.

The methods have been presented from a theoretical point of view, and an efficient implementation for each of them is freely available online, as part of a new software framework. A few simulation studies were presented here to illustrate the efficiency and the usability of the software. These properties will hopefully enable control designers in continuously bringing increasingly advanced optimization methods to real-world applications. We will conclude each chapter separately, and offer an outlook to future research directions.

Chapter 2 : Convexification for optimal control. Motivated by an equivalence result concerning convexity of QP problems, we presented a convexification method for indefinite QP problems as they arise in SQP-type methods for NMPC. It enables local quadratic convergence, under some assumptions, as was illustrated on a nonlinear control problem. Future research directions include:

- The automatic selection of the δ, γ, ϵ parameters, and a deeper understanding of the relation between them,
- A comparison study between our convexification method and (partial) condensing-based solvers.

Chapter 3 : Sequential convex quadratic programming. A new Hessian approximation for SQP-type methods was proposed. This approximation is applicable for general NLP problems with convex substructure, but are particularly useful in an embedded context, because of the low computational cost. A comparison between SCQP and GGN was made. Future work might comprise

- A convergence result stating that SCP and SCQP exhibit the same linear contraction rate,
- A more detailed comparison between SCP, SCQP, the method by [Martens and Sutskever, 2011] and GGN,
- An experimental study for comparing GGN and SCQP.

Chapter 4 : Time-optimal point-to-point motions. We proposed a new stability result for time-optimal NMPC for point-to-point motions. It is based on a new optimization formulation based on convex l_1 penalties and increasing weights. An HIL study shows that it is straightforward to implement and use. Motivated by the simulation study, we see the experimental application of the algorithm to a real plant, in combination with SCQP, as a promising direction.

Chapter 5 : Time-optimal path following for robotic manipulators. A spatial reformulation of the dynamics, as presented, has the following benefits: it ‘frees’ the time variable and parametrizes the optimal control problem with the path parameter instead; doing so, some difficult path constraints become trivial. The following points could be interesting follow-up activities:

- A generalization of the method to paths in $\mathbb{R}^n$,
- The practical application of it to a real robot system, some preliminary results are available in [van Duijkeren et al., 2016].

Chapter 6 : The `acados` software framework. We presented the new `acados` software framework. It is supposed to offer a more flexible experience than the `ACADO` Code Generation Tool. Although development is still ongoing, some preliminary computational results seem promising. Moreover, the ease of use is illustrated by the rapid development of the methods of Chapter 2 and Chapter 3 within `acados`. Many questions remain open, such as:

- integration with other ‘external’ NLP solvers,
- code generation of Simulink blocks for drag-and-drop functionality (some preliminary facilities are present already),
- interfaces to Julia and Octave,
- provision of ‘template’ files implementing a bare bones NMPC controller, to be tweaked by the control practitioner.

A corollary of Hofstadter’s Law (see beginning of the chapter) is that given an amount of time, you will always achieve less than you presumed at the outset. Nevertheless, looking back on the thesis text, I’m happy and proud with what I did achieve, and I’m looking forward to future cooperations with the many people I was lucky to meet during the last four years. Thanks for reading!

Bibliography

ABB (2015). IRB120 technical data. `http://new.abb.com/products/robotics`. [Online; accessed 22 September 2015].

Agrawal, A., Verschueren, R., Diamond, S., and Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60.

Albin, T., Ritter, D., Liberda, N., Quirynen, R., and Diehl, M. (2017). In-vehicle realization of nonlinear MPC for gasoline two-stage turbocharging airpath control. *IEEE Transactions on Control Systems Technology*, pages 1–13.

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2018). CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*.

Andersson, J. A. E. and Rawlings, J. B. (2018). Sensitivity analysis for nonlinear programming in casadi. In *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*.

Athans, M. and Falb, P. L. (1966). *Optimal Control - An Introduction to the Theory and Its Applications*. Dover.

Auernig, J. W. and Troger, H. (1987). Time optimal control of overhead cranes with hoisting of the load. *Automatica*, 23(4):437–447.

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

Beazley, D. M. (2003). Automated scientific software scripting with SWIG. *Future Gener. Comput. Syst.*, 19:599–609.

Bemporad, A. (2003). *Hybrid Toolbox for Matlab*.

Bemporad, A., Borrelli, F., and Morari, M. (1999). The explicit solution of constrained LP-based receding horizon control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, Sydney, Australia.

Bertsekas, D. P. (1999). *Nonlinear Programming.* Athena Scientific, 2nd edition.

Besselmann, T. J., de moortel, S. V., Almér, S., Jörg, P., and Ferreau, H. J. (2015). Model predictive control in the multi-megawatt range. *IEEE Transactions on Industrial Electronics*, 63(7):4641–4648.

Bobrow, J., Dubowsky, S., and Gibson, J. (1985). Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17.

Bock, H. (1987). *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, volume 183 of *Bonner Mathematische Schriften.* Universität Bonn, Bonn.

Bock, H., Kostina, E., and Schlöder, J. (2007). Numerical methods for parameter estimation in nonlinear differential algebraic equations. *GAMM Mitteilungen*, 30/2:376–408.

Bock, H. G. (1983). Recent advances in parameter identification techniques for ODE. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, pages 95–121. Birkhäuser.

Bock, H. G. and Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the IFAC World Congress*, pages 242–247. Pergamon Press.

Boggs, P. T. and Tolle, J. W. (1995). Sequential quadratic programming. *Acta Numerica*, pages 1–51.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization.* University Press, Cambridge.

Brenan, K. E., Campbell, S. L., and Petzold, L. R. (1987). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* Society for Industrial and Applied Mathematics.

Britannica, E. (2018). Linear programming. `https://www.britannica.com/science/linear-programming-mathematics`.

Byrd, R. H., Nocedal, J., and Waltz, R. A. (2006). KNITRO: An integrated package for nonlinear optimization. In Pillo, G. and Roma, M., editors, *Large Scale Nonlinear Optimization*, pages 35–59. Springer Verlag.

Chen, H. and Allgöwer, F. (1998). A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1218.

Chiang, N.-Y., Hang, R., and Zavala, V. M. (2017). An augmented lagrangian filter method for real-time embedded optimization. *IEEE Transactions on Automatic Control*, 62(12):6110–6121.

Cimini, G. and Bemporad, A. (2017). Exact complexity certification of active-set methods for quadratic programming. *IEEE Transactions on Automatic Control*, 62(12):6094–6109.

Conn, A., Gould, N., and Toint, P. (2000). *Trust-Region Methods*. MPS/SIAM Series on Optimization. SIAM, Philadelphia, USA.

Dahl, O. and Nielsen, L. (1990). Torque-limited path following by online trajectory time scaling. *IEEE Transactions on Robotics and Automation*.

Debrouwere, F., Van Loock, W., Pipeleers, G., and Swevers, J. (2014). Time-optimal tube following for robotic manipulators. In *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*.

Deng, H. and Ohtsuka, T. (2018). A highly parallelizable newton-type method for nonlinear model predictive control. In *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*.

Desoer, C. A. and Wing, J. (1961). An optimal strategy for a saturating sampled-data system. *IRE Trans. Automat. Control*, (1):5–15.

Di Cairano, S., Brand, M., and Bortoff, S. A. (2013). Projection-free parallel quadratic programming for linear model predictive control. *International Journal of Control*, 86(8):1367–1385.

Diehl, M. (2001). *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, University of Heidelberg.

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

Diehl, M., Ferreau, H. J., and Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In Magni, L., Raimondo, M., and Allgöwer, F., editors, *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 391–417. Springer.

Diehl, M., Walther, A., Bock, H. G., and Kostina, E. (2010). An adjoint-based SQP algorithm with quasi-Newton Jacobian updates for inequality constrained optimization. *Optimization Methods and Software*, 25(4):531–552.

Domahidi, A., Chu, E., and Boyd, S. (2013). ECOS: An SOCP solver for embedded systems. In *Proceedings of the European Control Conference (ECC)*, pages 3071–3076. IEEE.

Domahidi, A. and Perez, J. (2013). FORCES professional. http://embotech.com/FORCES-Pro.

Domahidi, A., Zgraggen, A., Zeilinger, M. N., Morari, M., and Jones, C. N. (2012). Efficient interior point methods for multistage problems arising in receding horizon control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 668–674, Maui, HI, USA.

dSPACE (2006). Homepage. http://www.dspace.com.

Duff, I. (2004). Ma57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144.

Englert, T., Völz, A., Mesmer, F., Rhein, S., and Graichen, K. (2018). A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (grampc). *ArXiv e-prints.*

Faulwasser, T., Kern, B., and Findeisen, R. (2009). Model predictive path-following for constrained nonlinear systems. In *Joint 48th IEEE conference on Decision and Control and 28th Chinese Control Conference.*

Ferreau, H., Lorini, G., and Diehl, M. (2006). Fast nonlinear model predictive control of gasoline engines. In *Proceedings of the IEEE International Conference on Control Applications, Munich*, pages 2754–2759.

Ferreau, H. J., Almer, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J. L., Stathopoulos, G., and Jones, C. (2017). Embedded optimization methods for industrial automatic control. In *Proceedings of the IFAC World Congress.*

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. (2014). qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363.

Forsgren, A. and Gill, P. E. (1998). Primal-dual interior methods for nonconvex nonlinear programming. *SIAM Journal of Optimization*, 8(4):1132–1152.

Forsgren, A., Gill, P. E., and Wright, M. H. (2002). Interior Point Methods for Nonlinear Optimization. *SIAM Review*, 44:525–597.

Franke, R. and Arnold, E. (1997). *Computer Intensive Methods in Control and Signal Processing*, chapter Applying new numerical algorithms to the solution of discrete-time optimal control problems, pages 105–117. Springer.

Frasch, J. V., Gray, A. J., Zanon, M., Ferreau, H. J., Sager, S., Borrelli, F., and Diehl, M. (2013). An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. In *Proceedings of the European Control Conference (ECC)*, pages 4136–4141.

Frasch, J. V., Sager, S., and Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computations*, 7(3):289–329.

Frison, G. (2015). *Algorithms and Methods for High-Performance Model Predictive Control.* PhD thesis, Technical University of Denmark (DTU).

Frison, G. (2017). HPIPM, High-performance interior-point qp solvers. https://github.com/giaf/hpipm.

Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A., and Diehl, M. (2018). BLASFEO: Basic linear algebra subroutines for embedded optimization. *ACM Transactions on Mathematical Software (TOMS)*, 44(4):42:1–42:30.

Frison, G., Sorensen, H. B., Dammann, B., and Jørgensen, J. B. (2014). High-performance small-scale solvers for linear model predictive control. In *Proceedings of the European Control Conference (ECC)*, pages 128–133.

Gao, Y., Gray, A., Frasch, J. V., Lin, T., Tseng, H. E., Hedrick, J., and Borrelli, F. (2012). Spatial predictive control for agile semi-autonomous ground vehicles. In *Proceedings of the 11th International Symposium on Advanced Vehicle Control.*

Geering, H. P., Guzzella, L., Hepner, S. A. R., and Onder, C. H. (1986). Time-optimal motions of robots in assembly tasks. *IEEE Trans. Automat. Control*, 31(6):512–518.

Gertz, E. M. and Wright, S. J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):58–81.

Gill, P., Murray, W., and Saunders, M. (2005). SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131.

Gill, P. E., Murray, W., and Saunders, M. A. (1995). *User's Guide For QPOPT 1.0: A Fortran Package For Quadratic Programming.*

Gill, P. E. and Robinson, D. P. (2013). A globally convergent stabilized sqp method. *SIAM Journal of Optimization*, 23(4):1983–2010.

Girrbach, F., Hol, J. D., Zandbergen, R., Verschueren, R., Bellusci, G., and Diehl, M. (2018). On the effect of stabilization methods for quaternion invariants on the uncertainty in optimization-based estimation. In *Joint 9th IFAC Symposium on Robust Control Design and 2nd IFAC Workshop on Linear Parameter Varying Systems.*

Gondzio, J. and Grothey, A. (2006). A new unblocking technique to warmstart interior point methods based on sensitivity analysis. *Siam J. Control Optim.*, 19(3):1184–1210.

Gould, N. and Robinson, D. (2010). A second derivative sqp method: Local convergence and practical issues. *SIAM Journal on Optimization*, 20(4):2049–2079.

Graber, S. (2018). Time-optimal control of electric machines with state and input constraints. Master's thesis, University of Freiburg.

Griewank, A. (2000). *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation.* Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia.

Guggenheimer, H. (1977). *Differential Geometry.* Dover.

Hairer, E., Nørsett, S., and Wanner, G. (1993). *Solving Ordinary Differential Equations I.* Springer Series in Computational Mathematics. Springer, Berlin, 2nd edition.

Hairer, E. and Wanner, G. (1991). *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems.* Springer, Berlin Heidelberg, 2nd edition.

Haynsworth, E. V. (1968). Determination of the inertia of a partitioned hermitian matrix. *Linear algebra and its applications*, 1(1):73–81.

Herceg, M., Kvasnica, M., Jones, C., and Morari, M. (2013). Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland. `http://control.ee.ethz.ch/~mpt`.

Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., and Woodward, C. (2005). SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396.

Houska, B., Ferreau, H. J., and Diehl, M. (2011). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10):2279–2285.

HSL (2011). A collection of Fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk`.

Kalmari, J., Backman, J., and Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering Practice*, 39:56–66.

Käpernick, B. and Graichen, K. (2014). The gradient based nonlinear model predicitive control software GRAMPC. In *Proceedings of the European Control Conference (ECC)*.

Kayacan, E., Kayacan, E., Ramon, H., and Saeys, W. (2014). Distributed nonlinear model predictive control of an autonomous tractor-trailer system. *Mechatronics*, 24(8):926–933.

Kehrle, F., Frasch, J. V., Kirches, C., and Sager, S. (2011). Optimal control of formula 1 race cars in a VDrift based virtual environment. In Bittanti, S., Cenedese, A., and Zampieri, S., editors, *Proceedings of the 18th IFAC World Congress*, pages 11907–11912.

Khusainov, B., Kerrigan, E. C., Suardi, A., and Constantinides, G. A. (2017). Nonlinear predictive control on a heterogeneous computing platform. In *Proceedings of the IFAC World Congress*.

Kraus, T., Ferreau, H. J., Kayacan, E., Ramon, H., Baerdemaeker, J. D., Diehl, M., and Saeys, W. (2013). Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles. *Computers and Electronics in Agriculture*, 98:25–33.

Kvamme, S. (2014). DuQuad Webpage. `http://sverrkva.github.io/duquad/`.

Lam, D., Manzie, C., and Good, M. (2010). Model Predictive Contouring Control. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Lau, M. A. and Pao, L. Y. (2003). Input shaping and time-optimal control of flexible structures. *Automatica*, 39:893—-900.

Leineweber, D. B., Bauer, I., Bock, H. G., and Schlöder, J. P. (2003). An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: theoretical aspects. *Computers and Chemical Engineering*, 27:157–166.

Li, W. C. and Biegler, L. T. (1989). Multistep, Newton-type control strategies for constrained nonlinear processes. *Chem. Eng. Res. Des.*, 67:562–577.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647.

Lipp, T. and Boyd, S. (2014). Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311.

Luenberger, D. (1969). *Optimization by Vector Space Methods.* Wiley.

Lynn, L. L. and Zahradnik, R. L. (1970). The use of orthogonal polynomials in the near-optimal control of distributed systems by trajectory approximation. *Int. J. Control*, 12(6):1079–1087.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *28th International Conference on Machine Learning.*

MathWorks, T. (2005). The model predictive control toolbox. `https://mathworks.com/products/mpc.html`.

Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, pages 1–27.

Mehrotra, S. (1992). On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601.

Melis, W. and Patrinos, P. (2018). C code generation for NMPC. `https://github.com/kul-forbes/nmpc-codegen`.

Morales, J. L., Nocedal, J., and Wu, Y. (2011). A sequential quadratic programming algorithm with an additional equality constrained phase. *IMA Journal of Numerical Analysis.*

Moravec, H. (1988). *Mind Children.* Harvard University Press.

Murray, D. and Yakowitz, S. J. (1984). Differential dynamic programming and newton's method for discrete optimal control problems. *Journal of Optimization Theory and Applications*, pages 395–414.

Nešić, D., Mareels, I. M. Y., Bastin, G., and Mahony, R. (1998). Output dead beat control for a class of planar polynomial systems. *SIAM J. Control Optim.*, 36(1):253–272. time-optimal control.

Nocedal, J. and Wright, S. (2000). *Numerical Optimization.* Springer-Verlag.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, 2 edition.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574.

Olofsson, B. (2015). *Topics in Machining with Industrial Robot Manipulators and Optimal Motion Control.* PhD thesis, Lund University, Department of Automatic Control.

O'Reilly, J. (1981). The discrete linear time invariant time-optimal control problem: An overview. *Automatica*, 17(2):363–370. time-optimal control.

Pannocchia, G., Rawlings, J., and Wright, S. (2007). Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43:852–860.

Quirynen, R. (2017). *Numerical Simulation Methods for Embedded Optimization.* PhD thesis, KU Leuven and University of Freiburg.

Quirynen, R., Knyazev, A., and Di Cairano, S. (2018). Block structured preconditioning within an active-set method for real-time optimal control. In *Proceedings of the European Control Conference (ECC).*

Rao, C. V., Wright, S. J., and Rawlings, J. B. (1998). Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, 99:723–757.

Rawlings, J. B. and Mayne, D. Q. (2009). *Model Predictive Control: Theory and Design.* Nob Hill.

Rawlings, J. B., Mayne, D. Q., and Diehl, M. M. (2017). *Model Predictive Control: Theory, Computation, and Design.* Nob Hill, 2nd edition edition.

Robinson, S. (1974). Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. *Mathematical Programming*, 7:1–16.

Rösmann, C., Hoffmann, F., and Bertram, T. (2015). Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *Proceedings of the European Control Conference (ECC)*, pages 3352–3357.

Sathya, A., Sopasakis, P., Themelis, A., Parys, R. V., Pipeleers, G., and Patrinos, P. (2018). Embedded nonlinear model predictive control for obstacle avoidance using PANOC. In *Proceedings of the European Control Conference (ECC).*

Schmid, C. and Biegler, L. T. (1994). Quadratic programming methods for reduced Hessian SQP. *Computers & chemical engineering*, 18(9):817–832.

Shahzad, A. and Goulart, P. J. (2011). A new hot-start interior-point method for model predictive control. In *Proceedings of the IFAC World Congress*.

Shin, K. G. and McKay, N. D. (1985). Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541.

Shukla, H., Khusainov, B., Kerrigan, E., and Jones, C. (2017). Software and hardware code generation for predictive control using splitting methods. In *Proceedings of the IFAC World Congress*.

Sousa, C. D. (2014). Sympybotics v1.0.

Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2006). *Robot modeling and control*. Wiley.

Steinbach, M. C. (1994). A structured interior point SQP method for nonlinear optimal control problems. In Bulirsch, R. and Kraft, D., editors, *Computation Optimal Control*, pages 213–222, Basel Boston Berlin. Birkhäuser.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2017a). OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*.

Stellato, B., Geyer, T., and Goulart, P. J. (2017b). High-speed finite control set model predictive control for power electronics. *IEEE Trans. Automat. Control*, 32(5):4007 – 4020.

Strogatz, S. H. (2001). *Nonlinear Dynamics and Chaos*. Westview Press.

Stryk, O. (1993). Numerical solution of optimal control problems by direct collocation. In *Optimal Control: Calculus of Variations, Optimal Control Theory and Numerical Methods*, volume 129. Bulirsch et al.

Torrisi, G., Frick, D., Robbiani, T., Grammatico, S., Smith, R., and Morari, M. (2017). FalcOpt: First-order Algorithm via Linearization of Constraints for OPTimization. `https://github.com/torrisig/FalcOpt`.

Torrisi, G., Grammatico, S., Smith, R., and Morari, M. (2016). A projected gradient and constraint linearilinear method for nonlinear model predictive control. *ArXiv e-prints*.

Tran-Dinh, Q. and Diehl, M. (2010). Local convergence of sequential convex programming for nonconvex optimization. In M.Diehl, F.Glineur, Jarlebring, E., and Michiels, W., editors, *Recent advances in optimization and its application in engineering*, pages 93–103. Springer-Verlag.

Ullmann, F. (2011). FiOrdOs: A Matlab toolbox for C-code generation for first order methods. Master's thesis, ETH Zurich.

Van den Broeck, L., Diehl, M., and Swevers, J. (2010). Embedded optimization for input shaping. *IEEE Transactions on Control System Technology*, 18:1146–1154.

Van den Broeck, L., Diehl, M., and Swevers, J. (2011). A model predictive control approach for time optimal point-to-point motion control. *IFAC Mechatronics*, 21:1203–1212.

van Duijkeren, N., Keviczky, T., Nilsson, P., and Laine, L. (2015). Real-time nmpc for semi-automated highway driving of long heavy vehicle combinations. In *Proceedings of the 5th IFAC Conference on Nonlinear Model Predictive Control 2015 (NMPC'15)*.

van Duijkeren, N., Verschueren, R., Pipeleers, G., Diehl, M., and Swevers, J. (2016). Path-following nmpc for serial-link robot manipulators using a path-parametric system reformulation. In *Proceedings of the European Control Conference (ECC)*.

Verscheure, D., Demeulenaere, B., Swevers, J., Schutter, J. D., and Diehl, M. (2009). Time-optimal path tracking for robots: a convex optimization approach. *IEEE Transactions on Automatic Control*, 54:2318–2327.

Verschueren, R. (2018). embench – benchmarking suite for embedded optimization. `https://github.com/roversch/embench`.

Verschueren, R., Bruyne, S. D., Zanon, M., Frasch, J. V., and Diehl, M. (2014). Towards time-optimal race car driving using nonlinear MPC in real-time. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 2505–2510.

Verschueren, R., Ferreau, H. J., Zanarini, A., Mercangöz, M., and Diehl, M. (2017a). A stabilizing nonlinear model predictive control scheme for time-optimal point-to-point motions. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Quirynen, R., and Diehl, M. (2018). Towards a modular software package for embedded optimization. In *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*.

Verschueren, R., van Duijkeren, N., Quirynen, R., and Diehl, M. (2016a). Exploiting convexity in direct optimal control: a sequential convex quadratic programming method. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*.

Verschueren, R., van Duijkeren, N., Swevers, J., and Diehl, M. (2016b). Time-optimal motion planning for n-dof robot manipulators using a path-parametric system reformulation. In *Proceedings of the American Control Conference (ACC)*, pages 2092–2097.

Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M. (2016c). Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm. In *Proceedings of the European Control Conference (ECC)*.

Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M. (2017b). A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control. *SIAM Journal of Optimization*, 27(3):2085–2109.

Vukov, M., Domahidi, A., Ferreau, H. J., Morari, M., and Diehl, M. (2013). Auto-generated algorithms for nonlinear model predictive control on long and on short horizons. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 5113–5118.

Vukov, M., Loock, W. V., Houska, B., Ferreau, H. J., Swevers, J., and Diehl, M. (2012). Experimental validation of nonlinear MPC on an overhead crane using automatic code generation. In *Proceedings of the American Control Conference (ACC)*, pages 6264–6269.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57.

Wan, W. and Biegler, L. T. (2016). Structured regularization for barrier nlp solvers. *Computational Optimization and Applications*.

Wirsching, L., Bock, H. G., and Diehl, M. (2006). Fast NMPC of a chain of masses connected by springs. In *Proceedings of the IEEE International Conference on Control Applications, Munich*, pages 591–596.

Zanelli, A., Domahidi, A., Jerez, J. L., and Morari, M. (2017a). FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*.

Zanelli, A., Horn, G., Frison, G., and Diehl, M. (2018). Nonlinear model predictive control of a human-sized quadrotor. In *Proceedings of the European Control Conference (ECC)*, pages 1542–1547.

Zanelli, A., Quirynen, R., Jerez, J., and Diehl, M. (2017b). A homotopy-based nonlinear interior-point method for NMPC. In *Proceedings of the IFAC World Congress*, Toulouse, France.

Zavala, V. M. and Biegler, L. T. (2009). The advanced step NMPC controller: Optimality, stability and robustness. *Automatica*, 45:86–93.

Zhao, J., Diehl, M., Longman, R., Bock, H., and Schlöder, J. (2004). Nonlinear model predictive control of robots using real-time optimization. In *Proceedings of the AIAA/AAS Astrodynamics Conference*, Providence, RI.

Curriculum Vitae

Robin Verschueren was born in Geel, Belgium in 1991. He attended the European School of Mol and graduated with greatest distinction (specialization Mathematics and Natural Sciences). After obtaining a Bachelor in Computer Science and Electrical Engineering (Cum Laude) in 2012 he pursued a Master of Science in Mathematical Engineering (Magna Cum Laude), both at the KU Leuven in Belgium. During his Master's thesis on autonomous racing ("Design and implementation of a time-optimal controller for model race cars") with Prof. Dr. Moritz Diehl, he was at Siemens PLM Software and KU Leuven. His work was awarded with the EOS prize for best master's thesis in science and engineering at a Flemish university.

In October 2014, he commenced doctoral studies at the Albert-Ludwigs-Universität in Freiburg, Germany, with prof. Diehl on an EU-funded Marie Curie-Skłodowska scholarship in the TEMPO Initial Training Network. His research focuses on efficient algorithms and software for nonlinear model predictive control. Within the TEMPO framework, he was a visiting researcher with prof. Jan Swevers at the Mechanical Engineering Department of KU Leuven (2015), at the ABB Corporate Research Center, Switzerland (2016), and at Stanford University (2017) under supervision of prof. Stephen Boyd. From September 2017, he is a research assistant (E13) at the Albert-Ludwigs-Universität Freiburg. He joins the ABB Corporate Research Center in Baden, Switzerland from October 2018.

List of publications

Journal publications

Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M. (2017b). A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control. *SIAM Journal of Optimization*, 27(3):2085–2109

Agrawal, A., Verschueren, R., Diamond, S., and Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60

In preparation/Submitted

Verschueren, R., Frison, G., Kouzoupis, D., Van Duijkeren, N., Zanelli, A., Novoselnik, B., Frey, J., Albin T., Quirynen, R., Diehl, M. (2018). `acados` – a modular open-source framework for fast embedded optimal control problem solvers.

Heitel, M., Verschueren, R., Diehl, M., Lebiedz, D. (2018). Considering slow manifold based model reduction for multiscale chemical optimal control problems.

Van Duijkeren, N., Verschueren, R., Pipeleers, G., Swevers, J. (2018). Sequential convex quadratic programming for online optimal control in robotics

Conference papers

Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Quirynen, R., and Diehl, M. (2018). Towards a modular software package for

embedded optimization. In *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*

Girrbach, F., Hol, J. D., Zandbergen, R., Verschueren, R., Bellusci, G., and Diehl, M. (2018). On the effect of stabilization methods for quaternion invariants on the uncertainty in optimization-based estimation. In *Joint 9th IFAC Symposium on Robust Control Design and 2nd IFAC Workshop on Linear Parameter Varying Systems*

Verschueren, R., Ferreau, H. J., Zanarini, A., Mercangöz, M., and Diehl, M. (2017a). A stabilizing nonlinear model predictive control scheme for time-optimal point-to-point motions. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*

Ferreau, H. J., Almer, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J. L., Stathopoulos, G., and Jones, C. (2017). Embedded optimization methods for industrial automatic control. In *Proceedings of the IFAC World Congress*

Verschueren, R., van Duijkeren, N., Quirynen, R., and Diehl, M. (2016a). Exploiting convexity in direct optimal control: a sequential convex quadratic programming method. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*

Verschueren, R., van Duijkeren, N., Swevers, J., and Diehl, M. (2016b). Time-optimal motion planning for n-dof robot manipulators using a path-parametric system reformulation. In *Proceedings of the American Control Conference (ACC)*, pages 2092–2097

Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M. (2016c). Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm. In *Proceedings of the European Control Conference (ECC)*

van Duijkeren, N., Verschueren, R., Pipeleers, G., Diehl, M., and Swevers, J. (2016). Path-following nmpc for serial-link robot manipulators using a path-parametric system reformulation. In *Proceedings of the European Control Conference (ECC)*

Verschueren, R., Bruyne, S. D., Zanon, M., Frasch, J. V., and Diehl, M. (2014). Towards time-optimal race car driving using nonlinear MPC in real-time. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 2505–2510

FACULTY OF ENGINEERING
DEPARTMENT OF MICROSYSTEMS ENGINEERING
SYSTEMS CONTROL AND OPTIMIZATION LABORATORY
Georges-Köhler-Allee 102
79110 Freiburg im Breisgau