KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Electrical Engineering

Model Predictive Control Algorithms for Applications with Millisecond Timescales

Hans Joachim Ferreau

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering Science

October 2011

Model Predictive Control Algorithms for Applications with Millisecond Timescales

Hans Joachim Ferreau

Jury:
Prof. Dr. Paula Moldenaers, chair
Dr. Paul Goulart (ETH Zürich)
Prof. Dr. Marc Van Barel
Prof. Dr. Jan Swevers
Prof. Dr. Marc Moonen
Prof. Dr. Joos Vandewalle, co-promotor
Prof. Dr. Moritz Diehl, promotor

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science

October 2011

Legal depot number: D/2011/7515/105
ISBN number: 978-94-6018-404-8

Acknowledgements

I would like to thank all persons who helped me while writing this thesis. First of all I thank my promotor Moritz Diehl and all other members of my PhD jury for intensive personal support and scientific advice. It has been a pleasure to discuss mathematical and algorithmic ideas and to identify challenging applications of embedded dynamic optimisation.

Moreover, I very much appreciated the opportunity to work on software that is open-source. In my opinion, this greatly increases its possible positive impact on society—the main final aim of any research (at least in the longer run). Thus, I also would like to thank everybody who supported and contributed to the two software packages described in this thesis. In particular my colleague Boris Houska, the other main developer of the `ACADO Toolkit`, and Andreas Potschka and Christian Kirches for contributing new ideas to `qpOASES`. There have been many more people that helped with beta-testing, contributing interfaces or making real-world applications running, but I hope they do not mind if I choose to refrain from mentioning them one by one.

During my PhD studies I enjoyed participating in two industry projects with Hoerbiger Control Systems AB and IPCOS NV. I would like to thank the respective persons-in-charge in Sweden and Belgium for our productive co-operations and interesting discussions. I also thank Stephen Boyd and his research group for hosting me three months at Stanford University.

Moreover, I gratefully acknowledge financial support of the *Fonds Wetenschappelijk Onderzoek – Vlaanderen (FWO)*, that granted a 4-year scholarship and covered travel expenses.

Finally, I cordially thank my family, my fiancé Nadia, my son Felix and my yet unborn daughter for all their love and encouragement during the last years.

Abstract

The last three decades have seen a rapidly increasing number of applications where model predictive control (MPC) led to better control performance than more traditional approaches. This thesis aims at lowering the practical burden of applying fast MPC algorithms in the real-world. To this aim, it contributes two software packages, which are released as open-source code in order to stimulate their widespread use. Both packages implement previously published methods but enrich them with a number of new theoretical and algorithmic ideas.

The first part of this thesis focusses on efficiently solving quadratic programs (QPs) as arising in linear MPC problems. To this end, it reviews the author's previous work on developing an online active set strategy to exploit the parametric nature of these QPs. This strategy is extended with ideas for initialising the solution procedure and treating QPs with semi-definite Hessian matrices. The software package `qpOASES` implements the online active set strategy and its extensions together with a number of tailored solution variants for special QP formulations. It offers interfaces to third-party software like MATLAB/SIMULINK and has been successfully used in a number of academic real-world MPC applications. Moreover, two industrial applications of `qpOASES`—dealing with emission control of integral gas engines and feasibility management for MPC in the process industry—are described. These industrial case studies also led to further theoretical ideas, namely the use of MPC with an asymmetric cost function and a novel method for handling infeasible QPs based on the online active set strategy.

The second part addresses nonlinear MPC problems and presents the `ACADO Toolkit`, a new software environment and algorithm collection for automatic control and dynamic optimisation. It has been designed for setting-up nonlinear optimal control and MPC problems in a user-friendly way and solving them efficiently. In particular, the `ACADO Toolkit` implements two algorithmic variants of the real-time iteration scheme: a Gauss-Newton approach for nonlinear MPC formulations involving a tracking objective function as well as an exact Hessian approach for tackling time-optimal formulations. The underlying QP subproblems are solved by means of the online active set strategy. The `ACADO Toolkit` features an

intuitive symbolic syntax for formulating MPC problems, which offers a couple of advantageous possibilities. Most importantly, it allows the user to automatically generate optimised, highly efficient C code that is tailored to each respective MPC problem formulation. Numerical results show that the exported code exhibits a promising computational performance allowing application of nonlinear MPC to non-trivial processes at kilohertz sampling rates.

Key words: model predictive control, numerical optimal control, dynamic optimisation, embedded optimisation, parametric quadratic programming, online active set strategy, infeasibility handling, real-time iteration algorithm, code generation, open-source software, `qpOASES`, `ACADO Toolkit`

Nomenclature

Selected Symbols

Model Predictive Control

A, B, C, D	state space matrices describing a linear ODE system
c, c^{term}, r	possibly nonlinear constraint functions
ε	slack variable
f	right-hand side describing nonlinear system dynamics
h	output function of nonlinear system dynamics
M, N	matrices describing linear constraints
n_{p}	length of discrete-time prediction horizon
n_{pa}	number of parameters
n_{u}	number of control inputs
n_{x}	number of differential states
n_{y}	number of outputs
p	vector of parameters
P, Q, R	weighting matrices of a tracking objective function
ψ	Lagrange term of objective function
ϕ	Mayer term of objective function
t	time
t_{p}	length of prediction horizon (fixed)
T	length of prediction horizon (free)
$\mathbb{T}_{\text{p}}$	prediction horizon
$\mathbb{T}_{\text{p}}^{\text{disc}}$	discrete-time prediction horizon

u	vector of control inputs
w_0	initial process state
x	vector of differential states
y	vector of outputs

Quadratic and Nonlinear Programming

$\mathbb{A}(\check{x})$	index set of active constraints at point $\check{x}$
b, $\underline{b_{\mathrm{B}}}$, $\overline{b_{\mathrm{B}}}$, $\underline{b_{\mathrm{C}}}$, $\overline{b_{\mathrm{C}}}$	constraint vectors
$\mathrm{CR}_{\mathbb{A}}$	critical region of an optimal active-set $\mathbb{A}$
$\mathcal{E}$	slack variable
F	NLP objective function
$\mathcal{F}$, $\mathcal{F}(w)$	feasible set of a (parametric) QP/NLP
g	QP gradient vector
G	QP constraint matrix *or* function defining NLP equalities
H	QP Hessian matrix *or* function defining NLP inequalities
$\mathbb{I}(\check{x})$	index set of inactive constraints at point $\check{x}$
m	number of QP constraints
n	number of QP/NLP variables
n_{G}	number of NLP equality constraints
n_{H}	number of NLP inequality constraints
$\mathcal{P}$	set of feasible parameters of a parametric QP/NLP
q	parameter for control parameterisation
s	parameter for state discretisation
τ	homotopy parameter *or* time instant of discretisation grid
V	function within least-squares objective
w	varying parameter
x, x^{opt}	(optimal) primal QP variables
X, X^{opt}	(optimal) primal NLP variables
y, y^{opt}	(optimal) dual QP variables
$\lambda, \lambda^{\mathrm{opt}}$	(optimal) dual NLP variables, equality constraints
μ, μ^{opt}	(optimal) dual NLP variables, inequality constraints

Mathematical Expressions

0	real matrix of appropriate dimensions with all elements zero
2^M	power set of set M
$\mathcal{D}$	domain of a real function
I_n	n-dimensional identity matrix
$\mathbb{N}$	set of natural numbers (greater than 0)
$\mathcal{O}(\cdot)$	big-O notation describing limiting behaviour
π	ratio of any circle's circumference to its diameter
$\mathbb{R}$	field of real numbers
$\mathbb{R}_{\geq 0}$	set of nonnegative real numbers
$\mathbb{R}_{>0}$	set of positive real numbers
$\mathbb{R}^n$	set of real n-dimensional vectors
$\mathbb{R}^{m\times n}$	set of real $m \times n$-dimensional matrices
$\mathcal{S}^n$	set of real symmetric $n \times n$-matrices
$\mathcal{S}^n_{\succeq 0}$	set of real symmetric positive semi-definite $n \times n$-matrices
$\mathcal{S}^n_{\succ 0}$	set of real symmetric positive definite $n \times n$-matrices
∞	infinity
$\forall$	for all
$\exists$	there exist
$\emptyset$	empty set
$[\cdot, \cdot]$	closed interval of real numbers
$(\cdot, \cdot)$	open interval of real numbers
$\cdot \stackrel{\text{def}}{=} \cdot$	defines the symbol on the left to equal the expression on the right
$\cdot \leftarrow \cdot$	assigns the value of the variable on the left to the variable on the right
$\dot{f}(t)$	first derivative of function f with respect to time t
∇f	gradient vector of function f
M'	transpose of vector or matrix M
M^{-1}	inverse of regular matrix M
$\lvert\cdot\rvert$	absolute value of a real number *or* cardinality of a set
$\lVert\cdot\rVert_2$	Euclidean norm of a matrix or vector

Abbreviations and Acronyms

Besides common expressions and SI units the following abbreviations and acronyms are used:

AD	automatic differentiation
BDF	backward differentiation formulae
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CP	convex program
CPU	central processing unit
CSTR	continuous stirred tank reactor
DAE	differential algebraic equation
GMRES	generalised minimal residual method
HJB	Hamilton-Jacobi-Bellmann
iff	if and only if
IP	interior-point
LGPL	Lesser General Public License
LICQ	linear independence constraint qualification
LP	linear program
KKT	Karush-Kuhn-Tucker
MIMO	multiple input multiple output
min	minimise
MPC	model predictive control
NLP	nonlinear programming (problem)
NMPC	nonlinear model predictive control
ODE	ordinary differential equation
PLC	programmable logic controller
QP	quadratic programming (problem)
rpm	revolutions per minute
RK	Runge-Kutta
RTI	real-time iteration
SISO	single input single output
SQP	sequential quadratic programming
s. t.	subject to

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The last three decades have seen a rapidly increasing number of applications where control techniques based on dynamic optimisation led to improved performance [195], e.g. maximising the process output or minimising energy use and emissions. These techniques use a mathematical model in the form of differential equations of the process to be controlled in order to predict its future behaviour and to calculate optimised control actions. Sometimes this can be done once before the runtime of the process to yield an offline controller. However, unknown or unmodelled disturbances often call for a feedback controller that repeatedly solves optimal control problems in real-time, i.e., during the runtime of the process. The notion model predictive control (MPC) refers to this second case.

MPC has a couple of advantages over traditional control approaches. Its key feature is that the control objective as well as desired limitations on the process behaviour can directly be specified within an optimal control problem. This mostly avoids the use of heuristics for designing the controller and also facilitates its tuning. Mathematical optimisation techniques are then used to obtain control actions that are optimal solutions of such optimal control problems. Moreover, MPC problem formulations can also directly include predictive information, allowing the controller to react pro-actively to future changes in the setup. Finally, MPC naturally handles processes with multiple inputs or outputs as it conceptually can be used with dynamic models of any dimension. These advantages come at the expense that optimal control problems need to be solved in real-time—possibly on embedded controller hardware. This becomes particularly challenging

if the controlled process dynamics are fast and thus require a controller that provides feedback at high sampling rates, like in many mechanical or automotive applications.

What makes solving MPC problems challenging? The answer is twofold: if the model features nonlinear dynamics, the resulting optimisation problem typically becomes nonconvex. Thus, the optimal solution might not be unique and, moreover, many sub-optimal local solutions might exist. This fact complicates both the solution procedure, resulting in higher computational load, and the theoretical analysis of the closed-loop behaviour of the process. Second, even if a linear dynamic model leads to a convex problem with unique optimal solution, solving it reliably within short sampling times—below a millisecond, say—is still computationally demanding for non-trivial systems. Besides the computational issues, a lot of effort has been spent to investigate theoretical properties of MPC algorithms. Among them, stability analysis has been by far the most important one, but also feasibility issues or guaranteed bounds on the computational runtime have been investigated. According to [198], several hundred articles on MPC have been published each year during the last decade. This sheds some light on the relevance of MPC and also gives rise to the hope that MPC theory is getting mature (in particular for the linear case).

The main objective of this thesis is to lower the practical burden of applying fast MPC algorithms in the real world. To this aim, it contributes two software packages: First, the quadratic programming solver `qpOASES` designed for solving linear MPC problems very efficiently, which has already been used for two industrial and numerous academic real-world applications. Second, the `ACADO Toolkit` for setting up and solving nonlinear MPC problems in a user-friendly way, which also offers the possibility to automatically generate optimised, highly efficient C code. Both packages implement previously published methods but enrich them with a number of new theoretical and algorithmic ideas. They are released as open-source code in order to stimulate their widespread use. All contributions of this thesis are discussed in more detail in the following section.

A last but important remark concerns the practical relevance of the topic and its possible benefits for the society. Though it is hard to predict the future (lacking a sufficiently accurate model of the world), there is some evidence that MPC will be able to contribute to some of mankind's main future challenges: Given the fact that MPC is already nowadays applied by many chemical companies to reduce energy and raw material consumption without jeopardising product quality, it will play its role in *saving natural resources*. Also increasing *individual mobility* calls for automatic control methods like MPC, not only to save fuel or reduce emissions, but also to realise new innovations such as autonomous driving. Yet another example is the *health care* sector, where possible benefits of MPC are currently investigated in pilot-projects, e.g. to optimally dose insulin injections.

1.2 Contribution

This thesis is divided into two parts focussing on algorithms for linear (Chapters 2–5) and nonlinear MPC (Chapters 6–8), respectively. We briefly summarise each chapter and mention their respective contribution.

Part I: Linear Model Predictive Control

- *Chapter 2* surveys existing numerical methods for solving linear MPC problems. It discusses a number of iterative solution methods and methods that explicitly pre-compute the solution as well as combinations of both. Also a basic introduction to quadratic programming (QP) is given, as basically all mentioned linear MPC methods rely on solving QP problems.

- *Chapter 3* starts with reviewing the author's previous work on developing an online active set strategy [75, 77] for solving QPs arising in linear MPC very efficiently. Afterwards, novel ideas to initialise the online active set strategy are described, which can substantially speed-up QP solution in case a good initial guess for the optimal solution is available. Also the presented idea of combining the online active set strategy with a proximal point method to obtain an efficient regularisation scheme for solving convex QPs is new.

- *Chapter 4* presents the main contribution of the first part of this thesis, namely a proper implementation of the online active set strategy within the open-source software `qpOASES`. It builds on the ad-hoc implementation mentioned in [77] but improves it in many important ways: greatly enhanced reliability and maintainability of the code, tailored solution variants for a number of special QP formulations to increase efficiency, several interfaces to frequently used third-party software packages and a detailed user's manual. Effectiveness and usability of the implementation is confirmed by mentioning a number of academic real-world applications of `qpOASES`, partly on embedded controller hardware. Finally, it is shown that `qpOASES` can significantly outperform other popular academic and commercial QP solvers on small- to medium-scale test examples.

- *Chapter 5* discusses several practical issues motivated by the use of `qpOASES` within two industrial case studies. The first case study describes the use of `qpOASES` on embedded controller hardware for reducing the emissions of integral gas engines. In this context, the use of linear MPC with an asymmetric cost function is proposed to improve control performance for processes described by Wiener models. The second case study briefly discusses MPC infeasibility handling for linear MPC applications in the process industry. For this aim, `qpOASES` has been integrated into an existing

commercial MPC software suite. Moreover, a novel strategy to efficiently handle infeasible QPs based on the online active set strategy is proposed.

Part II: Nonlinear Model Predictive Control

- *Chapter 6* summarises existing techniques to transform an infinite-dimensional optimal control problem into a finite-dimensional nonlinear programming problems (NLPs) by means of direct methods. It discusses the two main classes of methods, namely sequential quadratic programming and interior-point methods, for solving these specially structured NLPs and how these methods need to be adapted to work efficiently. Finally, a survey of existing algorithms for nonlinear MPC is given.

- *Chapter 7* introduces the newly developed open-source software package `ACADO Toolkit`. It provides a general and flexible framework for using a number of algorithms for dynamic optimisation, including nonlinear MPC. Moreover, it is able to directly handle symbolic problem formulations, which not only allows for a user-friendly syntax to setup optimisation problems but also provides interesting algorithmic possibilities (like the code generation and optimisation tool described in Chapter 8). This symbolic syntax and its flexible software design are distinguishing features of the `ACADO Toolkit` compared to other existing optimisation software packages. The `ACADO Toolkit` has been developed jointly with Boris Houska, who is the main author of the symbolic expression classes and the algorithms for solving offline optimal control problems. The implementation of algorithms for nonlinear MPC constitutes the first main contribution of the second part of this thesis. In particular, two previously proposed variants of the real-time iteration scheme to handle MPC formulations—that either comprise a tracking objective function or aim at time-optimal process behaviour—have been implemented within the `ACADO Toolkit`.

- *Chapter 8* introduces the idea to automatically generate optimised source-code of optimisation algorithms for speeding-up computations. Following this approach, a novel software tool to automatically export real-time iteration algorithms for use in nonlinear MPC is presented. This tool has been implemented as an add-on to the `ACADO Toolkit` and can thus make use of its symbolic syntax to setup the MPC problem. That way the symbolic problem formulation is used to export highly efficient and self-contained C code that is tailored to each respective MPC problem. It is illustrated that automatically generated NMPC algorithms can perform significantly faster than their counterparts implemented in a generic way—at least on small-scale NMPC examples. The `ACADO Code Generation` tool has been developed jointly with Boris Houska and is the second main contribution of the second part of this thesis.

A motivation for the research carried out in this thesis is given in *Chapter 1*, where also its contributions are summarised. Moreover, it introduces the mathematical framework to formulate linear and nonlinear MPC problems. *Chapter 9* concludes the thesis and discusses possible topics of future research.

1.3 Model Predictive Control

Model predictive control (MPC) repeatedly calculates control actions which optimise the forecasted process behaviour. The prediction is based on a dynamic model of the process to be controlled. At each sampling instant, this leads to an optimal control problem which needs to be solved online. Afterwards, the optimised control action is applied to the process until the next sampling instant when an updated optimal control problem, incorporating the new process state, is solved. Hence, model predictive control is a feedback control strategy, sometimes also referred to as receding horizon control. A thorough introduction can be found in several textbooks, e.g. [46, 165].

Unless otherwise stated, in this thesis we consider time-continuous process models described by an *ordinary differential equation* (ODE) and an *output function*. We assume the process model to be defined over a time interval $\mathbb{T} \stackrel{\text{def}}{=} [t_{\text{start}}, t_{\text{end}}] \subset \mathbb{R}$, and to have the following form:

$$x(t_{\text{start}}) = w_0\,, \tag{1.1a}$$

$$\dot{x}(t) = f\left(t, x(t), u(t), p\right) \quad \forall\, t \in \mathbb{T}\,, \tag{1.1b}$$

$$y(t) = h\left(t, x(t), u(t), p\right) \quad \forall\, t \in \mathbb{T}\,, \tag{1.1c}$$

with differential states $x : \mathbb{T} \to \mathbb{R}^{n_{\text{x}}}$, control inputs (or manipulated variables) $u : \mathbb{T} \to \mathbb{R}^{n_{\text{u}}}$, time-constant parameters $p \in \mathbb{R}^{n_{\text{pa}}}$, and outputs, or controlled variables, $y : \mathbb{T} \to \mathbb{R}^{n_{\text{y}}}$. Moreover, $w_0 \in \mathbb{R}^{n_{\text{x}}}$ is the initial value of the differential states, $f : \mathcal{D}_f \subseteq \mathbb{R} \times \mathbb{R}^{n_{\text{x}}} \times \mathbb{R}^{n_{\text{u}}} \times \mathbb{R}^{n_{\text{pa}}} \to \mathbb{R}^{n_{\text{x}}}$ the right-hand function of the ODE, and $h : \mathcal{D}_h \subseteq \mathbb{R} \times \mathbb{R}^{n_{\text{x}}} \times \mathbb{R}^{n_{\text{u}}} \times \mathbb{R}^{n_{\text{pa}}} \to \mathbb{R}^{n_{\text{y}}}$ denotes the output function of the process model.

There exists a great variety of different model types within the MPC context. They can be roughly divided into first principles (or white-box) models, identified (or black-box) models, or combinations of both (often called grey-box models). First principles models try to replicate, e.g., physical or chemical laws of nature, whereas identified models are based on measurements of the real process. Examples for different model types and their application in industry can be found in [195].

Based on dynamic models of form (1.1), we introduce

Definition 1.1 (optimal control problem): *An* optimal control problem *over the prediction horizon* $\mathbb{T}_\mathrm{p} \overset{\mathrm{def}}{=} [t_0, t_0 + t_\mathrm{p}]$*, with* $t_\mathrm{p} \in \mathbb{R}_{>0}$*, is the task of finding optimal control inputs* $u(t)$ *solving*

$$\mathrm{OCP}(t_0, w_0): \quad \min_{\substack{x(\cdot),u(\cdot),\\ y(\cdot),p}} \int_{t_0}^{t_0+t_\mathrm{p}} \psi\left(t, y(t), u(t), p\right)\, dt \,+\, \phi\left(t_0+t_\mathrm{p}, x(t_0+t_\mathrm{p}), p\right) \tag{1.2a}$$

$$\text{s.t.} \quad x(t_0) \;=\; w_0\,, \tag{1.2b}$$

$$\dot{x}(t) \;=\; f\left(t, x(t), u(t), p\right) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{1.2c}$$

$$y(t) \;=\; h\left(t, x(t), u(t), p\right) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{1.2d}$$

$$0 \;\geq\; c\left(t, y(t), u(t), p\right) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{1.2e}$$

$$0 \;\geq\; c^\mathrm{term}\left(t_0 + t_\mathrm{p}, x(t_0 + t_\mathrm{p}), p\right)\,, \tag{1.2f}$$

where w_0 *denotes the initial process state at the starting time* t_0*. It aims at minimising an objective function comprising a* Lagrange term $\psi: \mathcal{D}_\psi \subseteq \mathbb{R} \times \mathbb{R}^{n_\mathrm{y}} \times \mathbb{R}^{n_\mathrm{u}} \times \mathbb{R}^{n_\mathrm{pa}} \to \mathbb{R}$ *(sometimes called the "running cost") and a* Mayer term $\phi: \mathcal{D}_\phi \subseteq \mathbb{R} \times \mathbb{R}^{n_\mathrm{y}} \times \mathbb{R}^{n_\mathrm{pa}} \to \mathbb{R}$ *(or "terminal cost"), respectively. Moreover, optimisation can be subject to inequality constraints defined by the path constraints* $c: \mathcal{D}_c \subseteq \mathbb{R} \times \mathbb{R}^{n_\mathrm{y}} \times \mathbb{R}^{n_\mathrm{u}} \times \mathbb{R}^{n_\mathrm{pa}} \to \mathbb{R}^{n_\mathrm{c}}$ *and the terminal constraints* $c^\mathrm{term}: \mathcal{D}_{c^\mathrm{term}} \subseteq \mathbb{R} \times \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{pa}} \to \mathbb{R}^{n_\mathrm{c,term}}$*.* ○

The capability to enforce constraints on the control inputs, outputs or parameters is one of the most important features of MPC. Also additional equality constraints can be expressed using this formulation.

Let us assume that the process to be controlled by means of MPC starts at time instant t_start, ends at time instant t_end and that

$$t^{(0)} \overset{\mathrm{def}}{=} t_\mathrm{start} < t^{(1)} < \ldots < t^{(n_\mathrm{sample})} \overset{\mathrm{def}}{=} t_\mathrm{end}\,, \quad n_\mathrm{sample} \in \mathbb{N}\,, \tag{1.3}$$

is a sequence of sampling instants satisfying $t^{(i)} - t^{(i-1)} \leq t_\mathrm{p}$ for all $i \in \{1, \ldots, n_\mathrm{sample}\}$. If the sampling instants are chosen *equidistant*, i.e.

$$\delta \overset{\mathrm{def}}{=} \frac{t_\mathrm{end} - t_\mathrm{start}}{n_\mathrm{sample}}\,, \quad t^{(i)} \overset{\mathrm{def}}{=} t_\mathrm{start} + i \cdot \delta \quad \forall\, i \in \{1, \ldots, n_\mathrm{sample}\}\,, \tag{1.4}$$

we call $\delta \in \mathbb{R}_{>0}$ the *sampling time*.

Having solved OCP$(t^{(i)}, w_0)$, the optimal control input $u^{\mathrm{opt}}(t)$ is applied to the process until the next sampling instant $t^{(i+1)}$. Then the current process state is obtained (measured or estimated) and the optimal control problem OCP$(t^{(i+1)}, w_0)$ is solved with this updated initial value for the process state. This yields the model predictive control scheme which is summarised in Algorithm 1.1.

Algorithm 1.1 (model predictive control concept)

`input:` (open-loop) optimal control problem OCP$(t^{(0)}, w_0)$, sequence of sampling instants $t^{(0)}, t^{(1)}, \ldots, t^{(n_{\mathrm{sample}}-1)}$ as in (1.3)

`output:` piecewise defined optimised process inputs $u^* : [t_{\mathrm{start}}, t_{\mathrm{end}}] \to \mathbb{R}^{n_{\mathrm{u}}}$

(1) Set $i \leftarrow 0$.

(2) Obtain process state w_0 at time $t^{(i)}$ and formulate OCP$(t^{(i)}, w_0)$.

(3) Obtain $u^{\mathrm{opt}}(t)$, $t \in [t^{(i)}, t^{(i)} + t_{\mathrm{p}}]$, by solving OCP$(t^{(i)}, w_0)$.

(4) Set $u^*(t) \stackrel{\mathrm{def}}{=} u^{\mathrm{opt}}(t)\ \forall\, t \in [t^{(i)}, t^{(i+1)}]$ and apply $u^*(t)$ to the process until $t^{(i+1)}$.

(5) Stop if $i = n_{\mathrm{sample}} - 1$, otherwise set $i \leftarrow i + 1$ and continue with step (2).

One may ask why it is necessary to solve the optimal control problem repeatedly: If one would choose $t_{\mathrm{p}} \stackrel{\mathrm{def}}{=} t_{\mathrm{end}} - t_{\mathrm{start}}$, it would suffice to solve the first problem OCP$(t_{\mathrm{start}}, w_0)$ and to apply the resulting optimal control for all times in $[t_{\mathrm{start}}, t_{\mathrm{end}}]$. This objection is justified if one assumes that the model describes the real process *exactly* and that all inputs can be applied to the real process *instantaneously*.

However, these conditions are never satisfied in a real-world setup: there are always discrepancies between the model and real process, known as *model-plant mismatch*, as the real process is too complex to be modelled exactly. Sometimes the process dynamics are not even known completely making approximations or interpolations necessary. Moreover, unknown *disturbances* are almost always present in real-world and *measurement noise*[1] impedes the exact determination of the initial process state. On the other hand, the calculated optimal control inputs often cannot be applied exactly to the real process. Since actuators, valves and even electronic

[1] We should emphasise that the current process state w_0 is never known exactly in practice since it has to be obtained by means of more or less inaccurate sensors.

devices need a short time period, known as *dead time*, to react, there is always a short delay when applying the optimal control inputs (although this could be counteracted by prediction). A further delay stems from the fact that the controller needs time to calculate new optimised control inputs. And even if these delays are negligible, deviations between the optimised and the applied control inputs may occur because the actuators are not able to behave like the optimised function $u(t)$ possibly including discontinuities.

Without loss of generality, we eliminate the explicit dependencies of ψ, ϕ, f, h, c on t and p for ease of notation:

- Most presentations on MPC require the process model to be *time-invariant*. Explicit time-dependence can be eliminated by introducing an additional state $x_{n_\mathrm{x}+1}(t)$ and the additional differential equation:

$$x_{n_\mathrm{x}+1}(t_0) = t_0\,, \tag{1.5a}$$

$$\dot{x}_{n_\mathrm{x}+1}(t) = 1 \quad \forall\, t \in \mathbb{T}\,. \tag{1.5b}$$

- Also *parameters* can be written as differential states by introducing additional states $x_{n_\mathrm{x}+i}(t)$, $1 \leq i \leq n_\mathrm{pa}$, and imposing the additional equations:

$$x_{n_\mathrm{x}+i}(t_0) = p_i \quad \forall\, i \in \{1, \ldots, n_\mathrm{pa}\}\,, \tag{1.6a}$$

$$\dot{x}_{n_\mathrm{x}+i}(t) = 0 \quad \forall\, t \in \mathbb{T} \quad \forall\, i \in \{1, \ldots, n_\mathrm{pa}\}\,. \tag{1.6b}$$

We end our general introduction to model predictive control with the following frequently used definition:

Definition 1.2 (steady-state): *Every pair* $(\check{x}, \check{u})$ *satisfying*

$$0 = f\big(\check{x}, \check{u}\big) \tag{1.7}$$

is called a steady-state *of the ODE system* $\dot{x}(t) = f\big(x(t), u(t)\big)$. ○

This means that a process is at steady-state $(\check{x}\,, \check{u})$ if and only if it remains there when input $\check{u}$ is applied.

Special Case: Linear Model Predictive Control

The first part of this thesis deals with a special case of the general MPC setup introduced so far, namely *linear MPC*. It refers to situations in which a linear

time-invariant process model, linear constraints and a convex quadratic objective function is used. This does *not* imply that the real process to be controlled is governed by linear dynamics.

A (continuous-time) process model is called *linear time-invariant* if it can be written in the form

$$x(t_{\text{start}}) = w_0\,, \tag{1.8a}$$

$$\dot{x}(t) = Ax(t) + Bu(t) \quad \forall\, t \in \mathbb{T}\,, \tag{1.8b}$$

$$y(t) = Cx(t) + Du(t) \quad \forall\, t \in \mathbb{T}\,, \tag{1.8c}$$

with constant matrices[2] $A \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{x}}}$, $B \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{u}}}$, $C \in \mathbb{R}^{n_{\mathrm{y}} \times n_{\mathrm{x}}}$, $D \in \mathbb{R}^{n_{\mathrm{y}} \times n_{\mathrm{u}}}$. Since most real processes exhibit nonlinearities, linear process models are often obtained by linearising a nonlinear model at some working point (normally at a steady-state) or by employing system identification techniques [161].

Based on linear dynamic models of form (1.8), we introduce

Definition 1.3 (linear optimal control problem): *A* linear optimal control problem *over the prediction horizon* $\mathbb{T}_{\mathrm{p}} \stackrel{\text{def}}{=} [t_0, t_0 + t_{\mathrm{p}}]$*, with* $t_{\mathrm{p}} \in \mathbb{R}_{>0}$ *fixed, is the task of finding optimal control inputs* $u(t)$ *solving*

$$\mathrm{OCP}_{\mathrm{lin}}(t_0, w_0): \min_{\substack{x(\cdot),u(\cdot),\\ y(\cdot)}} \int_{t_0}^{t_0+t_{\mathrm{p}}} \hat{y}(t)'Q\hat{y}(t) + \hat{u}(t)'R\hat{u}(t)\,dt \;+\; \hat{x}(t_0+t_{\mathrm{p}})'P\hat{x}(t_0+t_{\mathrm{p}}) \tag{1.9a}$$

$$\text{s.t.} \quad x(t_0) = w_0\,, \tag{1.9b}$$

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad \forall\, t \in \mathbb{T}_{\mathrm{p}}\,, \tag{1.9c}$$

$$y(t) = Cx(t) + Du(t) \qquad \forall\, t \in \mathbb{T}_{\mathrm{p}}\,, \tag{1.9d}$$

$$b(t) \geq M(t)y(t) + N(t)u(t) \quad \forall\, t \in \mathbb{T}_{\mathrm{p}}\,, \tag{1.9e}$$

where we introduce the definitions $\hat{x}(t) \stackrel{\text{def}}{=} x(t) - x^{\mathrm{ref}}(t)$*,* $\hat{y}(t) \stackrel{\text{def}}{=} y(t) - y^{\mathrm{ref}}(t)$ *and* $\hat{u}(t) \stackrel{\text{def}}{=} u(t) - u^{\mathrm{ref}}(t)$*. We aim at minimising the convex quadratic objective function given by constant matrices* $Q \in \mathcal{S}^{n_{\mathrm{y}}}_{\succeq 0}$*,* $R \in \mathcal{S}^{n_{\mathrm{u}}}_{\succ 0}$*,* $P \in \mathcal{S}^{n_{\mathrm{y}}}_{\succeq 0}$ *and possibly time-varying* reference values $x^{\mathrm{ref}} : \mathbb{R} \to \mathbb{R}^{n_{\mathrm{x}}}$*,* $y^{\mathrm{ref}} : \mathbb{R} \to \mathbb{R}^{n_{\mathrm{y}}}$ *and* $u^{\mathrm{ref}} : \mathbb{R} \to \mathbb{R}^{n_{\mathrm{u}}}$*. Moreover, optimisation can be subject to linear inequality constraints defined by*

[2] Linear *time-variant* process models allow for time-varying matrices $A(t)$, $B(t)$, $C(t)$, $D(t)$.

possibly time-varying matrices $M : \mathbb{R} \to \mathbb{R}^{n_c \times n_y}$, $N : \mathbb{R} \to \mathbb{R}^{n_c \times n_u}$ *and upper bound vector* $b : \mathbb{R} \to \mathbb{R}^{n_c}$. ○

Linear MPC problems with this type of objective are often referred to as reference (or trajectory) tracking problems. In the special case of $y(t) \stackrel{\text{def}}{=} x(t)\ \forall\, t \in \mathbb{T}$ as well as $y^{\text{ref}} \stackrel{\text{def}}{=} 0$ and $u^{\text{ref}} \stackrel{\text{def}}{=} 0$, they aim at regulating the process to the origin. The matrices Q and R penalise deviations of the process outputs from desired reference values; whereas P is included for stability reasons (see e.g. [221, 143, 29, 50, 171] and the references therein).

As a special case of (1.9e), many linear MPC formulations comprise box-constraints of the form

$$\underline{u} \leq u(t) \leq \overline{u} \quad \forall\, t \in \mathbb{T}_{\text{p}}\,, \tag{1.10a}$$

$$\underline{y} \leq y(t) \leq \overline{y} \quad \forall\, t \in \mathbb{T}_{\text{p}}\,, \tag{1.10b}$$

where $\underline{u}, \overline{u} \in \mathbb{R}^{n_u}$ and $\underline{y}, \overline{y} \in \mathbb{R}^{n_y}$. Input bounds typically express physical limitations of the actuators, output bounds are often necessary to ensure safe process operating conditions.

Problem Discretisation

If $u(t)$ is allowed to be an arbitrary measurable real-valued function, both $\text{OCP}(t_0, w_0)$ and $\text{OCP}_{\text{lin}}(t_0, w_0)$ constitute infinite dimensional (over $\mathbb{R}$) optimisation problems. Although there exist necessary conditions—based on the *calculus of variations* or *Pontryagin's maximum principle* [124], [192]—for finding the optimal solution of such problems, these so-called *indirect methods* are often of limited use for MPC purposes (cf. [22, p. 85-87]). In contrast, *direct methods* parameterise the control functions in order to reduce the optimal control problem to a finite-dimensional one. This loss of degrees of freedom greatly simplifies the solution of the problem and is most often irrelevant for process performance in practice. A very popular *control parameterisation* is to require that the control functions are *piecewise constant* (or *piecewise linear*) on an equidistant grid.

Introducing a control parameterisation allows one to express the state and output trajectories $x(t)$ and $y(t)$ as functions of the initial value w_0 and finitely many optimisation variables representing the control inputs. Also the constraints need to be discretised and their fulfilment is ensured only at a finite number of time instants, e.g. at the grid points of the control parameterisation. Similarly, the continuous objective function is evaluated on a discrete time-grid only. That way the optimal control problem $\text{OCP}(t_0, w_0)$ is transformed into a *nonlinear programming problem* (NLP) with a nonlinear objective function and

possibly nonlinear constraints. In the linear MPC case, the optimal control problem $\mathrm{OCP}_{\mathrm{lin}}(t_0, w_0)$ is transformed into a *quadratic programming problem* (QP) comprising a convex quadratic objective function and linear constraints. A brief overview on direct methods is given in Subsection 6.1.2.

If we parameterise the control inputs as piecewise constant functions on an equidistant grid and evaluate the objective function and constraints on this grid only, we end up with

Definition 1.4 (discrete-time linear optimal control problem): *A* discrete-time linear optimal control problem *over the discrete-time prediction horizon* $\mathbb{T}_{\mathrm{p}}^{\mathrm{disc}} \stackrel{\mathrm{def}}{=} \{k_0, \ldots, k_0 + n_{\mathrm{p}} - 1\}$*, with* $n_{\mathrm{p}} \in \mathbb{N}$ *fixed, is the task of finding a sequence of constant optimal control inputs* $u_{k_0}, \ldots, u_{k_0+n_{\mathrm{p}}-1}$ *solving*

$$\mathrm{OCP}_{\mathrm{lin}}^{\mathrm{disc}}(k_0, w_0): \min_{\substack{x_{k_0},\ldots,x_{k_0+n_{\mathrm{p}}},\\ y_{k_0},\ldots,y_{k_0+n_{\mathrm{p}}},\\ u_{k_0},\ldots,u_{k_0+n_{\mathrm{p}}-1}}} \sum_{k=k_0}^{k_0+n_{\mathrm{p}}-1} \hat{y}_k' Q \hat{y}_k + \hat{u}_k' R \hat{u}_k + \hat{x}_{k_0+n_{\mathrm{p}}}' P \hat{x}_{k_0+n_{\mathrm{p}}} \tag{1.11a}$$

$$\text{s.t.} \quad x_{k_0} = w_0\,, \tag{1.11b}$$

$$x_{k+1} = A^{\mathrm{disc}} x_k + B^{\mathrm{disc}} u_k \quad \forall\, k \in \mathbb{T}_{\mathrm{p}}^{\mathrm{disc}}, \tag{1.11c}$$

$$y_k = C x_k + D u_k \quad \forall\, k \in \mathbb{T}_{\mathrm{p}}^{\mathrm{disc}}, \tag{1.11d}$$

$$b_k \geq M_k y_k + N_k u_k \quad \forall\, k \in \mathbb{T}_{\mathrm{p}}^{\mathrm{disc}}, \tag{1.11e}$$

$$b_{k_0+n_{\mathrm{p}}} \geq M_{k_0+n_{\mathrm{p}}} x_{k_0+n_{\mathrm{p}}}\,, \tag{1.11f}$$

where all quantities, except for $A^{\mathrm{disc}} \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{x}}}$ *and* $B^{\mathrm{disc}} \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{u}}}$*, are defined analogously to Definition 1.3.* ○

The discrete-time system matrices A^{disc} and B^{disc} can be easily calculated from their time-continuous counterparts (see e.g. [91]). The label "disc" is usually left out for ease of notation.

Definition 1.4 implies that the control inputs can vary along the whole prediction horizon on the same grid as the discretised differential states. In contrast to this, it is sometimes advantageous to use a *control horizon* that is a subset of the discrete-time prediction horizon $\mathbb{T}_{\mathrm{p}}^{\mathrm{disc}}$. A common choice is to let the control inputs vary only on a few intervals at the beginning of the prediction horizon and to fix them to a given value on all remaining intervals of $\mathbb{T}_{\mathrm{p}}^{\mathrm{disc}}$.

Part I

Linear Model Predictive Control

Chapter 2

Overview of Existing Methods for Linear MPC

This chapter introduces the main concepts of quadratic programming and gives an overview of existing numerical methods to solve linear MPC problems. This survey comprises a number of iterative solution methods, methods that explicitly pre-compute the solution as well as combinations of both.

2.1 Quadratic Programming

We introduce quadratic programming (QP) problems and show that discrete-time linear optimal control problems form a special, highly structured class of QPs. Efficient solution of these QPs in real-time is a key ingredient for fast linear MPC schemes. Moreover, several algorithms for nonlinear MPC rely on fast solutions of QP subproblems.

2.1.1 Definitions

We start by concisely summarising a couple of basic definitions:

Definition 2.1 (quadratic program): *The optimisation problem*

$$\text{QP}: \quad \min_{x \in \mathbb{R}^n} \quad \frac{1}{2} x' H x + x' g \tag{2.1a}$$

$$\text{s.t.} \quad Gx \ \geq \ b\,, \tag{2.1b}$$

with the Hessian matrix $H \in \mathcal{S}^n \stackrel{\text{def}}{=} \{M \in \mathbb{R}^{n\times n} \mid M = M'\}$, *the* gradient vector $g \in \mathbb{R}^n$, *the* constraint matrix $G \in \mathbb{R}^{m\times n}$, *and the* constraint vector $b \in \mathbb{R}^m$, *is called a* quadratic program. ○

Other equivalent definitions of a quadratic program exist. We only note that also equality constraints, bounds on single variables x_i, $1 \leq i \leq n$, or upper constraint limits can be expressed by means of the inequality constraints (2.1b) using a proper choice of G and b.

We denote the i-th row of the constraint matrix G by the row-vector G_i'. The matrix composed of the rows corresponding to constraints in any (ordered) index set $\mathbb{A} \subseteq \{1,\ldots,m\}$ is denoted by $G_{\mathbb{A}}$; the corresponding part of the constraint vector b is denoted by $b_{\mathbb{A}}$.

For describing basic properties of QPs, we introduce the following

Definition 2.2 (feasible, bounded and convex QPs): *A quadratic program of the form* (2.1) *is called*

- feasible *iff its feasible set*

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{\check{x} \in \mathbb{R}^n \mid G\check{x} \geq b\right\} \tag{2.2}$$

  is non-empty and infeasible *otherwise;*

- bounded (from below) *iff there exists a number* $\alpha \in \mathbb{R}$ *such that*

$$\alpha \leq \frac{1}{2}\check{x}'H\check{x} + \check{x}'g \quad \forall\, \check{x} \in \mathcal{F} \tag{2.3}$$

  and unbounded *otherwise;*

- convex *iff its Hessian matrix* H *is symmetric positive semi-definite, i.e.*

$$H \in \mathcal{S}^n_{\succeq 0}\,, \quad \mathcal{S}^n_{\succeq 0} \stackrel{\text{def}}{=} \left\{M \in \mathcal{S}^n \mid v'Mv \geq 0\ \forall\, v \in \mathbb{R}^n\right\} \tag{2.4}$$

  and nonconvex *otherwise;*

- strictly convex *iff its Hessian matrix* H *is symmetric positive definite, i.e.*

$$H \in \mathcal{S}^n_{\succ 0}\,, \quad \mathcal{S}^n_{\succ 0} \stackrel{\text{def}}{=} \left\{M \in \mathcal{S}^n \mid v'Mv > 0\ \forall\, v \in \mathbb{R}^n \setminus \{0\}\right\}. \tag{2.5}$$

○

It is trivial to see that every strictly convex QP is also bounded from below. Moreover, the Frank-Wolfe Theorem states that every strictly convex QP always has a solution if it is feasible [89].

For analysing the solution of a QP, it is often appropriate to follow [70] introducing

Definition 2.3 (dual quadratic program): *We define the* dual quadratic program *of a QP* (2.1) *to be the problem*

$$\mathrm{QP}^{\mathrm{dual}}: \qquad \max_{x\in\mathbb{R}^n,\, y\in\mathbb{R}^m} \quad -\tfrac{1}{2}x'Hx + y'b \tag{2.6a}$$

$$\text{s.t.} \qquad Hx + g = G'y\,, \tag{2.6b}$$

$$y \geq 0\,, \tag{2.6c}$$

where all quantities are defined as in Definition 2.1.

The notions of feasibility, boundedness and convexity (cf. Definition 2.2) also apply to the dual QP; its feasible set *is defined as*

$$\mathcal{F}^{\mathrm{dual}} \stackrel{\mathrm{def}}{=} \left\{ (\check{x}, \check{y}) \in \mathbb{R}^n \mid H\check{x} + g = G'\check{y},\ \check{y} \geq 0 \right\}, \tag{2.7}$$

accordingly. ○

2.1.2 Optimality Conditions

Duality is an important concept in linear programming and convex quadratic programming [70] that also has extensions to general nonlinear programming [249]. The main idea is to formulate a second, dual problem which can be shown (under mild conditions) to have the same optimal objective function value as the original, *primal* one. This and other theoretical properties are very helpful when proving optimality of a certain point and also lead to interesting practical methods for solving quadratic programs, as will be described in Section 2.3.

Since an extensive treatment of duality is beyond the scope of this thesis, we only summarise the main result [70]:

Theorem 2.1 (solution of primal and dual QP): *Let a convex primal and the corresponding dual quadratic program (as defined in Definitions 2.1 and 2.3) be given. Then the following holds:*

(i) If x^{opt} *is a solution to* QP (2.1) *then a solution* $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ *to* $\mathrm{QP}^{\mathrm{dual}}$ *exists.*

(ii) If a solution $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ *to* $\mathrm{QP}^{\mathrm{dual}}$ *exists, then a solution* x *to* QP (2.1) *satisfying* $Hx = Hx^{\mathrm{opt}}$ *exists.*

(iii) In either case

$$\frac{1}{2}\left(x^{\mathrm{opt}}\right)'H\left(x^{\mathrm{opt}}\right) + \left(x^{\mathrm{opt}}\right)'g \;=\; -\frac{1}{2}\left(x^{\mathrm{opt}}\right)'H\left(x^{\mathrm{opt}}\right) + \left(y^{\mathrm{opt}}\right)'b$$

holds. ○

Note that the second proposition simplifies to $x = x^{\mathrm{opt}}$ if the QP is strictly convex (implying that H is invertible). Using Theorem 2.1 it is easy to show that the objective function value of the dual QP at any feasible point provides a lower bound on the optimal objective function value of the primal QP. Another straightforward corollary is that a strictly convex QP is feasible if and only if its dual is bounded from above.

In order to formulate explicit optimality conditions for QPs, we need the following definitions:

Definition 2.4 (active and inactive constraints): *Let a feasible QP of the form* (2.1) *be given. A constraint* $G_i'\check{x} \geq b_i$, $1 \leq i \leq m$, *is called* active *at* $\check{x} \in \mathcal{F}$ *iff* $G_i'\check{x} = b_i$ *holds and* inactive *otherwise. The (disjoint) index sets*

$$\mathbb{A}(\check{x}) \stackrel{\mathrm{def}}{=} \left\{ i \in \{1,\ldots,m\} \mid G_i'\check{x} = b_i \right\},$$

$$\mathbb{I}(\check{x}) \stackrel{\mathrm{def}}{=} \left\{ i \in \{1,\ldots,m\} \mid G_i'\check{x} > b_i \right\}$$

are called set of active constraints*, or more common* active set*,* at $\check{x}$ *and* set of inactive constraints at $\check{x}$*, respectively. If* x^{opt} *is an optimal solution of the QP, we call the corresponding active set* $\mathbb{A}(x^{\mathrm{opt}})$ *the* optimal active set. ○

Now we can state the following optimality conditions which are special variants of the general nonlinear case (cf. [142, 151]):

Theorem 2.2 (Karush-Kuhn-Tucker conditions for QPs): *Let* QP (2.1) *be a strictly convex and feasible quadratic program. Then there exists a unique* $x^{\mathrm{opt}} \in \mathbb{R}^n$ *and at least one index set* $\mathbb{A} \subseteq \mathbb{A}(x^{\mathrm{opt}})$ *and a vector* $y^{\mathrm{opt}} \in \mathbb{R}^m$ *which satisfy the following conditions:*

$$Hx^{\mathrm{opt}} - G_{\mathbb{A}}' y_{\mathbb{A}}^{\mathrm{opt}} = -g\,, \tag{2.8a}$$

$$G_{\mathbb{A}} x^{\mathrm{opt}} = b_{\mathbb{A}}\,, \tag{2.8b}$$

$$G_{\mathbb{I}} x^{\mathrm{opt}} \geq b_{\mathbb{I}}\,, \tag{2.8c}$$

$$y_{\mathbb{I}}^{\mathrm{opt}} = 0\,, \tag{2.8d}$$

$$y_{\mathbb{A}}^{\mathrm{opt}} \geq 0\,, \tag{2.8e}$$

where $\mathbb{I} \stackrel{\mathrm{def}}{=} i \in \{1,\ldots,m\} \setminus \mathbb{A}$. *Furthermore,*

(i) x^{opt} *is the* unique *global minimiser of the primal QP* (2.1)*,*

(ii) $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ *is an optimal solution of the dual QP* (2.6)*, and*

(iii) the optimal objective function values of primal and dual QP are equal. ○

Proof: Virtually any textbook on optimisation shows that the KKT conditions are necessary and sufficient to characterise solutions of a convex QP, see e.g. [40]. The existence of a (unique) primal solution of strictly convex QPs is guaranteed by the before-mentioned Frank-Wolfe theorem. □

Note that neither the set $\mathbb{A}$ nor the dual solution y^{opt} are necessarily unique. However, if all rows of the matrix $G_{\mathbb{A}}$ are linearly independent for a fixed $\mathbb{A}$, y^{opt} is uniquely determined by Equations (2.8a) and (2.8b) [178]. If $\mathbb{A} = \mathbb{A}(x^{\text{opt}})$, the condition that $G_{\mathbb{A}}$ has full row rank is called *linear independence constraint qualification* (LICQ).

2.1.3 Parametric Quadratic Programming

Definition 2.1 of a quadratic program can be extended to QPs whose gradient and constraint vectors can depend affinely on a given parameter. We will see that this extension is perfectly suited to describe QPs given by a discrete-time linear optimal control problem formulation (1.9).

Definition 2.5 (parametric quadratic program): *The optimisation problem*

$$\text{QP}_{H,G}(w): \quad \min_{x\in\mathbb{R}^n} \quad \frac{1}{2}x'Hx + x'g(w) \tag{2.9a}$$

$$\text{s.t.} \quad Gx \geq b(w)\,, \tag{2.9b}$$

with fixed matrices $H \in \mathbb{R}^{n\times n}$, $G \in \mathbb{R}^{m\times n}$ and vectors

$$g(w) \overset{\text{def}}{=} f + Fw\,, \tag{2.10a}$$

$$b(w) \overset{\text{def}}{=} e + Ew\,, \tag{2.10b}$$

depending on a varying parameter $w \in \mathbb{R}^{n_x}$ is called a parametric quadratic program *(with $f \in \mathbb{R}^n$, $F \in \mathbb{R}^{n\times n_x}$, $e \in \mathbb{R}^m$, $E \in \mathbb{R}^{m\times n_x}$). For fixed matrices that are known from the context, we often just write* $\text{QP}(w)$ *for notational convenience.* ○

For an arbitrary but fixed $w = w_0$ we yield an ordinary quadratic program of the form (2.1) and therefore all definitions and results presented so far also carry over to parametric QPs. But since the gradient vector $g(w)$ and constraint vector $b(w)$ are both affine functions of the parameter w, the feasible set, its optimal solution,

the set of active and inactive constraints at a feasible point as well as its dual also depend on w. Therefore these quantities will be written as $\mathcal{F}(w)$, $x^{\mathrm{opt}}(w)$, $\mathbb{A}\big(x^{\mathrm{opt}}(w)\big)$, $\mathbb{I}\big(x^{\mathrm{opt}}(w)\big)$, and $\mathrm{QP}^{\mathrm{dual}}(w)$, respectively. However, for notational convenience, we will often drop this dependence if it is clear from the context.

As the feasible set of a parametric QP usually varies with its parameter, $\mathrm{QP}(w)$ might only be feasible for certain values of w. Thus, we introduce the following

Definition 2.6 (set of feasible parameters): *The set*

$$\mathcal{P} \stackrel{\mathrm{def}}{=} \big\{ w_0 \in \mathbb{R}^{n_x} \mid \mathcal{F}(w_0) \neq \emptyset \big\} \tag{2.11}$$

is called the set of feasible parameters *of a parametric quadratic program.* ○

In the following, we summarise a couple of properties of the set of feasible parameters that are crucial for the explicit MPC methods to be presented in 2.2 and the online active set strategy described in Chapter 3.

The first property was proven by [19]:

Theorem 2.3 (convexity and closedness of set of feasible parameters): *The set $\mathcal{P}$ of feasible parameters of a parametric quadratic program* $\mathrm{QP}(w)$ *as defined in Definition 2.6 is* convex *and* closed. ○

For further investigating the geometric structure of $\mathcal{P}$, we introduce the following definition [18]:

Definition 2.7 (critical region): *Let a strictly convex parametric quadratic program* $\mathrm{QP}(w)$ *with the set of feasible parameters $\mathcal{P}$ be given. Moreover, for each $w_0 \in \mathcal{P}$ let $x^{\mathrm{opt}}(w_0)$ denote its unique optimal (primal) solution and $\mathbb{A}\big(x^{\mathrm{opt}}(w_0)\big)$ the corresponding optimal active set. Then, for every index set $\mathbb{A} \subseteq \{1, \ldots, m\}$, the set*

$$\mathrm{CR}_{\mathbb{A}} \stackrel{\mathrm{def}}{=} \big\{ w_0 \in \mathcal{P} \mid \mathbb{A}\big(x^{\mathrm{opt}}(w_0)\big) = \mathbb{A} \big\} \tag{2.12}$$

○

is called a critical region *of $\mathcal{P}$.*

It is shown in [18, 170] that the set of feasible parameters $\mathcal{P}$ is not only convex and closed but can be subdivided into a collection of polyhedra (see, e.g., [77] for an elementary proof in case that the LICQ as introduced on page 2.1.2 holds):

Theorem 2.4 (partition of the set of feasible parameters): *For a strictly convex parametric quadratic program* $\mathrm{QP}(w)$ *the following hold:*

(i) All closures of critical regions $\mathrm{cl}\,(\mathrm{CR}_{\mathbb{A}_i})$ *are closed polyhedra with pairwise disjoint interiors.*

(ii) The set of feasible parameters $\mathcal{P}$ can be subdivided into a finite number of closures of critical regions:

$$\mathcal{P} \;=\; \bigcup_{i=1}^{2^m} \mathrm{cl}\,(\mathrm{CR}_{\mathbb{A}_i})\,, \quad \mathbb{A}_i \subseteq \{1,\ldots,m\}\,. \tag{2.13}$$

○

The proof of Theorem 2.4 also gives us some insight into the structure of the optimal solution $x^{\mathrm{opt}}(w)$ of the parametric quadratic program $\mathrm{QP}(w_0)$. We summarise this important result (see [18, 170]) in the following

Theorem 2.5 (piecewise affine optimal solution): *Let a strictly convex parametric quadratic program $\mathrm{QP}(w_0)$ and its set of feasible parameters $\mathcal{P}$ be given. Then the following is true:*

(i) Its optimal solution is a piecewise affine *and* continuous *function*

$$x^{\mathrm{opt}} : \mathcal{P} \;\longrightarrow\; \mathbb{R}^n\,,$$

(ii) its optimal objective function value is a piecewise quadratic *and* continuous *function*

$$\begin{aligned} \nu^{\mathrm{opt}} : \quad \mathcal{P} \;&\longrightarrow\; \mathbb{R} \\ w_0 \;&\longmapsto\; \frac{1}{2} x^{\mathrm{opt}}(w_0)' H x^{\mathrm{opt}}(w_0) + x^{\mathrm{opt}}(w_0)' g(w_0)\,. \end{aligned}$$

The notion "piecewise" means that there exists a finite partition of $\mathcal{P}$ into polyhedral critical regions such that the restrictions of x^{opt} and ν^{opt} to each critical region are affine or quadratic, respectively. ○

Continuity of the optimal solution function x^{opt} was already shown by Fiacco [84] in the context of sensitivity analysis in nonlinear programming; Zafiriou [253] proved that x^{opt} is piecewise affine in order to obtain stability results.

2.1.4 Remark on State-Elimination

Let us introduce the following short-cuts:

$$X \;\stackrel{\mathrm{def}}{=}\; \begin{pmatrix} x_{k_0} \\ x_{k_0+1} \\ \vdots \\ x_{k_0+n_{\mathrm{p}}} \end{pmatrix} \in \mathbb{R}^{(n_{\mathrm{p}}+1)\cdot n_{\mathrm{x}}}\,, \qquad U \;\stackrel{\mathrm{def}}{=}\; \begin{pmatrix} u_{k_0} \\ u_{k_0+1} \\ \vdots \\ u_{k_0+n_{\mathrm{p}}-1} \end{pmatrix} \in \mathbb{R}^{n_{\mathrm{p}}\cdot n_{\mathrm{u}}}\,. \tag{2.14}$$

Using these definitions, it is easy to see that a discrete-time linear optimal control problem (1.11) is a special parametric quadratic program QP(w) with optimisation variables[1] Its parameter vector w comprises at least the initial value of the differential states w_0 at time k_0. Additionally, it might contain the discrete-time values of the reference trajectories y_k^{ref} and u_k^{ref} or time-varying constraint limits b_k.

Using Equation (1.11c) all states are uniquely determined by their initial value x_{k_0} (which is given) and the control input sequence $u_{k_0}, \ldots, u_{k_0+n_\text{p}-1}$. Thus, though formally correct, it might look odd that the discrete-time state trajectory X is part of the optimisation variables.

However, it is often advisable to keep X in the formulation of the (parametric) QP as this leads to highly structured and sparse QP matrices[2] H and G: The contribution of the kth interval, $k_0 \leq k \leq k_0 + n_\text{p} - 1$, to the objective function value only depends on the values of the states x_k and controls u_k on this respective interval. Therefore, H is a block-diagonal matrix. Similarly, as x_{k+1} only depends on x_k and u_k, also the constraint matrix G has only a few dense blocks, while all other entries are zero. This structure can be exploited by dedicated sparse QP solvers leading to dramatically reduced solution times compared with standard dense implementations (see e.g. [196] for an interior-point implementation).

An alternative way to exploit this sparsity is to actually eliminate the discrete state trajectory from the QP formulation. After this step, also called *condensing* [38], the optimisation variables only comprise the control input sequence $x = U \in \mathbb{R}^{n_\text{p} \cdot n_\text{u}}$. Thus, the (parametric) QP to be solved has much fewer degrees of freedom but comes at the expense of dense matrices H and G. Note, however, that also these condensed matrices show a special rank structure that could be exploited for speeding-up computations.

The question whether the states should be eliminated from the QP formulation or not depends on:

- whether a sparse QP solver is available,
- whether the dense QP matrices change and need to be re-calculated during the runtime of the process,
- the length of the prediction horizon, i.e. n_p,

[1] Note that both the differential states as well as the primal optimisation variables are usually denoted by x. We stick to this convention here as the respective meaning should always be clear from the context.

[2] In fact, they turn out to be higher-order (quasi-)separable matrices [233] for which tailored representations and efficient structure-exploiting linear algebra operations have been developed (see, e.g., [61, 58]).

- the number of states and control inputs (in particular the ratio $\frac{n_x}{n_u}$).

Obviously condensing needs to be chosen if no sparse QP solver is available. As dense QP solver implementations are still more common, this is a question of practical relevance. However, condensing itself is computationally costly, so it becomes less attractive if it needs to be performed online. Furthermore, keeping the sparse formulation is more efficient for long prediction horizons as the computational load of the QP solution grows only linearly in the horizon length instead of quadratically to cubically when states are eliminated [196]. The prediction horizon should be regarded as being long if its length is much bigger than the ratio $\frac{n_x}{n_u}$.

2.2 Explicit Solution Methods

2.2.1 Main Concept

The third step of Algorithm 1.1 requires the solution of an optimal control problem at each sampling instant while the controlled process is running. Although this task reduces to solving a single convex QP in case of a linear MPC formulation, this might still be computationally prohibitive when sampling times become very short. Therefore, explicit MPC methods solve all possibly arising QP instances beforehand, i.e. they solve the parametric quadratic program $\mathrm{QP}(w_0)$, and simply look-up the respective stored solution when needed [18].

The parametric QP (also referred to as "*multi-parametric*" quadratic program to emphasise that w_0 is usually nonscalar) is solved as follows [18]: first, an arbitrary parameter $\hat{w}_0$ in the interior of a critical region is determined by solving an appropriate *linear programming problem* (LP). Then $\mathrm{QP}(\hat{w}_0)$ is solved providing a polyhedral representation $\{w \in \mathcal{P} \mid \hat{A}w \geq \hat{b}\}$ of the critical region $\mathrm{CR}_{\hat{\mathbb{A}}}$ with $\hat{w}_0 \in \mathrm{CR}_{\hat{\mathbb{A}}}$ as well as an affine representation $\hat{C}w + \hat{d}$ of the optimal solution over $\mathrm{CR}_{\hat{\mathbb{A}}}$. Afterwards, the complement $\mathcal{P} \setminus \mathrm{CR}_{\hat{\mathbb{A}}}$ can be easily divided into a partition of $\hat{m} \stackrel{\mathrm{def}}{=} \dim \hat{b}$ convex polyhedra $\mathcal{P}_1, \ldots, \mathcal{P}_{\hat{m}}$ by successive changes of the defining inequalities $\hat{A}_i w \leq \hat{b}_i$ into $\hat{A}_i w > \hat{b}_i$. These steps are recursively performed for $\mathcal{P}_1, \ldots, \mathcal{P}_{\hat{m}}$ until a full polytopic representation of the solution is obtained. Theorem 2.5 guarantees that only a finite number of critical regions and corresponding explicit affine representations of the solution have to be stored. Further refinements such as reduction of the number of QPs to be solved and linear dependency handling are described in [225, 219].

Solving the parametric QP and storing its piecewise affine solution within a look-up table is done offline. The only remaining computation to be performed during

the runtime of the process is to identify the critical region to which the current initial value w_0 belongs and to evaluate the corresponding affine solution function. This can be implemented straight-forwardly by simply checking all polyhedral representations, i.e. checking if $\hat{A}w_0 \geq \hat{b}$, until the correct critical region is found and then calculating the optimal solution via $\hat{C}w_0 + \hat{d}$.

The explicit method became popular for applications where the available online computing power is not sufficient to solve a QP within the required sampling period (and memory is cheap). For example, explicit MPC has been successfully applied to a permanent magnet synchronous motor with a sampling period of 25 microseconds [95]. A recent survey of explicit MPC methods is given in [5]; an open-source implementation of explicit methods can be found in the Multi-Parametric Toolbox (MPT) for MATLAB [152].

However, pre-calculating the optimal solution has a serious drawback: since the number of possible critical regions grows exponentially in the number of constraints (up to 2^m different optimal active sets), it is limited to low dimensional parameter spaces $\mathcal{P}$, i.e. to ODE models comprising only a few differential states. Otherwise both the offline computation and storage requirements as well as the online effort for finding the correct critical region soon become prohibitively large. A further relevant problem in practice is that online tuning becomes very difficult as the offline computation time blows up.

In order to speed-up online evaluation of the MPC law, the construction of a binary search tree has been proposed [226]. An efficient point location algorithm has been presented in [239].

2.2.2 Approximate Methods

Approximate explicit MPC methods have been developed to reduce the offline complexity at the expense of a suboptimal online performance or slight constraint violations. Early approaches aim at combining several "small" critical regions to a "bigger" one [17, 136, 226]. More recently, an iterative approximation of the explicit solution has been proposed [138] that permits gradual improvement of the quality of the approximation. Also ideas to directly approximate the nonconvex map between w_0 and the optimal solution has been developed [137, 30].

Approximate methods have extended the scope of explicit MPC to slightly higher dimensional problems. Moreover, it has been shown that they can drastically reduce the high computational load of pre-calculating the optimal solutions while still fulfilling application-dependent controller specifications. In order to further increase the scope of explicit techniques, also combinations of them with online QP solvers have been proposed (see the summary in Subsection 2.3.4).

Nevertheless, explicit MPC methods remain restricted to small-scale problems with short prediction horizons. Typically, they cannot treat problem formulations comprising more than 6–8 differential states or a control horizon much longer than 5 intervals [174]. For solving larger MPC problems, iterative solution methods need to be employed online.

2.3 Iterative Solution Methods

There exists a great variety of iterative methods for solving convex QPs as arising in linear MPC. Most of them are for general purpose, others are tailored to the MPC context. This section summarises the most important ones and also mentions corresponding software implementations.

2.3.1 Active-Set Methods

An important class of methods for solving convex QPs are active-set methods, which were originally developed as extensions of the simplex method for solving LPs [248, 54]. The fundamental idea of active-set methods is to fix a set of active constraints (often referred to as *working set*) and to solve the resulting equality constraint QP, which is computationally cheap. This set of active constraints is then repeatedly updated until the optimal one is found. One can distinguish at least two main variants of active-set methods: primal and dual ones, to which we restrict our presentation.

Once a feasible starting point has been found, *primal active-set methods* generate a sequence of primal feasible iterates until also dual feasibility and thus an optimal solution is obtained [178]. If no feasible starting point $x^{(0)}$ (and corresponding working set $\mathbb{A}^{(0)} \subseteq \mathbb{A}(x^{(0)})$) is provided by the user, a so-called *Phase I* is employed to generate one or to conclude that the QP is infeasible (see e.g. [87] for details). Starting from $x^{(0)}$, a step direction $\Delta x^{(0)}$ that keeps (primal) feasibility is computed based on the assumption that $\mathbb{A}^{(0)}$ is indeed an optimal active set. If $\Delta x^{(0)} \neq 0$, a step length is calculated that respects the inactive constraints. If no full step can be taken as it would violate an inactive constraint, this constraint is added to the current working set and the iteration continues. If $\Delta x^{(0)}$ is zero, it is checked whether all dual multipliers satisfy the KKT optimality conditions (2.8). If so, an optimal solution is found. Otherwise, a constraint corresponding to a multiplier violating the optimality conditions is dropped from the current working set. Also in this case, the iteration continues based on the new iterate and the updated working set.

For determining the current step direction, all active-set solvers need to solve a linear system; it is usually called the KKT system as it is implied by the KKT optimality condition. This KKT system can be dense or sparse depending on whether the Hessian and constraint matrix are dense or sparse. The way this system is solved is principally independent from the other parts of the algorithm, however certain combinations are usually more suited to yield an efficient implementation than others. For dense systems, it is common to either project the Hessian matrix to the range or the null space of the active constraints yielding *range space* [108, 97] or *null space methods* [98, 101, 88], respectively. For sparse systems, also iterative linear solvers might be applied. As factorising the KKT matrix has cubic complexity in the number of optimisation variables, most implementations make use of matrix updates that require only a quadratically growing number of floating-point operations.

A couple of implementations of primal active-set solvers exist, for example: `qpopt` [99] and `bqpd` [88] are FORTRAN implementations of a null-space method, where `bqpd` also provides sparse matrix algebra routines. Also MATLAB's `quadprog` function or `CPLEX` [53] can use a dense primal active-set solver.

Dual active-set methods generate a sequence of dual feasible iterates until also primal feasibility and thus an optimal solution is obtained. This is equivalent to solving the dual QP^{dual} with a primal active-set method [109]. This also implies that dual methods iterate on both the primal variables x and the dual variables y.

Obtaining a dual feasible starting point $\left(x^{(0)}, y^{(0)}\right)$ is trivial as the choice $\left(-H^{-1}g, 0\right)$ is always dual feasible. The fact that no Phase I is needed is a strong advantage of dual methods over primal ones, which comes at the price that only strictly convex QP can be handled (though [39] presented an extension to the general convex case). Analogous to primal methods, dual QP methods solve at each iteration k a linear system to obtain a step direction $\left(\Delta x^{(k)}, \Delta y^{(k)}\right)$, determine a step length and check whether a constraint needs to be added to or removed from the current working set $\mathbb{A}^{(k)}$. We refer to [109] for more details. As dual feasibility means primal optimality, a dual active-set method produces iterates that would be optimal for the given QP if the violated constraints were not present.

Dual active-set methods based on a dense range-space linear algebra are implemented within `QLD` [206] (in FORTRAN), `qpas` [242] (in C) and in the open-source C++ code `QuadProg++` [93]. Moreover, also `CPLEX` [53] can use a dual active-set solver. A primal-dual regularised method for sparse convex QPs is implemented in `QPBLUR` [166] (in FORTRAN and MATLAB). `QPSchur` is a C++ implementation of a dual active-set method for large-scale, strictly convex QPs based on a Schur complement approach [13].

Though none of these QP solvers has been specifically written for use in MPC, they nevertheless can often be used very successfully for this purpose. For example, the dual active-set solver `qpas` has been applied to MPC-based vibration suppression at sampling rates of 5 kHz [244, 243]. In contrast, [11] adapted a dual active-set solver for efficiently solving QPs within a branch and bound method for solving mixed-integer MPC problems. The online active set strategy as proposed in [75] has been developed to exploit the parametric nature of QPs arising in MPC for speeding-up the online QP solution. It has similarities to a dual active-set method and, for example, allows for warm-starting the QP solution without a Phase I. We will summarise the online active set strategy in Chapter 3, where also further theoretical extensions are presented. Its efficient open-source C++ implementation `qpOASES` is the topic of Chapter 4.

The QPs arising in the MPC context are specially structured and involve sparse QP matrices, provided that the states are kept within the formulation (see Subsection 2.1.4). Exploiting this structure within active-set QP solvers has been already proposed by [139], who used a sparse LR factorisation to solve the KKT system (2.8). Similar ideas have recently been proposed in [203]. For the case of input-constrained MPC problems, [47] uses Pontryagin's minimum principle [43] to solve the sparse KKT system via a recursion over state and co-state variables. An active-set QP solver for exploiting the block-structure arising from a multiple shooting discretisation for solving nonlinear MPC problems is described in [146].

The computational load of all these methods grows only linearly in the length of the prediction horizon n_p (instead of quadratically to cubically if states are eliminated without further exploiting the rank structure of the resulting dense QP matrices). However, they suffer from the drawback that each update of the current working set can become as expensive as a complete re-factorisation of the sparse KKT system. Therefore, QP sparsity arising from long prediction horizons is often more readily exploited by tailored interior-point methods.

2.3.2 Interior-Point Methods

Interior-point methods form a second important class of methods to solve convex QPs. Its modern form has been initially developed for linear programming [141] and was quickly extended to convex quadratic programming and general nonlinear programming. Interior point methods are mainly used in two different variants: *primal barrier methods* and *primal-dual methods*.

Primal barrier methods replace the inequality constraints $Gx \geq b$ of the QP by a weighted barrier function in the objective. This barrier function is constructed such that it becomes (infinitely) expensive to violate these constraints; most often

a logarithmic barrier function is used:

$$\frac{1}{2}x'Hx + x'g - \kappa^{(k)} \sum_{i=1}^{m} \log\left(G_i x - b_i\right) \tag{2.15}$$

The resulting equality constrained convex NLP is then solved by Newton's method [60]. In order to ensure convergence to the solution of the original QP, this procedure is repeated with a decreasing value of $\kappa^{(k)} \to 0$ for $k \to \infty$. By changing $\kappa^{(k)}$ moderately in each step k of the outer loop, it can be ensured that the inner Newton iteration always remains in the region of quadratic convergence and thus solves the inner problem within very few iterations [250]. In fact, a polynomial runtime guarantee can be given if $\kappa^{(k)}$ is updated in certain ways [177].

Primal-dual methods combine the inner and outer loop of primal barrier methods by reducing $\kappa^{(k)}$ at each iteration of Newton's method. Unlike primal barrier methods, both primal and dual variables are treated equally and updated within each Newton iteration. Primal-dual methods avoid the need of a feasible starting point and often work better in practice, though fewer convergence results exist. For a detailed description we refer to excellent textbooks [250, 40].

An early efficient implementation has been proposed by Mehrotra [172]. IPOPT [235, 236] is an open-source C/C++ implementation of a primal-dual interior-point method for general NLPs; another one for solving convex QPs is `OOQP` [94]. Also the C package `LOQO` [234] and the FORTRAN package `GALAHAD` [111] implement primal-dual methods. Moreover, many commercial implementations of IP methods exist.

Interior-point algorithms exhibit the great advantage of a relatively constant calculation time per QP solution. Therefore, they have been intensively used for MPC applications and a couple of modifications to speed-up the online QP solution have been proposed. We will summarise some of them.

For exploiting the above-mentioned sparsity of the QPs arising from MPC formulations, a discrete-time Riccati recursion has been proposed to solve the linear system in each Newton step [196, 140]. While the computational load of a standard implementation grows cubically in the length of the prediction horizon n_p, the cost of the Riccati recursion depends only linearly on n_p. Thus, this technique is particularly interesting for MPC problems formulated on long prediction horizons.

In contrast to active-set methods, interior-point methods suffer from the important drawback that it is difficult to warm-start them based on previous solution information. Recently, some progress has been achieved: [252] presents ideas to warm-start LP problems, [212] report computational savings of up to 70 % for

convex QPs. Similar ideas combined with the above-mentioned Riccati recursion have been presented in [238], where also computational results of a prototype implementation based on an infeasible start primal barrier method (see [40]) are given.

Also inexact interior-point methods, which are based on an inexact Newton method, have been proposed to trade of computational effort and solution accuracy. [211] proposes the use of an iterative method (the minimum residual method [185]) to solve the linear system and presents dedicated pre-conditioners to accelerate convergence. Numerical results indicate that their method outperforms the one presented in [196].

Yet another approach to reduce computation times is to automatically generate a customised interior-point method for each specific application, as recently proposed by [169]. We will further discuss this approach in Chapter 8.

Direct comparisons of interior-point with active-set methods indicate that it depends on the problem's characteristics which approach is more suited: "For large QPs with many active inequality constraints the interior-point approach is expected to require far fewer iterations than an active-set method to arrive at the solution. However, each of the interior-point iterations is many times more expensive than the iterations performed in an active-set method." [14].

2.3.3 Further Iterative Approaches

Recently, [189] proposed to formulate MPC problems as a system of piecewise affine equations. For doing so, the inequality equations within the KKT optimality conditions (2.8) are posed as piecewise affine equations by introducing a function that saturates whenever one of these inequalities becomes active. This allows one to solve the MPC problem by means of a regularised piecewise-smooth Newton method [158] using a line search globalisation based on a quadratic spline. At each iteration a linear system needs to be solved, whose dimension equals the cardinality of the current working set. As this number is often much smaller than the total number of optimisation variables, this is a key feature making the proposed method very fast in many cases. Moreover, exploiting the above-mentioned QP sparsity of MPC problems comprising a long prediction horizon, renders this approach very efficient for large-scale models and long control horizons [189].

For solving MPC problems efficiently, the use of a *fast gradient method* (see [176]) has been recently proposed [199, 200]. Classical gradient schemes do not rely on second order derivative information and take a damped steepest descent step in each iteration. Fast gradient methods modify this idea to yield faster convergence. For the MPC context this method offers two main advantages: First, they allow

us to derive reasonably tight a priori bounds on the number of iterations required to find an optimal solution, which is of great practical relevance. Second, they are particularly easy to implement as they only require to compute the gradient and to perform a projection operation in each step (and no Hessian matrix operations). However, computational efficiency of fast gradient methods depends crucially on the ability to efficiently project the QP gradient onto the feasible set $\mathcal{F}$. This projection is very cheap for box-constrained MPC problems, but treating general constraints makes this approach significantly more expensive.

2.3.4 Combinations of Explicit and Iterative Methods

In order to combine the fast QP solution of explicit methods with the generality of online QP solution, an approach called *partial enumeration* has been proposed in [187]. It is based on the following observation: although exponentially many critical regions exist, only a small fraction of them actually becomes relevant during the runtime of the process. Thus, instead of pre-calculating all critical regions, only those believed to be part of this relevant fraction are calculated and stored in a cache. If the critical region corresponding to the optimal solution of the current QP is found within the cache, its affine representation of the optimal solution is used to provide very fast feedback. Otherwise, while applying some suboptimal heuristic control action, the corresponding online QP is solved in the background using a standard QP solver. Once this QP is solved, its solution is used to obtain the corresponding critical region, which is then added to the cache. Recently, also robust stability of an MPC scheme based on partial enumeration has been shown [188].

Another approach to widen the applicability of explicit MPC by combining it with an iterative solver has been presented in [256]. It suggests to pre-calculate an approximate explicit control law that is used online to generate a feasible initial guess for warm-starting the online optimisation. The idea is presented for MPC formulations that lead to a parametric LP problem and a modified simplex algorithm is used as online LP solver. In order to meet hard real-time bounds on the computational load, the LP solver only performs a limited number of iterations which possibly leads to suboptimal control actions. This is justified by a procedure for bounding the suboptimality and for establishing stability guarantees of the suboptimal online control law. Moreover, the authors formulate another offline optimisation problem for approximately determining the optimal trade-off between the effort required to evaluate the explicit control law and the effort required to perform a certain number of LP iterations.

Chapter 3

Fast Linear MPC using the Online Active Set Strategy

This chapter starts with reviewing the main ideas of the online active set strategy as recently proposed by the author [75, 77]. It has been designed to solve strictly convex quadratic programs arising in linear MPC very efficiently. This review is followed by descriptions of novel theoretical extensions for initialising the online active set strategy (also in case the QP matrices change) and for solving convex QPs by means of a proximal point method. A proper implementation of the online active set strategy within the open-source software `qpOASES` and its application to industrial case studies constitute the main contribution of the first part of this thesis.

3.1 The Online Active Set Strategy

3.1.1 Main Idea

For solving (parametric) quadratic programs of the form (2.9) we propose to employ the online active set strategy. This strategy has been initially proposed in [75, 77] and we closely follow the presentation given therein. Its core idea, the introduction of a homotopy to traverse the critical regions while solving the QP, has been proposed independently by [21] in a different context.

The development of the online active set strategy for fast MPC applications has been motivated by the following observations:

- Linear MPC constitutes a parametric QP that has to be solved repeatedly for changing values of its parameter $w = w_0$. If QP solution is fast, thus allowing for (very) short sampling times, w_0 will usually change slowly leading to QP solutions that also vary moderately from one QP to the next. As this usually means that also the optimal active-set changes moderately, *warm-starting* the QP solver with the solution of the previous QP might speed-up QP solution drastically.
- Warm-starting still seems to be a big advantage of active-set methods (though a lot of effort has been made to warm-start interior-point methods [252, 212]). Moreover, active-set solvers are natural candidates to directly incorporate the theoretical knowledge on the geometric properties of parametric QP solutions as described in Subsection 2.1.3.
- As active-set QP solvers lack a practical runtime guarantee, there might occur situations where the MPC sampling time does not allow to solve the current QP exactly. In these situations it would be desirable to have an interpretation of the intermediate iterate. While standard primal or dual active-set solvers cannot provide a satisfactory justification for applying an intermediate iterate to the process (see Section 2.3), the introduction of a homotopy when traversing the critical regions does better justify the use of intermediate iterations for obtaining the feedback control.
- Using a homotopy eliminates the need for finding an initial feasible point like a Phase I for primal active-set QP solvers (cf. Subsection 2.3.1). Moreover, it allows us to re-use matrix factorisations not only within the solution of one QP but also across multiple QP solutions (we will refer to this as *hot-starting* the QP solver).

We will now describe the homotopy used within the online active set strategy for the transition from the solution of a quadratic program $\mathrm{QP}(w_0)$ to the next one $\mathrm{QP}(w_0^{\mathrm{new}})$. This homotopy can be interpreted as moving on a straight line in the parameter space $\mathcal{P}$ (see Definition 2.6). As this set is convex, see Theorem 2.3, we can be sure that all QPs on this line remain feasible and thus can be solved. As long as we stay in one critical region, the QP solution depends affinely on w_0. If we have to cross the boundaries of critical regions during our way along the line, Theorem 2.5 ensures that the solution can be continued continuously.

Using the notation from Definition 2.5, we introduce the homotopy parameter τ to re-parameterise w_0 as well as the gradient and constraint vector:

$$\tilde{w}_0 \colon [0,1] \to \mathbb{R}^{n_x}, \quad \tilde{w}_0(\tau) \stackrel{\mathrm{def}}{=} w_0 + \tau \Delta w_0 \,, \tag{3.1a}$$

$$\tilde{g} \colon [0,1] \to \mathbb{R}^{n}, \quad \tilde{g}(\tau) \stackrel{\mathrm{def}}{=} g(w_0) + \tau \Delta g \,, \tag{3.1b}$$

$$\tilde{b} \colon [0,1] \to \mathbb{R}^{m}, \quad \tilde{b}(\tau) \stackrel{\mathrm{def}}{=} b(w_0) + \tau \Delta b \,, \tag{3.1c}$$

where we use the following definitions:

$$\Delta w_0 \quad \stackrel{\text{def}}{=} \quad w_0^{\text{new}} - w_0\,, \tag{3.2a}$$

$$\Delta g \quad \stackrel{\text{def}}{=} \quad g(w_0^{\text{new}}) - g(w_0) \;=\; F\Delta w_0\,, \tag{3.2b}$$

$$\Delta b \quad \stackrel{\text{def}}{=} \quad b(w_0^{\text{new}}) - b(w_0) \;=\; E\Delta w_0\,. \tag{3.2c}$$

Let us assume that we have solved a parametric QP of the form (2.9) for a certain initial state w_0 and (after one sampling period) want to solve it again for a new initial state vector w_0^{new} with unknown solution $\left(x_{\text{new}}^{\text{opt}}, y_{\text{new}}^{\text{opt}}\right)$. We start from the known optimal solution x^{opt} and y^{opt} (and a corresponding index set $\mathbb{A} \subseteq \mathbb{A}(x^{\text{opt}})$) of the previous QP($w_0$) and aim at solving QP($w_0^{\text{new}}$). The core idea of the online active set strategy is to move from w_0 towards w_0^{new}, and thus from $(x^{\text{opt}}, y^{\text{opt}})$ towards $(x_{\text{new}}^{\text{opt}}, y_{\text{new}}^{\text{opt}})$, while keeping primal and dual feasibility (i.e. optimality) for all intermediate points. This means that we are introducing homotopies ($\mathbb{M} \stackrel{\text{def}}{=} \{1, \ldots, m\}$)

$$\tilde{x}^{\text{opt}}\colon [0,1] \to \mathbb{R}^n, \qquad \tilde{x}^{\text{opt}}(0) = x^{\text{opt}},\ \tilde{x}^{\text{opt}}(1) = x_{\text{new}}^{\text{opt}}, \tag{3.3a}$$

$$\tilde{y}^{\text{opt}}\colon [0,1] \to \mathbb{R}^m, \qquad \tilde{y}^{\text{opt}}(0) = y^{\text{opt}},\ \tilde{y}^{\text{opt}}(1) = y_{\text{new}}^{\text{opt}}, \tag{3.3b}$$

$$\tilde{\mathbb{A}}\colon [0,1] \to 2^{\mathbb{M}}, \qquad \tilde{\mathbb{A}}(0) = \mathbb{A},\ \tilde{\mathbb{A}}(\tau) \subseteq \mathbb{M}, \tag{3.3c}$$

$$\tilde{\mathbb{I}}\colon [0,1] \to 2^{\mathbb{M}}, \qquad \tilde{\mathbb{I}}(\tau) \stackrel{\text{def}}{=} \mathbb{M} \setminus \tilde{\mathbb{A}}(\tau), \tag{3.3d}$$

which satisfy the KKT optimality conditions (see Theorem 2.2) at every point $\tau \in [0,1]$:

$$\begin{pmatrix} H & G'_{\tilde{\mathbb{A}}(\tau)} \\ G_{\tilde{\mathbb{A}}(\tau)} & 0 \end{pmatrix} \begin{pmatrix} \tilde{x}^{\text{opt}}(\tau) \\ -\tilde{y}^{\text{opt}}_{\tilde{\mathbb{A}}(\tau)}(\tau) \end{pmatrix} \;=\; \begin{pmatrix} -\tilde{g}(\tau) \\ \tilde{b}_{\tilde{\mathbb{A}}(\tau)}(\tau) \end{pmatrix}, \tag{3.4a}$$

$$\tilde{y}^{\text{opt}}_{\tilde{\mathbb{I}}(\tau)}(\tau) \;=\; 0\,, \tag{3.4b}$$

$$G_{\tilde{\mathbb{I}}(\tau)}\tilde{x}^{\text{opt}}(\tau) \;\geq\; b_{\tilde{\mathbb{I}}(\tau)}(\tau)\,, \tag{3.4c}$$

$$\tilde{y}^{\text{opt}}_{\tilde{\mathbb{A}}(\tau)}(\tau) \;\geq\; 0\,. \tag{3.4d}$$

Since $\tilde{x}^{\mathrm{opt}}(\tau)$ and $\tilde{y}^{\mathrm{opt}}(\tau)$ are piecewise linear functions (and $\tilde{x}^{\mathrm{opt}}(\tau)$ even continuous according to Theorem 2.5), locally a relation of the form

$$\tilde{x}^{\mathrm{opt}}(\tau) \stackrel{\mathrm{def}}{=} x^{\mathrm{opt}} + \tau \Delta x^{\mathrm{opt}}, \tag{3.5a}$$

$$\tilde{y}_{\mathbb{A}}^{\mathrm{opt}}(\tau) \stackrel{\mathrm{def}}{=} y_{\mathbb{A}}^{\mathrm{opt}} + \tau \Delta y_{\mathbb{A}}^{\mathrm{opt}}, \tag{3.5b}$$

holds for sufficiently small $\tau \in [0, \tau_{\max}]$, $\tau_{\max} \geq 0$.

Because we start from an optimal solution, we know that conditions (3.4) are satisfied at $\tau = 0$. Therefore equality (3.4a) is satisfied for all $\tau \in [0, \tau_{\max}]$ if and only if

$$\begin{pmatrix} H & G_{\mathbb{A}}' \\ G_{\mathbb{A}} & 0 \end{pmatrix} \begin{pmatrix} \Delta x^{\mathrm{opt}} \\ -\Delta y_{\mathbb{A}}^{\mathrm{opt}} \end{pmatrix} = \begin{pmatrix} -\Delta g \\ \Delta b_{\mathbb{A}} \end{pmatrix} \tag{3.6}$$

holds. Because linear independence of the rows of $G_{\mathbb{A}}$ can easily be ensured [21], Equation (3.6) has a unique solution.

As long as one stays within a critical region, the QP solution depends affinely on w_0. However, the active-set changes whenever a previously inactive constraint becomes active, i.e.

$$G_i'(x^{\mathrm{opt}} + \tau \Delta x^{\mathrm{opt}}) = b_i(w_0) + \tau \Delta b_i$$

for some $i \in \mathbb{I}$, or a previously active constraint becomes inactive, i.e.

$$y_i^{\mathrm{opt}} + \tau \Delta y_i = 0$$

for some $i \in \mathbb{A}$. Thus, the maximum possible *homotopy step length* $\tau_{\max}$ is determined as follows[1]:

$$\tau_{\max}^{\mathrm{prim}} \stackrel{\mathrm{def}}{=} \min_{\substack{i \in \mathbb{I} \\ G_i' \Delta x^{\mathrm{opt}} < \Delta b_i}} \frac{b_i(w_0) - G_i' x^{\mathrm{opt}}}{G_i' \Delta x^{\mathrm{opt}} - \Delta b_i}, \tag{3.7a}$$

$$\tau_{\max}^{\mathrm{dual}} \stackrel{\mathrm{def}}{=} \min_{\substack{i \in \mathbb{A} \\ \Delta y_i < 0}} -\frac{(y^{\mathrm{opt}})_i}{\Delta y_i}, \tag{3.7b}$$

$$\tau_{\max} \stackrel{\mathrm{def}}{=} \min\left\{1, \tau_{\max}^{\mathrm{prim}}, \tau_{\max}^{\mathrm{dual}}\right\}. \tag{3.7c}$$

This choice of $\tau_{\max}$ ensures that conditions (3.4c), (3.4d) remain fulfilled. Moreover, by defining $\Delta y_{\mathbb{I}}^{\mathrm{opt}} \stackrel{\mathrm{def}}{=} 0$ also equality (3.4b) holds for all $\tau \in [0, \tau_{\max}]$.

[1] The minimum over an empty set is defined as ∞.

If $\tau_{\max}$ equals one, a full step along the homotopy path can be taken and the solution of the new quadratic program $\mathrm{QP}(w_0^{\mathrm{new}})$ is found. Otherwise, a primal or dual blocking constraint—which limits $\tau_{\max}$ to a value lower than one—indicates a constraint to be added to or removed from the index set $\mathbb{A}$. After updating $\mathbb{A}$ and the KKT matrix in Equation (3.6), a new step direction is calculated and the whole procedure repeats until the solution of $\mathrm{QP}(w_0^{\mathrm{new}})$ is found. This idea is summarised in Algorithm 3.1 and illustrated in Figure 3.1.

Algorithm 3.1 (online active set strategy)

`input:` data and solution $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ of $\mathrm{QP}(w_0)$ and associated index set of active constraints $\mathbb{A}$, new parameter $w_0^{\mathrm{new}} \in \mathcal{P}$

`output:` solution pair $(x_{\mathrm{new}}^{\mathrm{opt}}, y_{\mathrm{new}}^{\mathrm{opt}})$ of $\mathrm{QP}(w_0^{\mathrm{new}})$ and associated index set $\mathbb{A}^{\mathrm{new}}$

(1) Calculate Δw_0, Δg and Δb via Equations (3.2).

(2) Calculate *primal/dual step directions* Δx^{opt} and Δy^{opt} via Equation (3.6).

(3) Determine maximum *homotopy step length* $\tau_{\max}$ from Equations (3.7).

(4) Obtain *optimal solution* of $\mathrm{QP}(\tilde{w}_0)$:
Set $\tilde{w}_0 \leftarrow w_0 + \tau_{\max}\Delta w_0$, $\tilde{x}^{\mathrm{opt}} \leftarrow x^{\mathrm{opt}} + \tau_{\max}\Delta x^{\mathrm{opt}}$, $\tilde{y}^{\mathrm{opt}} \leftarrow y^{\mathrm{opt}} + \tau_{\max}\Delta y^{\mathrm{opt}}$.

(5) `switch` $\tau_{\max}$

- `case` $\tau_{\max} = 1$:
  Optimal solution of $\mathrm{QP}(w_0^{\mathrm{new}})$ found.
  Set $x_{\mathrm{new}}^{\mathrm{opt}} \leftarrow \tilde{x}^{\mathrm{opt}}$, $y_{\mathrm{new}}^{\mathrm{opt}} \leftarrow \tilde{y}^{\mathrm{opt}}$, $\mathbb{A}^{\mathrm{new}} \leftarrow \mathbb{A}$ and `stop!`
- `case` $\tau_{\max} = \tau_{\max}^{\mathrm{dual}}$:
  Remove a *dual blocking constraint*[2] $j \in \mathbb{A}$, with $\tau_{\max}^{\mathrm{dual}} = -\frac{y_j^{\mathrm{opt}}}{\Delta y_j}$, from index set $\mathbb{A} \leftarrow \mathbb{A} \setminus \{j\}$.
- `case` $\tau_{\max} = \tau_{\max}^{\mathrm{prim}}$:
  Add a *primal blocking constraint* $j \notin \mathbb{A}$, with $\tau_{\max}^{\mathrm{prim}} = \frac{b_j(w_0) - G_j' x^{\mathrm{opt}}}{G_j' \Delta x^{\mathrm{opt}} - \Delta b_j}$, to index set $\mathbb{A} \leftarrow \mathbb{A} \cup \{j\}$, while ensuring linear independence of the active constraint matrix $G_{\mathbb{A}}$ (see [21, 77]).

(6) Set $w_0 \leftarrow \tilde{w}_0$, $x^{\mathrm{opt}} \leftarrow \tilde{x}^{\mathrm{opt}}$, $y^{\mathrm{opt}} \leftarrow \tilde{y}^{\mathrm{opt}}$ and continue with step (1).

[2] If the Hessian matrix H is only positive semi-definite, one might ensure that the KKT matrix in Equation (3.6) remains invertible following the approach described in [21].

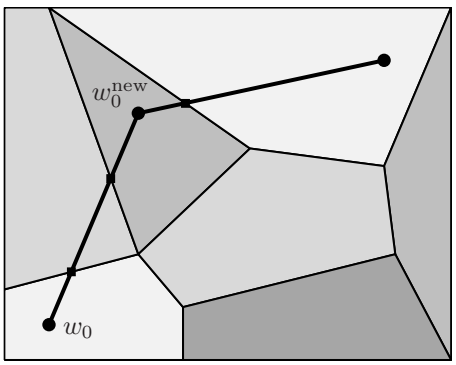


Figure 3.1: Homotopy paths from one QP to the next across multiple critical regions (taken from [75]).

3.1.2 Real-Time Variant

One advantage of the online active set strategy is that it produces a sequence of optimal solutions for QPs along the homotopy path. Thus, it is possible to interrupt this sequence after every iteration and to start a new homotopy from the current iterate towards the new QP. Thus, in a real-time setup, one can try to find the optimal solution of the current QP within a given sampling time. But if too many iterations are necessary to get from the solution of the previous QP to that of the current QP, one can simply stop the solution of the current QP and start a new homotopy towards the solution of the newest one. If solving the newest QP requires fewer iterations than computable within the given sampling time, the online active set strategy can make up for previously unperformed active-set changes.

This situation is illustrated in Figure 3.2, where we assume that only two active-set changes are allowed per QP solution. In [77, Chapter 5] a non-trivial example is given where the fully converged QP solution at a certain sampling instant requires 14 iterations and 6 iterations at the next. Limiting the number of active-set changes to 10 per QP solution yields a suboptimal solution at the first sample, while the optimal solution of the next QP can be found again within the given iteration limit.

We emphasise that also this real-time variant does not require re-factorisations of the KKT matrix in Equation (3.6). Instead matrix factorisations can be maintained based on matrix updates featuring a quadratic complexity in the number of QP variables can be used. As the computational effort per iteration

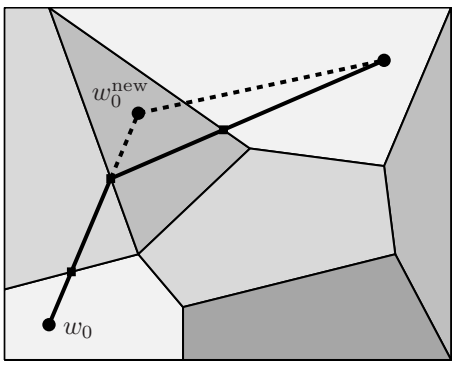


Figure 3.2: Homotopy paths (solid) from one QP to the next with limited number of active-set changes (taken from [75]).

can be determined exactly (see also Subsection 4.1.4), it is easy to estimate the maximum number of iterations that can be performed within a given sampling period. Before actually employing the online active set strategy in practice, realistic closed-loop simulations can give a hint whether—and if so: how often—this iteration limit is likely to be reached. In a reasonable setup, we expect that almost all QPs can be solved within the given number of iterations and that employing the real-time variant in the few other cases yields acceptable control inputs.

An intermediate iterate of the online active set strategy obtained by interrupting the homotopy prematurely might be infeasible with respect to the QP we actually want to solve[3], as also the constraint vector $b(w)$ might vary from one QP to the next. Though this is surely a drawback of the method, we note that this is also the case when a dual active-set method is employed. Moreover, even primal active-set methods (that produce a sequence of primal feasible iterates) might not guarantee that intermediate iterates are feasible: if they are initialised with an infeasible initial guess (e.g. the previous QP solution), already the employed Phase I to find a feasible starting point for the primal iterations might exceed the allowed computation time.

This discussion shows that any practical MPC scheme relying on an online QP solver, needs to ensure feasibility when intermediate results are passed to the process. However, for the online active set strategy we know that intermediate iterates solve a QP that is known to lie on the straight line between QP(w_0) and

[3]However, all intermediate iterates will be feasible if constraints do not change. This is true in the common situation when only input bounds are present, i.e. $G = \mathrm{I}_n$, that do not change over time.

$\mathrm{QP}(w_0^{\mathrm{new}})$ (while dual active-set methods deliver in each iteration the solution to a modified primal QP). Thus, we know that iterates monotonically get closer to the true solution in the parameter space $\mathcal{P}$. Recalling that the initial state w_0 is usually not known exactly but rather estimated and thus only known to lie within a certain confidence region (being a subset of $\mathcal{P}$), this property can be of practical relevance.

3.2 Initialisation of the Homotopy

The online active set strategy iterates through optimal solutions of intermediate QPs along the homotopy path. As the solution of each consecutive QP starts at the optimal solution of the previous one, this section addresses the question of how to initialise the homotopy of the very first QP to be solved. Besides the standard initialisation as presented in [77], we also present novel variants that significantly speed-up QP solution in case a sufficiently accurate guess for the optimal active-set is available. Moreover, we discuss how to obtain such guesses in the MPC context.

3.2.1 Initialisation Strategies

Standard Initialisation

Let us assume that we want to solve the quadratic program $\mathrm{QP}(w_0)$ for a given parameter w_0, being the first one of possibly a whole sequence of QPs. The main idea for initialising the online active set strategy is to setup an auxiliary parametric quadratic program $\mathrm{QP}^{\mathrm{aux}}(w)$ whose solution for $w = w_0^{\mathrm{aux}}$ is already known and whose solution for $w = w_0$ coincides with the optimal solution of $\mathrm{QP}(w_0)$. This allows one to perform a usual homotopy from the optimal solution of $\mathrm{QP}^{\mathrm{aux}}(w_0^{\mathrm{aux}})$ to that of $\mathrm{QP}(w_0)$.

It turns out that instead of constructing such an auxiliary parametric quadratic program $\mathrm{QP}^{\mathrm{aux}}(w)$ explicitly, it suffices to construct a single instance $\mathrm{QP}^{\mathrm{aux}}(w_0^{\mathrm{aux}})$, which is a standard QP. Let us assume that $\mathrm{QP}(w_0)$ does not comprise any (implicitly defined) equality constraints and that no prior information on its solution is available. In this situation it is natural to construct $\mathrm{QP}^{\mathrm{aux}}(w_0^{\mathrm{aux}})$ such that its optimal active-set is empty:

$$\mathrm{QP}^{\mathrm{aux}}(w_0^{\mathrm{aux}}): \quad \min_x \quad \tfrac{1}{2}x'Hx + x'0 \tag{3.8a}$$

$$\text{s.t.} \quad Gx \geq -\alpha \cdot 1\,, \tag{3.8b}$$

i.e. the gradient vector is set to zero and all components of the constraint vector to $-\alpha$ with $\alpha > 0$.

It can be easily verified that the choice $x^{\text{opt}} \stackrel{\text{def}}{=} 0$, $y^{\text{opt}} \stackrel{\text{def}}{=} 0$ and $\mathbb{A}^{\text{opt}} \stackrel{\text{def}}{=} \emptyset$ satisfies the KKT optimality conditions (2.8) of $\text{QP}^{\text{aux}}(w_0^{\text{aux}})$. Thus, we can now start from this optimal solution performing a homotopy to the optimal solution of the initial quadratic program $\text{QP}(w_0)$. If the unconstrained minimum is also optimal for this initial QP, its solution will be obtained after a single full step without any active-set change.

Starting with a Guess for the Optimal Active-Set

In case a good guess for the optimal active-set corresponding to the solution of $\text{QP}(w_0)$ is available, the standard initialisation described above may lead to unnecessarily many active-set changes before finding the optimal solution. In particular, this is the case if $\text{QP}(w_0)$ comprises (implicitly defined) equality constraints as these constraints are known to be active at the optimal solution. Assuming that a guess for the optimal active-set, denoted by $\widehat{\mathbb{A}}^{\text{opt}}$, is available, we therefore construct $\text{QP}^{\text{aux}}(w_0^{\text{aux}})$ in a different way:

$$\text{QP}^{\text{aux}}(w_0^{\text{aux}}): \quad \min_x \quad \tfrac{1}{2} x' H x + x' 0 \tag{3.9a}$$

$$\text{s.t.} \quad \begin{pmatrix} G_{\widehat{\mathbb{A}}^{\text{opt}}} \\ G_{\widehat{\mathbb{I}}^{\text{opt}}} \end{pmatrix} x \geq \begin{pmatrix} 0 \\ -\alpha \cdot 1 \end{pmatrix}, \tag{3.9b}$$

where $\alpha > 0$ and $\widehat{\mathbb{I}}^{\text{opt}} \stackrel{\text{def}}{=} \{1, \ldots, m\} \setminus \widehat{\mathbb{A}}^{\text{opt}}$ summarises the indices of constraints that are supposed to be inactive at the optimal solution.

This time we set the gradient vector to zero but the constraint vector is chosen such that constraints that are guessed to be active are indeed active and such that the remaining ones are inactive. It might happen that a constraint $i \in \widehat{\mathbb{I}}^{\text{opt}}$ is guessed to be inactive although it is linearly dependent from the other active ones (i.e. the ones with an index from $\widehat{\mathbb{A}}^{\text{opt}}$). If this is the case, also the i-th component of the constraint vector is set to 0 instead of $-\alpha$ rendering this constraint *weakly active*. Similarly, if $G_{\widehat{\mathbb{A}}^{\text{opt}}}$ does not have full row rank, only the indices of a linearly independent subset of these constraints are actually included in $\widehat{\mathbb{A}}^{\text{opt}}$.

Again, it can be easily verified that the choice $x^{\text{opt}} \stackrel{\text{def}}{=} 0$, $y^{\text{opt}} \stackrel{\text{def}}{=} 0$ and $\mathbb{A}^{\text{opt}} \stackrel{\text{def}}{=} \widehat{\mathbb{A}}^{\text{opt}}$ satisfies the KKT optimality conditions of (3.9). Thus, we can now start from this optimal solution performing a homotopy to the optimal solution of the initial quadratic program $\text{QP}(w_0)$. If the guess has been perfect, its solution will be obtained after a single full step without any active-set change.

Starting with a Guess for the Optimal Primal Solution

It is often more natural to obtain a guess for the optimal primal solution instead of a guess for the optimal active-set. Such a guess, say $\widehat{x}^{\text{opt}}$, can be easily transformed into one for the optimal active-set: We simply define

$$\widehat{\mathbb{A}}^{\text{opt}} \stackrel{\text{def}}{=} \left\{ i \in \{1, \ldots, n\} \mid G_i \widehat{x}^{\text{opt}} \leq b_i \right\}$$

to yield a consistent guess for the optimal active-set $\widehat{\mathbb{A}}^{\text{opt}}$. This means that we add constraints to $\widehat{\mathbb{A}}^{\text{opt}}$ that are active or violated at the guessed primal solution $\widehat{x}^{\text{opt}}$ (which is sometimes referred to as *clipping*). Using this definition we construct $\text{QP}^{\text{aux}}(w_0^{\text{aux}})$ as follows:

$$\text{QP}^{\text{aux}}(w_0^{\text{aux}}): \quad \min_x \quad \tfrac{1}{2} x' H x + x' g^{\text{aux}} \tag{3.10a}$$

$$\text{s.t.} \quad \begin{pmatrix} G_{\widehat{\mathbb{A}}^{\text{opt}}} \\ G_{\widehat{\mathbb{I}}^{\text{opt}}} \end{pmatrix} x \geq \begin{pmatrix} G_{\widehat{\mathbb{A}}^{\text{opt}}} \widehat{x}^{\text{opt}} \\ G_{\widehat{\mathbb{I}}^{\text{opt}}} \widehat{x}^{\text{opt}} - -\alpha \cdot 1 \end{pmatrix}, \tag{3.10b}$$

where $\alpha > 0$ and the gradient is defined to be $g^{\text{aux}} \stackrel{\text{def}}{=} -H\widehat{x}^{\text{opt}}$.

Looking at the KKT optimality conditions of $\text{QP}^{\text{aux}}(w_0^{\text{aux}})$, it can be easily verified that the choice $x^{\text{opt}} \stackrel{\text{def}}{=} \widehat{x}^{\text{opt}}$, $y^{\text{opt}} \stackrel{\text{def}}{=} 0$ and $\mathbb{A}^{\text{opt}} \stackrel{\text{def}}{=} \widehat{\mathbb{A}}^{\text{opt}}$ is an optimal solution. As before, we can thus start from this optimal solution performing a homotopy to the optimal solution of the initial quadratic program $\text{QP}(w_0)$. If the guess for the primal solution has been perfect, its solution will be obtained after a single full step without any active-set change.

3.2.2 Obtaining Good Initial Guesses

Having presented different strategies to start the homotopy, the question arises how to obtain good initial guesses to speed-up QP solution in the MPC context.

Part of the answer is that one often does not have to worry about this: as the online active set strategy can always start from the solution of the previous QP, the homotopy only needs to be initialised once at the solution of an auxiliary problem $\text{QP}^{\text{aux}}(w_0^{\text{aux}})$. So, if this occurs only once at the very start of the MPC controller, obtaining a sophisticated guess for the very first QP solution is usually not necessary from a computational point of view.

The initialisation strategies presented in Subsection 3.2.1 will be most interesting in the following situations:

- if not a whole sequence but only a single QP needs to be solved and computation time is crucial;
- if the QP sequence is re-initialised more frequently than only at the very beginning (e.g. due to accumulating numerical errors in the internal matrix factorisations as discussed in Subsection 4.3.1) and one is interested in speeding-up QP solution at the sampling instants where a re-initialisation is required;
- if one does not want the QP sequence to start each homotopy at the optimal solution of the previous QP but at a different initial guess instead.

The third situation typically occurs if the nature of the MPC problem to be solved asks to initialise QP solution with the *shifted* optimal solution of the previous QP. This is usually the case when the prediction horizon is long and the contribution of the Lagrange term to the objective function dominates that of the Mayer term (see also the discussion in [62]). Also the time-optimal MPC formulation in [230] makes use of an alternative initialisation of the homotopy.

3.3 Extension to Multiply Linearised MPC

Sometimes a nonlinear process cannot be modelled by a single linear ODE but using a fully nonlinear model is not desired. In this situation it can be appropriate to derive a collection of linear models, each of them valid for a certain operating range. Motivated by an MPC formulation comprising a collection of linear ODEs to describe a real-world Diesel engine test bench, we present an extension of the online active set strategy for dealing with such a setup [76].

Let us assume that we derived a number of linear ODEs that are locally valid within a certain operating range. A somewhat crude heuristic approach is to formulate a linear MPC problem (see Definition 1.3) for each of these models and to simply switch between these locally valid linear MPC formulations whenever the process enters a different operating range. Though this raises serious concerns about the stability of such a scheme, it allows us to use a standard linear MPC algorithm promising simpler implementation and faster execution than a fully nonlinear optimisation algorithm.

For extending the online active set strategy to this specific setup, we first note that we yield a standard parametric quadratic program $\mathrm{QP}(w_0) = \mathrm{QP}_{H,G}(w_0)$ as long as only one of the local linear MPC formulations is used. Thus, our extension focusses on the case where a switch to a different local formulation occurs, which corresponds to a different parametric quadratic program $\mathrm{QP}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0)$ with new Hessian and constraint matrix.

Let us assume that we have solved $\mathrm{QP}_{H,G}(w_0)$ with optimal primal-dual solution pair $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ and corresponding optimal active-set $\mathbb{A}^{\mathrm{opt}}$. The main observation is that this solution is also optimal for the following auxiliary QP comprising the new Hessian and constraint matrix:

$$\mathrm{QP}^{\mathrm{aux}}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{aux}}): \qquad \min_{x\in\mathbb{R}^n} \quad \tfrac{1}{2}x'H^{\mathrm{new}}x + x'g^{\mathrm{aux}} \tag{3.11a}$$

$$\text{s.t.} \qquad G^{\mathrm{new}}x \;\geq\; b^{\mathrm{aux}}\,, \tag{3.11b}$$

with

$$g^{\mathrm{aux}} \;=\; g - (H^{\mathrm{new}} - H)\,x^{\mathrm{opt}} + (G^{\mathrm{new}}_{\mathbb{A}^{\mathrm{opt}}} - G_{\mathbb{A}^{\mathrm{opt}}})'y^{\mathrm{opt}}_{\mathbb{A}^{\mathrm{opt}}}\,, \tag{3.12a}$$

$$b^{\mathrm{aux}} \;=\; b + (G^{\mathrm{new}}_{\mathbb{A}^{\mathrm{opt}}} - G_{\mathbb{A}^{\mathrm{opt}}})\,x^{\mathrm{opt}}\,. \tag{3.12b}$$

It is easy to show that the pair $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ actually is an optimal solution of $\mathrm{QP}^{\mathrm{aux}}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{aux}})$ by subtracting the KKT optimality conditions of $\mathrm{QP}_{H,G}(w_0)$ from the ones of $\mathrm{QP}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0)$.

Thus, it is possible to start a homotopy from the optimal solution $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ of $\mathrm{QP}^{\mathrm{aux}}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{aux}})$ towards the optimal solution of $\mathrm{QP}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{new}})$ as described in Algorithm 3.2.

Algorithm 3.2 (warm-start after QP matrix change)

`input:` optimal solution $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ to quadratic program $\mathrm{QP}_{H,G}(w_0)$ with corresponding index set $\mathbb{A}^{\mathrm{opt}}$, new parameter w_0^{new} and new QP matrices H^{new}, G^{new}

`output:` solution pair $(x^{\mathrm{opt}}_{\mathrm{new}}, y^{\mathrm{opt}}_{\mathrm{new}})$ of $\mathrm{QP}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{new}})$ and corresponding index set $\mathbb{A}^{\mathrm{opt}}_{\mathrm{new}}$

(1) Calculate matrix factorisations[4] of new Hessian matrix H^{new} and new constraint matrix G^{new} for the previously optimal index set $\mathbb{A}^{\mathrm{opt}}$.

(2) Calculate auxiliary gradient vector g^{aux} and auxiliary constraint vector b^{aux} via Equations (3.12).

(3) Perform a usual homotopy for $\mathrm{QP}_{H^{\mathrm{new}},G^{\mathrm{new}}}(w_0^{\mathrm{new}})$ as described in Algorithm 3.1, starting at $(x^{\mathrm{opt}}, y^{\mathrm{opt}})$ and $\mathbb{A}^{\mathrm{opt}}$.

[4] If $G^{\mathrm{new}}_{\mathbb{A}^{\mathrm{opt}}}$ has full row rank, otherwise any sub-matrix with same rank and linearly independent rows can be used instead.

This extension makes it possible to also warm-start the QP solution when switching between different local linear MPC formulations. The way this is done is similar to the initialisation strategies described in Subsection 3.2.1 but comes at the expense of additional computational effort, namely a re-factorisation of the new QP matrices with cubic complexity in the number of QP variables n. This effort is similar to that of a standard active-set QP solver, but if transition between the local MPC formulations is smooth, warm-starting usually leads to a reduced number of iterations.

An implementation of this extension of the online active set strategy has been successfully used to control a Diesel engine test bench at University of Linz, Austria. Results of real-world experiments using a collection of local linear MPC formulations have been presented in [76]; a simulation study based on linear parameter varying models is described in [184].

The proposed extension is not only applicable for switching between different local linear MPC formulations, but also for solving nonlinear MPC problems. In this case a nonlinear programming problem (NLP) has to be solved instead of a QP, as described in Chapter 6. This can be done efficiently using sequential quadratic programming (SQP) methods that require to solve a whole sequence of QPs with varying matrices and vectors at each sampling instant. When the iterates produced by these SQP methods converge towards the solution of the NLP problem, also the optimal active-set of the underlying QPs converges towards the optimal one [201]. Therefore, warm-starting these QPs with the solution of the previous one is expected to yield considerable computational savings even in the presence of changing QP matrices.

3.4 Regularisation Procedure for Convex QPs

The online active set strategy as outlined in Section 3.1 is designed to solve strictly convex QPs, i.e. it requires the QP Hessian matrix to be positive-definite. For solving convex QPs, we propose an iterative regularisation procedure that makes use of the homotopy framework and is easy to implement.

We can distinguish two different approaches to tackle convex QPs: either the Hessian matrix is regularised in a certain way such that it becomes positive-definite, or a direct handling of the semi-definiteness using suitable linear algebra operations is combined with additional algorithmic safeguards (see, e.g., [103, 39]). Regularisation schemes are easy to implement, whereas a direct handling often causes significant modifications of the linear algebra routines that also might increase the computational load of the algorithm. On the other hand, dealing with semi-definite Hessian matrices directly promises to yield very accurate solutions,

while each regularisation changes the QP problem and therefore will usually only produce less accurate approximations to a true solution. Based on the homotopy framework of the online active set strategy, we propose an iterative regularisation procedure that tries to combine the advantages of both approaches: an easy to implement, efficient modification that can produce very accurate solutions.

The core idea of all regularisation schemes is to add a symmetric, positive-definite matrix $H^{\text{reg}} \in \mathcal{S}^n_{\succ 0}$ to the QP Hessian matrix $H \in \mathcal{S}^n_{\succeq 0}$. A common choice is

$$H^{\text{reg}} \;=\; H_\varepsilon \;\stackrel{\text{def}}{=}\; \varepsilon \cdot \mathrm{I}_n \quad \text{with } \varepsilon > 0\,. \tag{3.13}$$

In case H has a special structure causing the semi-definiteness, more sophisticated choices than H_ε might be used. Sticking to $H^{\text{reg}} = H_\varepsilon$, this results in the following regularised (parametric) quadratic program:

$$\min_{x \in \mathbb{R}^n} \quad \tfrac{1}{2} x' H x + x' g(w) + \tfrac{1}{2} x' H_\varepsilon x \tag{3.14a}$$

$$\text{s.t.} \qquad G x \geq b(w)\,, \tag{3.14b}$$

where we use the same notation as in Definition 2.5. However, whether a solution of this regularised QP also is a solution for the original QP depends on the choice of ε (see for example [92]). At least we are sure that every solution of the regularised QP is feasible for the original one as both comprise identical constraints.

Choosing an appropriate value for ε is not straight-forward and problem dependent:

- Choosing ε too large changes the original QP formulation a lot and thus often leads to solutions that differ significantly from any optimal solution of the original QP.
- Choosing ε too small, say in the order of the machine precision, might cause a failure of a QP solver designed for strictly convex QPs as the Hessian matrix becomes too ill-conditioned (note that the condition number of $H + H_\varepsilon$ is proportional to $\frac{1}{\varepsilon}$).

In order to reduce the dependency of the solution quality on the choice of ε, we propose to "re-centre" the regularisation term $\frac{1}{2} x' H_\varepsilon x$ around a point $\bar{x}$ to yield the following regularised (parametric) quadratic program:

$$\min_{x \in \mathbb{R}^n} \quad \tfrac{1}{2} x' H x + x' g(w) + \tfrac{1}{2} (x - \bar{x})' H_\varepsilon (x - \bar{x}) \tag{3.15a}$$

$$\text{s.t.} \qquad G x \geq b(w)\,. \tag{3.15b}$$

The rationale behind this is the observation that if we could choose $\bar{x} = x^{\text{opt}}$ to be the optimal solution of the original QP, the regularised QP would yield an exact solution to the original QP independently of the choice of ε.

As we do not know x^{opt} in advance, we propose the following iterative regularisation procedure:

Algorithm 3.3 (iterative regularisation procedure)

`input:` (parametric) quadratic program QP(w_0), regularisation parameter $\varepsilon > 0$, maximum number of steps $k_{\max}$
`output:` approximative solution x^* to QP(w_0)

(1) Set $k = 0$, $\bar{x} = 0$.

(2) Use Algorithm 3.1 to solve regularised QP (3.15) and obtain its solution x^*.

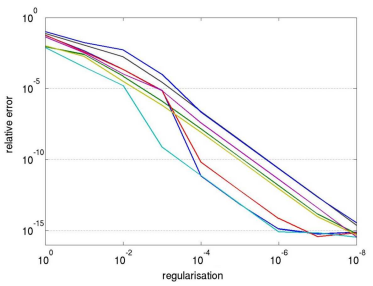
(3) Stop if $k = k_{\max}$; otherwise set $\bar{x} = x^*$, increase k by one and continue with step (2).

Algorithm 3.3 is a special case of a proximal point algorithm as introduced by [202]. Thus, the following convergence result holds: If all QPs (3.15) are solved exactly, Algorithm 3.3 converges at a linear rate (and even within a finite number of steps) towards a solution of the original QP [202].

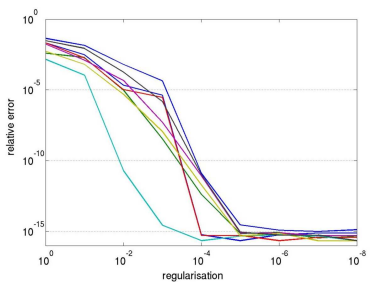
The iterative regularisation procedure as summarised in Algorithm 3.3 solves a sequence of QPs with changing values of the centring point $\bar{x}$. But as the regularisation term $\frac{1}{2}(x - \bar{x})' H_\varepsilon (x - \bar{x})$ only changes the gradient of the original quadratic program QP(w_0) (plus a constant offset), this sequence of regularised QPs can be solved within the usual homotopy framework of the online active set strategy. This means that we can hot-start each regularised QP solution at the optimal solution of the previous one which significantly speeds-up the overall procedure. Numerical tests indicate that, once the first regularised QP has been solved, only a small number of additional active-set changes are required to find the solutions of successive regularised QPs for $k \geq 1$. Thus, Algorithm 3.3 typically provides considerably more accurate solutions than a single standard regularisation at very limited extra computational load.

In order to illustrate the iterative regularisation procedure, its implementation within the software package `qpOASES` (see Chapter 4) has been applied to a collection of eight non-trivial convex QPs. These QPs comprise about 200 optimisation variables and up to 3000 constraints in order to handle infeasible

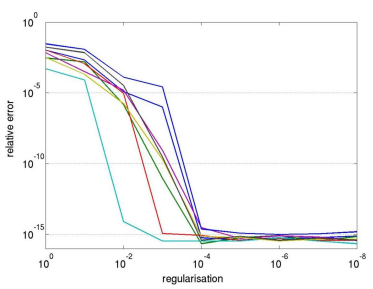
MPC problems as discussed in Section 5.2. Figures 3.3 shows the relative error in the optimal objective function value for different values of ε at different iterations of Algorithm 3.3. Using a standard regularisation of form (3.14) required very small values of ε to achieve acceptable results (see Subfigure 3.3(a)). In contrast, already one or two iterations of the iterative regularisation procedure recovered the optimal objective function value up to machine precision for moderate values of ε.



(a) 0th iteration



(b) 1st iteration



(c) 2nd iteration

Figure 3.3: Relative error in the optimal objective function value for a collection of non-trivial convex QPs when using the iterative regularisation procedure for different values of ε.

Chapter 4

The Open-Source Implementation qpOASES

One main contribution of this thesis is the software package `qpOASES` [79, 78], an open-source implementation of the online active set strategy as presented in Chapter 3. This chapter gives an overview of `qpOASES`, describing its main features and outlining its software design. We explain how the implementation is tailored to different QP types for solving them efficiently. Also an overview of the available interfaces to third-party software packages and successful applications of `qpOASES` to QPs arising in various control problems is given. Moreover, we summarise ideas recently presented in [193] to increase the reliability of the online active set strategy for ill-conditioned and degenerate QPs. These numerical modifications and additional functionality to handle sparse QP matrices have also been implemented into `qpOASES`, rendering it an efficient and reliable active-set solver for general convex QPs [83]. Finally, Section 4.5 illustrates that `qpOASES` can significantly outperform other popular academic and commercial QP solvers on small- to medium-scale test examples.

4.1 Overview of the Software Package

4.1.1 Algorithmic Description

`qpOASES` implements the online active set strategy for solving convex QPs. As described in Chapter 3, this strategy has been designed for solving QPs arising in MPC very efficiently. `qpOASES` solves instances of parametric QPs of the following

form:

$$\min_{x \in \mathbb{R}^n} \quad \tfrac{1}{2} x' H x + x' g(w_0) \tag{4.1a}$$

$$\text{s.t.} \quad \underline{b_{\mathrm{B}}}(w_0) \leq \; x \; \leq \overline{b_{\mathrm{B}}}(w_0)\,, \tag{4.1b}$$

$$\underline{b_{\mathrm{C}}}(w_0) \leq Gx \leq \overline{b_{\mathrm{C}}}(w_0)\,, \tag{4.1c}$$

with a positive semi-definite Hessian matrix $H \in \mathcal{S}^n_{\succeq 0}$, a gradient vector $g(w_0) \in \mathbb{R}^n$ and a constraint matrix $G \in \mathbb{R}^{m \times n}$. In order to fully exploit the problem structure, the constraint formulation was kept more general than the one in Definition 2.5. Thus, qpOASES distinguishes between *box constraints* or *bounds* on the variables given by the vectors $\underline{b_{\mathrm{B}}}(w_0)$, $\overline{b_{\mathrm{B}}}(w_0) \in \mathbb{R}^n$ and general constraints involving the constraint matrix G and the vectors $\underline{b_{\mathrm{C}}}(w_0)$, $\overline{b_{\mathrm{C}}}(w_0) \in \mathbb{R}^m$. It is worth to stress that qpOASES can also solve non-parametric QPs as its implementation uses the dependency of the QP data on w_0 only implicitly.

The distinction between bounds and constraints can lead to substantial computational savings (see Section 4.2) and is very natural in the MPC context: bounds on the control inputs also translate into bounds within the QP formulation, bounds and constraints on the outputs or the differential states lead to general constraints within the QP formulation.

qpOASES distinguishes two different ways to solve a QP of the form (4.1): First, it can be solved by performing a cold-start, i.e. without any prior solution information. This is the usual situation if just a single QP is to be solved or if the QP is the first one of a whole sequence of parameterised QPs (as typically arising in MPC). Second, provided that a QP with same dimensions has been already solved before, the current QP can be solved by performing a hot-start based on the optimal solution and the internal matrix factorisations of the previously solved QP.

In both cases, each QP iteration of the online active set strategy requires one to solve the KKT system (3.6). The way this is done does only affect computational speed and accuracy but does *not* affect the iterates produced along the homotopy path. qpOASES solves the KKT system by means of a null space approach (see Section 2.3.1), though other choices would have been possible. We briefly outline its main concept.

First, the active constraints matrix is decomposed by a modified *QR decomposition* as proposed in [101]:

$$G_{\mathbb{A}} Q \;=\; G_{\mathbb{A}} \begin{bmatrix} Z & Y \end{bmatrix} \;=\; \begin{bmatrix} 0 & T \end{bmatrix}, \tag{4.2}$$

where $Z \in \mathbb{R}^{n\times(n-|\mathbb{A}|)}$ forms an orthonormal basis of the null space of the active constraints and $T \in \mathbb{R}^{|\mathbb{A}|\times|\mathbb{A}|}$ is a reverse lower triangular matrix. Second, the Hessian matrix is projected to this null space and the resulting *projected Hessian matrix* is then factorised by means of a *Cholesky decomposition*:

$$Z'HZ \;=\; R'R\,, \tag{4.3}$$

where $R \in \mathbb{R}^{(n-|\mathbb{A}|)\times(n-|\mathbb{A}|)}$ is an upper triangular matrix. Once these two matrix factorisations have been set up, they allow for an efficient solution of the KKT system (3.6) in $\mathcal{O}(n^2)$ floating-point operations. In order to see this, we transform the coordinates of the KKT system by the orthonormal matrix

$$\begin{pmatrix} Q & 0 \\ 0 & \mathrm{I}_{|\mathbb{A}|} \end{pmatrix}. \tag{4.4}$$

This yields the following transformed KKT system:

$$\begin{pmatrix} R'R & Z'HY & 0 \\ Y'HZ & Y'HY & T' \\ 0 & T & 0 \end{pmatrix} \begin{pmatrix} \Delta x_Z^{\mathrm{opt}} \\ \Delta x_Y^{\mathrm{opt}} \\ -\Delta y_{\mathbb{A}}^{\mathrm{opt}} \end{pmatrix} \;=\; \begin{pmatrix} -\Delta g_Z \\ -\Delta g_Y \\ \Delta b_{\mathbb{A}} \end{pmatrix}, \tag{4.5}$$

where the subscripts Y and Z indicate a projection onto the range and null space of the active constraints, respectively. This system of equations can now be solved using forward and backward substitutions with the triangular matrices T and R:

$$\Delta x_Y^{\mathrm{opt}} \;=\; T^{-1}\Delta b_{\mathbb{A}}\,, \tag{4.6a}$$

$$\Delta x_Z^{\mathrm{opt}} \;=\; -R^{-1}(R')^{-1}\left(\Delta g_Z + Z'HY\Delta x_Y^{\mathrm{opt}}\right), \tag{4.6b}$$

$$\Delta y_{\mathbb{A}} \;=\; (T')^{-1}\left(\Delta g_Y + Y'HZ\Delta x_Z^{\mathrm{opt}} + Y'HY\Delta x_Y^{\mathrm{opt}}\right). \tag{4.6c}$$

Note that calculating these solutions can be further simplified by exploiting common subexpressions and other structural information. `qpOASES` actually solves an adapted variant of KKT system (3.6) to reflect the distinction between bounds and constraints (see [77] for more details).

The null space approach is particularly numerically stable and does not rely on a positive definite Hessian matrix. This facilitated the extension to convex QPs as presented in Section 4.3 and is also numerically advantageous for QPs comprising positive definite Hessian matrices with very small positive eigenvalues. However, this comes at the expense that iterations are more expensive (than ones of the range space approach) whenever the null space has high dimension, i.e. whenever only a few constraints are active. On the other hand, computational savings due to the distinction between bounds and constraints are "most readily achieved in null space methods" [87].

While moving along the homotopy path, the set of active constraints is modified by adding or removing a constraint in each iteration. However, it would be too expensive to re-compute the TQ factorisation and Cholesky decomposition in each iteration as they require $\mathcal{O}(n^3)$ floating-point operations. Instead, in order to compute the next step direction efficiently, these matrix decompositions are maintained after each change in the set of active constraints by means of *Givens plane rotations* [105, 110]. They reduce the effort to $\mathcal{O}(n^2)$ floating-point operations per iteration. `qpOASES` implements updating routines specially tailored to the context where bounds and constraints are distinguished as proposed in [101].

4.1.2 Features

`qpOASES` has been designed to be most efficient when applied to small- to medium scale[1], dense QPs as arising in MPC after eliminating the states from the QP formulation. It exhibits the following algorithmic and numerical features:

- `qpOASES` implements all features and extensions of the online active set strategy as described in Chapter 3. Thus, it exploits the fact that linear MPC leads to parametric QPs and allows hot-starting from previous QP solutions or from any other initial guess without a Phase I. Moreover, it implements the described real-time variant and the extensions to QPs where also the matrices are changing.
- `qpOASES` is able to solve QPs with arbitrary constraints and positive semi-definite Hessian matrices. In particular, this allows one to solve general MPC problems comprising state or output constraints and semi-definite weighting matrices Q, R and P.
- `qpOASES` has been tailored to fully exploit the structure of different QP types as described in Section 4.2. For example, the distinction between bounds and constraints can lead to substantial computational savings, in particular if only bounds (i.e. bounds on the control inputs in the MPC context) are present. Moreover, sparsity in constraint and Hessian matrix can be exploited within matrix-vector operations, though the underlying matrix factorisations are dense.
- The numerical modifications summarised in Section 4.3 provide provisions to even solve ill-conditioned and degenerated QPs very reliably and exactly. For example, `qpOASES` is able to solve all 70 test problems from the Maros-Mészáros test set [167] comprising less than 1000 variables and not more than 1001 two-sided constraints (not counting bounds) up to high accuracy (see

[1] Say, not more than several hundred optimisation variables and not more than one to two thousand constraints.

Subsection 4.5). On the other hand, if the QP formulation is known to be well-conditioned, any of these algorithmic modifications can be switched-off to avoid computational overhead.

In addition, the following properties facilitate the use of qpOASES:

- It is open-source software released under the GNU Lesser General Public License (LGPL) [134] and comes along with a detailed user's manual and a fully documented source code.
- It is implemented in an object-oriented manner as self-contained C++ code (see Subsection 4.1.3).
- It offers various interfaces to third-party software and has been tested on embedded hardware (see Sections 4.4 and 5.1).

4.1.3 Software Design

qpOASES is written as object-oriented C++ code as this offers advantages for both developers and users. From a developer's perspective, this allowed us to introduce classes corresponding to different QP types. By inheriting common functionality, code duplication is avoided and readability of the code is improved by introducing a clear programme structure. Moreover, all internal data is hidden from the user. This leads to a clean interface where the user provides the QP data along with a set of algorithmic options and, after solving the problem, obtains the desired solution information. Finally, encapsulating all data within a given QP object allows the user to instantiate more than one QP object at once, which can be useful in certain situations.

Another important design goal has been self-containedness of the code and compatibility with different, also embedded, hardware platforms (also see the discussion in Subsection 5.1.2). For this reason, qpOASES can be run stand-alone without linking external libraries, though coupling the BLAS or LAPACK libraries [31, 6] is supported. In fact, all matrix operations within qpOASES are performed by customised C implementations of the required BLAS/LAPACK routines. Therefore it does not come at a surprise that directly linking the BLAS/LAPACK libraries hardly affects computational performance. As the use of C++ can create a barrier to run the code on embedded hardware, advanced programming techniques such as templates or virtual inheritance have been avoided (or made optional) to increase the number of compatible compilers.

We will now outline the class structure of qpOASES as illustrated in Figure 4.1. The class QProblemB provides all functionality necessary for solving simply bounded

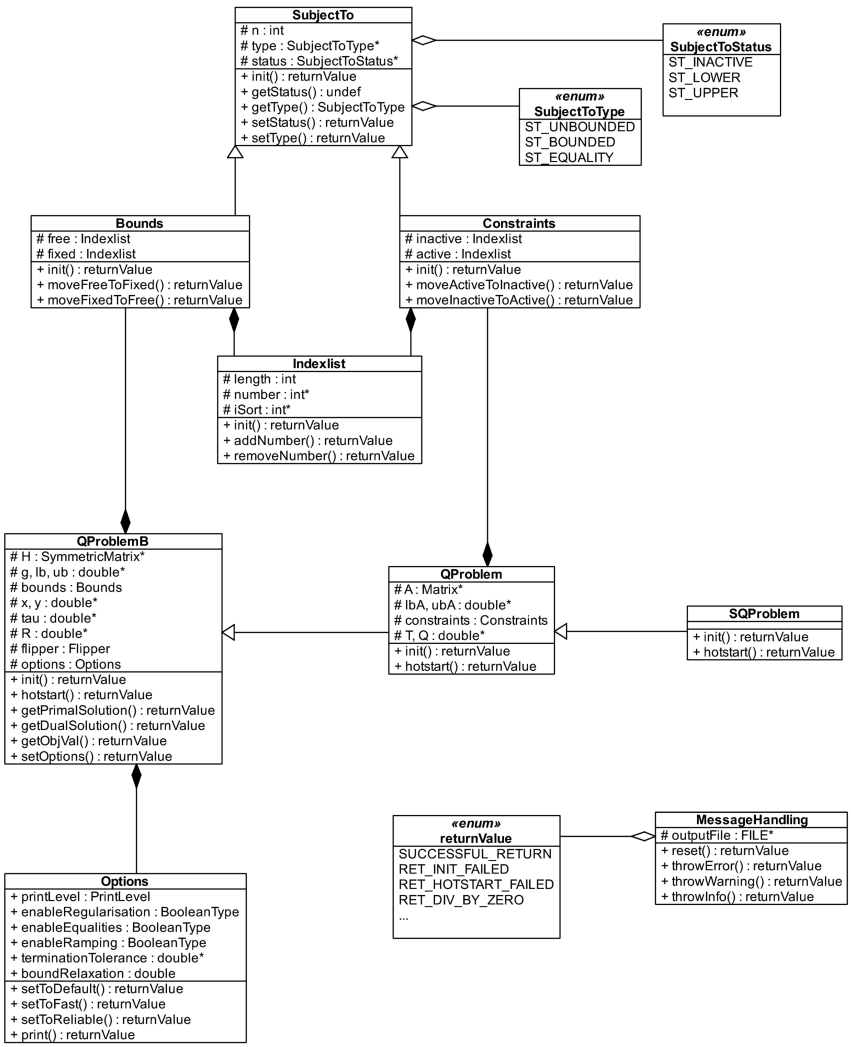


Figure 4.1: UML class diagram illustrating the main functionality of `qpOASES`. For clarity, only the most important members are shown for each class or enumeration.

QPs as described in Subsection 4.2.1. The class `QProblem` is derived from it and implements all additional functionality required for solving QPs comprising general constraints. Finally, the class `SQProblem` inherits all features of the `QProblem` class and provides further functionality for handling QPs with varying matrices as explained in Subsection 4.2.6. Hessian and constraint matrices are stored within

a minimal `Matrix` class implementation in order to use tailored linear algebra routines as explained in Subsection 4.2.5.

The user needs to instantiate an object of one of these three classes in order to specify his QP (sequence) to be solved. Before solving a QP, the user can pass an `Options` object specifying various algorithmic options [79] (otherwise default values are used). In order to assist the user with specifying these options, the class `Options` provides pre-defined settings optimised for solution speed or reliability, respectively.

All these three classes `QProblemB`, `QProblem` and `SQProblem` make use of further auxiliary classes: they possess a member of type `Bounds` or `Constraints`, respectively, in order to store information on the bounds or constraints of a QP. In particular, the classes `Bounds` and `Constraints` manage lists (of type `Indexlist`) of free and fixed variables or active and inactive constraints, respectively. Both classes are derived from a common base class `SubjectTo`.

Finally, all the above-mentioned classes use the `MessageHandling` class for providing error messages, warnings or other information to the user in a unified framework. This class makes use of the enumeration `returnValue`, which gathers all possible return values of all `qpOASES` functions.

4.1.4 Computational Complexity

Practical upper bounds on the number of iterations for finding an optimal solution are not available for active-set QP methods. In fact, even exponentially many iterations might become necessary as illustrated for active-set LP solvers in the famous example constructed in [149]. However, we stress that this worst-case behaviour has not been reported in practice. It is even possible to prove upper-bounds on the average number of LP iterations that only grow quadratically in the number of variables and constraints (see [213] and the references therein). This backs up the observation that the number of iterations required to solve convex QPs usually grows very moderately in the number of QP variables and constraints.

When analysing the runtime complexity of `qpOASES`, we restrict the presentation to the complexity of one single QP iteration as described in Algorithm 3.1: Obviously, the computational effort for performing steps (1), (4), (5) and (7) is $\mathcal{O}(n+m)$, i.e. it grows linearly in the number of variables n and the number of constraints m. As `qpOASES` makes use of matrix updates for solving the KKT system (3.6) as described in Subsection 4.1.1, both step (2) and (6) require $\mathcal{O}(n^2)$ floating-point operations (instead of $\mathcal{O}(n^3)$ and $\mathcal{O}(1)$, respectively, if the KKT matrix would be re-factorised at each iteration). Finally, computational load of step (3) is dominated

by the $\mathcal{O}(nm)$ floating-point operations required to form the matrix-vector product in Equation (3.7).

Thus, we can conclude that the overall runtime complexity of one iteration of `qpOASES` is $\mathcal{O}(n^2 + nm)$. The leading coefficient of the n^2 term depends on the number of active bounds and constraints and typically varies between 2 and 13 [77]. This also implies that the step length determination in step (3) might become the major share of the computational load of each iteration if the QP formulation comprises many more constraints than optimisation variables (see Subsections 4.2.4 and 4.2.5 for possible remedies).

The exact storage complexity of `qpOASES` depends on different aspects, in particular on whether the QP formulation comprises only box constraints or also general ones and on whether QP matrices are given in dense or sparse matrix format. For solving a dense QP formulation comprising general constraints, $4n^2 + nm + \mathcal{O}(n + m)$ floating-point numbers need to be stored; they represent the dense Hessian and constraint matrix as well as matrices comprising the dense Cholesky decomposition and the dense TQ factorisation, respectively.

4.2 Solution Variants for QPs with Special Properties

This section explains how `qpOASES` has been tailored to special QP types in order to speed-up computation by exploiting their respective properties.

4.2.1 Box Constraints

An important sub-class of general QPs of form (2.9) are those that only comprise bounds (or box constraints):

$$\min_{x \in \mathbb{R}^n} \quad \tfrac{1}{2} x' H x + x' g(w_0) \tag{4.7a}$$

$$\text{s.t.} \quad \underline{b_{\mathrm{B}}}(w_0) \leq \; x \; \leq \overline{b_{\mathrm{B}}}(w_0)\,. \tag{4.7b}$$

These QPs can arise from various applications, in particular from MPC formulations that only comprise bounds on the control inputs but no output or state constraints.

This special form can be exploited within the QP algorithm for speeding up the computation: First, as G becomes the identity matrix in Equation (4.2), the TQ factorisation becomes trivial. Thus, initially computing and updating can be done

at virtually no cost. Second, also projecting the Hessian matrix to the null space of active constraints becomes very cheap, as it suffices to only access the Hessian at indices corresponding to inactive bounds. Both also simplify the determination of the next step direction via Equation (3.6). These computational savings typically speed-up each single QP iteration by about a factor of three (compared to an implementation where bounds would be treated as general constraints). Moreover, as QP (4.7) only comprises n constraints, the number of active-set changes is typically lower than for QPs also comprising general constraints.

In order to fully benefit from these simplifications, `qpOASES` implements the special class `QProblemB`. It also requires less storage as no internal data members are allocated for storing general constraints and the TQ factorisation.

4.2.2 Trivial Hessian Matrix

The Hessian matrix is considered trivial if and only if it is the zero or identity matrix. If this is the case, several simplifications of the internal linear algebra operations are possible that cut down computational load by typically a factor of about two. Obviously, these simplifications affect all calculations involving the original or the projected Hessian matrix.

`qpOASES` automatically checks whether the Hessian matrix is trivial whenever it is passed to one of the QP objects. It is also possible to directly provide this information within the constructor call. This saves the small overhead for determining the Hessian type and also avoids allocating internal memory for storing the Hessian matrix.

If the Hessian matrix is zero, the (parametric) QP (2.9) is actually a (parametric) linear programming problem:

$$\mathrm{LP}(w_0): \quad \min_{x\in\mathbb{R}^n} \quad x'g(w_0) \tag{4.8a}$$

$$\text{s.t.} \quad Gx \geq b(w_0)\,. \tag{4.8b}$$

This kind of problems can be tackled using either of the strategies described in Section 4.2.3. However, as `qpOASES` is not a dedicated (parametric) LP solver, using it for solving LPs can be inefficient. Nevertheless, the ability to also solve LPs can be a useful feature in situations where computational time is not the main concern.

QPs whose Hessian is the identity matrix naturally arise from (unweighted) ℓ_2 norm minimisation problems. Moreover, every strictly convex QP can be

transformed into an equivalent one whose Hessian is the identity matrix by means of a suitable coordinate transformation. However, such a coordinate transformation only makes sense from a computational point of view if the QP comprises general constraints as it usually transforms box constraints into general ones.

4.2.3 Positive Semi-Definite Hessian Matrix

qpOASES provides two different strategies to deal with QPs that are convex but not strictly convex. The first one is the regularisation procedure as described in Section 3.4 and summarised in Algorithm 3.3. It is computationally cheap and works well if the regularised QP is numerically well-posed. A second strategy introduces so-called flipping bounds and zero-curvature tests and will be described in more detail in Section 4.3. This second one is usually computationally more expensive but tackles the semi-definite problem directly.

4.2.4 Many Constraints

As mentioned in Subsection 4.1.4, the computational load of each QP iteration can be dominated by the evaluation of the matrix-vector product Gx (with x being the current primal QP iterate) if the QP comprises many more constraints than optimisation variables. For such situations, qpOASES implements a technique that can speed-up computation by avoiding evaluation of the full matrix-vector product. We summarise its main idea and refer to [77] for mathematical details: Instead of computing $G_i' x$ for determining whether the inactive constraint $i \in \mathbb{I}$ might become active at the current iteration, an estimate for the distance of this product from its limits $\underline{b_{\mathrm{C}_i}}(w_0)/\overline{b_{\mathrm{C}i}}(w_0)$ is kept. Based on the norm of the current primal step Δx, one can reliably decide whether constraint i can become active or not. If so, the product $G_i' x$ needs to be computed to ensure a correct behaviour of the algorithm. Otherwise, this computation can be skipped and the distance estimate is updated at constant complexity.

In order to work efficiently, this technique requires normalisation of all rows of matrix G in a pre-processing step. Normalisation needs to be done with respect to a norm that is dual to the one used to calculate $\|\Delta x\|$. The current version of qpOASES relies on an ℓ_1 norm normalisation of the constraint rows and employs the computationally cheap maximum norm for the step direction.

The described technique only makes sense if evaluation of $G_i' x$ has $\mathcal{O}(n)$ complexity. If G is sparse or structured in a way that allows for a faster evaluation, the functionality described in Subsection 4.2.5 should be used instead.

4.2.5 Sparse QP Matrices

qpOASES has been developed for small- to medium scale QPs resulting from MPC formulations after the differential states have been eliminated. These QPs usually feature a fully dense Hessian matrix and a lower triangular constraint matrix. Consequently, the internal matrix factorisations have been implemented as dense linear algebra routines.

In order to exploit possible additional sparsity in the constraint matrix, the class ConstraintProduct has been introduced. It allows the user to provide a routine that calculates the matrix-vector product Gx more efficiently than a standard multiplication. This can save a considerable amount of computation per iteration in case the QP formulation comprises many constraints (cf. Subsection 4.1.4). A typical source of sparsity in the constraint matrix are bounds on the input rate. They couple exactly two optimisation variables—and thus cannot be expressed as box constraints—leading to exactly two non-zero entries in each corresponding column of G. Another reason for a partly sparse constraint matrix can be the introduction of slack variables (e.g. to handle QP infeasibilities as discussed in Section 5.2).

For enhancing qpOASES's applicability to general QPs, the idea to exploit sparsity of the constraint matrix has been recently extended in order to support general sparse QP matrices. For doing so, a minimal Matrix base class has been introduced[2] that encapsulates all matrix operations. This framework allows to easily switch between special linear algebra routines for dense and sparse QP matrices, respectively, and to exploit possible symmetry of them (see Figure 4.2). It would also facilitate to include further specialisations of the linear algebra operations, e.g. for exploiting the rank structure as mentioned in Subsection 2.1.4. Sparse matrices are stored in column compressed storage format [71].

4.2.6 Varying QP Matrices

As explained in Section 3.3, the online active set strategy can also be extended to parametric quadratic programs $\mathrm{QP}_{H,G}(w_0)$ with varying matrices. Instead of performing a cold-start for a QP with new matrices, this idea allows one to warm-start at the previous QP solution based on the previous optimal active-set and can help to reduce the computational effort. qpOASES implements this extension within the class SQProblem as summarised in Algorithm 3.2.

[2]This extension has been initiated by Andreas Potschka, who also realised most of the implementation.

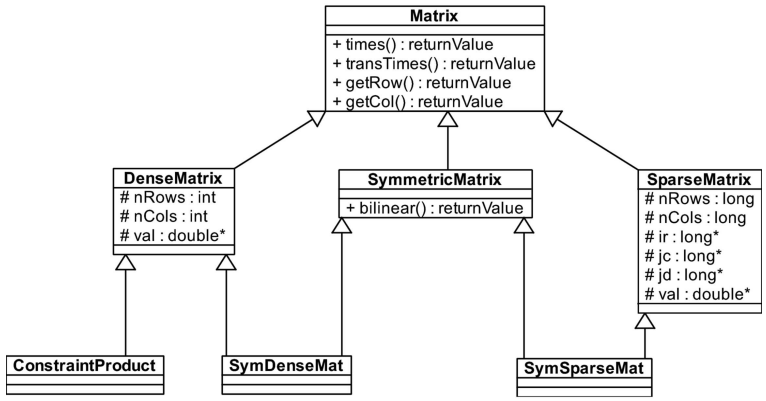


Figure 4.2: UML class diagram illustrating the matrix class hierarchy of qpOASES in order to use tailored linear algebra routines. For clarity, only the most important members are shown for each class.

4.3 Numerical Modifications to Increase Reliability

Though qpOASES is based on numerically stable matrix factorisations, namely a modified QR factorisation and a Cholesky decomposition, the online active set strategy can encounter numerical difficulties when solving certain QPs. Two main reasons can be distinguished:

- First, the QP can be ill-conditioned leading to projected Hessian matrices with large condition numbers. This can either be due to Hessian matrices with greatly varying eigenvalues or due to constraints that are very close to being linearly dependent[3].
- Second, the minimum determining the homotopy step length in Equations (3.7) can be attained for more than one constraint at once. In this situation, usually called a *tie* [237], it is not clear which of these constraints shall be added or removed. This can lead to repeated addition and deletion within a fixed subset of constraints causing *cycling* of the algorithm. The phenomenon of cycling is common to all active-set QP methods and "most QP implementations simply ignore the possibility of cycling" [178]. However, for solving QPs reliably, this issue needs to be addressed (for example by following one of the approaches described in [102, 88]).

[3]Exact linear dependence of the constraints is usually less of a problem as the linear independence test in step (5) of Algorithm 3.1 can prevent a linearly dependent constraint from being added to the current working set.

In the following subsections we briefly summarise ideas proposed in [193] to tackle these issues for increasing the reliability of the QP solution. These ideas have also been implemented into `qpOASES` and can optionally be used if necessary or desired [83].

4.3.1 Dealing with Rounding Errors and Ill-Conditioning

In order to reduce the influence of ill-conditioned projected Hessian matrices on the computation of the step direction, *iterative refinement* can be employed when solving the linear system (3.6). Iterative refinement (as originally proposed in [241]) recovers in each iteration a fixed number of extra valid digits of the solution of the linear system, provided that the refinement step is performed with higher precision. However, numerical accuracy is typically also improved if residuals are computed only at working precision (so-called *fixed precision iterative refinement*) [217, 125]. Thus, iterative refinement can be helpful whenever the condition number of the KKT matrix in (3.6) is only a few orders of magnitude smaller than the inverse of the machine precision. Its additional computational cost amounts in each iteration to one matrix-vector multiplication and one backward substitution with the matrix decomposition of the KKT matrix, which is moderate compared to the computational cost of a full QP iteration. `qpOASES` allows one to specify the desired non-negative number of fixed precision iterative refinement steps per QP iteration. Monitoring the norm of the iterative refinement residuals in consecutive iterations could also help to detect increasing ill-conditioning of the matrix factorisations due to the use of matrix updates.

A second idea aims at preventing the accumulation of rounding errors in the QP solution over multiple iterations, which can become large for ill-conditioned problems. Repeatedly updating the constraint vector $b(w_0)$ and the dual solution vector $y(w_0)$ in each QP iteration might render them inconsistent. For example, a component $b_i(w_0)$ corresponding to an active constraint i might not exactly match the value $G_i' x(w_0)$ or the corresponding dual multiplier $y_i(w_0)$ might have a small negative entry (instead of a non-negative one). The so-called *drift correction* changes the values of $b(w_0)$ and $y(w_0)$ straight-forwardly to consistent values and afterwards modifies the QP gradient as follows:

$$g(w_0) \stackrel{\text{def}}{=} Hx(w_0) - G'y(w_0)\,. \tag{4.9}$$

That way the current QP is slightly perturbed such that the current solution is an exact one, i.e. the drift correction actually introduces tiny kinks into the homotopy path. However, this allows the algorithm to annihilate all rounding errors in the QP solution before performing the next QP iteration. The drift correction increases computational load mainly by the two extra matrix-vector multiplications in Equation (4.9), which might become quite expensive compared

to the overall QP iteration in certain situations. Therefore, and because it is often superfluous to perform a drift correction in each QP iteration even for ill-conditioned QPs, qpOASES allows the user to specify a frequency at which drift corrections are to be performed (if at all).

A third idea to avoid ill-conditioning of the Cholesky factors R due to tiny eigenvalues of the Hessian matrix is the introduction of *flipping bounds*. The main idea is to check upon removal of an active constraint (or bound) whether the new diagonal elements of R drops below a certain threshold. If so, the constraint remains active but the intermediate QP data is changed such that it is active at its opposite limit (e.g. an active upper bound will become an active lower bound). qpOASES provides options to enable the use of flipping bounds and to adjust this threshold. As all matrix decompositions remain valid in case one needs to flip, the computational overhead is low.

The flipping bound strategy requires that the projected Hessian matrix is numerically positive definite at the beginning. This can be easily achieved by starting the homotopy such that all bounds are fixed in the beginning (as described in Section 3.2) leading to a projected Hessian matrix of dimension zero. Moreover, it requires that all bounds and constraints do have lower and upper limits. To ensure this, Potschka et al. [193] suggest to introduce so-called *far bounds*. Artificial limits are introduced for all bounds and constraints that do not have such limits within the QP formulation. By successively increasing the absolute value of these limits, it can be ensured that no such artificial limit will stay active at the solution (otherwise it can be concluded that the QP is unbounded).

4.3.2 Dealing with Ties

A rigorous approach for resolving ties has been presented in [237] where an auxiliary QP is solved at each tie. However, as this would be an inadequate computational overhead for the online context, qpOASES by default uses a much simpler heuristic that works very well in practice: if a tie occurs, constraint removal is preferred over constraint addition and if this is not sufficient to break the tie, the constraint with smallest index is removed.

An alternative heuristic, called *ramping*, has been proposed in [193] and is also implemented within qpOASES. It basically aims at avoiding ties by adding mutually different offsets to the constraint and dual solution vectors. Afterwards, the gradient vector is modified in a very similar way as described in Equation (4.9) to restore a consistent QP solution.

One should note that ties cannot be avoided in certain cases, e.g. if the QP solution lies at a point where the LICQ does not hold. In such cases the ramping strategy

aims at postponing the occurrence of ties to the very last step along the homotopy, where there is no need to resolve them anymore as the QP solution is already found. However, this reasoning is questionable in the online context when a whole sequence of QPs is to be solved.

4.4 Interfaces and Applications

qpOASES comes along with a couple of interfaces to third-party software that also facilitate to run the code on embedded hardware. This section gives an overview of these interfaces and outlines a couple of real-world applications.

4.4.1 Interfaces for Matlab, Octave, Scilab and YALMIP

qpOASES can be directly compiled and used within MATLAB, allowing a user to run the solver without touching its C++ source code. For example, a single QP can be solved by calling

```
[x,fval,exitflag,iter,lambda] = qpOASES( H,g,A,lb,ub,lbA,ubA,x0,options )
```

Besides the usual data specifying a QP of the form (4.1), an initial guess for the primal solution and a set of options can be passed. If no initial guess is given, the usual homotopy starting at the origin is performed (see Subsection 3.2.1). Options can be generated using the `qpOASES_options` command in order to retrieve the full functionality of the C++ version. The output arguments contain the optimal primal solution vector as well as optionally the optimal objective function value, a status flag, the number of iterations actually performed, and the optimal dual solution vector, respectively.

The MATLAB interface automatically detects whether the QP only comprises box constraints and internally instantiates the corresponding QP object. Moreover, it is possible to pass QP matrices in sparse format. This standard interface always performs a cold-start taking into account the guess for the primal solution (if specified), but also variants exist for solving whole sequences of QPs using all features of the online active set strategy.

In order to support open-source alternatives to MATLAB, qpOASES provides similar interfaces to OCTAVE [72] and SCILAB. Moreover, qpOASES has been interfaced to YALMIP [162], a modelling language for solving convex and nonconvex optimisation problems.

4.4.2 Running qpOASES on dSPACE and xPC Target

qpOASES also provides an interface to SIMULINK that allows the user to compile the code within a C MEX S-function. Different variants interfacing the QProblem, QProblemB and SQProblem class to the SIMULINK workspace are available (see Figure 4.3). The S-function block expects all QP data to be given in signal form. It outputs the first piece of the primal solution (corresponding to optimised control inputs on the first interval of the prediction horizon), the optimal objective function value, a status flag and the number of actually performed QP iterations.

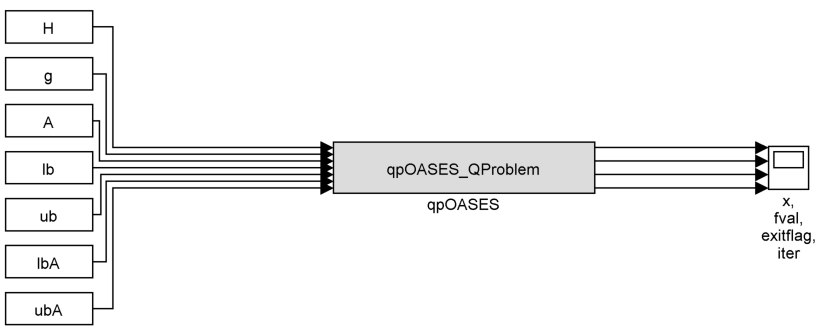


Figure 4.3: Illustration of the SIMULINK interface of qpOASES.

The SIMULINK interface also allows one to conveniently compile qpOASES onto DSPACE or XPC target hardware by means of the SIMULINK Real-Time Workshop [135]. The main requirement is the availability of a C++ compiler for the respective hardware. Compilation of qpOASES has been tested for DSPACE boards version 5.3 or higher together with the dSPACE C++ Integration Kit version 1.0.2 or higher. Also successful use on XPC target hardware has been reported (see below).

4.4.3 Real-World Applications

During the last few years, qpOASES has been used by many researchers for a wide range of applications. We only mention a number of academic real-world applications:

- MPC of a Diesel engine test bench at University of Linz, Austria, on DSPACE hardware at sampling times of 50 milliseconds [76, 184];
- MPC of beam tip vibrations at Slovak University of Technology, Bratislava, Slovakia, on an XPC target at sampling times of 10 milliseconds [223, 222];

- trajectory planning for a boom crane at University of Stuttgart, Germany, similar to the one described in [9], on dSPACE hardware at sampling times in the order of 100 milliseconds;
- time-optimal control of machine tools at K.U. Leuven, on dSPACE and xPC hardware at sampling times in the order of 4–10 milliseconds [231, 232, 230];
- solving QPs for controlling a tendon-driven robot platform at ETH Zurich, Switzerland, on a Standard PC [175, 197];
- simulations on optimisation-based clipping of audio signals at K.U. Leuven, on a Standard PC [57].

Moreover, two industrial applications of `qpOASES` are presented in detail in Chapter 5.

4.5 Numerical Performance

This section concludes the description of `qpOASES` by commenting on its numerical performance, namely its reliability and computational efficiency. The presented examples show that `qpOASES` can significantly outperform existing QP solvers, but we do *not* state that this is true for arbitrary problems.

4.5.1 Reliability

In order to investigate `qpOASES`'s reliability, we use it to solve convex QP problems from the Maros-Mészáros test set [167]. As `qpOASES` has been designed for solving small- to medium-scale QPs, we restrict our analysis to the 70 convex test problems comprising less than 1000 variables and not more than 1001 two-sided constraints (not counting bounds). We determine the fraction of successfully solved problems[4] and compare this number to the ones obtained by employing a couple of popular academic and commercial QP solvers to the same test problems, namely MATLAB's `quadprog`, `OOQP` [94] and `CPLEX` [53] (using its primal and dual active-set as well as its interior-point solver). Each solver is run via its MATLAB interface using its algorithmic default settings[5].

Table 4.1 summarises the obtained results[6], which allow to draw the following two conclusions: First, though all QP problems of the Maros-Mészáros test set are

[4] A QP problem is considered successfully solved if the provided optimal solution does not violate the KKT optimality conditions (see Theorem 2.2) by more than 10^{-2}.

[5] Better results might be obtained by adjusting these settings.

[6] These tests have been set up and performed by Andreas Potschka and the complete results are to be published in [83].

convex, it contains plenty of challenging problems that are hard to solve even for well-established QP solvers. These problems are either numerically ill-conditioned, highly degenerated or nearly infeasible. Second, only qpOASES is able to solve all 70 small- to medium-scale QPs of this test set, which illustrates the mature state of its implementation. Moreover, we note that QP solutions obtained by qpOASES are usually very accurate (with a maximum violation of the KKT optimality conditions of less than 10^{-5}).

Solver name	Fraction of problems successfully solved
quadprog	62 %
OOQP	70 %
CPLEX-IP	73 %
CPLEX-Primal	96 %
CPLEX-Dual	97 %
qpOASES	100 %

Table 4.1: Fraction of small- to medium-scale problems from the Maros-Mészáros test set successfully solved by the respective solver with default settings.

4.5.2 Computational Efficiency

In order to judge on the computational speed of qpOASES, the above test setup is not particularly suited for several reasons. First, qpOASES has been designed for use in model predictive control but the stand-alone QP instances of the test set do not allow it to benefit from its hot-starting features. Second, the QP problems of the test set are not very representative for those arising in MPC with respect to both numerical conditioning and problem structure (e.g. most of them comprise equality constraints). Third, the runtimes obtained via CPLEX's MATLAB interface either show a significant offset (when measured externally) or are accurate only up to 0.01 seconds (when measured internally), rendering them useless for small-scale problems. Nevertheless, we can summarise that qpOASES is competitive with OOQP and outperforms quadprog significantly.

It is more reasonable to explore qpOASES's computational efficiency based on QP problems arising within the MPC context. We compare its runtime with those of QP solvers generated by the CVXGEN tool [169] (see Section 8.2 for more details). For doing so, we run each of the two nonlinear MPC scenarios as presented in Section 8.3 with each of these two online QP solvers and compare their respective worst-case runtime. We use an embedded variant of qpOASES (using static memory only) and allow for warm-starting the QP solution (note that hot-starting is not

possible as QP matrices are changing in each MPC loop). All tests are performed on a standard PC (Intel Core 2 Duo P9700) having a 2.8 GHz dual-core processor and 4 GB RAM.

Table 4.2 summarises the obtained worst-case runtimes and shows that qpOASES seems to be competitive for these QP problems. Similar observations have been made in [251].

	qpOASES	CVXGEN[7]
CSTR, 2 control inputs, 20 intervals	0.15 ms	0.25 ms
Kite, 2 control inputs, 10 intervals	0.04 ms	0.09 ms

Table 4.2: Worst-case runtime of qpOASES (using warm-starts) and CVXGEN when running the two nonlinear MPC scenarios of Section 8.3.

Finally, we note that computational performance of qpOASES has been also compared to that of the interior-point solver of MOSEK [8] for time-optimal MPC of a linear motor drive system. The following runtimes have been reported in [230]: while qpOASES performed slightly faster than MOSEK when using cold-starts, it outperformed MOSEK by a factor of 10 when using dedicated hot-starts as described in Section 3.2.

[7]The reported runtimes for CVXGEN have been obtained by compiling the solvers using the -O3 flag of the GNU C compiler as this led to much better runtimes than the default option -Os. Moreover, the CVXGEN option better_start has been set to the value that resulted in the lowest runtime.

Chapter 5

Practical Issues and Industrial Case Studies

Having introduced the theoretical background of the online active set strategy in Chapter 3 and its open-source implementation `qpOASES` in Chapter 4, we will now focus on a number of further practical issues arising in real-world applications. We present two industrial case studies in which MPC based on `qpOASES` has been successfully applied: emission control of an integral gas engine using embedded controller hardware (see Section 5.1) as well as feasibility management for linear MPC for applications in the process industry (see Section 5.2). Both case studies not only provide deeper insight into practical requirements of MPC software like implementation issues or the need to handle infeasible QPs. They have also led to novel theoretical ideas for dealing with the encountered challenges. In particular, the first case study motivated the use of an asymmetric cost function to improve control performance for processes described by Wiener models, while the second case study motivated the development of a novel strategy to efficiently handle infeasible QPs.

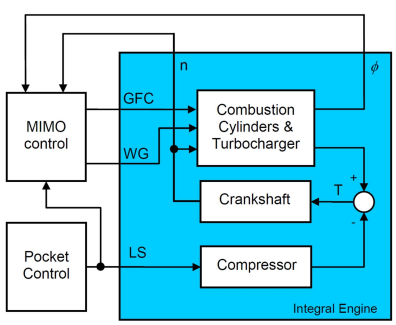
5.1 Industrial Case Study I: Emission Control of Integral Gas Engines

Our first case study aims at reducing the amount of nitrogen oxide (NO_x) emitted when running integral gas engines. For doing so, an adapted version of `qpOASES` has been integrated into an embedded engine controller developed by Hoerbiger Control Systems AB in Sweden. After summarising the MPC setup,

we discuss a couple of software issues that seem to be crucial for MPC software running on embedded hardware in industrial applications. Finally, we show how control performance of the considered gas engine was improved by introducing an asymmetric cost function to the MPC formulation. We generalise this idea to processes described by Wiener models and formulate conditions under which these processes can be controlled by linear MPC despite the model nonlinearities.

5.1.1 MPC of Integral Gas Engines

We consider an integral gas engine of type Clark TLA 6 as it is used for example in the pipeline network of the United States (see Figure 5.1). We closely follow the description in [2] and refer to [7, 132] for more details: The integral gas engine consists of a turbocharged 2-stroke reciprocating gas engine with counterflow scavenging and a reciprocating compressor. Both act on the same crankshaft, as the name "integral engine" suggests. The engine has 6 combustion and 3 compressor cylinders. It is typically operated in a narrow range with engine speeds of 280–300 rpm and relative fuel to air ratio of 0.55–0.62.



(a) Schematic drawing (taken from [7]), abbreviations are explained below.

(b) Clark TLA 6 at site in Newberry Springs, United States (taken from [2, 132]).

Figure 5.1: Integral gas engine of first industrial case study.

The engine speed n is affected by the crankshaft torque T and can be controlled by the so-called governor fuel command (GFC). The GFC represents the valve position of the fuel supply and directly influences the injection pressure and thus also the amount of fuel injected to the combustion cylinders. The fuel to air ratio ϕ is controlled via the wastegate (WG), i.e. the opening position of the turbocharger turbine bypass, while the ignition timing is usually kept constant. The power output can be adjusted with the compressor load by adding or removing discrete amounts of compressor clearances (pockets). In standard operation these

pockets are used to maintain almost constant engine power according to the slowly changing pipeline conditions. The considered integral engine has 6 small and 3 big pockets resulting in 28 possible load stages (LS). The steady-state values of two adjacent load stages differ by about 3 %. However, due to imperfect synchronisation of the pocket switching, substantial NO_x peaks arise during the transition between certain load stage changes.

Standard control configurations implemented as programmable logic controller (PLC) use *single input single output* (SISO) PID controllers, neglecting the fact that in particular ϕ has a strong coupling with the injected fuel amount [2]. Therefore, a—multiple input multiple output (MIMO)—linear MPC controller has been designed based on a discrete-time linear optimal control problem similar to (1.11).

The optimal control problem formulation comprises a linear grey-box model of the gas engine. This model is composed of parts obtained via system identification techniques and dynamic equations based on physical insight. The model comprises at least $n_x = 22$ differential states (including auxiliary states to model time delays) and $n_u = 2$ control inputs: the rates of change of the waste gate position and the governor fuel command. Both the rates of changes as well as the (integrated) actual value of the waste gate position and the governor fuel command are limited by bounds. Deviations of two controlled variables, the engine speed n and the fuel to air ratio ϕ, from their respective desired setpoints are penalised by a quadratic objective function. More details can be found in [7].

The MPC formulation uses a prediction horizon of about 300 time steps and a control horizon of 5–7 intervals (i.e. the control inputs at later time steps are fixed and thus do not enter the QP as optimisation variables). As the ratio $\frac{n_x}{n_u}$ is very large compared to the length of the control horizon, it is much more efficient to eliminate the states from the QP formulation before the runtime of the process (see Subsection 2.1.4). The resulting small-scale, dense QP comprises 10–14 (bounded) optimisation variables and up to several hundred constraints. For applying MPC to a real integral gas engine, this QP needs to be solved reliably in a small fraction of the sampling time of 100 milliseconds as several other tasks like communication and state estimation need to be performed by the engine controller during each sampling time. Moreover, all computations are performed on an embedded PowerPC whose computational power is many times lower than that of a standard PC.

For solving the resulting dense QPs in real-time, `qpOASES` has been integrated into the Hoerbiger Advanced Engine Controller [1]. Afterwards, the complete MPC controller has been successfully tested at the SoCalGas compressor station in Newberry Springs, California, United States. We summarise the results presented in [7]: both when decreasing the compressor work load as well as when increasing

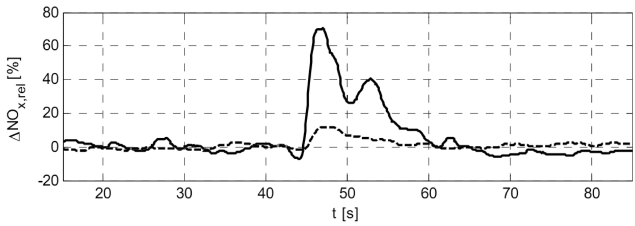


Figure 5.2: Comparison of NO_x emissions during a load increase: standard PLC (solid) vs. MPC using `qpOASES` (dashed) [7].

it, the MPC controller clearly outperformed the standard PLC. MPC was able to reduce the variations of the fuel to air ratio by 20–80 %, which led to significantly reduced NO_x emissions as shown in Figure 5.2. During this field test, the embedded variant of `qpOASES` worked without problems. The allowed maximum number of iterations per QP solution was only sporadically reached and the real-time variant described in Subsection 3.1.2 turned out to be a reasonable heuristic in these cases.

5.1.2 Software for Embedded Optimisation

Optimisation software written for use in embedded applications[1] has to meet different specifications than software written for academic purposes. This is particularly true if the code is going to be used in an industrial context. This subsection aims at summarising and discussing a number of issues that have been important for reliably integrating `qpOASES` to an embedded MPC controller. As most of them were also encountered during different projects with industrial partners, they are believed to be of general relevance.

The following first set of requirements applies to basically any code used in an industrial context and is independent of the employed hardware:

- The software has to include a *detailed documentation* (see [218]) describing how to use the software, making the user aware of actions that might cause an unwanted or even unsafe behaviour of the software, and enabling (external) developers to maintain it.

[1] The notion "embedded" is used in various contexts. For our discussion on embedded software for MPC, we assume the following, informal definition: An *embedded* application requires the optimisation software to run reliably without user-interaction, on PC-like hardware with computing power much lower than that of a standard PC, possibly with less accurate arithmetic.

- The software has to run *highly reliably* under all circumstances that might occur during its use on the target hardware. Most often this can only be verified by exhaustive testing but also certain programming techniques can help reducing possible bugs. Additionally, a thorough exception handling has to ensure that the software stops in a well-defined manner in case of a failure, such that higher-level logic can react to such an event (e.g. by switching to a stabilising default controller).
- The software should come with an *on-target verification procedure* allowing one to check whether it is operational after installing it on a target hardware.
- The software should be released under a suitable *licence*. For example, the GNU Lesser General Public License [134] offers the possibility to keep the optimisation part of the code open-source for scientific use while all other parts can remain proprietary to the company using it.

In addition to these more general requirements, the following guidelines enhance usability of the optimisation software on embedded hardware and help reducing the risk of software bugs:

- The software should be written in a *compilable (higher-level) programming language* to ensure maximum efficiency. Most rapid prototyping solutions support C or C++, sometimes also FORTRAN or small ADA can be used.
- The code should be as *self-contained* as possible in order to mostly avoid linking external libraries. These libraries often increase the size of the compiled code significantly and might even not be available for the embedded platform (due to compiler or licence issues). Even the number of included standard library headers should be reduced to a minimum.
- Restricting the code to *static memory allocations* excludes the risk of memory leaks and also avoids the computational overhead of dynamic memory allocations [59]. Moreover, when using C++, local variables requiring a significant amount of memory should be made static to avoid large automatic memory allocations on the stack.
- Many embedded hardware platforms do not support double precision arithmetic or need to emulate it in software (greatly slowing down execution). Thus, the code should be written in a way that allows to *switch between single and double precision arithmetic*. It goes without saying that a well-conditioned formulation of optimisation problems to be solved is even more crucial when using single precision.
- Depending on the concrete setup, *certain programming constructs might not be allowed*, e.g. the use of virtual functions, templates, Boolean variables etc.

- Console output or exchanging data from files is useful for testing the software on a standard PC but is often not available on embedded hardware. Thus, this functionality might be made optional, e.g. by introducing a corresponding compiler flag.

5.1.3 Linear MPC of Wiener Systems

We have seen in Subsection 5.1.1 that MPC was able to significantly reduce NO_x emissions of an integral gas engine compared to standard PLC. However, as NO_x emissions depend nonlinearly on the actually controlled fuel to air ratio, we will present an idea of how linear MPC can be modified to perform even better. Our description is based on [2] but presents a simpler and more efficient re-formulation of the MPC problem.

Model predictive control of the integral gas engine can make use of rather accurate linear ODE models describing the fuel to air ratio ϕ. As the amount of NO_x emissions is roughly proportional to ϕ within the relevant operating range (see Figure 5.3), linear MPC can be used to control ϕ in order to reduce NO_x emissions. In the following, we develop a better description of the NO_x emissions by adding a static nonlinearity to the linear model describing ϕ. This leads to a special class of ODE models, where we restrict the presentation to discrete-time formulations:

Definition 5.1 (discrete-time Wiener model): *A discrete-time linear ODE system (with linear output $z_k \in \mathbb{R}^{n_y}$) together with a static nonlinear map $h : \mathbb{R}^{n_y} \to \mathbb{R}^{n_y}$ describing the output $y_k \in \mathbb{R}^{n_y}$ of the system*

$$x_{k+1} = Ax_k + Bu_k \tag{5.1a}$$

$$z_k = Cx_k + Du_k \tag{5.1b}$$

$$y_k = h(z_k) \tag{5.1c}$$

is called a Wiener model. *Within this section, we make the standing assumption that h is invertible.* ○

There are two practical interpretations of Wiener models [2]:

- The nonlinear map might be interpreted as the sensor characteristic when measuring the actual linear output z_k. Assuming the function h to be invertible, the nonlinearity can be easily compensated in such a case.
- If the nonlinear map describes the actual output to be regulated, like the NO_x emission in our case, the controller has to take it into account in order to avoid linearisation errors.

Also in the second case, regulating the linear output z to the time-constant reference value $z_k^{\mathrm{ref}} \stackrel{\mathrm{def}}{=} h^{-1}(y_k^{\mathrm{ref}})$ is equivalent to regulating the nonlinear system to y_k^{ref}. However, this equivalence does not hold during transient conditions or in case of output constraints.

Though Wiener models can provide superior descriptions of systems with nonlinear gains, only little work has been done focussing on MPC based on Wiener models. Norquay et al. [179] suggest to eliminate the static nonlinearity and apply standard linear MPC to a pH neutralisation experiment. The use of robust linear MPC employing bounds on the static nonlinearities has been proposed in [32].

If the controller needs to be based on system dynamics modelled in form of a Wiener model, a standard linear MPC formulation cannot be used anymore. In fact, without additional assumptions on the nonlinear static output map, the discretised optimal control problem will usually result in a nonconvex, nonlinear programming problem (NLP). This loss of structure increases the computational load for solving these problems significantly and, due to the lack of convexity, the solver might even get stuck in a local minimum. While keeping convexity for systems with nonlinear dynamics is almost hopeless (see [12] for a few limited exceptions), this property remains for optimal control problems comprising a Wiener system if the nonlinear map h satisfies the following conditions:

Lemma 5.1 (MPC with Wiener models): *Let a discrete-time MPC problem of the form* (1.11) *but comprising a discrete-time Wiener model* (5.1) *be given. If* $\|h\|_2^2$ *is convex, then the resulting NLP has a convex objective function* (1.11a).

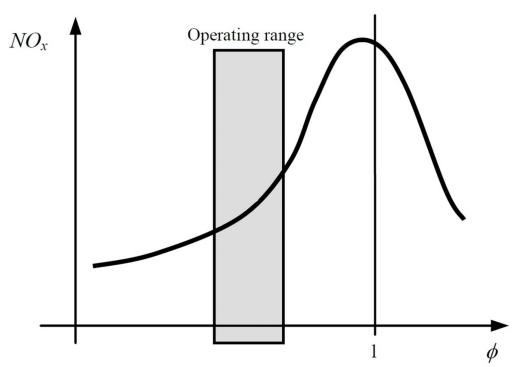


Figure 5.3: NO_x emission of the considered integral gas engine as a function of the fuel to air ratio ϕ (taken from [2]). Note that this function is invertible within the considered operating range only.

Moreover, if h is chosen such that also the inequality constraints (1.11e) *describe a convex set of feasible points, the resulting NLP is also convex.* ○

Proof: Convexity of the NLP objective function follows directly from the following observation: If $\|h\|_2^2$ is convex, also $h(z_k)'Qh(z_k)$ is so. This also remains true if all variables z_k are eliminated from the NLP formulation by expressing them as a linear transformation of the variables x_k and u_k (see e.g. [40]). That the whole resulting NLP is convex if the inequality constraint describes a convex set of feasible points is trivial. □

Note that the first condition is satisfied whenever $\|h\|_2$ is convex, but does not need to hold if h is itself convex and takes on negative values. The second condition is even more restrictive. For example, if h is itself a convex function, one can impose upper bounds on the nonlinear output y, but lower bounds would lead to a non-convex feasible set. A simple example satisfying the conditions of Lemma 5.1 is a convex, piecewise linear function consisting of two linear pieces that meet at the origin (see Figure 5.4).

Let us assume that a scalar function $h : \mathbb{R} \to \mathbb{R}$ satisfies both conditions of Lemma 5.1. Then there is some theoretical justification for trying to approximate the resulting convex NLP by a convex QP, i.e. sticking to a linear MPC formulation. If h is a piecewise linear function, such an approximation can be even made exact. In order to see that, let

$$h(z_k) = \begin{cases} a_{m_\mathrm{n}}\hat{z}_k + b_{m_\mathrm{n}} & \hat{z}_k \in \hat{Z}_{m_\mathrm{n}} \\ \dots & \\ a_{-1}\hat{z}_k + b_{-1} & \hat{z}_k \in \hat{Z}_{-1} \\ a_0\hat{z}_k & \hat{z}_k \in \hat{Z}_0 \\ a_1\hat{z}_k + b_1 & \hat{z}_k \in \hat{Z}_1 \\ \dots & \\ a_{m_\mathrm{p}}\hat{z}_k + b_{m_\mathrm{p}} & \hat{z}_k \in \hat{Z}_{m_\mathrm{p}} \end{cases} \tag{5.2}$$

be a piecewise linear function satisfying both conditions of Lemma 5.1 and, without loss of generality, crossing the origin (with m_n and m_p being non-negative integers and the $\hat{Z}_i$ form a mutually disjoint partition of the domain of $\hat{z}_k \stackrel{\mathrm{def}}{=} z_k - z_k^{\mathrm{ref}}$). In order to yield an exact representation as a convex QP, we introduce slack variables ε_k for the kth interval of the prediction horizon. Moreover, we add the following linear constraints to our linear optimal control problem (1.11):

$$1.\quad \varepsilon_j \geq \quad a_i\hat{z}_j + b_i \quad \forall j \in \{k_0, \dots, k_0 + n_\mathrm{p} - 1\}\ \forall i \in \{0, \dots, m_\mathrm{p}\}\,, \tag{5.3a}$$

$$2.\quad \varepsilon_j \geq -a_i\hat{z}_j - b_i \quad \forall j \in \{k_0, \dots, k_0 + n_\mathrm{p} - 1\}\ \forall i \in \{m_\mathrm{n}, \dots, -1\}\,. \tag{5.3b}$$

Our reformulation is exact, whenever all these inequality constraints hold with equality on the respective pieces $\hat{Z}_i$. This is ensured by modifying the objective function to

$$\min_{\substack{x_{k_0},\dots,x_{k_0+n_\mathrm{p}},\\ \varepsilon_{k_0},\dots,\varepsilon_{k_0+n_\mathrm{p}-1},\\ u_{k_0},\dots,u_{k_0+n_\mathrm{p}-1}}} \sum_{k=k_0}^{k_0+n_\mathrm{p}-1} \varepsilon_k' Q \varepsilon_k + \hat{u}_k' R \hat{u}_k + x_{k_0+n_\mathrm{p}}' P x_{k_0+n_\mathrm{p}}\,, \tag{5.4}$$

where we require $Q \in \mathcal{S}_{\succ 0}$.

Introducing the constraints (5.3) and the objective function (5.4), we arrive at a modified MPC formulation where the constrained slack variables implicitly describe an asymmetric convex quadratic objective function. This asymmetry captures the nonlinearity of h and is exact if h is a piecewise linear function. Otherwise, one can try to approximate h by a piecewise linear function. If h is differentiable, this approximation can be made arbitrarily good by introducing many linear pieces.

Our main motivation to introduce this modified linear MPC formulation is to avoid the need to solve a costly NLP problem. However, also our modified linear formulation increases the computational load of solving the resulting convex QP. In particular, we introduce an additional optimisation variable ε_j per time interval and an additional linear constraint per time interval and per linear piece of h (both for each output component respectively). When employing `qpOASES`, the computational load per QP iteration increases quadratically to cubically with the number of optimisation variables and linearly in the number of constraints. Moreover, a larger number of constraints will usually also lead to a larger number of QP iterations required to solve the problem. Thus, one can summarise that the main additional computation cost of our modified formulation stems from introducing $n_\mathrm{p} \cdot n_\mathrm{y}$ slack variables, while the $(m_\mathrm{n}+m_\mathrm{p}) \cdot n_\mathrm{p} \cdot n_\mathrm{y}$ additional constraints usually have less impact.

Nevertheless, if h (or its piecewise linear approximation) consists of many linear pieces, the number of additional constraints can become very large. We propose two possible remedies for this case: The first one is to use `qpOASES`'s feature to handle QP comprising many constraints (see Subsection 4.2.4). The second one is to introduce slack variables only on the first $n_\mathrm{sl} \leq n_\mathrm{p}$ intervals of the prediction horizon, i.e. one ignores the nonlinear dependence described by h on later intervals. This heuristic takes h into account only during transient behaviour of the process, which is supposed to take place at the beginning of the prediction horizon. On later time intervals, the heuristic relies on an explicit inversion of h when the system is expected to be close to a steady-state as discussed before.

The proposed modification for MPC formulations comprising a Wiener model has been applied to the integral gas engine. For doing so, the NO_x emissions

were approximated within the relevant operating range by a suitable piecewise linear function consisting of two linear pieces (see Figure 5.4). In [2] a prediction horizon of 300 and a shorter slack horizon of 50 intervals have been used, which increased computation time only moderately compared to a standard linear MPC formulation. It was shown that the modified objective function causes the gas engine to tend running lean and thus could reduce the NO_x peaks by up to 50 % (compared to standard linear MPC).

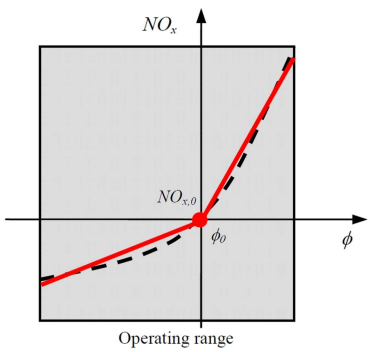


Figure 5.4: Piecewise linear approximation (solid) of the NO_x emission (dashed) within the relevant operating range as a function of the fuel to air ratio ϕ (taken from [2]).

5.2 Industrial Case Study II: MPC Feasibility Management in the Process Industry

Our second case study explores how to deal with MPC formulations that lead to infeasible QPs. This is an important question for practical setups and each MPC software for industrial use needs to be able to handle QP infeasibilities. In Subsection 5.2.2 we also briefly discuss the need to impose constraints with different priorities. Our considerations are motivated by a project with IPCOS NV, Belgium, aiming at coupling `qpOASES` to their INCA MPC software for advanced process control. Subsection 5.2.3 concludes this section by presenting a novel approach to efficiently handle infeasible QPs arising in MPC based on the online active set strategy.

5.2.1 Infeasibility Handling for Linear MPC

The ability to directly incorporate constraints into the control problem formulation is one of the major advantages of MPC over conventional feedback control approaches. By incorporating them into the underlying optimisation problem—a QP in the linear MPC case—the MPC controller makes sure that all these constraints will be met when applying the optimised control inputs to the process. However, this becomes impossible whenever the imposed constraints are too restrictive, i.e. if there do not exist any control inputs that satisfy all of them simultaneously. In this case, the underlying QP will become infeasible and thus does not have a solution.

The occurrence of infeasibility can often be avoided by a careful problem formulation. However, in real-world applications the MPC controller is confronted with noisy measurements, model-plant mismatch, discretisation errors or other phenomena that can render the resulting QP infeasible. As MPC usually operates processes close to their constraints for improving control performance, all industrial controllers need to implement a dedicated infeasibility handling strategy that deals with the resulting failure of the QP solver.

During the last two decades, several MPC infeasibility handling strategies have been proposed that modify the imposed constraints such that the modified optimisation problem can be solved [56, 208, 130]. All these strategies introduce *slack variables* as additional degrees of freedom into the QP to allow for constraint violation. In order to keep this violation at a minimum, non-zero values of the slack variables are penalised in the objective function of the QP. Selection and tuning of these penalisations is crucial for good control performance and at least two measures of optimality for these constraint relaxations exist: one usually either wants to minimise the amplitude (measured in a certain norm) or the duration of the constraint violations.

It has been shown that for certain systems, e.g. non-minimum phase systems, these are contrary design goals and [208] elaborates on the multi-objective nature of infeasibility strategies. We skip the so-called minimal time approach minimising the predicted duration of constraint violations and focus on briefly summarising strategies that aim at minimising the maximal predicted constraint violation (ℓ_∞ strategy) and the (weighted) ℓ_1 or ℓ_2 norm of the predicted constraint violations, respectively.

A first class of strategies always relaxes all constraints to become soft constraints in order to solve a QP that is guaranteed to be feasible. In this case, slack variables

ε_k are introduced into problem formulation (1.11) as follows:

$$\min_{\substack{x_{k_0},\dots,x_{k_0+n_\mathrm{p}},\\ y_{k_0},\dots,y_{k_0+n_\mathrm{p}},\\ \varepsilon_{k_0},\dots,\varepsilon_{k_0+n_\mathrm{p}},\\ u_{k_0},\dots,u_{k_0+n_\mathrm{p}-1}}} \sum_{k=k_0}^{k_0+n_\mathrm{p}-1} \hat{y}_k' Q \hat{y}_k + \hat{u}_k' R \hat{u}_k + \varepsilon_k' s_k + \varepsilon_k' S_k \varepsilon_k \; + \; \hat{x}_{k_0+n_\mathrm{p}}' P \hat{x}_{k_0+n_\mathrm{p}} \tag{5.5a}$$

$$\text{s.t.} \quad x_{k_0} = w_0\,, \tag{5.5b}$$

$$x_{k+1} = Ax_k + Bu_k \qquad \forall k \in \{k_0, \dots, k_0 + n_\mathrm{p} - 1\}\,, \tag{5.5c}$$

$$y_k = Cx_k + Du_k \qquad \forall k \in \{k_0, \dots, k_0 + n_\mathrm{p} - 1\}\,, \tag{5.5d}$$

$$b_k \geq M_k y_k + N_k u_k - \varepsilon_k \qquad \forall k \in \{k_0, \dots, k_0 + n_\mathrm{p} - 1\}\,, \tag{5.5e}$$

$$b_{k_0+n_\mathrm{p}} \geq M_{k_0+n_\mathrm{p}} x_{k_0+n_\mathrm{p}} - \varepsilon_{k_0+n_\mathrm{p}}\,, \tag{5.5f}$$

$$\varepsilon_k \geq 0 \qquad \forall k \in \{k_0, \dots, k_0 + n_\mathrm{p}\}\,, \tag{5.5g}$$

where the component-wise non-negative vectors s_k and the symmetric positive semi-definite matrices S_k define the penalties on the slack variables ε_k. Formulation (5.5) can capture ℓ_1 and ℓ_2 penalty norms (or mixtures of them), where each slack variable ε_k comprises one scalar slack variable per output constraint. In case of the ℓ_∞ norm only one slack variable ε for the whole prediction horizon is introduced also containing one scalar slack per output constraint. Also adaptations of these ideas to explicit MPC have been proposed [131].

Though practically appealing, these variants require a careful tuning of the weighting of the slack variables within the objective function. Otherwise control performance can deteriorate significantly or constraint violations occur though the original QP is feasible. In particular, exactness of the relaxation can only be achieved when using the ℓ_1 or ℓ_∞ norm [87]. We call an infeasibility handling strategy exact if the constraints of the MPC problem (5.5) remain unrelaxed whenever it is feasible. In order to facilitate the tuning, the use of time-varying weights combined with the ℓ_∞ norm is investigated in [130].

A second class of the strategies first checks for infeasibility, e.g. by solving a linear programming problem. In this case, relaxations are introduced only if the current QP is actually infeasible. These two-step approaches are exact, but require an additional LP to be solved at every time instant, even when the problem is feasible. A variant of this concept is to let the QP solver detect infeasibility and to re-run it with a relaxed problem in such a case. Also here two calls to an optimisation routine are required.

5.2.2 Handling of Prioritised Constraints

The strategies presented so far neglect a special aspect when designing strategies to handle QP infeasibilities: In a practical setup, it often turns out that satisfaction of certain constraints has higher priority than satisfaction of others. This is particularly true for QPs arising in our second case study dealing with MPC for chemical processes.

The INCA MPC software has been designed to control batch and other chemical processes and allows the user to set up control problems in a hierarchical manner [41]. It starts with optimising an MPC formulation containing only operating specifications with the highest priority. Optimising these specifications has absolute priority over further specifications with lower priority. Only if the solution found leaves further degrees of freedom to be optimised, the operating specification at the next lower priority level are included into the MPC formulation. A typical example for such a hierarchical formulation in the process industry could be: first optimise plant safety with higher priority than product quality, then optimise product quality with higher priority than energy saving or cost reduction. As both safety requirements as well as minimum standards for product qualities are typically formulated as constraints within an MPC formulation, we will restrict our discussion to prioritised constraints (a discussion on multi-objective formulations can be found in [145]):

$$\min_{x\in\mathbb{R}^n} \quad \tfrac{1}{2}x'Hx + x'g(w) \tag{5.6a}$$

$$\text{s.t.} \quad G_1 x \geq b_1(w)\,, \tag{5.6b}$$

$$G_2 x \geq b_2(w)\,, \tag{5.6c}$$

$$\dots$$

$$G_{n_{\mathrm{PL}}} x \geq b_{n_{\mathrm{PL}}}(w)\,, \tag{5.6d}$$

where $n_{\mathrm{PL}} \in \mathbb{N}$ is the number of *priority levels* within the constraints. Here, we assume for notational simplicity that all constraints are expressed as inequality constraints.

A straight-forward approach to tackle formulation (5.6) is by solving a sequence of QPs. It starts with solving a QP comprising only the constraints with highest priority, which are relaxed as described in Subsection 5.2.1, i.e. $G_1 x \geq b_1(w_0) - \varepsilon_1$ with $\varepsilon_1 \geq 0$. Preferably a slack penalisation that guarantees exactness is used. Let $\varepsilon_1^{\mathrm{opt}}$ be the value of the slack variables at the optimal solution of this first QP. Then the next QP is solved where $G_1 x \geq b_1(w_0) - \varepsilon_1^{\mathrm{opt}}$, $\varepsilon_1^{\mathrm{opt}}$ fixed, are imposed as hard constraints and the inequalities of the next lower priority level $G_2 x \geq b_2(w_0) - \varepsilon_2$,

$\varepsilon_2 \geq 0$, are added as soft constraints. This procedure is continued until the constraints of all priority levels are added.

This approach is easy to implement and guarantees that constraints at a higher priority level are not violated more than necessary for satisfying constraints at lower levels. However, a major drawback is the significant computational load as n_{PL} QPs (augmented with slack variables) need to be solved.

In order to overcome the need to solve up to n_{PL} QPs, Vada et al. [228, 229] proposed an approach that only requires the solution of one additional LP problem. The solution of this LP gives optimal relaxations of all constraints of the MPC formulation such that the resulting QP is feasible and respects the imposed priorities. However, the presented algorithm for designing the objective weights of this LP offline relies on a polytopic partition of the MPC state space. Thus, it does not seem to be practical for MPC problems with higher state dimensions as typically arising in advanced process control.

`qpOASES` cannot handle constraint priorities directly. However, when following the straight-forward approach as described above, solving the sequence of n_{PL} QPs can be speed-up by using the warm-start capabilities of the online active set strategy as described in Section 3.2.

5.2.3 Infeasibility Handling using the Online Active Set Strategy

`qpOASES` can be used as it is to solve a relaxed problem (5.5) for handling infeasibilities arising in MPC. As these problems typically comprise a large number of slack variables, the Hessian and constraint matrix of the resulting QP is partly sparse. Therefore, the features mentioned in Subsection 4.2.5 can be used to speed-up computation by exploiting this structure. Moreover, the QP sequence to be solved to tackle formulation (5.6) including prioritised constraints often comprises more constraints than optimisation variables. Therefore the trick described in Subsection 4.2.4 to speed-up QP solution can be applied.

Alternatively, we will now present a novel approach based on the online active set strategy to relax QPs penalising the ℓ_1 or ℓ_2 norm of the slack variables. It allows one to implicitly relax QPs without causing any extra computational cost in cases where the QP is feasible. If the QP turns out to be infeasible while solving it, a minimal number of slacks is added to yield a feasible relaxed QP which can be solved very efficiently within the usual homotopy framework. Moreover, by automatically adapting the weights of the ℓ_1 penalty based on the dual solution of

the current intermediate QP, this infeasibility handling strategy can be basically made exact[2].

We start our presentation by noting that the soft-constrained MPC problem (5.5) can be written as parametric QP, too:

$$\min_{x,\,\mathcal{E}} \quad \frac{1}{2}x'Hx + x'g(w) + \sum_{i=1}^{m}\frac{1}{2}S_{ii}\mathcal{E}_i^2 + \sum_{i=1}^{m} s_i\mathcal{E}_i \tag{5.7a}$$

$$\text{s.t.} \quad G'x \geq b(w_0) - \mathcal{E}\,, \tag{5.7b}$$

$$\mathcal{E} \quad \geq 0\,, \tag{5.7c}$$

where we introduce slack variables $\mathcal{E} \in \mathbb{R}^m$. Moreover, we assume a constant positive weighting vector $s \in \mathbb{R}^m$ representing the ℓ_1 penalty as well as a time-constant, diagonal weighting matrix $S \in \mathcal{S}^m$ with non-negative entries for the ℓ_2 penalty. For the moment we regard s and S to be given but we will later use s as variable parameter.

We propose to start solving QP (2.9) corresponding to an unrelaxed MPC formulation by means of the online active set strategy. While moving along the homotopy path towards the solution of the current QP, QP infeasibility corresponds to reaching the boundary of the convex set of feasible parameters $\mathcal{P}$ (see Definition 2.6). Algorithmically, this is indicated by either of the following two possibilities:

1. A constraint needs to be added to the active set in step (6) of Algorithm 3.1 which is linearly dependent from the other active ones but no constraint can be removed to resolving this linear dependence.

2. The limits of active constraints change in such a way that their range become conflicting. This can be detected by rapidly growing dual multipliers corresponding to the involved active constraints [240].

In both cases, we propose to only introduce a slack variable for the constraint that causes the (current) infeasibility. In the first case, this means that we only relax the constraint to be added to the active set; in the second case, the active constraint whose dual multiplier tends to infinity is relaxed. This can be interpreted as switching to the relaxed formulation (5.7) and one can continue to solve this relaxed QP by further proceeding along the usual homotopy path. While doing so, possibly more constraints need to be relaxed. Conversely, relaxed constraints might become feasible in their original form again.

[2]Thomas Wiese contributed to this research during his Bachelor Thesis project at K.U. Leuven [240]. He also wrote a prototype implementation of the presented approach within `qpOASES`.

Introducing only one slack variable per QP iteration if necessary can be implemented efficiently. The reason is that one can show that the KKT optimality conditions of (5.7) are similar to those of (2.9), with the following analogue to Equation (3.4) (see appendix of [240]):

$$\begin{pmatrix} H + \sum_{i \in \mathbb{J}} S_{ii} G_i G_i' & G_{\mathbb{A}\setminus\mathbb{J}}' \\ G_{\mathbb{A}\setminus\mathbb{J}} & 0 \end{pmatrix} \begin{pmatrix} x \\ -y_{\mathbb{A}\setminus\mathbb{J}} \end{pmatrix} = \begin{pmatrix} -g(w_0) + \sum_{i \in \mathbb{J}} G_i (s + S_{ii} b_i(w_0)) \\ b_{\mathbb{A}\setminus\mathbb{J}} \end{pmatrix}, \quad (5.8)$$

where we define $\mathbb{J} \stackrel{\text{def}}{=} \{i \in k \in \{1, \ldots, m\} \mid \mathcal{E}_i > 0\}$, i.e. the index set of relaxed constraints. It is important to note that the slack variables $\mathcal{E}$ enter this KKT system only implicitly and thus do not increase the computational load to determine the next step direction along the homotopy path. In particular, no primal step in the slack variables $\Delta\mathcal{E}$ needs to be computed.

This raises the question of how the online active set strategy can detect whether an index enters or leaves $\mathbb{J}$. From the analysis of exact penalty functions (see [87]), it can be seen that $\mathcal{E}_j > 0$ if and only if $y_j > s$ for $j \in \mathbb{A}$ (exact penalty property). Therefore, we can use the condition $y_j + \tau \Delta y_j \leq s$ to detect whether an index $j \in \mathbb{A}$ enters $\mathbb{J}$ and thus can add it to the step length determination implied by Equations (3.7). As complementary condition for checking whether an index j leaves $\mathbb{J}$ (and enters $\mathbb{A}$), we use $G_j'(x + \tau\Delta x) = b_j(w_0) + \tau \Delta b_j$, i.e. we look for the closest point where constraint j can be met again without relaxation.

In order to yield an efficient implementation, we need to be able to update the KKT system (5.8) efficiently whenever the cardinality of $\mathbb{J}$ changes. If only an ℓ_1 penalty term is present, i.e. $S = 0$, only the gradient in the right-hand side vector changes. This can be perfectly captured within the usual parametric QP framework of the online active set strategy at no significant additional cost. If (also) an ℓ_2 penalty term is present in formulation (5.7), we need to update the Hessian matrix. Adding or removing a constraint j to or from $\mathbb{J}$ implies that we have to add or subtract $S_{jj} G_j G_j'$ from the Hessian matrix. This means that also `qpOASES`'s Cholesky factors of the projected Hessian matrix need to be updated accordingly. Fortunately, as both operations describe symmetric rank-1 updates, it is possible to update the Cholesky decomposition efficiently (requiring $\mathcal{O}(n^2)$ operations) by means of rank-1 update and downdates of the Cholesky factors [209]. We repeat that these additional computations are not necessary as long as the original QP is feasible, i.e. $\mathbb{J} = \emptyset$.

The fact that the set of feasible parameters $\mathcal{P}$ is convex (cf. Theorem 2.3) ensures that our infeasibility handling strategy is "semi-exact" in the sense that it uses non-zero slack variables only if the original QP is infeasible. However, it might happen that slack variables (falsely) remain non-zero after w_0 has entered $\mathcal{P}$ again.

Referring to [87], this effect can be avoided by choosing the weights of the ℓ_1 penalty terms s larger than the largest absolute value of the dual multipliers at the optimal solution. For small-size problems this value might be pre-computed as suggested in [145]. However, in most cases this value is not known beforehand and standard implementations simply choose (very) large weights in order to yield an exact penalty function. As this might lead to numerical difficulties, we propose an adaptive choice of s: Once `qpOASES` detects infeasibility of the current QP, it chooses all entries of s to be larger than the largest absolute value of the dual multipliers of the current intermediate QP solution. This value is kept constant until w_0 re-enters $\mathcal{P}$.

Evaluation of the practical benefits of this novel strategy to handle infeasible MPC problems is still ongoing research. Therefore, we conclude with summarising its theoretical strengths:

- only one QP needs to be solved at each sampling instant and computational load is not increased whenever the current QP is feasible;
- slack variables are only introduced implicitly up to an extent actually required, which significantly reduces the problem dimension compared to the fully relaxed formulation (5.7) and should thus reduce computation time;
- the adaptive choice of s mimics an exact penalty function while avoiding numerical difficulties;
- when leaving the set of feasible parameters $\mathcal{P}$, the primal QP solution keeps on changing continuously as in the standard variant of the online active set strategy.

Part II

Nonlinear Model Predictive Control

Chapter 6

Overview of Existing Methods for Nonlinear MPC

This chapter outlines general approaches to solve nonlinear MPC problems. The focus lies on so-called direct methods that first discretise the problem (1.2) and then solve the resulting nonlinear programming problem (NLP). For doing so, tailored methods to solve linearised subproblems and for efficient derivative computation become crucial to yield a sufficiently accurate approximate solution in real-time. This general discussion is concluded by reviewing a number of efficient NMPC schemes that have been proposed during the last two decades for fast real-time control.

6.1 Tackling the Infinite-Dimensional Optimisation Problem

Nonlinear MPC requires the repeated solution of optimal control problems of the form (1.2) to obtain optimised control inputs u^{opt}. This constitutes an infinite-dimensional optimisation problem over the function space to which u belongs. In certain situations it can be possible to derive the exact optimal solution analytically, but in general numerical methods for computing an approximate solution have to be employed. These methods are usually divided into the following three classes [27]:

1. indirect methods based on Pontryagin's minimum principle,
2. Hamilton-Jacobi-Bellman (HJB) approaches,

3. direct methods based on a finite-dimensional control parameterisation.

Methods belonging to either of the first two classes initially treat u as a function and formulate infinite-dimensional optimality conditions that are discretised afterwards to numerically find a solution (often summarised in the phrase "first optimise, then discretise"). In contrast, direct methods first approximate u by a finite-dimensional parameterisation and then numerically solve the resulting finite-dimensional NLP problem ("first discretise, then optimise").

6.1.1 Indirect and Hamilton-Jacobi-Bellman Approaches

Indirect methods compute the optimal control inputs based on Pontryagin's minimum principle [192] by introducing the *Hamiltonian function*[1]

$$\mathcal{H}(x,u,\lambda) \stackrel{\text{def}}{=} \psi\left(x(t),u(t)\right) + \lambda(t)' f\left(x(t),u(t)\right), \tag{6.1}$$

with *co-states* (or adjoint variables) $\lambda : \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$. Then necessary first-order optimality conditions for solving (1.2) can be obtained in form of the following boundary value problem [43]:

$$\frac{\partial}{\partial t} x^{\mathrm{opt}}(t) = \nabla_\lambda \mathcal{H}(x^{\mathrm{opt}}(t), u^{\mathrm{opt}}(t), \lambda^{\mathrm{opt}}(t)), \tag{6.2a}$$

$$\frac{\partial}{\partial t} \lambda^{\mathrm{opt}}(t) = -\nabla_x \mathcal{H}(x^{\mathrm{opt}}(t), u^{\mathrm{opt}}(t), \lambda^{\mathrm{opt}}(t)), \tag{6.2b}$$

$$x^{\mathrm{opt}}(t_0) = w_0(t_0), \tag{6.2c}$$

$$\lambda^{\mathrm{opt}}(t_0 + t_\mathrm{p}) = \nabla_x \phi(x^{\mathrm{opt}}(t_0 + t_\mathrm{p})), \tag{6.2d}$$

which is also known as *Euler-Lagrange* differential equation. If optimal state and co-state trajectories $x^{\mathrm{opt}}(t)$ and $\lambda^{\mathrm{opt}}(t)$ are found, the optimal control input trajectory can be obtained by a point-wise minimisation of the Hamiltonian function:

$$u^{\mathrm{opt}}(\check{t}) = \underset{u(\check{t})}{\arg\min}\ \mathcal{H}\left(x^{\mathrm{opt}}(\check{t}), u(\check{t}), \lambda^{\mathrm{opt}}(\check{t})\right) \quad \forall\, \check{t} \in [t_0, t_0 + T]. \tag{6.3}$$

A number of indirect approaches are summarised in [27]. Therein, also the following practical drawbacks are mentioned: it is non-trivial to formulate the boundary value problem (6.2) in a numerically stable way; for handling inequality

[1] For simplicity, we assume that $y(t) = x(t)$ for all $t \in \mathbb{T}_\mathrm{p}$ and leave out the inequality constraints.

constraints, their switching structure describing when they become active and inactive has to be guessed before-hand; suitable initial guesses for the optimal state and co-state trajectories need to be provided to ensure convergence of iterative solution methods. We will discuss a numerical approach based on Equations (6.2) in Subsection 6.4.4.

The second class of methods to tackle the infinite-dimensional optimisation problem (1.2) formulates the so-called *Hamilton-Jacobi-Bellman* nonlinear partial differential equation. It is based on the *principle of optimality of subarcs*, which can be informally stated as follows [16]: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision[2]". *Dynamic programming* methods aim at solving a discretised HJB equation [20].

According to [48, 27], solving the HJB equation is usually numerically intractable for higher state dimensions and analytic solutions are known for special cases only (e.g. for unconstrained linear MPC problems). Therefore, we will not further consider them in this thesis.

6.1.2 Direct Methods

Most MPC algorithms are based on direct methods that first transform the infinite-dimensional optimal control problem into a finite-dimensional NLP (see Section 6.2 for a formal definition). All direct methods parameterise the control input trajectory u by a finite number of parameters $q \in \mathbb{R}^{n_\mathrm{q}}$ to yield a suitable approximation of u:

$$u^{\mathrm{appr}}(t;q) \;\approx\; u(t) \quad \forall\, t \in \mathbb{T}_\mathrm{p}\,. \tag{6.4}$$

Depending on the treatment of the differential states x, one can distinguish two main sub-classes of direct methods: sequential and simultaneous approaches, but also combinations of both exist.

Sequential direct methods make use of the fact that the state trajectory x is uniquely determined by Equations (1.2b)–(1.2c) for any given parameterised control input trajectory $u^{\mathrm{appr}}(t;q)$ and initial value $s_0 \stackrel{\mathrm{def}}{=} x(t_0)$. Thus, by performing a single system simulation using a numerical integrator, an approximate state trajectory $x^{\mathrm{appr}}(t;s_0,q)$ can be obtained on $\mathbb{T}_\mathrm{p}$. The same holds for the output trajectory for which an approximation $y^{\mathrm{appr}}(t;s_0,q)$ can be obtained using Equation (1.2d).

[2]Where "optimal policy" refers to an optimal control input trajectory and "decision" refers to any piece of such an optimal trajectory.

This results in the following finite-dimensional optimisation problem[3]:

$$\min_{\substack{s_0 \in \mathbb{R}^{n_x}, \\ q \in \mathbb{R}^{n_q}}} \int_{t_0}^{t_0+t_\mathrm{p}} \psi\left(y^{\mathrm{appr}}(t; s_0, q), u^{\mathrm{appr}}(t; q)\right) dt + \phi\left(x^{\mathrm{appr}}(t_0 + t_\mathrm{p}; s_0, q)\right) \tag{6.5a}$$

$$0 \geq c\left(y^{\mathrm{appr}}(t; s_0, q), u^{\mathrm{appr}}(t; q)\right) \quad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{6.5b}$$

$$0 \geq c^{\mathrm{term}}\left(x^{\mathrm{appr}}(t_0 + t_\mathrm{p}; s_0, q)\right)\,, \tag{6.5c}$$

where we eliminated a possible explicit dependence on t or additional parameters p as described in Section 1.3.

This optimisation problem comprises only a finite number of optimisation variables but still comprises infinitely many constraints. Thus, constraints are usually only enforced at the points of a finite grid covering $\mathbb{T}_\mathrm{p}$ to yield a finite-dimensional problem. Sequential direct methods, like the described *direct single shooting* method, thus solve the optimal control problem (1.2) by first simulating the system to eliminate the states and outputs and second optimise the resulting discretised problem (6.5). This idea was already proposed more than 30 years ago in [205].

In contrast, *simultaneous direct methods* perform these two steps (system simulation and optimisation) at once. Instead of performing a system simulation explicitly, the differential equation (1.2c) and the output equation (1.2d) are discretised in time and are kept within the optimisation problem as equality constraints. This state discretisation is usually done based on *direct collocation* or *direct multiple shooting* methods.

Collocation methods, as introduced into optimal control in [227] and used for MPC in [25], divide the prediction horizon $\mathbb{T}_\mathrm{p}$ into a fine time grid and approximate both the state and control trajectory on each interval i of this grid by (low-order) polynomials $x^{\mathrm{poly}}(t; s_i)$ and $u^{\mathrm{poly}}(t; q_i)$ with coefficients s_i and q_i. In order to guarantee that these polynomials result in a continuous approximation of the state trajectory, a number of equality constraints are imposed on their coefficients. They ensure that the polynomials satisfy the differential equation at all grid points and that they fit together continuously. Enforcing the inequality constraints (1.2e) only on a finite grid like in the sequential approach, direct collocation yields a finite-dimensional optimisation problem in the variables s_i and q_i. That way, the system simulation is done implicitly while solving this optimisation problem.

Also multiple shooting methods, as introduced into optimal control by [191, 38], divide the prediction horizon $\mathbb{T}_\mathrm{p}$ into a finite grid, which is usually chosen much

[3] We keep the optimisation variable s_0 within the problem formulation though it could easily be eliminated. This formulation will allow us to describe the real-time iteration algorithm of Section 7.2 more conveniently.

coarser than the one used for collocation methods. On each multiple shooting interval $[\tau_i, \tau_{i+1}]$, $0 \leq i < n_\mathrm{p}$, the system is simulated like in single shooting starting from artificial initial values $s_i \in \mathbb{R}^{n_\mathrm{x}}$ based on a local control trajectory parameterisation $u_i^{\mathrm{appr}}(t; q_i)$. This yields smooth state trajectories $x_i^{\mathrm{appr}}(t; s_i, q_i)$ and corresponding output trajectories $y_i^{\mathrm{appr}}(t; s_i, q_i)$ on each multiple shooting interval. So-called *matching conditions*, requiring that each final value of the simulated state trajectories equals the artificial initial value of the next interval, guarantee that a continuous state trajectory is obtained. Also here the inequality constraints (1.2e) are only enforced on a finite grid yielding a finite-dimensional optimisation problem in the variables s_i and q_i:

$$\min_{\substack{s_0,\dots,s_{n_\mathrm{p}} \\ q_0,\dots,q_{n_\mathrm{p}-1}}} \sum_{i=0}^{n_\mathrm{p}-1} \int_{\tau_i}^{\tau_{i+1}} \psi\left(y_i^{\mathrm{appr}}(t; s_i, q_i), u_i^{\mathrm{appr}}(t; q_i)\right) dt \; + \; \phi\left(s_{n_\mathrm{p}}\right) \tag{6.6a}$$

$$\text{s.t. } s_0 \;=\; w_0\,, \tag{6.6b}$$

$$s_{i+1} \;=\; x_i^{\mathrm{appr}}(\tau_{i+1}; s_i, q_i) \qquad \forall\, i \in \{0, \dots, n_\mathrm{p} - 1\}\,, \tag{6.6c}$$

$$0 \;\geq\; c\left(y_i^{\mathrm{appr}}(\tau_i; s_i, q_i), u_i^{\mathrm{appr}}(\tau_i; q_i)\right) \;\forall\, i \in \{0, \dots, n_\mathrm{p} - 1\}\,, \tag{6.6d}$$

$$0 \;\geq\; c^{\mathrm{term}}\left(s_{n_\mathrm{p}}\right)\,. \tag{6.6e}$$

Also multiple shooting techniques perform the overall system simulation simultaneously with the optimisation procedure, but simulation of the single trajectory pieces is done sequentially as in single shooting. Therefore, multiple shooting is sometimes rather categorised as a hybrid method and not as a fully simultaneous one.

The different direct methods have their respective theoretical and numerical strengths and disadvantages, so it depends on the specific application context which one is most suited. We will elaborate on this in Section 6.3.3.

6.2 Nonlinear Programming

The ability to efficiently solve nonlinear programming problems (NLPs) is an algorithmic key ingredient to applying nonlinear MPC at high sampling rates in real-time. This section formally introduces NLPs and briefly sketches different Newton-type optimisation algorithms for solving them. A thorough treatment of this topic can be found in several excellent textbooks [87, 104, 178].

6.2.1 Definitions and Optimality Conditions

This section introduces algorithms to solve optimisation problems given in the following form:

Definition 6.1 (nonlinear program): *The optimisation problem*

$$\text{NLP}: \quad \min_{X \in \mathbb{R}^n} \quad F(X) \tag{6.7a}$$

$$\text{s.t.} \quad G(X) \;=\; 0\,, \tag{6.7b}$$

$$H(X) \;\leq\; 0 \tag{6.7c}$$

with an objective function $F : \mathbb{R}^n \to \mathbb{R}$ *and constraint functions* $G : \mathbb{R}^n \to \mathbb{R}^{n_\mathrm{G}}$ *and* $H : \mathbb{R}^n \to \mathbb{R}^{n_\mathrm{H}}$ *is called a* nonlinear program. *All functions are assumed to be twice continuously differentiable.* ○

We call an NLP (6.7) convex if and only if its defining functions F and H are convex and G is linear. For analysing its optimal solutions, we introduce the following:

Definition 6.2 (feasible, regular and optimal points of an NLP): *Let an NLP of the form* (6.7) *and any point* $X^\star \in \mathbb{R}^n$ *be given.* $X^\star$ *is called*

- feasible *iff it satisfies all constraints, i.e.*

  $$G(X^\star) \;=\; 0 \quad \textit{and} \quad H(X^\star) \;\leq\; 0 \tag{6.8}$$

  and infeasible *otherwise (the set of all feasible points is denoted by* $\mathcal{F}$ *as in the case of quadratic programs);*

- regular *iff it is feasible and the Jacobian matrix of the active constraints at* $X^\star$*, i.e.* $\nabla(G(X^\star)', H_{\mathbb{A}}(X^\star)')$*, has full row rank, i.e. the LICQ holds at* $X^\star$ *(where,*

  $$\mathbb{A} \;\stackrel{\text{def}}{=}\; \big\{ i \in \{1, \ldots, n_\mathrm{H}\} \;\big|\; H_i(X^\star) = 0 \big\} \tag{6.9}$$

  denotes the index set of all inequality constraints that are active at $X^\star$*);*

- *a* global optimum *iff it minimises* F *among all feasible points, i.e.*

  $$F(X^\star) \;\leq\; F(\check{X}) \quad \forall\, \check{X} \in \mathcal{F}\,; \tag{6.10}$$

- *a* local optimum *iff it minimises* F *among all feasible points in a neighbourhood of* $\check{X}$*, i.e.*

  $$F(X^\star) \;\leq\; F(\check{X}) \quad \forall\, \check{X} \in \mathcal{F} \cap \{X \in \mathbb{R}^n \;|\; \|X - X^\star\| < \varepsilon\} \tag{6.11}$$

  for some $\varepsilon > 0$. ○

Not much is known about optimality conditions characterising global optima of general NLPs and only local optima can be satisfactorily described. An important exception form convex NLPs, for which it can be shown that every local optimum is also a global one. We will now state the famous KKT optimality conditions that can be shown to be necessary for any local minimum [142, 151].

Theorem 6.1 (Karush-Kuhn-Tucker conditions for NLPs): *If a regular point $X^{\text{opt}} \in \mathbb{R}^n$ is a local optimum of the* NLP (6.7)*, then there exist multiplier vectors $\lambda^{\text{opt}} \in \mathbb{R}^{n_{\text{G}}}$ and $\mu^{\text{opt}} \in \mathbb{R}^{n_{\text{H}}}$ such that the following conditions are satisfied:*

$$\begin{aligned}
\nabla_X \mathcal{L}\left(X^{\text{opt}}, \lambda^{\text{opt}}, \mu^{\text{opt}}\right) &= 0\,, && \text{(6.12a)}\\
G(X^{\text{opt}}) &= 0\,, && \text{(6.12b)}\\
H(X^{\text{opt}}) &\leq 0\,, && \text{(6.12c)}\\
\mu^{\text{opt}} &\geq 0\,, && \text{(6.12d)}\\
H_i(X^{\text{opt}})\mu_i^{\text{opt}} &= 0 \qquad \forall\, i \in \{1, \ldots, n_{\text{H}}\}\,, && \text{(6.12e)}
\end{aligned}$$

where the so-called Lagrange function *is defined as*

$$\mathcal{L}(X, \lambda, \mu) \stackrel{\text{def}}{=} F(X) + \lambda' G(X) + \mu' H(X)\,. \qquad \text{(6.13)}$$

○

Sufficient local optimality conditions and a further discussion on regularity can be found in [178].

Analogous to the QP case, we also introduce the following

Definition 6.3 (parametric nonlinear program): *The optimisation problem*

$$\begin{aligned}
\text{NLP}(w): \quad \min_{X \in \mathbb{R}^n} \quad & F(X) && \text{(6.14a)}\\
\text{s.t.} \quad & G(X; w) = 0\,, && \text{(6.14b)}\\
& H(X) \leq 0 && \text{(6.14c)}
\end{aligned}$$

depending on a varying parameter $w \in \mathbb{R}^{n_{\text{x}}}$ is called a parametric nonlinear program. ○

6.2.2 Newton-Type Optimisation

Let us ignore the inequality constraints defined by H for a moment. Then a reasonable approach to search for local minima is to identify points that satisfy

the KKT optimality conditions (6.12a)–(6.12b). For solving this set of nonlinear equations, Newton's method has proven to be well-suited. For arbitrary nonlinear equations $C(W) = 0$, it works as follows: Starting from an initial guess $W^{(0)}$, it generates a sequence of iterates $W^{(k)}$ that each solve a linearisation of the equations at the previous iterate. This means that for any given iterate $W^{(k)}$, $k \geq 0$, the next iterate $W^{(k+1)}$ is required to satisfy

$$C(W^{(k)}) + \nabla C(W^{(k)})' \left(W^{(k+1)} - W^{(k)} \right) \; = \; 0 \,, \tag{6.15}$$

which can be equivalently formulated as

$$\nabla C(W^{(k)})' \Delta W^{(k)} = -C(W^{(k)}) \,, \quad W^{(k+1)} = W^{(k)} + \Delta W^{(k)} \,. \tag{6.16}$$

The hope is that the linearisations are sufficiently good approximations of the original nonlinear system and that the iterates converge towards a solution W^{opt}. *Globalisation strategies* aim at ensuring convergence to local optima from arbitrary initial guesses by taking so-called *damped Newton steps* $W^{(k+1)} = W^{(k)} + \alpha^{(k)} \Delta W^{(k)}$ with suitable $\alpha^{(k)} \in (0, 1]$. An excellent overview of Newton's methods including convergence analysis, algorithmic variants and solution methods is given in [60].

If Newton's method is started sufficiently close to a solution, it converges at quadratic rate towards this solution. In order to speed-up computation, the Jacobian $\nabla C(W^{(k)})'$ might not be computed or inverted exactly. This yields cheaper iterations but leads to slower convergence rates. These variants are known as *Newton-type methods*.

Most MPC formulations comprise inequality constraints leading to NLP problems that involve inequalities $H(X) \leq 0$. Thus, we are interested in extensions to Newton-type optimisation methods that can also deal with the KKT optimality conditions (6.12c)–(6.12e). Depending on how these additional conditions are treated, one can distinguish two main classes of Newton-type optimisation methods: sequential quadratic programming (SQP) methods and interior-point (IP) methods. We will discuss them in the following subsections, which follow the presentation given in [65].

6.2.3 Sequential Quadratic Programming

A first Newton-type variant to iteratively solve the KKT system (6.12) is to linearise all nonlinear functions (i.e. also the ones corresponding to the NLP inequalities) at each iteration k. It can be shown that the resulting linear complementarity system can be interpreted as the KKT conditions of the following

quadratic program:

$$\min_{X\in\mathbb{R}^n} \quad \frac{1}{2}(X - X^{(k)})'\nabla^2_X\mathcal{L}(X^{(k)},\lambda^{(k)},\mu^{(k)})(X - X^{(k)}) + \nabla F(X^{(k)})'X \quad (6.17a)$$

$$\text{s.t.} \quad G(X^{(k)}) + \nabla G(X^{(k)})'(X - X^{(k)}) \;=\; 0\,, \quad (6.17b)$$

$$H(X^{(k)}) + \nabla H(X^{(k)})'(X - X^{(k)}) \;\leq\; 0\,, \quad (6.17c)$$

where $X^{(k)}$ denotes the kth primal solution iterate. In case the Hessian matrix $\nabla^2_X\mathcal{L}(X^{(k)},\lambda^{(k)},\mu^{(k)})$ is positive semi-definite, this QP is convex and global solutions can be found reliably. Thus, most of the QP solvers mentioned in Section 2.3 can be used to obtain the current step direction $\Delta X^{(k)} \stackrel{\text{def}}{=} X^{\text{opt}}_{\text{QP}} - X^{(k)}$. Moreover, the multipliers are set to the optimal QP multipliers: $\lambda^{(k+1)} \stackrel{\text{def}}{=} \lambda^{\text{opt}}_{\text{QP}}$, $\mu^{(k+1)} \stackrel{\text{def}}{=} \mu^{\text{opt}}_{\text{QP}}$. This general approach to address the NLP problem (6.7) is called *sequential quadratic programming* (SQP) and goes back to [245].

Apart from this *exact Hessian* SQP method presented above, several other SQP variants exist that make use of inexact Hessian or Jacobian matrices. One of the most successfully used SQP variants has been proposed in [194]. It replaces the Hessian matrix $\nabla^2_X\mathcal{L}(X^{(k)},\lambda^{(k)},\mu^{(k)})$ by a symmetric approximation $A^{(k)}$. Each new Hessian approximation $A^{(k+1)}$ is obtained from the previous one by an update formula that uses the difference of the Lagrange gradients

$$\Delta\mathcal{L}^{(k)} \stackrel{\text{def}}{=} \nabla_X\mathcal{L}(X^{(k+1)},\lambda^{(k+1)},\mu^{(k+1)}) - \nabla_X\mathcal{L}(X^{k},\lambda^{(k+1)},\mu^{(k+1)}) \quad (6.18)$$

and the step $\Delta X^{(k)}$. The aim of these *variable metric* or *quasi-Newton methods* is to collect second order information in $A^{(k+1)}$ by satisfying the secant equation $A^{(k+1)}\Delta X^{(k)} = \Delta\mathcal{L}^{(k)}$. The most widely used update formula is the *Broyden-Fletcher-Goldfarb-Shanno (BFGS) update* [42, 86, 107, 214]

$$A^{(k+1)} \stackrel{\text{def}}{=} A^k + \frac{\Delta\mathcal{L}^{(k)}\left(\Delta\mathcal{L}^{(k)}\right)'}{\left(\Delta\mathcal{L}^{(k)}\right)'\Delta X^{(k)}} - \frac{\left(A^{(k)}\Delta X^{(k)}\right)\left(A^{(k)}\Delta X^{(k)}\right)'}{\left(\Delta X^{(k)}\right)'A^{(k)}\Delta X^{(k)}}\,. \quad (6.19)$$

When starting with a symmetric, positive definite approximation $A^{(0)}$, it ensures that the approximations remain symmetric and positive definite as long as $\left(\Delta\mathcal{L}^{(k)}\right)'\Delta X^{(k)} > 0$ holds. Quasi-Newton methods can be shown to converge superlinearly to a local optimum under mild conditions [178], and had a tremendous impact in the field of nonlinear optimisation.

The *constrained* (or *generalised*) *Gauss-Newton method* is another particularly successful SQP variant, which is also based on approximations of the Hessian matrix. It is applicable when the objective function is a sum of squares:

$$F(X) \;=\; \frac{1}{2}\,\|V(X)\|_2^2 \quad (6.20)$$

as in this case it is reasonable to approximate the Hessian matrix by

$$A^{(k)} \stackrel{\text{def}}{=} \nabla V(X^{(k)}) \nabla V(X^{(k)})'. \tag{6.21}$$

The corresponding QP objective is easily seen to be

$$\frac{1}{2} \left\| V(X^{(k)}) + \nabla V(X^{(k)})'(X - X^{(k)}) \right\|_2^2 . \tag{6.22}$$

The constrained Gauss-Newton method has been developed in [33, 34] and converges linearly towards the solution, which means that the contraction rate of the error between two successive iterates is asymptotically bounded by a positive constant smaller than one. Though this is worse than the superlinear rate of Quasi-Newton methods, the error contracts at a high rate whenever the residual norm $\|V(X^{\text{opt}})\|$ or the second derivatives of the problem-defining functions V, G, H are small [33].

All SQP variants mentioned so far are based on exact first-order derivatives $\nabla G(X^{(k)})$ and $\nabla H(X^{(k)})$ of the constraint functions. But also variants that approximate these Jacobian matrices have been proposed [118, 35]. This can be advantageous if evaluating G or H is computationally expensive; for example in shooting methods for solving optimal control problems, where evaluating G requires the integration of a dynamic system.

SQP methods have been implemented for general NLPs in the proprietary FORTRAN packages `NPSOL` [96] and `SNOPT` [100]. The proprietary C/C++ package `MUSCOD-II` [157, 68] implements several SQP variants tailored to optimal control problems.

6.2.4 Interior-Point Methods

An alternative Newton-type variant to address the solution of the KKT system (6.12) is to replace the last nonsmooth equality (6.12e) by a smooth nonlinear approximation:

$$\begin{aligned} \nabla_X \mathcal{L}(X^{\text{opt}}, \lambda^{\text{opt}}, \mu^{\text{opt}}) &= 0\,, && \text{(6.23a)} \\ G(X^{\text{opt}}) &= 0\,, && \text{(6.23b)} \\ H_i(X^{\text{opt}})\, \mu_i^{\text{opt}} &= \kappa \quad \forall\, i \in \{1, \ldots, n_{\text{H}}\}\,, && \text{(6.23c)} \end{aligned}$$

with a suitable $\kappa > 0$. This system is then solved with Newton's method and satisfaction of the two remaining KKT inequality conditions is ensured by choosing

suitable step lengths $\alpha^{(k)}$. Analogue to the QP case (see Subsection 2.3.2), the obtained solution is not a solution to the original problem, but to the problem

$$\min_{X\in\mathbb{R}^n} \quad F(X) - \kappa \sum_{i=1}^{n_{\mathrm{H}}} \log\left(-H_i(X)\right) \tag{6.24a}$$

$$\text{s.t.} \quad G(X) \;=\; 0\,. \tag{6.24b}$$

Thus, the solution lies in the interior of the set described by the inequality constraints and approaches the true solution as κ is reduced. The crucial feature of these *interior-point methods* is the fact that, once a solution for a given κ is found, the parameter κ can be reduced by a constant factor without jeopardising convergence of Newton's method. After only a limited number of Newton iterations a reasonably accurate solution of the original NLP is obtained. We refer to the excellent textbooks [250, 40] for further details.

It is interesting to note that the linearisation of the smoothed KKT system (6.23)—after elimination of the variable $\mu^{(k+1)}$—constitutes a linear system that is equivalent to the KKT conditions of an equality constrained QP. Thus, most structure exploiting features of SQP methods also have an equivalent in IP methods.

A widely used implementation of nonlinear IP methods is the open-source code IPOPT [235, 236] originally written in FORTRAN and later ported to C/C++. Further implementations are the proprietary package `KNITRO` [45], the C packages `LOQO` [234] and the FORTRAN package `GALAHAD` [111].

6.3 Numerical Optimal Control

In order to yield efficient implementations of optimal control algorithms, Newton-type optimisation methods need to take advantage of the special structure of optimal control problems (1.1). In all Newton-type optimisation routines there are two crucial and often costly computational steps, namely the solution of linearised subproblems and the computation of derivatives. Depending on which direct method is employed to transform the optimal control problem into an NLP, different approaches to exploit their specific structure exist. This is particularly important for simultaneous direct methods as they result in larger-scale NLPs in the variables s_i and q_i (see Subsection 6.1.2). Our presentation will mainly consider multiple shooting discretisations, while we refer to [23, 24] for a detailed description of NLPs resulting from using collocation.

6.3.1 Solving Linearised Subproblems

Let us consider a multiple shooting discretisation of the optimal control problem (1.2). This leads to a parametric nonlinear program NLP(w) of the form (6.6) that depends linearly on the initial value $w = w_0$. By introducing artificial initial values s_i and the choice of a control parameterisation q_i with local support as suggested in [38], one obtains a discretised objective function that on each multiple shooting interval j depends on the local variables s_j and q_j only. Therefore, its Hessian is a block-diagonal matrix. Very similar observations apply to problem discretisations based on collocation as it also uses local variables s_i and q_i to parameterise the state and control input trajectory.

At each iteration of a Newton-type method, a linearisation of problem (6.6) needs to be solved. As discussed in Section 6.2, this subproblem can be interpreted as a QP no matter whether an SQP or an IP method is used. It turns out that due to the dynamic system structure the Hessian of the Lagrange function has the same separable structure as the Hessian of the original objective function. Therefore, the QP objective can still be written as a sum of linear-quadratic stage costs. We write down the QP subproblem for an SQP method, where the iteration index k is left out for notational simplicity:

$$\min_{s,\,q} \quad \sum_{i=0}^{n_\mathrm{p}-1} \psi_i^{\mathrm{QP}}(s_i, q_i) \;+\; \phi^{\mathrm{QP}}\left(s_{n_\mathrm{p}}\right) \tag{6.25a}$$

$$\text{s.t.} \quad s_0 \;=\; w_0\,, \tag{6.25b}$$

$$s_{i+1} \;=\; F_i + F_{s_i} s_i + F_{q_i} q_i \qquad \forall\, i \in \{1, \ldots, n_\mathrm{p}\}\,, \tag{6.25c}$$

$$0 \;\geq\; C_i + C_{s_i} s_i + C_{q_i} q_i \qquad \forall\, i \in \{1, \ldots, n_\mathrm{p}\}\,, \tag{6.25d}$$

with suitable vectors F_i, C_i and Jacobians F_{s_i}, F_{q_i}, C_{s_i}, C_{q_i} (see the description in Subsection 7.2.1 for more details). The summands of the objective each are linear-quadratic.

The fact that the Hessian matrix of the discretised optimal control problem is block-diagonal does not only allow one to write down the QP objective in a separable form and to exploit this sparsity in the linear algebra. When quasi-Newton Hessian updates are used, it also allows one to perform *partitioned variable metric* or *high rank updates* by updating all Hessian blocks separately [117, 38].

The above linearised QP subproblem (6.25) is basically a discrete-time linear optimal control problem (1.4), which in turn can be interpreted as a parametric quadratic program QP(w_0). Thus, for solving it efficiently, the considerations of

Subsection 2.1.4 apply. Following the discussion in [65], we briefly recall the two main solution variants and relate them to approaches proposed in the literature.

The first one is *state-elimination* or *condensing* and is based on the fact that all state variables s_i, $1 \leq i \leq n_\mathrm{p}$, can be uniquely expressed by s_0 and the optimisation variables q_i corresponding to the control inputs. Thus, the states can be eliminated from the linearised problem resulting in a smaller, but dense QP [38]. It is interesting to note that a multiple shooting discretisation used in combination with condensing leads to QP subproblems of exactly the same dimension as a single shooting discretisation[4]. The cost of solving these subproblems with a dense QP solver grows with $\mathcal{O}(n_\mathrm{p}^3 n_\mathrm{u}^3)$, i.e. cubically with the horizon length n_p. This becomes inefficient whenever n_p is large, but allows one to employ general-purpose QP solvers or the online QP solver `qpOASES` described in Chapter 4.

Instead of condensing the linearised problem, one can opt to keep the constraints (6.25b) and (6.25c) and the variables s_i as unknowns in the QP. The corresponding KKT optimality conditions (2.8) can be shown to have almost block diagonal structure, allowing to factorise this linear system very efficiently. For doing so, the use of a discrete-time *Riccati recursion* has been proposed both within an active-set [106] and within an interior-point framework [220, 196]. The cost of this factorisation is $\mathcal{O}(n_\mathrm{p}(n_\mathrm{x} + n_\mathrm{u})^3)$, i.e. it grows only linearly with the horizon length n_p. Alternatively, the same complexity can also be achieved by using a direct sparse solver, as done in the general-purpose NLP solver IPOPT [235, 236].

6.3.2 Derivative Computation

Looking at the QP subproblem (6.17), we see that at least first-order derivatives of the functions F, G and H need to be computed at each SQP iteration. The same also holds for interior-point methods, where a similar equality constrained QP needs to be solved at each iteration. If an exact Hessian scheme is used (which is particularly common for IP methods), even second-order derivatives are needed. As function evaluations always form the basis of computing corresponding derivatives, computational effort for generating derivatives is proportional with the effort of a single function evaluation. While evaluating the objective function F or the path constraints H is usually relatively cheap, evaluation of the dynamic equations gathered in G is typically more expensive. This is especially true for shooting methods, where evaluating G requires a system simulation.

Several standard techniques for computing derivatives exist. A first group of methods calculates derivatives analytically, which might either be done manually

[4]However, when controlling systems with unstable or highly nonlinear dynamics, the numerical conditioning of condensed QPs arising from a multiple shooting discretisation is usually better than those of QPs arising in single shooting.

or automated using a symbolic representation of the function. A related approach is *automatic differentiation* [114, 115] which generates source code for computing the derivative by transforming the source code of the respective function. Alternatively, this can also be achieved by making use of operator overloading. It comes in two main variants, the so-called *forward* and *backward mode*, respectively, but also combinations of both exist. The forward mode is more efficient if the function maps into a space whose dimension is higher than its domain (as it is typically the case for H), while the backward or reverse mode is more efficient in the opposite case (as it is the case for the scalar-valued function F). Both variants enjoy the common feature that evaluating the derivative of a function becomes at most five times more expensive than a single function evaluation. A popular open-source implementation working with C/C++ code is `ADOL-C` [116], which uses operator overloading.

The second group of methods approximates derivatives by numerical schemes like the straight-forward *finite-differences* approach. It requires one function evaluation for each directional derivative and the approximated derivative is computed with much less accuracy than the function itself. To overcome the latter disadvantage, *internal numerical differentiation* has been proposed to generate sensitivities of dynamic equations (i.e. derivatives of the function G) [34]. It basically derives the numerical integration scheme itself such that sensitivities are obtained with the same accuracy as the solution of the differential equation.

In the context of multiple shooting discretisations, reduction techniques to compute derivatives more efficiently have been proposed based on the following observation: if the linearised subproblems are solved employing state-elimination, computation of the full derivatives F_{s_i} and F_{q_i} in Equation (6.25c) is not necessary as only the quantities of the resulting condensed QP are required. This can save a significant amount of computation if the number of states is large compared to the number of controls [207]. A related idea for optimal control problems comprising differential-algebraic equations (DAEs) has been presented in [157].

6.3.3 Comparison of Newton-Type Optimal Control Methods

Our discussion of direct optimal control methods can be summarised by categorising these methods into different classes that depend on the way the main algorithmic steps are performed [65]:

1. *problem discretisation*: sequential or simultaneous;
2. *treatment of inequalities*: SQP or nonlinear IP method;
3. *derivative computation*: full or reduced;

4. *linear algebra of linearised subproblems*: state-elimination or direct sparsity exploitation.

Sequential methods have the advantage that each NLP iteration is based on a continuous state trajectory; in contrast, the equality constraints on the coefficients s_i and q_i imposed by simultaneous methods are usually only satisfied at the solution and the corresponding state trajectories thus have no direct physical interpretation at intermediate iterates. On the other hand, simultaneous approaches allow one to specify initial guesses not only for the optimal control input trajectory (which are usually difficult to obtain) but also for the optimal state trajectory (for which often good estimates exist). Simultaneous methods are also much better suited to optimise highly nonlinear or unstable systems as, roughly speaking, the nonlinearity is equally distributed over the nodes. Moreover, they often show faster local convergence of the Newton iterations.

Shooting methods can employ state-of-the-art integrators with an adaptive choice of the step-size for simulating the system (e.g. `DAESOL-II` [3] or the `SUNDIALS` suite [126]), while collocation methods correspond to implicit Runge-Kutta methods. As sequential methods directly lead to smaller-sized NLPs, they can be easily combined with general-purpose NLP/QP solvers. This is contrast to simultaneous approaches, which lead to larger NLPs whose particular structure needs to be exploited in order to solve them efficiently. However, simultaneous approaches also offer better possibilities to parallelise computations on multi-core processors: while collocation methods allow for parallelisation of the sparse linear algebra within the NLP solver, integration of the state trajectory and corresponding derivative computation can be performed independently on each interval, and thus in parallel, when employing a multiple shooting discretisation.

Single shooting methods can be used in combination with both SQP methods (see [205, 224]) and nonlinear IP methods. The same holds true for multiple shooting methods, however, they are typically used together with SQP methods, where the underlying QP subproblems might be solved using an active-set or an interior-point QP solver. As historically most active-set QP solvers were dense, active-set SQP methods were used in combination with state elimination [38, 207], while SQP methods using interior-point QP solvers were proposed to exploit the sparsity of the linearised subproblems directly [220]. Collocation methods were initially also used in combination with active-set SQP methods [227, 25, 22]. However, the fact that they lead to very large and sparse KKT systems makes them most suited for IP methods based on a sparse linear algebra (like the approach proposed in [235] based on IPOPT).

6.4 Algorithms Tailored to Nonlinear MPC

So far all described methods were designed for (offline) optimal control. This section discusses suitable modifications to tailor these methods to nonlinear MPC and surveys a number of NMPC algorithms proposed in the literature (closely following the presentation given in [65]).

6.4.1 Online Initialisations

NMPC requires the solution of a sequence of "neighbouring" nonlinear programs NLP(w_0) that only differ in the initial state w_0. For exploiting this fact, one can initialise subsequent problems based on previous solution information. A first and obvious way is based on the principle of optimality of subarcs as mentioned in Subsection 6.1.1: Let us assume that we have computed an optimal solution $\left(s_0^{\mathrm{opt}}, q_0^{\mathrm{opt}}, s_1^{\mathrm{opt}}, q_1^{\mathrm{opt}}, \ldots, s_{n_\mathrm{p}}^{\mathrm{opt}}\right)$ of the NMPC problem (6.25) starting with initial value w_0. If we consider a shortened NMPC problem without the first interval starting with the initial value $w_0^{\mathrm{new}} \stackrel{\mathrm{def}}{=} s_1^{\mathrm{opt}}$, then for this shortened problem the vector $\left(s_1^{\mathrm{opt}}, q_1^{\mathrm{opt}}, \ldots, s_{n_\mathrm{p}}^{\mathrm{opt}}\right)$ is the optimal solution. Based on the expectation that the measured initial value w_0^{new} for the shortened NMPC problem does not differ much from s_1^{opt}, this *shrinking horizon initialisation* is canonical and it is used in MPC of batch or finite time processes (see e.g. [123, 63]).

However, in the case of moving (finite) horizon problems, the principle of optimality is not strictly applicable and we have to think about how to initialise the appended new variables q_{n_p}, $s_{n_\mathrm{p}+1}$. Often, they are obtained by setting $q_{n_\mathrm{p}} \stackrel{\mathrm{def}}{=} q_{n_\mathrm{p}-1}$ or setting q_{n_p} to the steady-state control. The state $s_{n_\mathrm{p}+1}$ is then obtained by forward simulation. This transformation of the variables from one problem to the next is called *shift initialisation*. It is often used in practice (see e.g. [160, 26, 173, 69]), though the shifted solution is not optimal for the new, undisturbed problem. However, in the case of long horizon lengths n_p we can expect the shifted solution to be a good initial guess for the new solution. Moreover, for most NMPC schemes with stability guarantee (for an overview see e.g. [171]) there exists a canonical choice of u_{n_p} that implies at least feasibility of the shifted solution for the new, undisturbed problem.

A comparison of shifted and non-shifted initialisations was performed in [36] with the result that for autonomous NMPC problems that regulate a system to steady-state, there is usually no advantage of a shift initialisation compared to the simple warm-start initialisation that leaves the variables at the previous solution. In the case of short horizon lengths it turns out to be even advantageous not to shift the previous solution, as subsequent solutions are less dominated by the initial values than by the terminal conditions. On the other hand, shift initialisation is a crucial

prerequisite in periodic tracking applications [69] and whenever the system or cost function are not autonomous.

6.4.2 Parametric Sensitivities and Tangential Predictors

In the shift initialisation discussed above we did assume that the new initial value equals the model prediction. This is of course never the case, as exactly the presence of unknown disturbances is the main motivation to use feedback control strategies like MPC. Thus, unpredictable changes in the initial value w_0 are the most important change from one parametric optimisation problem NLP(w_0) to the next one.

It is possible to use the concept of parametric NLP sensitivities to construct a new initial guess. To illustrate the idea, let us first consider the parameterised root finding problem $C(W; w) = 0$. In the NMPC context, this problem depends on the uncertain initial value $w = w_0$ and we denote its solution manifold by $W^{\mathrm{opt}}(w)$. It is a well-known fact from parametric optimisation that the solution manifold is smooth (if bifurcations are excluded). Moreover, if the root finding problem is extended by inequality constraints to cover the (parameterised) KKT optimality conditions (6.12), non-differentiable points occur whenever the optimal active-set changes [119].

As IP methods solve the approximate (parameterised) KKT equality conditions (6.23), the corresponding solution manifold $W^{\mathrm{opt}}(w)$ is smooth. Thus, if a solution $W^{\mathrm{opt}}(w_0)$ for a previous initial value w_0 has been found, a good solution predictor for a new initial value w_0^{new} is provided by

$$\widehat{W^{\mathrm{opt}}}(w_0^{\mathrm{new}}) \stackrel{\mathrm{def}}{=} W^{\mathrm{opt}}(w_0) - \frac{dW^{\mathrm{opt}}}{dw_0}(w_0)\,, \tag{6.26}$$

where

$$\frac{dW^{\mathrm{opt}}}{dw_0}(w_0) = \left(\frac{\partial C}{\partial W}(W^{\mathrm{opt}}(w_0); w_0)\right)^{-1} \frac{\partial C}{\partial w_0}(W^{\mathrm{opt}}(w_0); w_0)\,(w_0^{\mathrm{new}} - w_0) \tag{6.27}$$

is given by the implicit function theorem. Moreover, an important practical observation is that an approximate tangential predictor can also be obtained when it is computed at a point W^* that does not exactly lie on the solution manifold. This more general predictor is given by the formula

$$\widehat{W^{\mathrm{opt}}}(w_0^{\mathrm{new}}) \stackrel{\mathrm{def}}{=} W^* - \left(\frac{\partial C}{\partial W}(W^*; w_0)\right)^{-1} \left[\frac{\partial C}{\partial w_0}(W^*; w_0)\,(w_0^{\mathrm{new}} - w_0) + C(W^*; w_0)\right]. \tag{6.28}$$

This fact, that is illustrated in Figure 6.1(a), leads to a combination of a predictor and corrector step in one linear system. If $C(W^*; w_0) = 0$ holds, the

formula simplifies to the tangential predictor of the implicit function theorem. Unfortunately, the IP solution manifold is strongly nonlinear at points where the optimal active-set changes. Therefore, the tangential predictor is not a good approximation when we linearise at such points, as visualised in Figure 6.1(b). One remedy would be to increase the path parameter κ, which decreases the nonlinearity, but comes at the expense of generally less accurate IP solutions. This is illustrated in Figure 6.2 for the same two linearisation points as before. In Figure 6.2(b) we see that the tangent is approximating the IP solution manifold well in a larger area around the linearisation point, but that the IP solution itself is more distant to the true NLP solution.

SQP methods directly tackle the exact (parameterised) KKT optimality conditions. Thus, at points with weakly active constraints, we have to consider directional derivatives of the solution manifold, or *generalised tangential predictors*. These can be computed by suitable QPs [119] and are visualised in Figure 6.3(b). The theoretical results can be made a practical algorithm by the following procedure proposed in [62]: first, we have to make sure that the parameter w_0 enters the NLP linearly, which is automatically the case for simultaneous discretisations of the optimal control problem (see Equation (6.25b)). Second, we address the problem with an exact Hessian SQP method. Third, we just take our current solution guess $W^*(w_0)$, and then solve the condensed QP subproblem for the new parameter value w_0^{new}, but initialised at $W^*(w_0)$. It can be shown that this *initial value embedding* procedure delivers exactly the generalised tangential predictor when started at a solution $W^{\text{opt}}(w_0)$ [62], as in Figure 6.3(b). The predictor becomes approximately tangential whenever we do not start on the solution manifold (see Figure 6.3(a)) or we do not use an exact Hessian or Jacobian matrix or we do not evaluate the Lagrange gradient or constraint residuals exactly.

6.4.3 Ideas to Reduce the Feedback Delay

When using NMPC, each solution to a new optimal control problem should ideally be available instantly, which is impossible due to computational delays. Closely following the presentation in [65], we summarise several ideas that have been proposed to reduce this *feedback delay*:

- *Offline pre-computations*: As consecutive NMPC problems are similar, some computations might be done once and for all before the controller starts. In the extreme case, this leads to an explicit pre-computation of the NMPC control law (extending the explicit MPC approaches discussed in Section 2.2), or a solution of the Hamilton-Jacobi-Bellman equation. Both approaches are usually prohibitive for higher state dimensions, say above 6–8 [174, 48]. But also when online optimisation is used, it is sometimes

possible to pre-compute and factorise Hessians or even Jacobians in Newton-type optimisation methods, in particular in the case of neighbouring feedback control along reference trajectories [150, 44].

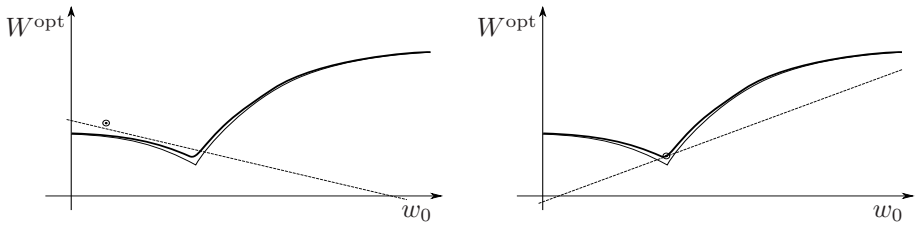
- *Delay compensation by prediction*: When it is known how long solving an NMPC problem will take, it is a good idea not to address a problem starting at the current state but to simulate at which state the system will be when the problem will have been solved. This can be done using the NMPC system model and the open-loop control inputs that are also applied in the meantime [85]. This feature is used in many practical NMPC schemes with non-negligible computation time. Alternatively, computational delays might be interpreted as actuator delays and modelled within the dynamic system of the optimiser.

- *Division into preparation and feedback phase*: Another trick used in several NMPC algorithms is to divide the computations in each sampling time into a preparation phase and a feedback phase following [64]. The more CPU intensive preparation phase is performed with an old predicted state w_0 before the new state estimate w_0^{new} is available, while the feedback phase then delivers quickly an approximate solution to the optimisation problem corresponding to w_0^{new}. This approximation is often based on a tangential predictor as discussed in Subsection 6.4.2.

- *Iterating while the problem changes*: An important ingredient of some NMPC algorithms is the idea to solve the NLP problem while it changes. Instead of performing Newton-type iterations till convergence for an NMPC problem getting older and older during the iterations, the most current information is used in each new iteration. This idea is used in [160, 64, 180].

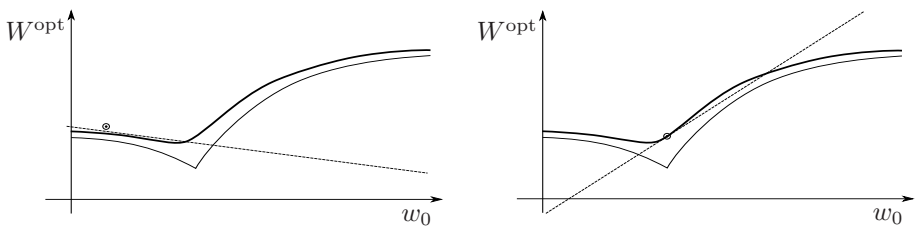
6.4.4 Sequential Approaches

One of the first true online algorithms for NMPC has been proposed about two decades ago in [159]. This *Newton-Type Controller* is based on a sequential direct method and uses an SQP-type procedure with Gauss-Newton Hessian approximation and a line search globalisation. It uses a shift initialisation and performs only one SQP iteration at each sampling instant. The result of each SQP iterate is used to give an approximately optimal feedback to the real process. Closed-loop stability was proven for open-loop stable processes [160].



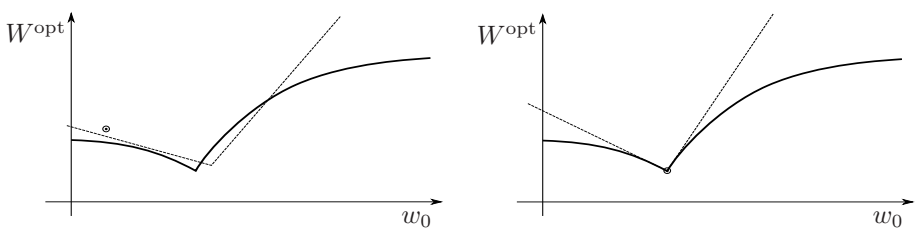
(a) Linearising at approximate solution. (b) Linearising at active-set change.

Figure 6.1: Qualitative illustration of tangential predictors (dashed) for interior-point methods using a small κ (based on [65]). The IP solution manifold (thick line) approximates the exact solution manifold (thin line) very well.



(a) Linearising at approximate solution. (b) Linearising at active-set change.

Figure 6.2: Qualitative illustration of tangential predictors (dashed) for interior-point methods using a larger κ (based on [65]). The IP solution manifold (thick line) approximates the exact solution manifold (thin line) less accurate.



(a) Linearising at approximate solution. (b) Linearising at active-set change.

Figure 6.3: Qualitative illustration of generalised tangential predictors (dashed) for exact Hessian SQP methods (based on [65]). By treating the KKT optimality conditions exactly, the optimal solution manifold (thick line) is recovered.

Similar to the Newton-Type Controller, the *Continuation/GMRES method* as proposed in [180, 210] performs only one Newton-type iteration at each sampling instant and is based on a sequential formulation. However, instead of solving linearised subproblems as discussed in Subsection 6.3.1, it iterates on a finite difference approximation of the Euler-Lagrange differential equation (6.2). Thus, it might also be classified as an indirect optimal control method, but it works with a piecewise constant control parameterisation. In order to treat inequality constraints it uses the following IP-like formulation for a fixed κ:

$$\min_{X\in\mathbb{R}^n,\, Y\in\mathbb{R}^{n_\mathrm{H}}} \quad F(X) - \kappa \sum_{i=1}^{n_\mathrm{H}} Y_i \tag{6.29a}$$

$$\text{s.t.} \qquad G(X) \;=\; 0\,, \tag{6.29b}$$

$$H_i(X) + Y_i^2 \;=\; 0 \quad \forall\, i \in \{1, \ldots, n_\mathrm{H}\}\,. \tag{6.29c}$$

The Continuation/GMRES method uses an exact Hessian matrix and an iterative GMRES method [144] for solving only one linear system at each sampling instant. As noted in [180], this can be interpreted as performing a Newton-type iteration with a tangential predictor (6.28) (without shifting). This single combined predictor-corrector step at each sampling instant seems to be sufficient to follow the nonlinear IP solution manifold well, as illustrated in Figure 6.4. A variant of the method using a simultaneous approach and condensing is proposed in [215], which shows improved accuracy and lower computational cost in each Newton-type iteration.

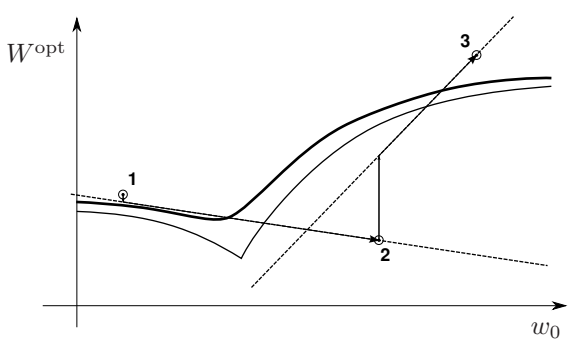


Figure 6.4: Subsequent solution approximations of the continuation/GMRES method (based on [65]); formatting as in Figure 6.2.

An interesting cross-over method combining typical features of sequential and simultaneous methods has been proposed in [224]. It is based on a simultaneous formulation within an SQP-type framework, but perturbs each SQP iterate in order to make the state trajectory continuous. This is done by a certain open-loop or closed-loop forward simulation of the dynamic system given the new iterate of the control inputs. In the open-loop variant, this nearly results in a sequential approach, but it allows one to exploit the same sparsity structure as a simultaneous approach as discussed in Subsection 6.3.1.

6.4.5 Simultaneous Approaches

Like the Newton-Type Controller, the *Real-Time Iteration (RTI) scheme* presented in [62, 64] performs one SQP-type iteration with Gauss-Newton Hessian per sampling instant. However, it employs a simultaneous NLP parameterisation using direct multiple shooting with full derivatives and state-elimination. Moreover, it uses the generalised tangential predictor resulting from an *initial value embedding* as discussed in Subsection 6.4.2 to correct for the mismatch between the expected and the actual state. In contrast to the Continuation/GMRES method, where the predictor is based on one linear system solve, the Real-Time Iteration scheme solves an inequality constrained QP. Thus, the tangential predictor even works across changes of the optimal active-set (see Figure 6.5). Thus, roughly spoken, the RTI scheme works well when the optimal active-set changes faster than the matrices of the linearised system. The computations of each iteration are divided into a long *preparation phase*, in which the system linearisation and state-elimination are performed, and a much shorter *feedback phase* that solves just one condensed QP. Depending on the application, the feedback phase can be several orders of magnitude shorter than the feedback phase. Note that only one system linearisation and one QP solution are performed in each sampling time. The QP corresponds to a linear MPC feedback along a time-varying trajectory and the RTI scheme gives the same feedback as linear MPC in the limiting case of a linear system model. Error bounds and closed-loop stability of the RTI scheme have been established for shrinking horizon problems in [63] and for NMPC with shifted and non-shifted initialisations in [67] and [66], respectively.

Multi-level RTI schemes have been proposed in [35, 147] to further reduce the computations performed in each sampling time. For doing so, the multiple shooting based SQP-type iterations are divided into hierarchical levels: At the lowest level A, only one condensed QP is solved for the most current initial value w_0. This provides a form of linear MPC at the base level, taking at least active-set changes into account at a very high sampling frequency. On the next two intermediate levels, only the nonlinear constraint residuals are evaluated (level B) for improving feasibility similar to [44], and the Lagrange gradient is evaluated (level C) for improving optimality based on an adjoint-based inexact SQP method.

At all these three levels A–C, no new QP matrices are computed and even system factorisations can be reused again and again. This also makes is possible to hot-start QP solution based on the online active set strategy [247]. Note that Level C iterations are still considerably cheaper than one full SQP iteration [246]; but also for them optimality and NMPC closed-loop stability can be guaranteed by the results in [66] as long as the system matrices are accurate enough to guarantee contraction of the Newton-type iterations. Only if this is not the case anymore, an iteration on the highest level D has to be performed, which includes a full system linearisation and is as costly as a usual Newton-type iteration.

In order to avoid the convergence issues of predictor-corrector path-following interior-point methods, the *Advanced Step Controller* makes a more conservative choice [255, 254]: in each sampling time, a complete IP procedure is iterated to convergence (with $\kappa \to 0$). In this respect, it behaves like an offline optimal control algorithm using a simultaneous method (with exact Hessians, full derivatives and sparse linear algebra). However, two features qualify it as an online algorithm: first, it takes computational delay into account by solving an "advanced" problem with the expected initial value $\widehat{w_0}^{\text{new}}$ in a preparation phase (similar to real-time iterations with shift). Second, during the feedback phase, it applies the obtained solution not directly, but computes first a tangential predictor which corrects for the differences between the expected state $\widehat{w_0}^{\text{new}}$ and the actual state w_0^{new}, as described in Equation (6.26). We illustrate the behaviour of the Advanced Step Controller in Figure 6.6, where the tiny arrows indicate that several Newton iterations are performed during the preparation phase. The very short feedback phase computes the tangential predictor by only one linear system solve based on the factorisation of the last Newton iteration's KKT matrix. As the IP predictor does not predict optimal active-set changes accurately, one can say that the Advanced Step Controller gives priority to the nonlinearities of the system

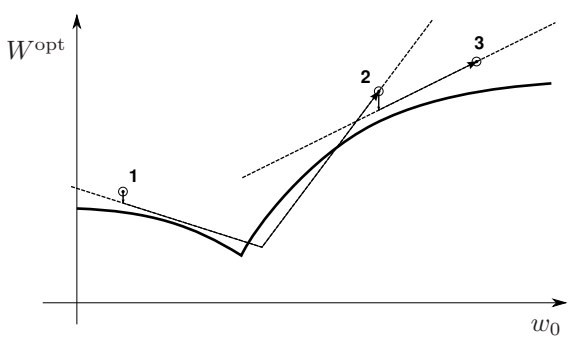


Figure 6.5: Subsequent solution approximations of the real-time iteration scheme (based on [65]); formatting as in Figure 6.3.

over the ones introduced by changes in the optimal active-set. As it solves each problem exactly, classical NMPC stability theory [171] can be extended to this scheme [254].

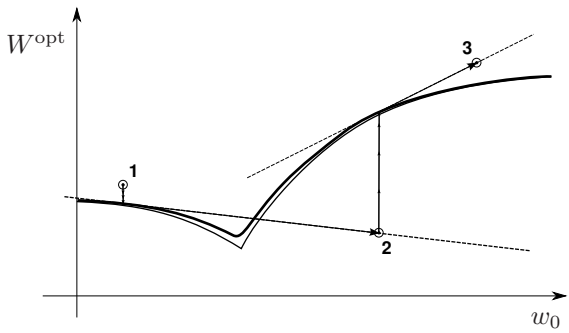


Figure 6.6: Subsequent solution approximations of the advanced step controller (based on [65]); formatting as in Figure 6.1.

Chapter 7

ACADO Toolkit

This chapter describes the open-source package `ACADO Toolkit`, a new software environment and algorithm collection for automatic control and dynamic optimisation [127, 128]. It provides a general framework for using a number of algorithms for direct optimal control and is implemented as self-contained C++ code. One distinguishing feature is its ability to handle symbolic expressions, which not only provides interesting algorithmic possibilities but also allows for a user-friendly syntax to setup optimisation problems. We focus on the implementation of previously published algorithms for nonlinear model predictive control [82], which are the first main contribution of the second part of this thesis[1]. Computational performance of these algorithms is illustrated by controlling the start-up of a continuous stirred tank reactor model.

7.1 Overview of the Software Package

We motivate the development of the `ACADO Toolkit`, summarise its scope and elaborate on its algorithmic features. Moreover, its software design is outlined. Our presentation closely follows the description given in [82, 128].

[1]The `ACADO Toolkit` has been jointly coded with Boris Houska, who is main author of the symbolic expressions, differentiation routines, numerical integrators, and routines for parameter estimation and robust optimisation. Moreover, Filip Logist has contributed to the multi-objective optimisation and David Ariens wrote a flexible interface for MATLAB.

7.1.1 Introduction and Scope

While linear MPC generally requires a dedicated QP solver, a couple of additional algorithmic routines are required to tackle nonlinear MPC problems as outlined in Chapter 6 for direct methods. When searching the literature, a number of optimisation software packages for solving nonlinear optimal control problems (1.2) can be found. In order to motivate the development of the `ACADO Toolkit`, we briefly discuss some frequently used packages:

- A very popular interior-point algorithm for optimising large-scale differential algebraic systems is the open-source package IPOPT [235, 236], originally developed by Andreas Wächter and Larry Biegler. It can be combined with collocation methods for discretising continuous-time dynamic systems and implements a filter strategy as globalisation technique. IPOPT is written in C/C++ or FORTRAN, but uses modelling languages like `AMPL` or MATLAB to provide a more convenient user interface and to allow for automatic differentiation.

- The MATLAB package `PROPT` [183] receives increasing attention. `PROPT` is a commercial software tool developed by Tomlab Optimization Inc. It solves optimal control problems based on collocation techniques, and uses existing NLP solvers such as `KNITRO` [45] or `SNOPT` [100]. The use of MATLAB syntax increases the user-friendliness of `PROPT` but restricts its use to platforms on which MATLAB is available.

- Recently, the open-source code `dsoa` has been published by Brian Fabien [74]. This package is written in C/C++ and discretises differential algebraic systems based on implicit Runge-Kutta methods [121]. Unfortunately, the package only implements a single shooting method, which is often not advisable for nonlinear optimal control problems. An SQP method is used to solve the resulting NLP.

- Another SQP-based optimal control package is `MUSCOD-II`, originally developed by Daniel Leineweber [156]. It discretises the differential algebraic systems based on backward differentiation formula (BDF) [4] or Runge-Kutta integration methods and uses direct multiple shooting [38]. `MUSCOD-II` is proprietary software written in C/C++. Its numerical algorithms seem to be more elaborated than those of `dsoa` and also support optimal control problems comprising integer-valued variables.

- Finally, the development of software packages dedicated to nonlinear MPC in the process industry has been reported in the literature: The open-source packages `OptCon` [216] and `NEWCON` [204] both employ a multiple shooting method in combination with the sparse SQP method `HQP` [90].

Each of the above packages has its particular strengths and all of them have proven successful for a specific range of applications. As they are all tailored to a certain choice of underlying numerical algorithms, it is usually problem-dependent which one is most suited. Moreover, their specialised software design renders it difficult to combine algorithmic ideas from different packages or to extend them with new mathematical concepts. Finally, most packages aim at users with a sound programming background and some of them are either not freely available or rely on proprietary external packages (like the sparse linear system solver within IPOPT).

The `ACADO Toolkit` has been designed to overcome these issues. Besides the efficiency of the implementation, four key properties have been identified that are believed to be crucial for any automatic control software based on dynamic optimisation. These properties have guided the implementation of the `ACADO Toolkit`:

1. *Open-source.* As such a software package should be freely available at least to academic users, the `ACADO Toolkit` is distributed under the GNU Lesser General Public Licence (LGPL) [134]. Releasing the software as open-source code also allows other researchers to reproduce all results easily, to check whether all steps of an algorithm have been implemented as stated and to try out own modifications.

2. *User-friendly.* The syntax to formulate optimal control problems should be as intuitive as possible. Thus, the `ACADO Toolkit` makes intensive use of the object-oriented capabilities of C++, in particular operator overloading, to allow the user to formulate control problems in a way that is very close to the usual mathematical syntax (see the example listing in Section 7.3). For experienced users this might only be a question of convenience. But given the fact that dynamic optimisation is more and more widely used in many different engineering applications, also non-experts with little programming experience should be able to formulate their control problems within a reasonable period of time. Another aspect of user-friendliness requires that the software should, as far as possible, also make consistent default choices for algorithmic settings and initialisations automatically in case they are not provided by the user.

3. *Extensible code.* A favourable feature is a software design that makes it easy to extend the package by linking existing algorithms and adding new developments while avoiding the duplication of code. The `ACADO Toolkit` realises this requirement by a careful interface design guaranteeing that almost all algorithmic parts could also be used stand-alone. This is done using well-established object-oriented software design concepts such as abstract base classes and inheritance.

4. *Self-contained.* In particular for use in model predictive controllers on embedded hardware, it is important that the software must only depend on external packages if really inevitable (see also the discussion in Subsection 5.1.2). While linking additional libraries during offline simulations is usually less of a problem, the main package should provide a mode to run stand-alone on the controller hardware if possible. The `ACADO Toolkit` is written in a completely self-contained manner and optionally allows one to use external packages for graphical output or specialised linear algebra operations (e.g. sparse solvers for linear systems).

The latter three design goals are typically conflicting with the efficiency of the implemented algorithms. Let us briefly summarise how the `ACADO Toolkit` deals with this trade-off: First, the symbolic syntax introduces a certain runtime overhead, which can become considerable if the problem formulation comprises very lengthy expressions. However, as all symbolic expressions need to be pre-processed only once in an initialisation phase, this overhead might be compensated by the ability to automatically detect and exploit structural information of the problem formulation (see Subsection 7.1.2). Second, some algorithmic overhead is introduced by the object-oriented design of the `ACADO Toolkit` (e.g. due to virtual inheritance) that keeps the package extensible. Though it is hard to quantify this effect, it is believed to be minor as executing the respective parts of a numerical algorithm is typically much more expensive than switching between different variants of them. Third, keeping the code self-contained (and open-source) can introduce a serious loss of efficiency as it prevents the use of existing highly efficient implementations of certain algorithmic components. This seems to be particularly crucial for the implementation of linear algebra routines (such as `BLAS` [31] or `LAPACK` [6]), which are at the base of any numerical algorithm. By encapsulating these operations in dedicated classes, the `ACADO Toolkit` in principle allows to replace the built-in implementation by existing linear algebra packages (as has been done with `CSPARSE` [55]). This promises significant runtime improvements for larger-scale problem formulations.

The `ACADO Toolkit` can deal with the following four problem classes:

1. *Optimal control problems* of the following form:

$$\begin{aligned}
\min_{\substack{x(\cdot),z(\cdot),\\ u(\cdot),p,T}} \quad & \int_{t_0}^{t_0+T} \psi\left(t,x(t),z(t),u(t),p\right) dt \;+\; \phi\left(x(t_0+T),z(t_0+T),p,t_0+T\right) \\
\text{s.t.} \quad \dot{x}(t) &= f\left(t,x(t),z(t),u(t),p\right) \quad \forall\, t\in[t_0,t_0+T]\,, \\
0 &= g\left(t,x(t),z(t),u(t),p\right) \quad \forall\, t\in[t_0,t_0+T]\,, \\
0 &\geq c\left(t,x(t),z(t),u(t),p\right) \quad \forall\, t\in[t_0,t_0+T]\,, \\
0 &\geq r\left(x(t_0),z(t_0),x(t_0+T),z(t_0+T),p,t_0+T\right).
\end{aligned} \tag{7.1}$$

This formulation extends the one given in Definition 1.1 by also including *algebraic states* $z : \mathbb{R} \to \mathbb{R}^{n_z}$ for dealing with differential algebraic equation (DAE) models. Moreover, the horizon length does not need to be fixed but might also be subject to optimisation.

The algorithms implemented in `ACADO Toolkit` assume that the differential and algebraic right-hand side functions f and g are sufficiently smooth. Moreover, it is assumed that the function $\frac{\partial g}{\partial z}$ is always regular, i.e. that the DAE is of index one. The remaining functions, namely the Lagrange term ψ, the Mayer term ϕ, the path constraint function c, as well as the boundary constraint function r are assumed to be at least twice continuously differentiable in all their arguments.

2. *Multi-objective optimisation and optimal control* problems, which require the simultaneous minimisation of more than one objective. These multi-objective optimisation problems typically result in a set of Pareto optimal solutions instead of one single (local) optimum. More details can be found in [163] and the references therein.

3. *Parameter and state estimation* problems, where parameters, unknown control inputs or initial states are to be identified by measuring an output of a given (nonlinear) dynamic system.

4. *Model predictive control* problems and online state estimation, where parameterised dynamic optimisation problems have to be solved repeatedly to obtain a dynamic feedback control law. We will focus on this problem class in Section 7.2.

We will now outline the main algorithmic features of the `ACADO Toolkit` and also give an overview of its software design.

7.1.2 Algorithmic Features

Symbolic Syntax

One fundamental requirement for any optimal control package is that functions such as objectives, differential equations or constraint functions can be provided by the user in a convenient manner. This is often realised by allowing the user to link C functions, for example, implementing the respective mathematical functions via a pre-defined interface. Also the `ACADO Toolkit` provides this possibility but implements a more powerful *symbolic syntax* in addition to that. The whole optimal control problem (7.1) can be formulated based on symbolic expressions that can be easily used to build up complex model equations as well as user-defined objective and constraint functions. Due to operator overloading and carefully

designed C++ classes, the syntax can be used very similar to usual C/C++ code. This is illustrated for a simple function in Listing 7.1 and for a complete optimal control problem formulation in Listing 7.2 on page 133.

The symbolic syntax of the `ACADO Toolkit` offers the following benefits:

- It allows to *automatically detect structural information* for all symbolically specified functions. For example, the implementation offers routines that check whether a function is convex or concave based on the concept of disciplined convex programming [112]. Moreover, sparsity patterns or dependencies on certain arguments might be detected.

- It provides a *user-friendly* way to formulate optimisation problems that mimics the usual mathematical syntax. It allows to give user-defined names to all occurring variables, which can greatly increase the readability of the code, in particular for small- to medium-size problem formulations. Moreover, the above-mentioned auto-detection routines allow the user to determine the structure and dimensions of all functions and intermediate values internally, taking away this effort from the user.

- The symbolic notation of functions allows the `ACADO Toolkit` to provide routines for *automatic- and symbolic differentiation* (in addition to numeric differentiation). Automatic differentiation (AD) [28, 115, 116] is implemented in its forward and adjoint mode for first and (mixed) second order derivatives. Moreover, all expressions can be differentiated symbolically returning an expression again, like AD with source code transformation. This functionality can be used recursively leading to symbolic derivatives of arbitrary order. Nevertheless, by also allowing to link user-defined C functions by means of the `Function` class, the `ACADO Toolkit` also supports existing AD packages such as `ADOL-C` [116].

- Within optimal control algorithms, the model equations are typically evaluated many times. Thus, another useful benefit of the symbolic syntax is the possibility to *speed-up function evaluation* by detecting and exploiting sparsity patters when performing matrix-vector operations. Moreover, it allows the user to define intermediate variables, like the variable `tmp` in Listing 7.1, occurring several times within a function definition that has to be evaluated only once.

- Implementing functions within the symbolic `ACADO` syntax also allows to export these functions in the form of optimised plain C code. This feature will be used and discussed in detail within the `ACADO Code Generation` tool presented in Chapter 8.

Listing 7.1: Illustration of ACADO Toolkit's symbolic syntax by defining a simple two-dimensional function.

```
int main( )
{
    DifferentialState x;
    Control           u;
    IntermediateState tmp;
    Function          f;

    tmp = 0.5*x + 1.0;

    f << exp( x ) + u;
    f << exp( tmp + exp( tmp ) );

    // ...

    return 0;
}
```

Integration Algorithms

As discussed in Chapter 6, direct single- or multiple shooting methods for optimal control require to simulate ODE (or DAE) systems. Moreover, sensitivities of the state trajectory with respect to initial values, control inputs or parameters have to be obtained efficiently. Consequently, the ACADO Toolkit comes along with state-of-the-art integration routines, namely several Runge-Kutta methods [120] and a BDF method. The BDF integrator is based on the algorithmic ideas in [10, 15, 190] and can be used for stiff differential or differential algebraic equations. It can also deal with fully implicit DAEs of index 1:

$$\forall\, t \in [t_0, t_0 + T]: \quad f(t, \dot{x}(t), x(t), z(t), u(t), p, T) = 0\,. \tag{7.2}$$

However, note that in most optimal control problems arising in practice the right-hand side f is linear in $\dot{x}$.

All integrators can compute first and second order sensitivities. The differentiation can either be based on internal numerical differentiation [34, 15] or on (internal) automatic differentiation (provided that the right-hand side function is given in ACADO syntax). It should be mentioned that the integration routines that are currently implemented within the ACADO Toolkit are similar to existing integrator packages like the SUNDIALS suite [126] or DAESOL-II [3] with respect to both the algorithmic strategies as well as performance.

Discretisation of Dynamic Systems

The ACADO Toolkit implements both a single- and a multiple shooting discretisation of the infinite-dimensional optimal control problem (7.1). For doing so, either of the above-mentioned integration methods can be used. Moreover, also a discrete-time variant of formulation (7.1) is supported (where integration becomes a simple iterative function evaluation). As a suitable software design completely decouples the state discretisation from the optimisation algorithm for solving the resulting NLP, also direct collocation schemes can be added easily.

Nonlinear Optimisation Algorithms

Discretising the optimal control problem by any direct method leads to specially structured NLPs of the form (6.7). For solving them efficiently, the ACADO Toolkit currently provides different SQP-type methods and offers interfaces to couple further SQP or IP implementations.

The implemented SQP-type methods offer different possibilities to approximate the Hessian of the underlying QP: either the exact Hessian is computed, (full or blockwise) BFGS updates are employed [194] or a Gauss-Newton approximation [33] is used. For solving the underlying QP, the ACADO Toolkit exploits the sparsity introduced by a multiple shooting discretisation via state-elimination. The resulting dense QPs are solved by means of qpOASES as presented in Chapter 4. In case an underlying QP becomes infeasible during the SQP iterations, all QP constraints are automatically relaxed using slack variables whose ℓ_1 or ℓ_2 norm is penalised in the objective function. The SQP method can either always take full-steps or employs a line search globalisation routine as described in [194, 122]. Moreover, auto-initialisation techniques are implemented to increase the reliability of the optimisation routines.

For solving parameterised NLPs arising in the nonlinear MPC context, two variants of the real-time iteration scheme [62] have been implemented. Section 7.2 discusses them in detail.

7.1.3 Software Design

The ACADO Toolkit has been designed as algorithm collection, whose generic interfaces shall allow the user to extend the package easily. This can either be done by adding functionality based on existing algorithmic building blocks or by (optionally) linking external packages. Thus, from a developers point of view, an object-oriented design providing interfaces based on (abstract) base classes was a natural choice. Moreover, the object-oriented design combined with extensive

operator overloading allowed to implement the user-friendly symbolic syntax for formulating optimal control problems. Further design goals and decisions have been discussed in Subsection 7.1.1.

For reflecting the different steps for solving optimal control problems based on direct methods, the following hierarchical algorithmic layers can be distinguished. Typically, classes at lower levels serve as members of higher-level classes:

1. *Low-level data structures*: All basic expressions of the symbolic syntax (represented by the base classes `Expression` and `Operator`) as well as the classes `Matrix`, `Vector` and `EvaluationPoint`.

2. *Function evaluations*: The base class `Function` that evaluates both symbolic functions and functions given as C code and their derivatives. Also the class `VariablesGrid` representing a discrete-time sequence of `Vector`s as well as the classes `ObjectiveElement` and `ConstraintElement` for composing the objective and constraint formulation, respectively, can be categorised here.

3. *Integration routines*: The class `Integrator` provides a base class for interfacing numerical integrators for use in shooting methods or for plain simulations. They integrate ODE/DAE systems given in form of a `DifferentialEquation` (derived from the class `Function`).

4. *Dynamic discretisation routines*: The class `DynamicDiscretization` serves as a base class for interfacing discretisation methods such as single and multiple shooting and collocation.

5. *NLP solvers*: The base class `NLPsolver` provides a generic interface for algorithms solving NLP problems, like SQP or IP methods. Moreover, it turned out to be advantageous to also introduce the base classes `BandedCPsolver` and `DenseCPsolver` for interfacing solvers for banded and dense *convex programming problems* (CPs) (a convex generalisation of QPs).

6. *High-level optimisation tools*: Classes that organise the selection of underlying algorithmic components providing numerous options to the user, such as the class `OptimizationAlgorithm` for optimal control problems and the class `RealTimeAlgorithm` for MPC problems.

Having a look at this hierarchy leads to an important design issue: It would be highly desirable that classes at intermediate levels can be used both within algorithmic building blocks at higher levels and stand-alone. In the first case, developers of new algorithms need to rely on highly efficient implementation avoiding any overhead as far as possible. The second case mainly occurs when users, for example, want to run plain simulations using one of the integrators in a convenient manner.

For resolving these potentially conflicting use cases of intermediate classes, the following design has been realised: Algorithmic building blocks are tailored to the needs of algorithm developers and are implemented without dedicated user interface. The most important ones are depicted in Figure 7.1, which also shows that they are derived from the common abstract base class `AlgorithmicBase` to collect common functionality. For example, we find back the above-mentioned classes `Integrator`, `DynamicDiscretization` and `NLPsolver`. On the other hand, additional user interface classes have been introduced to provide conveniently usable stand-alone versions of the respective algorithmic building block, which they hold as a private member. All user interface classes are derived from the common base class `UserInteraction` which provides generic functionality to log and plot algorithmic information and to handle user-defined options (see Figure 7.2). Note that the current design does not make the distinction between developer and user class for the top-level class `OptimizationAlgorithm` and that no separate user interfaces for solving general NLPs or performing an arbitrary dynamic discretisation are provided yet. The class `SimulationBlock` will be discussed later in Subsection 7.2.4.

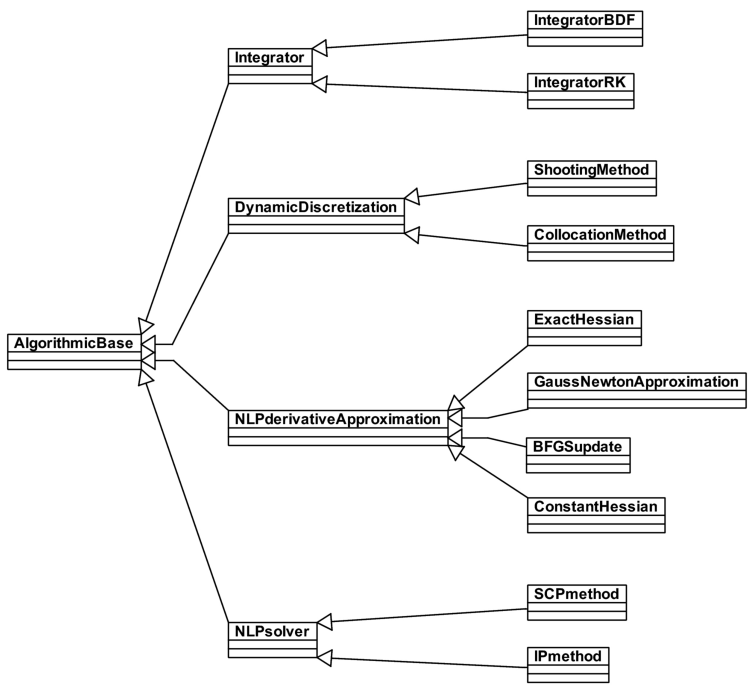


Figure 7.1: UML class diagram illustrating the most important algorithmic building blocks of the `ACADO Toolkit`.

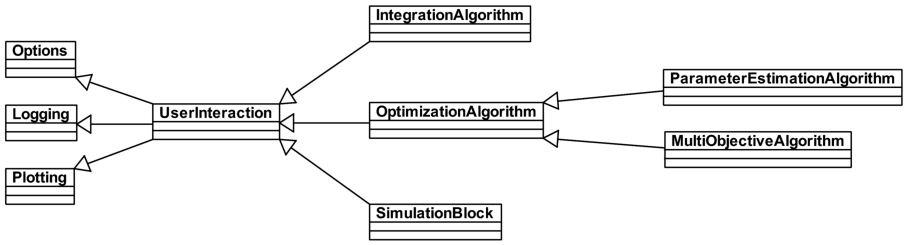


Figure 7.2: UML class diagram illustrating a couple of user interfaces of the ACADO Toolkit.

7.2 MPC Algorithms

This section outlines two algorithmic variants of the real-time iteration scheme [62] that have been implemented within the ACADO Toolkit. Moreover, its built-in simulation environment for performing realistic closed-loop simulations is introduced.

7.2.1 Generalised Gauss-Newton Method

The ACADO Toolkit implements a generalised Gauss-Newton method [33, 34] that turned out to be suited for optimal control problems (1.2) comprising a so-called *tracking objective function* of the form:

$$\int_{t_0}^{t_0+t_\mathrm{p}} \hat{y}(t)'Q\hat{y}(t) + \hat{u}(t)'R\hat{u}(t)\,dt \;+\; \hat{x}(t_0+t_\mathrm{p})'P\hat{x}(t_0+t_\mathrm{p})\,, \tag{7.3}$$

where all quantities are defined as in the linear case in Definition 1.3. Single and multiple shooting discretisations are implemented to transform the problem into a (parametric) NLP. We will describe the real-time iteration algorithms based on a multiple shooting method.

The shooting methods as implemented in the ACADO Toolkit, introduce a piecewise constant control parameterisation

$$u_i^{\mathrm{appr}}(\tau_i + t; q_i) \;\stackrel{\mathrm{def}}{=}\; q_i \quad \forall\, t \in [0, \tau_{i+1} - \tau_i) \;\; \forall\, i \in \{0, \ldots, n_\mathrm{p} - 1\}\,, \tag{7.4a}$$

$$u_i^{\mathrm{appr}}(t_0 + t_\mathrm{p}; q_i) \;\stackrel{\mathrm{def}}{=}\; q_{n_\mathrm{p}-1}\,, \tag{7.4b}$$

with $q_i \in \mathbb{R}^{n_\mathrm{u}}$ for all i, $0 \leq i \leq n_\mathrm{p}$, on an arbitrary but fixed time grid

$$t_0 = \tau_0 < \tau_1 < \ldots < \tau_{n_\mathrm{p}} = t_0 + t_\mathrm{p} \,. \tag{7.5}$$

Introducing piecewise constant control inputs is very common in the MPC context, though [186] elaborates on the advantage of piecewise linear parameterisations. Continuous piecewise linear control inputs could be formulated within the `ACADO Toolkit` by introducing additional auxiliary states.

The multiple shooting method, as introduced in Subsection 6.1.2, looks on each interval $i \in \{0, \ldots, n_\mathrm{p} - 1\}$ for a solution $x_i^{\mathrm{appr}}(t; s_i, q_i)$ to the initial value problem

$$\begin{aligned} x_i^{\mathrm{appr}}(\tau_i; s_i, q_i) &= s_i \,, & \text{(7.6a)} \\ \dot{x}_i^{\mathrm{appr}}(t; s_i, q_i) &= f\left(x_i^{\mathrm{appr}}(t; s_i, q_i), q_i\right) \quad \forall\, t \in [\tau_i, \tau_{i+1}] \,, & \text{(7.6b)} \end{aligned}$$

with $s_i \in \mathbb{R}^{n_\mathrm{x}}$ for all $i \in \{0, \ldots, n_\mathrm{p}\}$. The solution $x_i^{\mathrm{appr}}(\tau_{i+1}; s_i, q_i)$ at each multiple shooting node is evaluated numerically by using a Runge-Kutta or BDF integrator. As common for shooting methods, the `ACADO Toolkit` employs numerical integrators with adaptive step-size control in order to ensure a given integration accuracy.

Based on this discretisation, the tracking objective function (7.3) is evaluated at the multiple shooting nodes only and is thus approximated by

$$\begin{aligned} \|V(s_0, X, U)\|_2^2 \stackrel{\mathrm{def}}{=} \sum_{i=0}^{n_\mathrm{p}-1} & \Big[\big(h(s_i, q_i) - y^{\mathrm{ref}}(\tau_i)\big)' Q \left(h(s_i, q_i) - y^{\mathrm{ref}}(\tau_i)\right) \\ & + \big(q_i - u^{\mathrm{ref}}(\tau_i)\big)' R \left(q_i - u^{\mathrm{ref}}(\tau_i)\right)\Big] \\ & + \big(s_{n_\mathrm{p}} - x^{\mathrm{ref}}(t_0 + t_\mathrm{p})\big)' P \left(s_{n_\mathrm{p}} - x^{\mathrm{ref}}(t_0 + t_\mathrm{p})\right) , \end{aligned} \tag{7.7}$$

where we summarise all control inputs in $U \stackrel{\mathrm{def}}{=} \left(q_0', \ldots, q_{n_\mathrm{p}-1}'\right)' \in \mathbb{R}^{n_\mathrm{u} \cdot n_\mathrm{p}}$ and all artificial initial values in $X \stackrel{\mathrm{def}}{=} \left(s_1', \ldots, s_{n_\mathrm{p}}'\right)' \in \mathbb{R}^{n_\mathrm{x} \cdot n_\mathrm{p}}$. Moreover, multiple shooting introduces matching conditions, ensuring a continuous state trajectory at the solution, that can be summarised as

$$G(s_0, X, U) \stackrel{\mathrm{def}}{=} \begin{pmatrix} s_1 - x_0^{\mathrm{appr}}(\tau_1; s_0, q_0) \\ s_2 - x_1^{\mathrm{appr}}(\tau_2; s_1, q_1) \\ \vdots \\ s_{n_\mathrm{p}} - x_{n_\mathrm{p}-1}^{\mathrm{appr}}(t_0 + t_\mathrm{p}; s_{n_\mathrm{p}-1}, q_{n_\mathrm{p}-1}) \end{pmatrix} . \tag{7.8}$$

Also path and endpoint constraints (1.2e)–(1.2f) are discretised and imposed at the multiple shooting nodes only, which can be written as

$$H(s_0, X, U) \stackrel{\text{def}}{=} \begin{pmatrix} c(s_0, q_0) \\ \vdots \\ c\left(s_{n_\text{p}-1}, q_{n_\text{p}-1}\right) \\ c^\text{term}\left(s_{n_\text{p}}\right) \end{pmatrix}. \tag{7.9}$$

Altogether, this leads to the following specially structured, parameterised NLP with least-squares objective function:

$$\begin{aligned} \text{NLP}(w_0): \quad \min_{s_0, X, U} \quad & \|V(s_0, X, U)\|_2^2 && \text{(7.10a)} \\ \text{s.t.} \quad & s_0 = w_0\,, && \text{(7.10b)} \\ & G(s_0, X, U) = 0\,, && \text{(7.10c)} \\ & H(s_0, X, U) \leq 0\,, && \text{(7.10d)} \end{aligned}$$

The ACADO Toolkit solves this least-squares NLP with a generalised Gauss-Newton method as outlined in Subsection 6.2.3. An offline full-step version of this method starts from an initial guess $\left(s_0^{(0)}, X^{(0)}, U^{(0)}\right)$ and updates the iterates in the form

$$\left(s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}\right) \stackrel{\text{def}}{=} \left(s_0^{(k)}, X^{(k)}, U^{(k)}\right) + \left(\Delta s_0^{(k)}, \Delta X^{(k)}, \Delta U^{(k)}\right), \tag{7.11}$$

where $\left(\Delta s_0^{(k)}, \Delta X^{(k)}, \Delta U^{(k)}\right)$ solves the convex QP

$$\begin{aligned} \min_{\Delta s_0, \Delta X, \Delta U} \quad & \|V + V_{s_0}\Delta s_0 + V_X \Delta X + V_U \Delta U\|_2^2 && \text{(7.12a)} \\ \text{s.t.} \quad & s_0 + \Delta s_0 = w_0\,, && \text{(7.12b)} \\ & G + G_{s_0}\Delta s_0 + G_X \Delta X + G_U \Delta U = 0\,, && \text{(7.12c)} \\ & H + H_{s_0}\Delta s_0 + H_X \Delta X + H_U \Delta U \leq 0\,. && \text{(7.12d)} \end{aligned}$$

Here, we dropped the iteration index k and introduced the following short hands for notational convenience:

$$V \stackrel{\text{def}}{=} V(s_0, X, U)\,, \quad V_v \stackrel{\text{def}}{=} \partial_v V(s_0, X, U) \quad \forall\, v \in \{s_0, X, U\}\,, \quad \text{etc.} \tag{7.13}$$

As discussed in Subsection 6.3.1, the sparsity of this large-scale QP (7.12) can be exploited in different ways. The ACADO Toolkit currently implements an approach

that eliminates the variable ΔX from the QP formulation in order to yield the smaller-scale but dense convex QP:

$$\min_{\Delta s_0, \Delta U} \quad \| D + D_{s_0} \Delta s_0 + D_U \Delta U \|_2^2 \tag{7.14a}$$

$$\text{s.t.} \quad s_0 + \Delta s_0 \;=\; w_0 \,, \tag{7.14b}$$

$$E + E_{s_0} \Delta s_0 + E_U \Delta U \;\leq\; 0 \,, \tag{7.14c}$$

where the following "condensing matrices" are used:

$$\begin{aligned} D &\stackrel{\text{def}}{=} V - V_X G_X^{-1} G \,, \quad D_{s_0} \stackrel{\text{def}}{=} V_{s_0} - V_X G_X^{-1} G_{s_0} \,, \quad D_U \stackrel{\text{def}}{=} V_U - V_X G_X^{-1} G_U \,, \\ E &\stackrel{\text{def}}{=} H - H_X G_X^{-1} G \,, \quad E_{s_0} \stackrel{\text{def}}{=} H_{s_0} - H_X G_X^{-1} G_{s_0} \,, \quad E_U \stackrel{\text{def}}{=} H_U - H_X G_X^{-1} G_U \,. \end{aligned} \tag{7.15}$$

Note that the matrix G_X has a special structure with unit matrices on its diagonal blocks. Thus is it always invertible and can be inverted without any matrix factorisation.

In a second condensing step, also the variable Δs_0 can be eliminated leading to a dense QP with only $n_\mathrm{u} \cdot n_\mathrm{p}$ optimisation variables. By default, the `ACADO Toolkit` solves the dense QP (7.14) with `qpOASES` (see Chapter 4), which performs this second condensing step implicitly by fixing the variable Δs_0 to $w_0 - s_0$.

Solving the dense QP (7.14) delivers step directions $\Delta s_0^{(k)}$ and $\Delta U^{(k)}$, the corresponding step direction for the state trajectory can be obtained by

$$\Delta X^{(k)} \;=\; -G_X^{-1} \left(G + G_{s_0} \Delta s_0^{(k)} + G_U \Delta U^{(k)} \right) . \tag{7.16}$$

After this *expansion step*, the next Gauss-Newton iterate can be obtained.

7.2.2 Real-Time Iteration Algorithm

For the MPC context, the described offline Gauss-Newton method is slightly modified to perform real-time iterations as summarised in Subsection 6.4.5. For doing so, all computations are divided into a preparation and a usually much shorter feedback phase. The preparation phase starts with expanding the previous solution of the dense QP (7.14) to obtain $\Delta X^{(k-1)}$. Afterwards, the current iterate $\left(s_0^{(k)}, X^{(k)}, U^{(k)} \right)$ is obtained, at which the functions R, G and H are evaluated and all derivatives for setting up the large-scale QP (7.12) are computed. Moreover,

the variable $\Delta X^{(k)}$ is eliminated to yield an updated condensed, smaller-scale QP (7.14). Once the current initial value w_0 is known, the feedback phase only solves this updated dense QP and immediately returns the new approximately optimal control input $q_0^{(k+1)} \stackrel{\text{def}}{=} q_0^{(k)} + \Delta q_0^{(k)}$ to the process. The `ACADO Toolkit` implements this Gauss-Newton real-time iteration algorithm, which is summarised in Algorithm 7.1.

Algorithm 7.1 (Single Real-Time Iteration)

`input:` current initial value w_0 and starting iterate $\left(s_0^{(k)}, X^{(k)}, U^{(k)}\right)$, e.g. obtained from previous real-time iteration

`output:` next Gauss-Newton iterate $\left(s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}\right)$, including the first piece of an approximately optimal control input $q_0^{(k+1)} \in \mathbb{R}^{n_u}$ for current sampling instant

(P2) Evaluate V, G, H and all first-order derivatives V_{s_0}, V_X, V_u, G_{s_0} etc. at $\left(s_0^{(k)}, X^{(k)}, U^{(k)}\right)$.

(P3) Eliminate variable ΔX, i.e. compute the condensing matrices (7.15).

(P4) Wait for current initial value w_0.

(F1) Solve the dense convex QP (7.14) to yield $\left(\Delta s_0^{(k)}, \Delta U^{(k)}\right)$.

(F2) Send first control piece $q_0^{(k+1)} \leftarrow q_0^{(k)} + \Delta q_0^{(k)}$ immediately to the process.

(P1) Obtain $\Delta X^{(k)}$ via Equation (7.16) and compute next iterate $\left(s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}\right)$, possibly including a shift in time.

Within step (P2), first-order derivatives can be calculated by means of internal numerical or automatic differentiation (both in forward or backward mode).

7.2.3 Time-Optimal NMPC

The `ACADO Toolkit` also implements a variant of the real-time iteration scheme for a modified MPC formulation aiming at time-optimal behaviour of the process:

$$\min_{\substack{x(\cdot),u(\cdot),\\ y(\cdot),T}} \quad T \;+\; \alpha \cdot \int_{t_0}^{t_0+T} \hat{y}(t)'Q\hat{y}(t) + \hat{u}(t)'R\hat{u}(t)\,dt \;+\; \alpha \cdot \hat{x}(t_0+t_\mathrm{p})'P\hat{x}(t_0+t_\mathrm{p}) \tag{7.17a}$$

$$\text{s.t.} \quad x(t_0) \;=\; w_0\,, \tag{7.17b}$$

$$\dot{x}(t) \;=\; f\big(x(t),u(t)\big) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{7.17c}$$

$$y(t) \;=\; h\big(x(t),u(t)\big) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{7.17d}$$

$$0 \;\geq\; c\big(y(t),u(t)\big) \qquad \forall\, t \in \mathbb{T}_\mathrm{p}\,, \tag{7.17e}$$

$$x(t_0+T) \;=\; x^\mathrm{ref}(t_0+T)\,, \tag{7.17f}$$

$$T \;\geq\; T_\mathrm{min}\,, \tag{7.17g}$$

where we follow the notation of Definition 1.3. Moreover, T denotes the free length of the prediction horizon; $\alpha \geq 0$ and $T_\mathrm{min} > 0$ are tuning parameters that will be discussed now.

This time-optimal NMPC formulation has been proposed in [257] and keeps the length of the prediction horizon free for optimisation (in contrast to the standard formulation (1.2)). Therefore, it needs to explicitly include the constraint that the state reaches the desired reference value at the end of the time horizon. Moreover, a lower bound T_min on T needs to be introduced as the problem could otherwise become degenerated for $T = 0$. Finally, the tuning parameter weights the tracking term of the objective, where $\alpha = 0$ would lead to a truly time-optimal formulation.

If time-optimality is the true control objective, the advantage of formulation (7.17) is that it explicitly searches for time-optimal solutions. However, leaving the length of the prediction horizon free for optimisation introduces a significant source of nonlinearity and nonconvexity into the optimisation problem. Thus, it becomes more likely that the optimisation algorithm gets stuck in a sub-optimal local minimum. Therefore, this formulation seems only to be practical if a good initial guess for the (global) solution is available.

One might ask why a weighted tracking term is kept within the time-optimal NMPC formulation. The reason to introduce a lower bound on T along with the addition of a small least-squares objective is to avoid deadbeat behaviour of the controller when close to the steady-state. Instead, it is easy to see that once $T = T_\mathrm{min}$ is possible, i.e. the steady-state could be reached within the time T_min, the controller behaviour automatically switches from time-optimal to conventional

model predictive control: if no further disturbance occurs, the controller will behave from that moment on exactly as a tracking MPC controller with a control horizon of length $T_{\min}$.

Algorithmically, the horizon length T can be introduced as additional differential state $x_{n_\mathrm{x}+1}$ that satisfies the trivial differential equation $\dot{x}_{n_\mathrm{x}+1}(t) = 0$ and whose initial value $x_{n_\mathrm{x}+1}(t_0)$ is free. Additionally including the terminal constraint $x(t_0 + T) = x^{\mathrm{ref}}(t_0 + T)$ and the lower bound on $T = x_{n_\mathrm{x}+1}$ is straight-forward. Thus, the time-optimal formulation (7.17) results in a sparse NLP that is similar to formulation (7.10):

$$\mathrm{NLP}(w_0): \quad \min_{s_0,X,U} \quad \underbrace{s_{n_\mathrm{p},n_\mathrm{x}+1} + \alpha \cdot \|V(s_0,X,U)\|_2^2}_{=F(s_0,X,U)} \tag{7.18a}$$

$$\text{s.t.} \quad s_{0,i} \;=\; w_{0,i} \quad \forall\, i \in \{1,\ldots,n_\mathrm{x}\}\,, \tag{7.18b}$$

$$G(s_0,X,U) \;=\; 0\,, \tag{7.18c}$$

$$H(s_0,X,U) \;\leq\; 0\,, \tag{7.18d}$$

where $s_{j,i}$ denotes the ith component of $s_j \in \mathbb{R}^{n_\mathrm{x}+1}$ for all $j \in \{0,\ldots,n_\mathrm{p}\}$. Again, all control inputs are summarised in $U \stackrel{\mathrm{def}}{=} \left(q_0',\ldots,q_{n_\mathrm{p}-1}'\right)' \in \mathbb{R}^{n_\mathrm{u}\cdot n_\mathrm{p}}$ and $X \stackrel{\mathrm{def}}{=} \left(s_1',\ldots,s_{n_\mathrm{p}}'\right)' \in \mathbb{R}^{(n_\mathrm{x}+1)\cdot n_\mathrm{p}}$ summarises all (augmented) artificial initial values. The functions V, G and H are adapted variants of the ones defined in (7.7)–(7.9) that reflect the augmented dimensions of s_0 and X as well as the additional constraints.

The optimal solution of (7.18) usually depends strongly nonlinearly on the choice of T. Thus, using only first-order derivative information as the Gauss-Newton method of Subsection 7.2.1 might not lead to convergence. Therefore, the `ACADO Toolkit` allows one to rather formulate the QP subproblems (6.17) based on an exact computation of the Hessian matrix of the Lagrange function as summarised in Algorithm 7.2. Calculating also second-order derivatives increases the computational load of the SQP method but the other ideas of the real-time iteration scheme remain valid. The only remaining modification is that the expansion step described in Equation (7.16) also needs to expand the dual solution information of the dense QP (see [155]).

Algorithm 7.2 (Single Real-Time Iteration for Time-Optimal NMPC)

`input:` current initial value w_0, starting primal/dual iterate $\left(s_0^{(k)}, X^{(k)}, U^{(k)}\right)$ and $\lambda^{(k)}$, $\mu^{(k)}$, e.g. obtained from previous real-time iteration

`output:` next primal/dual SQP iterate $\left(s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}\right)$, $\lambda^{(k+1)}$, $\mu^{(k+1)}$ including the first piece of an approximately optimal control input input $q_0^{(k+1)} \in \mathbb{R}^{n_\mathrm{u}}$ for current sampling instant

(P2) Evaluate F, G, H and all first-order derivatives at $\left(s_0^{(k)}, X^{(k)}, U^{(k)}\right)$. Moreover, compute Hessian matrix $\nabla^2_{(s_0,X,U)} \mathcal{L}\left((s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}), \lambda^{(k)}, \mu^{(k)}\right)$ of the Lagrange function (6.13).

(P3) Eliminate variable ΔX from resulting QP, i.e. compute condensing matrices similar to (7.15).

(P4) Wait for current initial value w_0.

(F1) Solve dense convex QP similar to (7.14) to yield $\left(\Delta s_0^{(k)}, \Delta U^{(k)}\right)$.

(F2) Send first control piece $q_0^{(k+1)} \leftarrow q_0^{(k)} + \Delta q_0^{(k)}$ immediately to the process.

(P1) Expand QP solution to obtain $\Delta X^{(k)}$, $\lambda^{(k+1)}$, $\mu^{(k+1)}$ and compute next primal iterate $\left(s_0^{(k+1)}, X^{(k+1)}, U^{(k+1)}\right)$, possibly including a shift in time (see [155] for details).

7.2.4 Simulation Environment

The `ACADO Toolkit` also provides a built-in simulation environment for performing realistic closed-loop simulations [82]. Its main components are the `Process` class for setting up a simulation of the process to be controlled and the `Controller` class for implementing the closed-loop controller. This controller can later be used stand-alone for real-world feedback control applications.

The `Process` class has as members a dynamic system, comprising a differential equation and an optional output function, modelling the process as well as an integrator capable of simulating these model equations. The simulation uses (optimised) control inputs from the controller, which might be subject to noise or delays that can be introduced via an optional `Actuator`. In addition, so-called process disturbances can be specified by the user for setting up arbitrary

disturbance scenarios for the simulation. Finally, the outputs obtained by integrating the model equations can again be subject to noise or delays introduced via an optional `Sensor`. It is important to note that the model used for simulating the process does not need to be the same as specified within the optimal control formulations within the controller.

The `Controller` class consists of three major blocks: first, an online state/parameter estimator uses the outputs of the process to obtain estimates for the differential states or other parameters. This estimation can be based on online optimisation, e.g. moving horizon estimation, but also a linear or extended Kalman filter could be used. Second, a reference trajectory can be provided to the control law. These references can either be statically given by the user according to a desired simulation scenario or can be calculated dynamically based on information from the estimator. Finally, both the state/parameter estimates as well as the reference trajectory are used by the control law class to compute optimised control inputs. The control law will usually be a `RealTimeAlgorithm` based on the real-time iteration algorithms as described in Subsections 7.2.1–7.2.3, but can also be something as simple as a linear state feedback.

While the `Process` is conceptionally thought to produce a continuous output stream, output of the `Controller` comes in sampled form. Its sampling times are usually pre-defined by the user. Communication between `Process` and `Controller` is orchestrated by an instance of the `SimulationEnviroment` class. It also features the simulation of computational delays, i.e.i.e. it can delay the control input to the `Process` by the amount of time the `Controller` took to determine the control inputs. This feature seems to be crucial for realistic closed-loop simulations of fast processes where the sampling time is not negligible compared to the settling time of the controlled process.

Figure 7.3 illustrates the mentioned classes of the built-in simulation environment, which are all derived from the base class `SimulationBlock`. In the programming philosophy introduced in Subsection 7.1.3, all classes of the simulation environment are user interfaces, thus the `SimulationBlock` is derived from the `UserInteraction` class. Figure 7.3 also shows the class `RealTimeAlgorithm`, which implements the mentioned variants of the real-time iteration scheme.

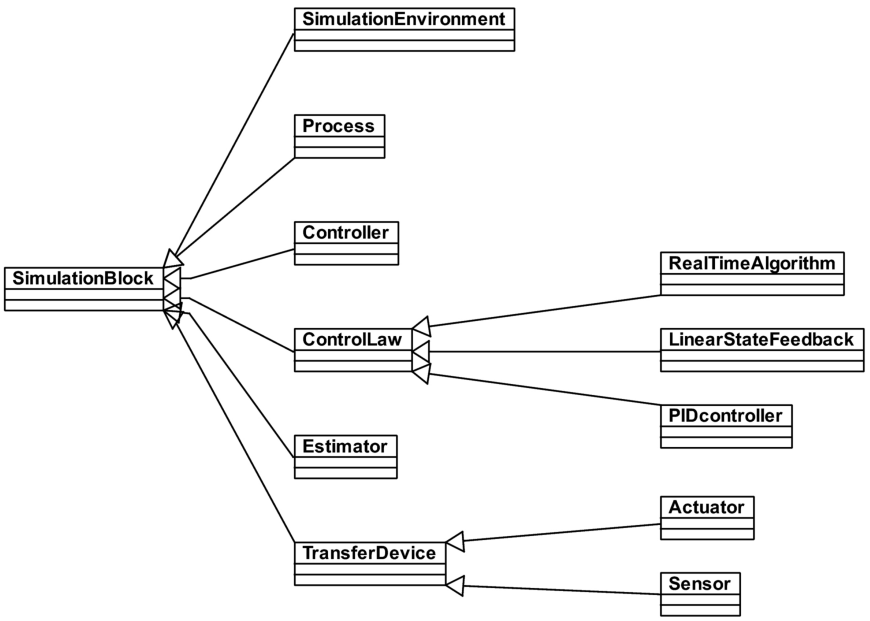


Figure 7.3: UML class diagram illustrating the main building blocks of the built-in simulation environment of the ACADO Toolkit.

7.3 Numerical Example

7.3.1 Start-Up of a Continuous Stirred Tank Reactor

For illustrating the two variants of the real-time iteration algorithm as implemented within the ACADO Toolkit, we consider the nonlinear ODE model of a continuous stirred tank reactor (CSTR). The formulation we use comprises four states, $x \in \mathbb{R}^4$, and two controls, $u \in \mathbb{R}^2$. Here, the first two states, c_A and c_B, are the concentrations of cyclopentadiene (substance A) and cyclopentenol (substance B), respectively, while the other two states, ϑ and ϑ_K, denote the temperature in the reactor and temperature in the cooling jacket of the tank reactor. Thus, the state vector is given by

$$x(t) \stackrel{\text{def}}{=} \begin{pmatrix} c_A(t) \\ c_B(t) \\ \vartheta(t) \\ \vartheta_K(t) \end{pmatrix}. \tag{7.19}$$

The first input is the feed inflow which is controlled via its scaled rate $u_1(t) \stackrel{\text{def}}{=} \frac{\dot{V}(t)}{V_R}$, while the temperature ϑ_K is held down by an external heat exchanger whose heat removal rate $u_2(t) \stackrel{\text{def}}{=} \dot{Q}_K(t)$ can be controlled as well.

The model is given by the following nonlinear ODE system:

$$\begin{aligned}
\dot{c}_A(t) &= u_1(t)(c_{A0} - c_A(t)) - k_1(\vartheta(t))c_A(t) - k_3(\vartheta(t))(c_A(t))^2, \\
\dot{c}_B(t) &= -u_1(t)c_B(t) + k_1(\vartheta(t))c_A(t) - k_2(\vartheta(t))c_B(t)\,, \\
\dot{\vartheta}(t) &= u_1(t)(\vartheta_0 - \vartheta(t)) + \tfrac{k_w A_R}{\rho C_p V_R}(\vartheta_K(t) - \vartheta(t)) \\
&\quad -\tfrac{1}{\rho C_p}\left[\, k_1(\vartheta(t))c_A(t)H_1 + k_2(\vartheta(t))c_B(t)H_2 + k_3(\vartheta(t))(c_A(t))^2 H_3 \,\right], \\
\dot{\vartheta}_K(t) &= \tfrac{1}{m_K C_{PK}}\left(u_2(t) + k_w A_R(\vartheta(t) - \vartheta_K(t))\right)\,.
\end{aligned} \tag{7.20}$$

Therein, the reaction rate functions $k_i : \mathbb{R} \to \mathbb{R}$ are given by

$$k_i(\vartheta(t)) \;=\; k_{i0} \cdot \exp\left(\frac{E_i}{\vartheta(t) + 273.15\,^\circ\mathrm{C}}\right), \quad i \in \{1,2,3\}\,.$$

These equations along with appropriate values for all parameters can be found in [148, 62] and have been proposed as a benchmark example for NMPC in [51].

For performing the NMPC simulations, we control the start-up of this CSTR model. The aim to is steer the reactor in minimal time from its start configuration

$$x(t_{\text{start}}) \stackrel{\text{def}}{=} \begin{pmatrix} 0.0\,\frac{\text{mol}}{\text{l}} \\ 0.0\,\frac{\text{mol}}{\text{l}} \\ 85.0\,^\circ\text{C} \\ 85.0\,^\circ\text{C} \end{pmatrix} \tag{7.21}$$

to a stable steady-state given by

$$x^{\text{ref}} \stackrel{\text{def}}{=} \begin{pmatrix} 2.14\,\frac{\text{mol}}{\text{l}} \\ 1.09\,\frac{\text{mol}}{\text{l}} \\ 114.2\,^\circ\text{C} \\ 112.9\,^\circ\text{C} \end{pmatrix}, \quad u^{\text{ref}} \stackrel{\text{def}}{=} \begin{pmatrix} 14.19\,\text{h}^{-1} \\ -1113.5\,\frac{\text{kJ}}{\text{h}} \end{pmatrix}. \tag{7.22}$$

While doing so, we require the control inputs to satisfy the following constraints:

$$\begin{pmatrix} 3.0\,\text{h}^{-1} \\ -9000.0\,\frac{\text{kJ}}{\text{h}} \end{pmatrix} \;\leq\; u(t) \;\leq\; \begin{pmatrix} 35.0,\text{h}^{-1} \\ 0.0\,\frac{\text{kJ}}{\text{h}} \end{pmatrix}. \tag{7.23}$$

Listing 7.2 illustrates the formulation of the corresponding (offline) optimal control problem in `ACADO` syntax.

For all simulations, we employ a multiple shooting discretisation, where the prediction horizon $[t_0, t_0 + 1500\,\mathrm{s}]$ is divided into 20 control intervals of equal length. The sampling time is chosen to equal the length of one control interval, i.e. 75 s. In order to focus on the main algorithmic effects, we restrict ourselves to nominal simulations: no model-plant mismatch, no disturbances, all states can be measured.

We present numerical results comparing both the tracking formulation (7.3) and the time-optimal formulation (7.17) with respect to control performance and computational load. For both formulations, we compare different algorithmic variants. All simulations have been performed on a standard PC having a 2.8 GHz dual-core processor and 4 GB RAM.

7.3.2 Using Fully Converged Solutions

Figure 7.4 shows states and control inputs of the simulated CSTR start-up controlled using online optimisation with fully converged solutions. We compare the tracking NMPC formulation and the time-optimal NMPC formulation with $\alpha = 1$ and $\alpha = 0.5$. As expected, the time-optimal NMPC formulation is able to regulate the CSTR faster to the desired steady-states. This comes at the expense of a larger overshoot at the beginning. We see that decreasing the tuning parameter α, i.e. reducing the influence of the tracking component in the objective function, leads to a more aggressive behaviour such that the desired steady-state is reached even faster.

As mentioned in Subsection 7.2.3, we need to choose a lower bound $T_{\min}$ on the horizon length T when employing the time-optimal NMPC formulation. For our simulations, we use a constant value of $T_{\min} = 150\,\mathrm{s}$ which does not constrain T in the first samples but ensures a smooth tracking NMPC behaviour once the steady-state can be reached within 150 s.

Figure 7.5 compares the computational load of different algorithmic variants by illustrating the total runtime per sampling instant required to solve the respective optimal control problem. The tracking NMPC formulation is solved once employing the Gauss-Newton algorithm described in Subsection 7.2.1 and once using an exact Hessian computation. Solution of the time-optimal NMPC formulation is always based on an exact Hessian computation.

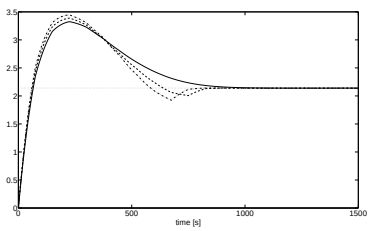
We can observe that runtimes vary greatly depending on the number of Newton-type iterations needed to solve the problem, which becomes smaller when we approach the steady-state. In case of the tracking NMPC formulation, calculation

Listing 7.2: Symbolic formulation in ACADO syntax of an optimal control problem comprising the CSTR model and a least-squares objective function.

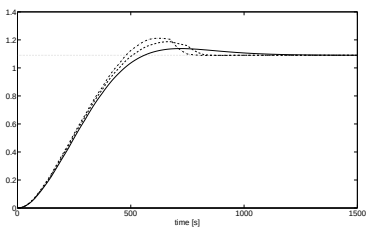
```
int main( ){
    // ...

    // DEFINE CSTR MODEL:
    // ------------------
    DifferentialEquation f;

    DifferentialState cA, cB, theta, thetaK;
    Control u(2);
    IntermediateState k1, k2, k3;

    k1 = k10 * exp( E1/(273.15+theta) );
    k2 = k20 * exp( E2/(273.15+theta) );
    k3 = k30 * exp( E3/(273.15+theta) );

    f << dot(cA)     == u(0)*(cA0-cA) - k1*cA - k3*cA*cA;
    f << dot(cB)     == - u(0)*cB + k1*cA - k2*cB;
    f << dot(theta)  == u(0)*(theta0-theta)
                        -(1/(rho*Cp))*(k1*cA*H1 + k2*cB*H2 + k3*cA*cA*H3)
                        +(kw*AR/(rho*Cp*VR))*(thetaK -theta);
    f << dot(thetaK) == (1/(mK*CPK))*(u(1) + kw*AR*(theta-thetaK));

    // DEFINE LEAST-SQUARE FUNCTION:
    // -----------------------------
    Function h;
    h << cA;
    h << cB;
    h << theta;
    h << thetaK;
    h << u(0);
    h << u(1);

    Matrix S = eye(6);
    Vector r = zeros(6);
    // ...

    // DEFINE AN OPTIMAL CONTROL PROBLEM:
    // ----------------------------------
    const double tStart = 0.0;
    const double tEnd   = 1500.0;

    OCP ocp( tStart,tEnd, 20 );

    ocp.minimizeLSQ( S,h,r );

    ocp.subjectTo( f );
    ocp.subjectTo( AT_START, cA     ==  0.0 );
    ocp.subjectTo( AT_START, cB     ==  0.0 );
    ocp.subjectTo( AT_START, theta  == 85.0 );
    ocp.subjectTo( AT_START, thetaK == 85.0 );
    ocp.subjectTo(     3.0 <= u(0) <= 35.0 );
    ocp.subjectTo( -9000.0 <= u(1) <=  0.0 );

    // DEFINE AN OPTIMIZATION ALGORITHM AND SOLVE THE OCP:
    // ---------------------------------------------------
    OptimizationAlgorithm algorithm( ocp );

    algorithm.set( HESSIAN_APPROXIMATION, GAUSS_NEWTON );
    algorithm.set( KKT_TOLERANCE, 1e-5 );
    algorithm.solve( );

    return 0;
}
```

of the exact Hessian does not seem to speed-up convergence while making each iteration more expensive. Changing α moderately in the time-optimal NMPC formulation does not seem to affect the overall runtime significantly.

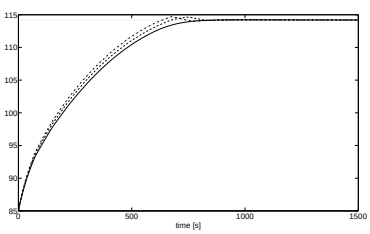
Note that the computational performance of the `ACADO Toolkit` on this example is competitive with other existing software packages: For example, solving the



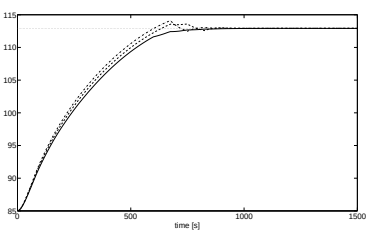
(a) Concentration of substance A in mol/l.



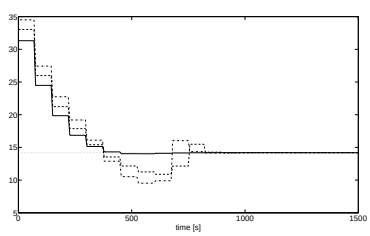
(b) Concentration of substance B in mol/l.



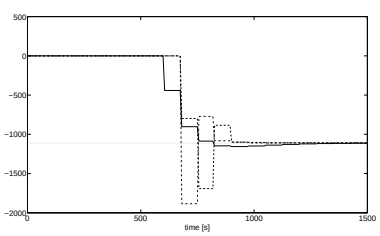
(c) Temperature in the reactor in °C.



(d) Temperature of the jacket in °C.



(e) Scaled feed flow u_1 in 1/s.



(f) Heat removal rate u_2 in kJ/s.

Figure 7.4: CSTR start-up controlled using online optimisation with fully converged solutions: tracking NMPC (solid), time-optimal NMPC with $\alpha = 1$ (dashed), time-optimal NMPC with $\alpha = 0.5$ (dash-dotted). Also the respective reference values are shown (dotted, grey).

optimal control problem at the first sampling instant with the nonlinear optimal control package `dsoa` takes about 0.86 seconds to obtain the same solution accuracy. `dsoa` implements a single-shooting approach and solves the resulting small-scale NLP by means of an inexact SQP algorithm [73] based on BFGS updates (6.19).

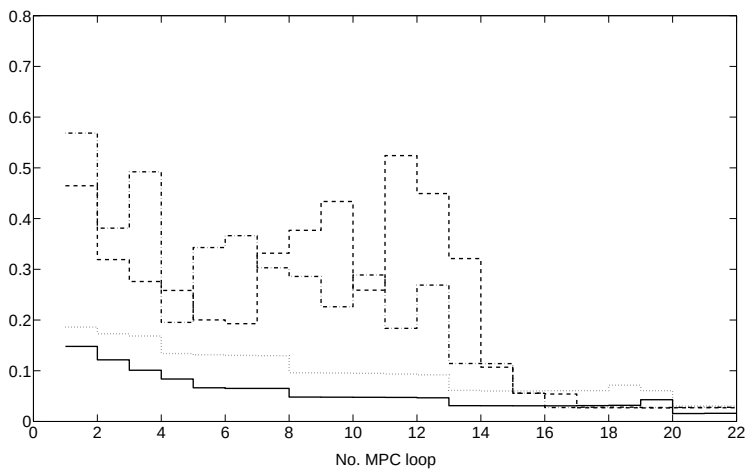


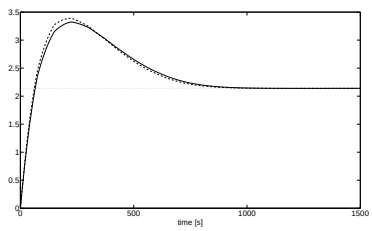
Figure 7.5: Runtimes in seconds for the CSTR start-up controlled using online optimisation with fully converged solutions: tracking NMPC (solid), tracking NMPC using exact Hessians (dotted), time-optimal NMPC with $\alpha = 1$ (dashed), time-optimal NMPC with $\alpha = 0.5$ (dash-dotted).

7.3.3 Employing the Real-Time Iteration Algorithm

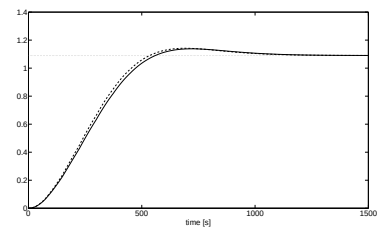
Next we employ the real-time iteration algorithm performing only one Newton-type iteration per NMPC sampling instant. Figure 7.6 illustrates states and control inputs for the CSTR start-up when using the tracking NMPC formulation; for comparison also the ones obtained by using the fully converged solutions are depicted. Figure 7.7 illustrates the corresponding results for the time-optimal NMPC formulation with $\alpha = 1$.

We can see that control performance is hardly affected by restricting the number of Newton-type iterations to one per NMPC sampling instant. This is both due to the only mildly nonlinear behaviour of the CSTR and the good contraction properties of the real-time iteration scheme using initial value-embedding. Figure 7.8 illustrates that the computational load can be reduced

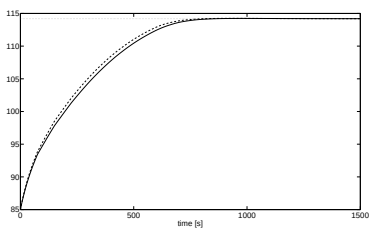
significantly when employing the real-time iteration scheme instead of using the fully converged solutions. Again we note that runtimes of the ACADO real-time iteration algorithms seem to be competitive, when comparing them with results reported earlier in the literature: For example, computation times in the order of



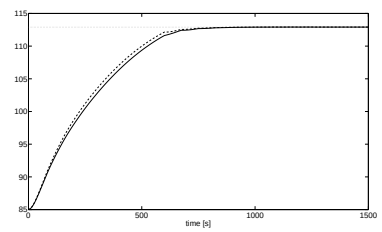
(a) Concentration of substance A in mol/l.



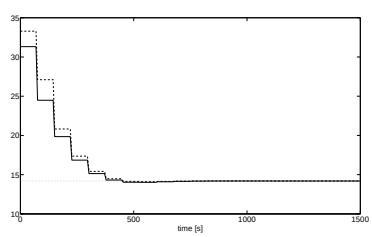
(b) Concentration of substance B in mol/l.



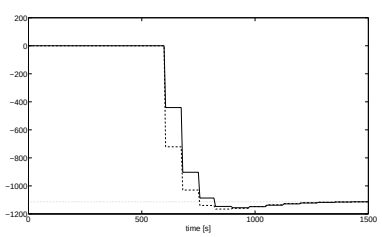
(c) Temperature in the reactor in °C.



(d) Temperature of the jacket in °C.



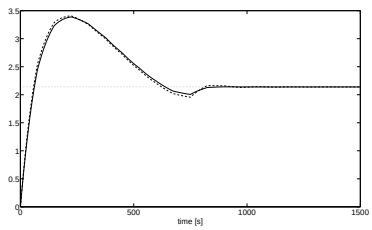
(e) Scaled feed flow u_1 in 1/s.



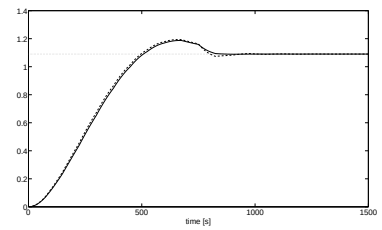
(f) Heat removal rate u_2 in kJ/s.

Figure 7.6: CSTR start-up controlled using online optimisation based on a tracking NMPC formulation: fully converged solution (solid) and solution obtained by employing the real-time iteration scheme performing only one Gauss-Newton iteration per sampling instant (dashed). Also the respective reference values are shown (dotted, grey).

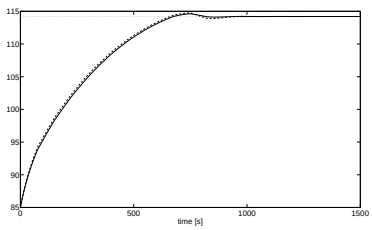
minutes for a similar setup have been reported fifteen years ago in [49], while [37, 62] report computation times of about one second (on a PC that was at most 10 times slower than the one we used to run the ACADO simulations). The next subsection will provide a closer look at the computational load of one real-time iteration.



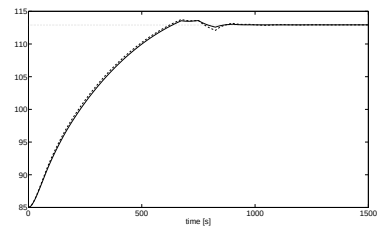
(a) Concentration of substance A in mol/l.



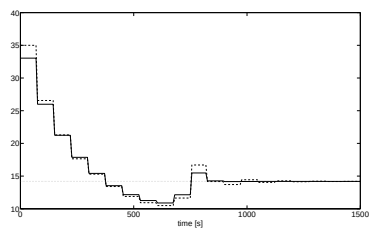
(b) Concentration of substance B in mol/l.



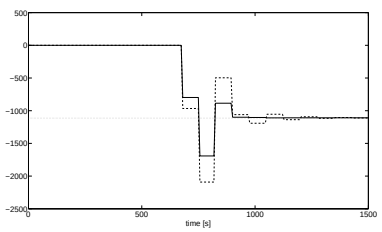
(c) Temperature in the reactor in °C.



(d) Temperature of the jacket in °C.



(e) Scaled feed flow u_1 in 1/s.



(f) Heat removal rate u_2 in kJ/s.

Figure 7.7: CSTR start-up controlled using online optimisation based on a time-optimal NMPC formulation with $\alpha = 1$: fully converged solution (solid) and solution obtained by employing the real-time iteration scheme performing only one SQP iteration per sampling instant (dashed). Also the respective reference values are shown (dotted, grey).

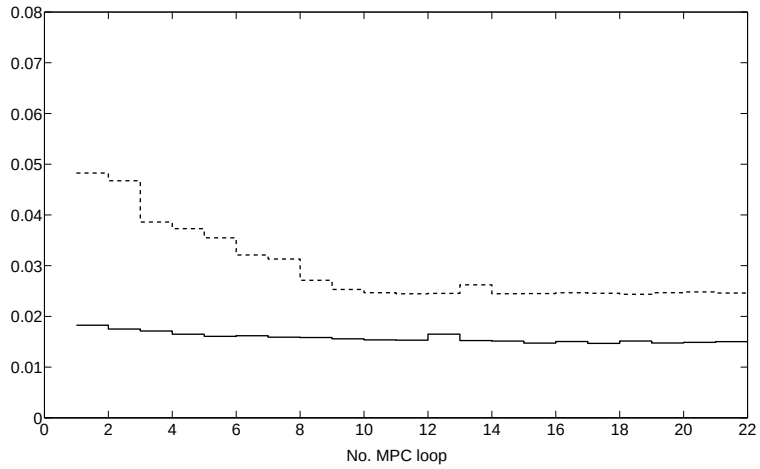


Figure 7.8: Runtimes in seconds for the CSTR start-up controlled using online optimisation employing the real-time iteration scheme: tracking NMPC with Gauss-Newton Hessian approximation (solid) and time-optimal NMPC with $\alpha = 1$ with exact Hessian computation (dashed).

7.3.4 Computational Load of Real-Time Iterations

As described in Section 7.2.2, each real-time iteration consists of different algorithmic steps. The most important ones are the integration of the dynamic system including the sensitivity generation, the condensing step and the solution of the condensed QP. Table 7.1 lists the runtimes for each of these steps for a typical real-time iteration when controlling the CSTR start-up using the tracking NMPC formulation. We see that the major fraction of the computation time is spent for the sensitivity generation.

Table 7.2 reports the runtimes for a typical real-time iteration with exact Hessian computation using the tracking NMPC formulation. When comparing the runtimes with the ones of Table 7.1, we see that the exact Hessian computation causes the integration and sensitivity generation to take longer, while the effort for all other algorithmic parts basically stays the same.

Next, Table 7.3 reports the runtimes for a typical real-time iteration with exact Hessian computation using the time-optimal NMPC formulation. The reported runtimes are very similar to the ones reported in Table 7.2. This comes at no

surprise as all algorithmic steps are basically the same (however, the time-optimal NMPC formulation comprises T as an additional degree of freedom).

	CPU time	%
Integration and sensitivity generation	10.5 ms	69 %
Condensing	3.0 ms	20 %
QP solution (with `qpOASES`)	0.6 ms	4 %
Remaining operations	1.1 ms	7 %
One complete real-time iteration	15.2 ms	100 %

Table 7.1: Typical runtime performance of one Gauss-Newton real-time iteration when controlling the CSTR start-up using the tracking NMPC formulation.

	CPU time	%
Integration and sensitivity generation	29.1 ms	86 %
Condensing	3.0 ms	9 %
QP solution (with `qpOASES`)	0.6 ms	2 %
Remaining operations	1.1 ms	3 %
One complete real-time iteration	33.8 ms	100 %

Table 7.2: Typical runtime performance of one real-time iteration with exact Hessian computation when controlling the CSTR start-up using the tracking NMPC formulation.

	CPU time	%
Integration and sensitivity generation	31.7 ms	87 %
Condensing	3.3 ms	9 %
QP solution (with `qpOASES`)	0.7 ms	2 %
Remaining operations	0.9 ms	2 %
One complete real-time iteration	36.6 ms	100 %

Table 7.3: Typical runtime performance of one real-time iteration with exact Hessian computation when controlling the CSTR start-up using the time-optimal NMPC formulation.

Chapter 8

Code Generation for Nonlinear MPC

This chapter presents the ACADO Code Generation tool [80, 129] for generating real-time iteration algorithms for nonlinear MPC. Based on the symbolic syntax of the ACADO Toolkit, it allows the user to export highly efficient and self-contained C code that is tailored to each respective MPC problem formulation. We explain its algorithmic components and investigate its computational performance. It is illustrated with small-scale NMPC examples that automatically generated NMPC algorithms can significantly outperform their counterparts implemented in a generic way. The ACADO Code Generation tool has been developed jointly with Boris Houska and is the second main contribution of the second part of this thesis.

8.1 Introduction

The idea to automatically generate tailored source code in order to speed-up the numerical solution of optimisation problems is not new. More than 20 years ago, a code generation environment to export tailored implementations of Karmarkar's algorithm for solving LPs was presented [182]. It exported PASCAL source code implementing a customised Cholesky decomposition as well as further matrix-vector operations, but does not seem to have received much attention.

About one decade ago, code generation was proposed for use in nonlinear MPC. The software AutoGenU by Ohtsuka and Kodama [181, 180] provides Mathematica scripts to generate customised C code implementing the Continuation/GMRES method as described in Subsection 6.4.4. AutoGenU allows the user to specify the

MPC problem formulation together with initialisations in the symbolic syntax of `Mathematica`. Solving only one linear system per sampling instant, the exported NMPC algorithm has been used to control an experimental hovercraft setup at sampling times of 1.5 milliseconds [210].

Recently, code generation has attracted great attention due to the software package `CVXGEN` [169]. It is based on the MATLAB front-end `CVX` [112, 113]—which provides a symbolic syntax to formulate and solve convex optimisation problems—and allows the user to generate customised interior-point solvers for small-scale LP and QP problems. In particular, linear MPC problems can be solved based on highly efficient, auto-generated code. The main ingredient of the exported code is a sparse Cholesky decomposition of the interior-point KKT system (6.23). `CVXGEN` automatically detects the respective sparsity pattern and determines a fixed pivoting that aims at avoiding fill-in in the Cholesky factors as far as possible. `CVXGEN` is not freely available but can be tested by academic users via a dedicated web interface [168].

After this brief review of previous approaches to auto-generate code for solving optimisation problems, we want to summarise the two main advantages that motivate automatic code generation:

- *Speeding-up computations* by tailoring the code to each specific optimisation problem formulation. This can comprise an optimised memory management as problem dimensions and sparsity patterns can be detected and hard-coded beforehand. Moreover, computations of the exported code might be organised in such a way that they maximise the cache usage of the respective target hardware. Also loop unrolling and the correct choice of compiler settings might speed-up computations significantly. Finally, the numerical algorithm itself might be tailored to the specific application by leaving away unnecessary computations.

- *Increased adaptivity* of the numerical algorithm to be exported by influencing its programming syntax. For example, the code generator might offer options to choose between single and double precision arithmetic or to avoid certain programming constructs or calls to functions from standard libraries that are not available on the respective target hardware[1]. It might even be desired to export one and the same optimisation code in different programming languages (like the Multi-Parametric Toolbox for explicit MPC that allows the user to generate the online look-up table both in C and in a language compatible with certain programmable logic controllers [153]).

[1] Up to a certain extend this could also be achieved by using preprocessor directives. However, this might greatly reduce readability and maintainability of the code.

These benefits of code generation seem to be most relevant for real-time optimisation algorithms that need to solve optimisation problems at very high sampling rates. This is particularly true if these algorithms are designed to run on embedded hardware, as this imposes specific requirements on the implementation (see the discussion of Subsection 5.1.2). Thus, code generation looks like a promising technique, especially for small- to medium-scale nonlinear MPC problems.

8.2 Auto-Generated Real-Time Iteration Algorithms

We will now present the `ACADO Code Generation` tool for automatically generating Gauss-Newton real-time iteration algorithms for nonlinear MPC (see Section 7.2). It allows the user to export highly efficient and self-contained C code that is tailored to each respective MPC problem formulation. It generalises previous work on code generation (as in [169]) to nonlinear MPC algorithms, which (in contrast to [181]) are based on solving a full QP at each sampling instant for improving control performance. This QP solution can either be done using an embedded variant of `qpOASES` or a dense QP solver as exported by `CVXGEN`. Our presentation extends the ones given in [129, 81].

8.2.1 Symbolic Problem Formulation

All three code generation tools mentioned in Section 8.1 rely on a symbolic formulation of the optimisation problem to be solved. This comes as no surprise as only a symbolic representation keeps all structural information that should be exploited as far as possible to increase efficiency of the generated code. The `ACADO Code Generation` tool is written as an add-on to the `ACADO Toolkit` and therefore can make use of its symbolic syntax (see Subsection 7.1.2). Listing 8.1 illustrates this with a slightly adapted variant of Listing 7.2 (see page 133) for exporting a tailored Gauss-Newton real-time iteration algorithm. All structure-exploiting features of the `ACADO Code Generation` tool will be discussed in the following subsections.

Currently, the `ACADO Code Generation` tool allows the user to export optimised C code for solving nonlinear MPC problems of the following form:

$$\min_{x(\cdot),u(\cdot)} \quad \int_{t_0}^{t_0+t_\mathrm{p}} \left\|x(t)-x^{\mathrm{ref}}(t)\right\|_Q^2 + \left\|u(t)-u^{\mathrm{ref}}(t)\right\|_R^2 \, dt \tag{8.1a}$$

$$+ \left\|x(t_0+t_\mathrm{p})-x^{\mathrm{ref}}(t_0+t_\mathrm{p})\right\|_P^2$$

$$\text{s. t.} \quad x(t_0) \;=\; w_0\,, \tag{8.1b}$$

$$\dot{x}(t) \;=\; f(x(t),\, u(t))\,, \tag{8.1c}$$

$$\underline{u}(t) \;\leq\; u(t) \;\leq\; \overline{u}(t) \qquad \forall\, t \in [t_0, t_0+t_\mathrm{p}]\,, \tag{8.1d}$$

$$\underline{x}(t) \;\leq\; x(t) \;\leq\; \overline{x}(t) \qquad \forall\, t \in [t_0, t_0+t_\mathrm{p}]\,. \tag{8.1e}$$

Here, $x : \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$ denotes the differential state, $u : \mathbb{R} \to \mathbb{R}^{n_\mathrm{u}}$ the control input and $w_0 \in \mathbb{R}^{n_\mathrm{x}}$ the current initial state. For all $t \in [t_0, t_0+t_\mathrm{p}]$ possibly time-varying reference values for states and controls are denoted by $x^{\mathrm{ref}}(t) \in \mathbb{R}^{n_\mathrm{x}}$, $u^{\mathrm{ref}}(t) \in \mathbb{R}^{n_\mathrm{u}}$, respectively, and $\underline{u}(t) \leq \overline{u}(t) \in \mathbb{R}^{n_\mathrm{u}}$, $\underline{x}(t) \leq \overline{x}(t) \in \mathbb{R}^{n_\mathrm{x}}$ denote lower and upper limits on control inputs and states, respectively. The right-hand side function f defining a system of ordinary differential equations (ODEs) can be nonlinear in both states and controls, while only least-squares tracking objectives are currently supported (with $\|\cdot\|_M$ denoting the Euclidean norm weighted by a symmetric, positive semi-definite matrix M).

Note that the `ACADO Code Generation` tool currently does not support the solution of time-optimal NMPC problems. However, generating algorithms as described in Subsection 7.2.3 is possible, though computing exact second derivatives would require some extra programming effort.

8.2.2 Integration and Sensitivity Generation

As the real-time iteration algorithm is based on a shooting discretisation of the optimal control problem (8.1), integrating the ODE system and computing first-order derivatives with respect to the initial value(s) and the parameterised control inputs are a main computational step. The code exported by the `ACADO Code Generation` tool computes these first-order derivatives (or sensitivities) by means of the *variational differential equation*:

Let an initial value problem of the form

$$x(t_i) \;=\; s_i\,, \tag{8.2a}$$

$$\dot{x}(t) \;=\; f\left(x(t), q_i\right) \quad \forall\, t \in [t_i, t_{i+1}]\,, \tag{8.2b}$$

Listing 8.1: Slightly adapted variant of Listing 7.2 for exporting a tailored real-time iteration algorithm using the `ACADO Code Generation` tool.

```
int main( ){
    // ...

    // DEFINE CSTR MODEL:
    // ------------------
    DifferentialEquation f;

    DifferentialState cA, cB, theta, thetaK;
    Control u(2);
    IntermediateState k1, k2, k3;

    k1 = k10 * exp( E1/(273.15+theta) );
    k2 = k20 * exp( E2/(273.15+theta) );
    k3 = k30 * exp( E3/(273.15+theta) );

    f << dot(cA)      == u(0)*(cA0-cA) - k1*cA - k3*cA*cA;
    f << dot(cB)      == - u(0)*cB + k1*cA - k2*cB;
    f << dot(theta)   == u(0)*(theta0-theta)
                         -(1/(rho*Cp))*(k1*cA*H1 + k2*cB*H2 + k3*cA*cA*H3)
                         +(kw*AR/(rho*Cp*VR))*(thetaK -theta);
    f << dot(thetaK) == (1/(mK*CPK))*(u(1) + kw*AR*(theta-thetaK));

    // DEFINE WEIGHTING MATRICES:
    // --------------------------
    Matrix Q  = eye(4);
    Matrix R  = eye(2);
    // ...

    // DEFINE AN OPTIMAL CONTROL PROBLEM:
    // ----------------------------------
    const double tStart  = 0.0;
    const double tEnd    = 1500.0;
    const int nIntervals = 20;

    OCP ocp( tStart,tEnd, nIntervals );

    ocp.minimizeLSQ( Q, R );

    ocp.subjectTo( f );
    ocp.subjectTo(      3.0 <= u(0) <= 35.0  );
    ocp.subjectTo( -9000.0 <= u(1) <=  0.0  );

    // SETUP MPC CODE EXPORT AND GENERATE THE CODE:
    // --------------------------------------------
    MPCexport mpc( ocp );

    mpc.set( DISCRETIZATION_TYPE,    SINGLE_SHOOTING );
    mpc.set( HESSIAN_APPROXIMATION, GAUSS_NEWTON     );
    mpc.set( INTEGRATOR_TYPE,        INT_RK4         );
    mpc.set( NUM_INTEGRATOR_STEPS,   40              );
    mpc.set( QP_SOLVER,              QP_QPOASES      );
    mpc.set( HOTSTART_QP,            NO              );
    mpc.set( GENERATE_TEST_FILE,     YES             );
    mpc.set( GENERATE_MAKE_FILE,     YES             );

    mpc.exportCode( "cstr_export" );

    return 0;
}
```

with $s_i \in \mathbb{R}^{n_\mathrm{x}}$, $q_i \in \mathbb{R}^{n_\mathrm{u}}$ and $f : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{u}} \rightarrow \mathbb{R}^{n_\mathrm{x}}$, be given and let us assume that a unique solution exists. Then, under mild regularity conditions, the following matrix-valued initial value problems

$$G_s(t_i) = \mathrm{I}_{n_\mathrm{x}}\,, \tag{8.3a}$$

$$\dot{G}_s(t) = \nabla_x f\left(x(t), q_i\right)' G_s \quad \forall\, t \in [t_i, t_{i+1}] \tag{8.3b}$$

and

$$G_q(t_i) = 0\,, \tag{8.4a}$$

$$\dot{G}_q(t) = \nabla_x f\left(x(t), q_i\right)' G_q + \nabla_{q_i} f\left(x(t), q_i\right)' \quad \forall\, t \in [t_i, t_{i+1}] \tag{8.4b}$$

also have unique solutions [52], which satisfy:

$$G_s(t_{i+1}) = \frac{\partial x}{\partial s_i}(t_{i+1})\,, \qquad G_q(t_{i+1}) = \frac{\partial x}{\partial q_i}(t_{i+1})\,. \tag{8.5}$$

Thus, first-order sensitivities can be obtained by integrating these additional $n_\mathrm{x} \times n_\mathrm{x}$ ODEs (8.3) and $n_\mathrm{u} \times n_\mathrm{x}$ ODEs (8.4).

The `ACADO Code Generation` tool generates an optimised C function for evaluating the original ODE system as well as the corresponding variational differential equations (8.3) and (8.4). The required gradients are calculated and symbolically simplified beforehand using the automatic differentiation features of the `ACADO Toolkit`.

In a second step, a tailored explicit Runge-Kutta integrator is generated for integrating the resulting augmented ODE system. In order to guarantee a deterministic runtime of the online integrator, the `ACADO Code Generation` tool only supports the choice of fixed step-sizes. However, this requires a careful tuning of these step-sizes in offline simulations in order to obtain the desired integration accuracy. Moreover, explicit Runge-Kutta integrators are not suited for integrating stiff ODE systems [121]. Consequently, only nonstiff ODE systems can be handled by the current implementation of the `ACADO Code Generation` tool.

From a computational point of view, Runge-Kutta integrators whose Butcher tableau contains many zero entries are preferable. For example, exploiting the four zero-entries in the Butcher tableau of order 4 (see Figure 8.1) reduces the computational load of each integrator step by about one third. At the same time, a fourth order Runge-Kutta integrator usually leads to sufficient integration accuracy with a relatively small number of integrator steps. Thus, it is used as default integrator.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Figure 8.1: Butcher tableau for an explicit Runge-Kutta integrator of order four for fixed step-sizes.

8.2.3 Solving the Linearised Subproblem

The `ACADO Code Generation` tool automatically generates source code which implements a Gauss-Newton real-time iteration algorithm based on a single or multiple shooting discretisation. For solving the linearised subproblem, a state-elimination approach in combination with a dense QP solver has been chosen (see Subsection 6.3.1). This choice was mainly made because both available online QP solvers, that are suited for embedded applications, are dense: the QP solvers generated by `CVXGEN` as well as an embedded variant of `qpOASES`. Nevertheless, the presented ideas for auto-generating real-time iteration algorithms could also be combined with a sparse QP solver for solving the linearised subproblem. This would lead to even more efficient code in case the prediction horizon is long (see also the discussion in Subsection 2.1.4).

In order to obtain the dense QP (7.14), a condensing routine is generated that performs the required matrix-vector and matrix-matrix operations. It turns out that auto-generating these operations can make them significantly faster than generic implementation. This is due to the fact that all dimensions can be hard-coded which avoids dynamic memory allocation. Moreover, (inner) multiplication-loops can be unrolled leading to linear code without any conditional statements. Finally, certain operations can be made more efficient by hard-coding known numerical values or sparsity patterns (e.g. in the case of constant diagonal weighting matrices in the objective function).

For solving the resulting small-scale, dense and convex QP, the `ACADO Code Generation` tool interfaces two QP solvers based on different algorithmic strategies:

1. The first one is a primal-dual interior-point solver as auto-generated by the software `CVXGEN` [169]. The exported algorithm is implemented in highly efficient plain C code that only makes use of static memory. Moreover, interior-point algorithms exhibit the advantage of relatively constant calculation times for each occurring QP [40], which is particularly desirable in embedded applications.

2. Active-set algorithms form a second class of suitable QP solvers, thus also the open-source package qpOASES as described in Chapter 4 is interfaced. Following the guidelines given in Subsection 5.1.2, an embedded variant of qpOASES has been interfaced to the ACADO Code Generation. It uses hard-coded QP dimensions and allocates static memory only. Moreover, it automatically uses the more efficient solution variant for box-constrained QPs in case the MPC formulation does not comprise state bounds. Calculation times of active-set solvers strongly depend on the number of required active-set changes. But as each active-set iteration is much faster than an interior-point iteration, active-set solvers can significantly outperform IP methods for specific applications if dedicated hot-starting procedures turn out to work well.

8.2.4 The Auto-Generated Code

Once an MPC problem has been formulated, as illustrated in Listing 8.1, it can be passed to the MPCexport class of the ACADO Code Generation tool[2]. This class will generate optimised C code taking into account a couple of user-defined options (or default values).

The source code generated by the ACADO Code Generation tool is fully based on hard-coded dimensions and uses static global memory only. Thus, there are no malloc/free or new/delete statements, which ensures that we cannot possibly encounter any segmentation faults or out-of-memory errors while running the algorithm on embedded controller hardware. It also avoids loops whenever reasonable in order to ensure maximum efficiency. Moreover, except for the QP solver, the generated code does not contain any conditional statements. Besides the increased efficiency, this guarantees that the program cannot run into a part of code which has accidentally never been tested.

In order to illustrate how the exported code looks like, Listing 8.2 shows a snap-shot of an automatically generated initial value embedding step in plain C code. One might still be able to guess that this piece of code implements a hard-coded matrix-vector multiplication for a system with four states. However, auto-generated code is usually hard to read by a human as it is not designed to be easily readable but to be efficient and reliable. Of course, this comes at the price that debugging the exported code becomes very difficult and has to be done at the level of the code exporting functions instead.

[2]This class provides a user-interface and is thus derived from the UserInteraction class (see Figure 7.2). It organises the code export relying on the code generation functionality of lower-level classes.

Listing 8.2: A snap-shot of automatically generated code: hard to read but performing efficiently (taken from [129]).

```
void initialValueEmbedding( ){
params.g[0] = acadoWorkspace.g[4] +
acadoWorkspace.H[4]*acadoWorkspace.deltaY[0] +
acadoWorkspace.H[18]*acadoWorkspace.deltaY[1] +
acadoWorkspace.H[32]*acadoWorkspace.deltaY[2] +
acadoWorkspace.H[46]*acadoWorkspace.deltaY[3];
params.g[1] = acadoWorkspace.g[5] +
acadoWorkspace.H[5]*acadoWorkspace.deltaY[0] +
/* ... */
```

The `ACADO Code Generation` tool can export the following files:

- Source and header files containing the augmented ODE system, the fixed-step integrator, the condensing routine, the Gauss-Newton real-time iteration algorithm based on a shooting method as well as an interface to the specified online QP solver;
- a sample main function for testing the exported code;
- a `Makefile` for facilitating compilation of the exported code;
- a tailored SIMULINK interface for compiling the exported code into a C MEX S-function.

Moreover, the `ACADO Code Generation` tool offers an option to export code using single precision arithmetic and also the embedded variant of `qpOASES` can run in single precision. This can be advantageous for at least two reasons: First, certain hardware platforms simply do not support double precision arithmetic or need to emulate it in software, which can increase computation times significantly. Second, many microprocessors can perform considerably faster in single precision arithmetic; speed-ups of up to a factor of two on Intel's Pentium IV or even up to a factor of ten on IBM's Cell Broad Engine processor have been reported in [154]. However, the use of single precision arithmetic limits the applicability of (exported) optimisation code to more well-conditioned problem formulations, also because round-off errors become more critical.

The numerical examples presented in Subsection 8.3 actually show that code exported in single precision executes faster than its double precision counterpart. Thus, the possibility to export the NMPC algorithm in different floating point representations gives the user some freedom to trade-off solution accuracy and runtime complexity.

8.3 Performance of the Generated Code

We illustrate the performance of the auto-generated code with a couple of examples and discuss briefly how its computational complexity scales with changing problem dimensions. All simulations have been performed on a standard PC (Intel Core 2 Duo P9700) having a 2.8 GHz dual-core processor and 4 GB RAM.

8.3.1 Start-Up of a CSTR (Revisited)

First of all, we apply the `ACADO Code Generation` tool to export optimised C code for controlling the start-up of a continuous stirred tank reactor (CSTR) model. We use the same dynamic model and MPC scenario as described in Subsection 7.3.1; the corresponding symbolic formulation for exporting the tailored real-time iteration algorithm is given in Listing 8.1. We use 40 integrator steps of fixed length—which turns out to provide sufficient integration accuracy—and employ the embedded variant of `qpOASES` as online QP solver.

As the resulting control performance is very similar to the one depicted in Figure 7.6, we only discuss the computational performance as summarised in Table 8.1. When comparing these figures with the ones given in Table 7.1, we can see that one complete real-time iteration of the auto-generated NMPC algorithm only took about 620 microseconds, thus being about 24 times faster than the generic implementation of the `ACADO Toolkit`. Employing single precision arithmetic as discussed in Subsection 8.2.4 would further reduce the runtime to about 550 microseconds. A significant speed-up can be observed for all algorithmic components. In particular, the auto-generated RK integrator with fixed step-size performs more than 40 times faster than its generic counterpart with adaptive step-size control. Also the condensing routine (about 14 times faster) and `qpOASES` gain from hard-coding problem dimensions and using static memory only. The embedded variant of `qpOASES` also performs faster because, unlike in the generic implementation of the `ACADO Toolkit`, the special solution variant for box-constrained QPs is automatically employed.

Note that the reported runtimes for `qpOASES` correspond to six active-set changes; each additional active-set change would require an extra runtime of about 15 microseconds. Moreover, Table 8.1 indicates that eliminating the states from the QP took a big share of the computational load of each real-time iteration. This is because the chosen control horizon of 20 intervals is already quite long compared to the state and control dimensions. Thus, real-time iterations based on a sparse QP solution could probably be made even faster.

	CPU time	%
Integration and sensitivity generation	0.24 ms	38 %
Condensing	0.22 ms	35 %
QP solution (with qpOASES)[3]	0.15 ms	25 %
Remaining operations	0.01 ms	2 %
One complete real-time iteration	0.62 ms	100 %

Table 8.1: Worst-case runtime for controlling the CSTR start-up when performing Gauss-Newton real-time iterations based on auto-generated code.

Based on the presented CSTR model, results of a nominal MPC simulation involving several set-point changes have been reported in [129]. Therein, a modified problem formulation comprising only 10 control intervals has been used and only 20 integrator steps were performed along the prediction horizon. Using the same PC hardware as our simulations, the worst-case runtime per real-time iteration reduced to about 175 microseconds.

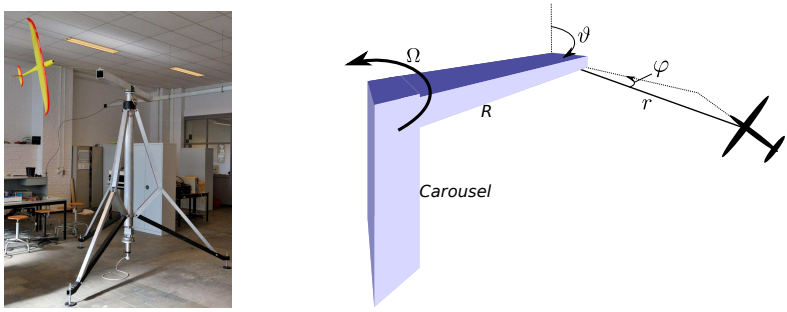
In order to illustrate that the ACADO Code Generation tool can also export code for MPC problem formulations comprising state constraints, the MPC scenario in [129] also introduced lower bounds on ϑ and ϑ_K. Moreover, one of the set-points were chosen infeasible with respect to these state constraints. This led to many active constraints causing qpOASES to perform up to 21 active-set changes in order to find the optimal solution to the dense QP. Still the worst-case runtime per real-time iteration increased only moderately to about 365 microseconds.

8.3.2 Real-Time Control of a Kite Carousel Model

A second example for illustrating the computational performance of auto-generated real-time iteration algorithms has been given in [81]. Therein, the ACADO Code Generation tool has been applied to a simplified model of a kite carousel, which will form the basis for future experiments at a real-world prototype (see Figure 8.2(a)). We briefly summarise the results and refer to [81] for all details.

The setup to be controlled aims at using kites for wind power generation as previously motivated by [164]. The kite, anchored to a ground-based generator, delivers a high force on the tether while it is being unrolled. This drives the generator to produce electricity. A kite carousel prototype has been built at K.U. Leuven in order to test control systems for kites. It consists of a rotating device driving an aeroplane (instead of a kite) which is attached to one of its arms (see Figure 8.2(b)). As the whole carousel construction exhibits nonlinear

[3] In Subsection 4.5.2 also the runtimes for CVXGEN are given.



(a) Experimental setup. (b) Sketch of the carousel model.

Figure 8.2: Picture and sketch of the prototype kite carousel at K.U. Leuven (both taken from [81]).

dynamics while the aeroplane (or kite) is moving relatively fast along its orbits, controlling the system reliably is challenging.

The kite carousel is described by a simplified, but still quite involved nonlinear ODE model comprising four differential states (the spherical coordinates φ and ϑ of the plane and the associated angular velocities $\dot{\varphi}$ and $\dot{\vartheta}$ and two control inputs effecting pitch and roll of the plane [81]:

$$\ddot{\vartheta} = \frac{1}{r}\left[R\Omega^2\cos(\vartheta)\cos(\varphi) + r(\Omega+\dot{\varphi})^2\sin(\vartheta)\cos(\vartheta) + g\sin(\vartheta) + \frac{F_{\vartheta}^{\text{aer}}}{m}\right], \quad (8.6a)$$

$$\ddot{\varphi} = \frac{1}{r\sin(\vartheta)}\left[-2r(\Omega+\dot{\varphi})\dot{\vartheta}\cos(\vartheta) - R\Omega^2\sin(\phi) + \frac{F_{\varphi}^{\text{aer}}}{m}\right] \quad (8.6b)$$

where g is the gravitational constant, R the length of the carousel arm, which rotates with a constant angular velocity Ω in a horizontal plane. Moreover, the aerodynamic forces denoted by F^{aer} are modelled in the form

$$F^{\text{aer}} = \frac{\rho A\|w_{\text{e}}\|^2}{2}\,\psi(x,u)\,, \quad (8.7)$$

with ρ being the density of the air, A the wing area, w_{e} the effective wind at the plane in local coordinates, and $\psi : \mathbb{R}^4 \times \mathbb{R}^2 \to \mathbb{R}^2$ a function depending nonlinearly on the states and controls. The effective wind can be written as

$$w_{\text{e}} = \begin{pmatrix} \dot{\vartheta}r + R\Omega\cos(\vartheta)\sin(\varphi) \\ r(\Omega+\dot{\varphi})\sin(\vartheta) + R\Omega\cos(\varphi) \\ R\Omega\sin(\vartheta)\sin(\varphi) \end{pmatrix}. \quad (8.8)$$

Motivated by the fact that the aerodynamic force F^{aer} is mainly influenced by the sum of a lift component, which is used to pull the plane in forward direction, and a drag component, which breaks the plane in the opposite direction, ψ is approximated by

$$\psi\,(x,u) \;\approx\; \begin{pmatrix} -C_{\text{L}}u_1 - b\dot{\vartheta} \\ -C_{\text{D}}(1+\alpha_1 u_2) - C_{\text{L}}(1+\alpha_2 u_2)\varphi \end{pmatrix}, \tag{8.9}$$

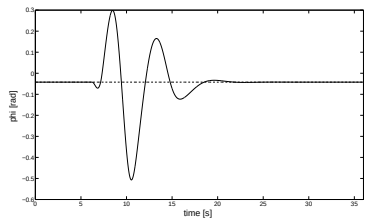
with C_{L} and C_{D} denoting the nominal lift and drag coefficient, b a roll stabilisation coefficient, and α_1 and α_2 the relative influence of the pitch control u_2 on the plane's lift-over-drag efficiency. The other control variable u_1, is assumed to adjust the roll of the plane.

For testing the closed-loop behaviour of the auto-generated Gauss-Newton real-time iteration algorithm, we aim at tracking a given steady-state reference while an unknown disturbance occurs. For doing so, an MPC problem comprising a control horizon of 2π seconds (which amounts to the time the carousel arm needs to complete one full turn) divided into 10 equidistant control intervals is formulated. Moreover, it formulates upper and lower limits on both control inputs as well as a tracking objective function comprising a terminal penalty weight.

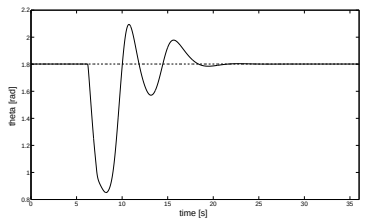
Figure 8.3 shows the closed-loop results simulating the following MPC scenario with a sampling time of 50 milliseconds: We start at the reference position and no disturbance occurs during the first 2π seconds. At the time 2π seconds an external wind gust has been simulated, which is switched off again after 1 second. For the chosen model parameters (see [81]), this wind gust of $1.5\frac{\text{m}}{\text{s}}$ is large enough to cause a crash of the kite if it would fly open-loop.

Using a standard PC with a 2.8 GHz processor and 4 GB RAM, performing one real-time iteration for the kite carousel formulation never took more than 0.76 milliseconds and was sufficient to track the desired set points. Table 8.2 summarises the worst-case runtimes of all algorithmic components for a complete real-time iteration. Employing single precision arithmetic would further reduce the runtime to about 0.65 milliseconds. Note that the computation times obtained by employing the auto-generated NMPC algorithm are orders of magnitude faster than the ones reported in [133]. Therein, one real-time iteration took about 300 milliseconds using an only slightly more complicated ODE model.

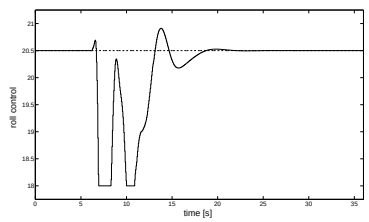
The major share of the computation was spent within the auto-generated Runge-Kutta integrator (using a fixed number of 30 steps). This is due to the lengthy mathematical expressions (8.6)–(8.9) required to formulate the nonlinear ODE model. The reported QP runtimes obtained with the embedded variant of `qpOASES` correspond to four active-set changes as required when warm-starting the QP; cold-started QP solution would have required up to 14 iterations taking about 0.1 milliseconds.



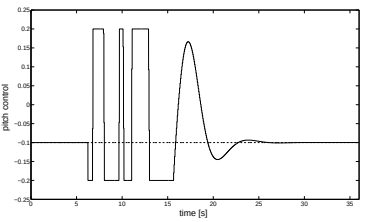
(a) Angular position φ.



(b) Angular position ϑ.



(c) Roll control u_1.



(d) Pitch control u_2.

Figure 8.3: Simulated states and optimised control inputs of the kite carousel (solid lines) together with their respective reference values (dotted lines); similar to [81] but with a larger wind disturbance of $1.5\frac{\mathrm{m}}{\mathrm{s}}$ and corrected subfigure titles.

	CPU time	%
Integration and sensitivity generation	0.59 ms	78 %
Condensing	0.09 ms	12 %
QP solution (with `qpOASES`)[4]	0.04 ms	5 %
Remaining operations	0.04 ms	5 %
One complete real-time iteration	0.76 ms	100 %

Table 8.2: Worst-case runtime of the auto-generated real-time iteration algorithm applied to the kite carousel model (with QP warm-starts).

8.3.3 Scalability

The previous examples show that auto-generated code can be executed very fast for small-scale MPC problem formulations. Thus, the question naturally arises how its performance scales with the problem dimensions, namely the number of

[4] In Subsection 4.5.2 also the runtimes for `CVXGEN` are given.

states n_{x}, the number of control inputs n_{u} and the length of the prediction/control horizon n_{p}.

Integration and sensitivity generation requires to perform a fixed number of evaluations of the augmented ODE system. Thus, its computational effort grows quadratically in n_{x} and linearly in n_{u}. The effort also grows linearly in n_{p}, if we make the reasonable assumption that the number of integrator steps is proportional to the horizon length. State-elimination as implemented in the condensing routine has cubic complexity in n_{x} and quadratic in n_{u} and n_{p}, respectively. The effort for solving the resulting dense QP grows quadratically to cubically in n_{u} and n_{p} and does not depend on the number of states. All remaining operations require at most $\mathcal{O}(n_{\mathrm{x}}^2 + n_{\mathrm{u}}^2 + n_{\mathrm{p}})$ operations.

These considerations show that code size and execution time will increase significantly when the number of states becomes larger. The quadratic to cubic growth in the number of control inputs and the horizon length could be avoided by directly solving the sparse QP subproblem, instead of eliminating the states. However, for moderate problem dimensions—say, $n_{\mathrm{x}} \leq 10$, $1 \leq n_{\mathrm{u}} \leq \frac{n_{\mathrm{x}}}{2}$, and $n_{\mathrm{p}} \leq 5\frac{n_{\mathrm{x}}}{n_{\mathrm{u}}}$—the NMPC algorithms as currently auto-generated by the `ACADO Code Generation` tool should be very efficient.

Chapter 9

Conclusions

This last chapter concludes both parts of this thesis and discusses possible directions for future research.

9.1 Linear MPC

9.1.1 Summary

Linear MPC has been proven to be a very useful feedback control strategy within many practical applications during the last decades, in particular in the process industry. Thus, researchers and practitioners keep on aiming at extending its scope to new application areas or to processes with more challenging time-scales. Linear MPC requires to solve a sequence of optimal control problems that can be expressed as convex quadratic programs. These QPs have a common mathematical structure for a given problem formulation, namely their gradients and constraint vectors depend parametrically on the current system state. When applying linear MPC to processes that ask for high sampling rates, this specific structure often needs to be exploited for solving such QPs in real-time.

The online active set strategy is a dedicated QP algorithm that has been developed to exploit this parametric dependency for speeding-up computations. It allows for efficient hot-starting of the QP solution procedure based on solution information of the previous problem (including matrix factorisations). Moreover, it allows for a transparent interpretation of intermediate iterates in case the QP solution needs to be stopped prematurely. Chapter 3 reviewed the main ideas of the online active

set strategy and presented novel extensions for initialising the solution procedure and treating QPs with semi-definite Hessian matrices.

Chapter 4 presented the software package `qpOASES`, an efficient and reliable open-source implementation of the online active set strategy. It is written as self-contained C++ code and implements a number of tailored solution variants for special QP formulations. Moreover, it offers user-friendly interfaces to third-party software packages like MATLAB and SIMULINK; the latter also facilitates using the software on dSPACE and xPC hardware. `qpOASES` has been successfully used in a number of academic and industrial real-world MPC applications, sometimes outperforming general-purpose QP solvers significantly. As it is based on dense linear algebra routines, `qpOASES` is most suited for small- to medium-scale MPC formulations with control horizons of moderate length.

Chapter 5 discussed two industrial case studies to which `qpOASES` has been successfully applied: The first one, MPC-based emission control of integral gas engines, required to solve small-scale QPs in a small fraction of the sampling time of 100 milliseconds on an embedded PowerPC (whose computational speed is much lower than that of a standard PC). Based on experience gathered with this and other embedded applications, a couple of programming guidelines for writing embedded optimisation software were given. Also a modified MPC formulation based on an asymmetric cost function has been proposed to increase control performance for processes that can be suitably described by Wiener models. Though this is only a minor theoretical contribution, it can be of significant practical relevance for certain processes, like the described integral gas engines. The second case study outlines several existing strategies to handle infeasible MPC problems; the focus lies on MPC problems as arising in the process industry, where prioritised constraint satisfaction is important. As infeasibility handling is a crucial ingredient of basically all practical MPC schemes, a novel method based on the special properties of the online active set strategy has been developed. It keeps computational overhead small as it introduces slack variables only if needed and weights these slacks in the objective function adaptively.

9.1.2 Directions for Future Research

When discussing possible directions of future research, one should be aware that many different algorithms for efficiently solving linear MPC problems have been proposed during the last decades. Most of them focus on a certain sub-class of problems with special properties. For example, explicit MPC methods have turned out to be extremely efficient but limited to MPC formulations comprising not more than a couple of states or control inputs. Interior-point methods offer the great advantage of relatively constant and predictable computational load but might easily be outperformed by active-set methods if hot-starting works

well for a specific application. Fast gradient methods seem to offer practical runtime guarantees but their efficient use is currently limited to MPC formulations without state constraints. Structure-exploiting linear algebra is usually tailored to problems with a long or a short prediction horizon, respectively. By exploiting the special rank structure of the QP matrices, it might be possible to further speed-up the QP solution in case of long prediction horizons.

Thus, there already exist suitable algorithms for many possible real-world applications of linear MPC and often only a reliable and efficient implementation of them is missing. Future research should further investigate the limitations of each respective approach and possible combinations of them. This can lead to new algorithms that find better trade-offs between practical aspects such as solution speed, solution accuracy, runtime/stability guarantees, implementation effort, flexibility of the implementation and others. All these aspects are influenced by the controller hardware used for running the MPC algorithm. For example, multi-core architectures or graphics processing units (GPUs) ask for algorithms that can be easily parallelised to obtain high efficiency. Therefore, there is need for adapting existing algorithms to match these additional requirements imposed by the controller hardware. Moreover, parallelisation might also be the most promising way for further significant reduction of the computation time, as most traditional approaches for solving QPs seem to have been pushed close to their efficiency limits with respect to exploiting the MPC problem structure.

Besides these computational issues, the question of how to address infeasible MPC problems remains an important practical question. Usually one of the mentioned techniques to relax the resulting infeasible QPs is employed. However, from a system theoretic viewpoint, it does not seem to be clear at all how these methods need to be tuned in order to ensure an acceptable and stable system behaviour. As the possibility to directly impose constraints is one of the main advantages of MPC, it is surprising that this issue has been mostly neglected in literature (except for several ad-hoc approaches).

9.2 Nonlinear MPC

9.2.1 Summary

Though nonlinear MPC theory and algorithms are still in a less mature state than their linear counterparts, they are being applied more frequently in practice. This is often done by using extensions of existing linear MPC schemes. In most cases only nonlinear system dynamics are incorporated—allowing for more accurate models of the controlled process—but the remaining constraints are linear and a convex quadratic (tracking) objective function is used. Thus, in the simplest

case, one might simply linearise the ODE system at each sampling instant to yield a time-varying linear MPC problem that can be solved reliably with techniques mentioned in the first part of this thesis. Chapter 6 surveyed several direct NMPC schemes that extend this basic idea to yield algorithms that find more accurate approximate solutions to the nonlinear optimisation problem or enjoy better convergence properties. It also addressed a number of numerical issues—such as derivative computation, sparsity exploitation or online initialisations—that are important to yield efficient implementations. Also ideas to reduce the feedback delay were summarised.

Chapter 7 introduced the `ACADO Toolkit`, a novel open-source software package for automatic control and dynamic optimisation. It implements efficient algorithms for optimal control and model predictive control and also supports parameter estimation and multi-objective optimisation. Problem formulations can make use of an intuitive symbolic syntax, which is a key feature of the `ACADO Toolkit`. This does not only increase user-friendliness but also allows for a couple of algorithmic possibilities like automatic/symbolic differentiation, convexity detection or structure exploitation. The `ACADO Toolkit` also implements two algorithmic variants of the real-time iteration scheme: a Gauss-Newton approach for NMPC formulations involving a tracking objective function as well as an exact Hessian approach for tackling time-optimal formulations. Computational performance of these algorithms, which both employ `qpOASES` as underlying QP solver, has been illustrated with a small-scale benchmark example.

Based on the symbolic syntax of the `ACADO Toolkit`, an open-source tool for automatically generating Gauss-Newton real-time iteration algorithms has been presented in Chapter 8. The `ACADO Code Generation` tool allows the user to export highly efficient and self-contained C code that is tailored to each respective MPC problem formulation. Computational speed is increased by hard-coding all problem dimensions, avoiding dynamic memory allocations, loop unrolling, symbolic simplifications and the use of a fixed-step integrator. This can lead to significant speed-ups compared to generic implementations as illustrated with numerical examples. It was shown that the exported algorithms in their current form are most suited for small-scale NMPC problems comprising non-stiff ODE models. However, the basic ideas behind generating customised optimisation algorithms seem to be extendible to more general problem classes.

9.2.2 Directions for Future Research

As mentioned before, most NMPC algorithms are designed for problem formulations comprising a tracking objective function. However, from a practical viewpoint, the minimisation of an arbitrary *economic* objective function would often be of greater interest. An important special case of such non-tracking

objectives are formulations that minimise the time to reach a certain process state. Thus, future research should further develop theory and efficient algorithms for NMPC with non-tracking objective functions.

A second remark concerns the available nonlinear MPC software. While many mature implementations of general-purpose NLP solvers exist, there still seems to be a lack of efficient and reliable codes for optimal control and NMPC. As it is problem-dependent which combination of algorithmic components is most suited, it would be highly desirable if such implementations are written in an open and extensible form with well-defined interfaces. The development of the `ACADO Toolkit` should be seen as a first step into this important direction, though a single code will never be able to provide all desired features. In any case, there is strong evidence that problem formulations based on symbolic syntax offer many important advantages, in particular regarding user-friendliness and automated structure exploitation.

Code generation seems to be a promising technique for linear and nonlinear MPC applications, in particular for applications on embedded hardware. First results for small-scale problem formulations are very encouraging; nevertheless, this approach still has to prove its usefulness for larger-scale problems. In case of longer prediction horizons, auto-generating or coupling an embedded solver for sparse QPs will probably provide better results. Moreover, future research should further investigate possibilities to optimise the exported code for a given processor and cache architecture. Furthermore, it is currently not clear which parts of the code optimisation can be reasonably done by high-level code generation tools and which parts should rather be left to the compiler. Finally, one should explore whether it might make sense to auto-generate only certain parts of an optimisation algorithm and to leave the remaining parts generic in order to reduce the size of the exported code.

Bibliography

[1] HOERBIGER Holding AG. Company webpage. http://www.hoerbiger.com, 2008–2011.

[2] D. Alberer, H. Kirchsteiger, L. del Re, H.J. Ferreau, and M. Diehl. Receding horizon optimal control of Wiener systems by application of an asymmetric cost function. In *IFAC Workshop on Control Applications of Optimisation, Agora, Finland*, 2009.

[3] J. Albersmeyer and H.G. Bock. Sensitivity generation in an adaptive BDF-method. In *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 6-10, 2006, Hanoi, Vietnam*, 2008.

[4] J. Albersmeyer and H.G. Bock. Sensitivity Generation in an Adaptive BDF-Method. In H.G. Bock, E. Kostina, X.H. Phu, and R. Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 6-10, 2006, Hanoi, Vietnam*, pages 15–24. Springer, 2008.

[5] A. Alessio and A. Bemporad. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter A Survey on Explicit Model Predictive Control, pages 345–369. Springer, 2009.

[6] E. Anderson, Z. Bai C., Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[7] J. Ängeby, M. Huschenbett, and D. Alberer. *Automotive Model Predictive Control*, volume 402 of *Lecture Notes in Control and Information Sciences*, chapter MIMO Model Predictive Control for Integral Gas Engines, pages 257–272. Springer, 2010.

[8] MOSEK ApS. MOSEK webpage. http://www.mosek.com, 2011.

[9] E. Arnold, J. Neupert, O. Sawodny, and K. Schneider. Trajectory tracking for boom cranes based on nonlinear control and optimal trajectory generation. In *IEEE International Conference on Control Applications*, pages 1444–1449, 2007.

[10] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential–Algebraic Equations*. SIAM, Philadelphia, 1998.

[11] D. Axehill. *Applications of Integer Quadratic Programming in Control and Communication*. PhD thesis, Linköpping University, 2005.

[12] V. Azhmyakov and J. Raisch. Convex Control Systems and Convex Optimal Control Problems With Constraints. *IEEE Transactions on Automatic Control*, 53(4):993–998, 2008.

[13] R.A. Bartlett and L.T. Biegler. QPSchur: A Dual, Active Set, Schur Complement Method for Large-scale and Structured Convex Quadratic Programming Algorithm. *Optimization and Engineering*, 7:5–32, 2006.

[14] R.A. Bartlett, L.T. Biegler, J. Backstrom, and V. Gopal. Quadratic programming algorithms for large-scale model predictive control. *Journal of Process Control*, 12:775–795, 2002.

[15] I. Bauer. *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik*. PhD thesis, Universität Heidelberg, 1999.

[16] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[17] A. Bemporad and C. Filippi. Suboptimal Explicit Receding Horizon Control via Approximate Multiparametric Quadratic Programming. *Journal of Optimization Theory and Applications*, 117(1):9–38, 2003.

[18] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38:3–20, 2002.

[19] A.B. Berkelaar, K. Roos, and T. Terkaly. *Recent Advances in Sensitivity Analysis and Parametric Programming*, chapter 6: The Optimal Set and Optimal Partition Approach to Linear and Quadratic Programming. Kluwer Publishers, Dordrecht, 1997.

[20] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Belmont, MA, 1995.

[21] M.J. Best. *Applied Mathematics and Parallel Computing*, chapter An Algorithm for the Solution of the Parametric Quadratic Programming Problem, pages 57–76. Physica-Verlag, Heidelberg, 1996.

[22] J.T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, Philadelphia, 2001.

[23] J.T. Betts. *Practical Methods for Optimal control and Estimation Using nonlinear Programming*. SIAM, 2nd edition, 2010.

[24] Lorenz T. Biegler. *Nonlinear Programming*. MOS-SIAM Series on Optimization. SIAM, 2010.

[25] L.T. Biegler. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Computers and Chemical Engineering*, 8:243–248, 1984.

[26] L.T. Biegler and J.B Rawlings. Optimization approaches to nonlinear model predictive control. In W.H. Ray and Y. Arkun, editors, *Proc. 4th International Conference on Chemical Process Control - CPC IV*, pages 543–571. AIChE, CACHE, 1991.

[27] T. Binder, L. Blank, H.G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J.P. Schlöder, and O.v. Stryk. Introduction to Model Based Optimization of Chemical Processes on Moving Horizons. In M. Grötschel, S.O. Krumke, and J. Rambau, editors, *Online Optimization of Large Scale Systems: State of the Art*, pages 295–340. Springer, 2001.

[28] C.H. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR Generating derivative codes from Fortran programs. *Scientific Programming*, 1:11–29, 1992.

[29] R.R. Bitmead, M. Gevers, and V. Wertz. *Adaptive optimal control: the thinking man's GPC*. Prentice Hall, Sydney, 1990.

[30] J. Björnberg and M. Diehl. Approximate robust dynamic programming and robustly stable MPC. *Automatica*, 42(5):777–782, May 2006.

[31] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, and et. al. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Soft.*, 28:135–151, 2002.

[32] H.H.J. Bloemen, T.J.J. Van Den Boom, and H.B. Verbruggen. Model-based predictive control for Hammerstein-Wiener systems. *Int. J. Control*, 74(5):482–95, 2001.

[33] H.G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser, Boston, 1983.

[34] H.G. Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, volume 183 of *Bonner Mathematische Schriften*. Universität Bonn, Bonn, 1987.

[35] H.G. Bock, M. Diehl, E.A. Kostina, and J.P. Schlöder. Constrained Optimal Feedback Control of Systems Governed by Large Differential Algebraic Equations. In L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, editors, *Real-Time and Online PDE-Constrained Optimization*, pages 3–22. SIAM, 2007.

[36] H.G. Bock, M. Diehl, D.B. Leineweber, and J.P. Schlöder. Efficient direct multiple shooting in nonlinear model predictive control. In F. Keil, W. Mackens, H. Voß, and J. Werther, editors, *Scientific Computing in Chemical Engineering II*, volume 2, pages 218–227, Berlin, 1999. Springer.

[37] H.G. Bock, M. Diehl, D.B. Leineweber, and J.P. Schlöder. A direct multiple shooting method for real-time optimization of nonlinear DAE processes. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*, pages 246–267, Basel Boston Berlin, 2000. Birkhäuser.

[38] H.G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.

[39] N. L. Boland. A dual-active-set algorithm for positive semi-definite quadratic programming. *Mathematical Programming*, 78:1–27, 1997.

[40] S. Boyd and L. Vandenberghe. *Convex Optimization*. University Press, Cambridge, 2004.

[41] W. Van Brempt, P. Van Overschee, T. Backx, J. Ludlage, P. Hayot, L. Oostvogels, and S. Rahman. Grade-change control using INCA model predictive controller: application on a dow polystyrene process model. In *Proceedings of the IEEE American Control Conference, Denver, Colorado*, pages 5411–5416, 2003.

[42] C.G. Broyden. The convergence of a class of double rank minimization algorithms, part I and II. *J. Inst. Maths. Applns.*, 6:76–90 and 222–231, 1970.

[43] A.E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Wiley, New York, 1975.

[44] C. Büskens and H. Maurer. SQP-methods for solving optimal control problems with control and state constraints: adjoint variables, sensitivity analysis and real-time control. *Journal of Computational and Applied Mathematics*, 120:85–108, 2000.

[45] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. KNITRO: An integrated package for nonlinear optimization. In Gianni Pillo and Massimo

Roma, editors, *Large Scale Nonlinear Optimization*, pages 35–59. Springer Verlag, 2006.

[46] E.F. Camacho and C. Bordons. *Model Predictive Control*. Springer, 2nd edition, 2007.

[47] M. Cannon, W. Liao, and B. Kouvaritakis. Efficient mpc optimization using pontryagin's minimum principle. *International Journal of Robust and Nonlinear Control*, 18(8):831–844, 2008.

[48] Mark Cannon. Efficient nonlinear model predictive control algorithms. *Annual Reviews in Control*, 28:229–237, 2004.

[49] H. Chen. *Stability and Robustness Considerations in Nonlinear Model Predictive Control*. Fortschr.-Ber. VDI Reihe 8 Nr. 674. VDI Verlag, Düsseldorf, 1997.

[50] H. Chen and F. Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1218, 1998.

[51] H. Chen, A. Kremling, and F. Allgöwer. Nonlinear predictive control of a benchmark CSTR. In *Proc. 3rd European Control Conference ECC'95*, pages 3247–3252, Rome, 1995.

[52] E.A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. McGraw-Hill, New York, 1955.

[53] IBM Corp. *IBM ILOG CPLEX V12.1, User's Manual for CPLEX*, 2009.

[54] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[55] T.A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[56] N.M.C. de Oliveira and L.T. Biegler. Constraint Handling and Stability Properties of Model-Predictive Control. *AIChE Journal*, 40(7):1138–1155, 1994.

[57] B. Defraene, T. van Waterschoot, H.J. Ferreau, M. Diehl, and M. Moonen. Perception-based clipping of audio signals. In *Proceedings of the 18th European Signal Processing Conference (EUSIPCO '10), Aalborg, Denmark*, pages 517–521, 2010.

[58] S. Delvaux and M. Van Barel. A givens-weight representation for rank structured matrices. *SIAM J. Matrix Anal. Appl.*, 29:1147–1170, 2007.

[59] D. Detlefs, A. Dosser, and B. Zorn. Memory allocation costs in large C and C++ programs. *Software: Practice and Experience*, 24(6):527–542, 1994.

[60] P. Deuflhard. *Newton Methods for Nonlinear Problems*. Springer, New York, 2004.

[61] P. Dewilde and A.-J. van der Veen. *Time-varying systems and computations*. Kluwer Academic Publishers, Boston, 1998.

[62] M. Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*, volume 920 of *Fortschr.-Ber. VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik*. VDI Verlag, Düsseldorf, 2002.

[63] M. Diehl, H.G. Bock, and J.P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005.

[64] M. Diehl, H.G. Bock, J.P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations. *J. Proc. Contr.*, 12(4):577–585, 2002.

[65] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pages 391–417. Springer, 2009.

[66] M. Diehl, R. Findeisen, and F. Allgöwer. A Stabilizing Real-time Implementation of Nonlinear Model Predictive Control. In L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, editors, *Real-Time and Online PDE-Constrained Optimization*, pages 23–52. SIAM, 2007.

[67] M. Diehl, R. Findeisen, F. Allgöwer, H.G. Bock, and J.P. Schlöder. Nominal Stability of the Real-Time Iteration Scheme for Nonlinear Model Predictive Control. *IEE Proc.-Control Theory Appl.*, 152(3):296–308, 2005.

[68] M. Diehl, D.B. Leineweber, and A.A.S. Schäfer. MUSCOD-II Users' Manual. IWR-Preprint 2001-25, Universität Heidelberg, 2001.

[69] M. Diehl, L. Magni, and G. De Nicolao. Online NMPC of unstable periodic systems using approximate infinite horizon closed loop costing. *IFAC Annual Reviews in Control*, 28:37–45, 2004.

[70] W.S. Dorn. Duality in quadratic programming. *Quarterly of Applied Mathematics*, 18:155–162, 1960.

[71] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[72] John W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[73] B.C. Fabien. Implementation of a robust sqp algorithm. *Optimization Methods & Software*, 23(6):827–846, 2008.

[74] B.C. Fabien. dsoa: The implementation of a dynamic system optimization algorithm. *Optimal Control Applications and Methods*, 31:231–247, 2010.

[75] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[76] H. J. Ferreau, P. Ortner, P. Langthaler, L. del Re, and M. Diehl. Predictive control of a real-world diesel engine using an extended online active set strategy. *Annual Reviews in Control*, 31(2):293–301, 2007.

[77] H.J. Ferreau. An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control. Master's thesis, University of Heidelberg, 2006.

[78] H.J. Ferreau. qpOASES – An Open-Source Implementation of the Online Active Set Strategy for Fast Model Predictive Control. In *Proceedings of the Workshop on Nonlinear Model Based Control – Software and Applications, Loughborough*, pages 29–30, 2007.

[79] H.J. Ferreau. qpOASES User's Manual. http://www.qpOASES.org, 2007–2011.

[80] H.J. Ferreau and B. Houska. ACADO Code Generation User's Manual. http://www.acadotoolkit.org, 2011.

[81] H.J. Ferreau, B. Houska, K. Geebelen, and M. Diehl. Real-time control of a kite-carousel using an auto-generated nonlinear MPC algorithm. In *Proceedings of the IFAC World Congress*, Milan, Italy, 2011.

[82] H.J. Ferreau, B. Houska, T. Kraus, and M. Diehl. Numerical Methods for Embedded Optimisation and their Implementation within the ACADO Toolkit. In W. Mitkowski R. Tadeusiewicz, A. Ligeza and M. Szymkat, editors, *7th Conference - Computer Methods and Systems (CMS'09)*, Krakow, Poland, November 2009. Oprogramowanie Naukowo-Techniczne.

[83] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active set strategy for quadratic programming. *ACM Transactions on Mathematical Software*, 2011 (to be submitted).

[84] A.V. Fiacco. *Introduction to sensitivity and stability analysis in nonlinear programming*. Academic Press, New York, 1983.

[85] R. Findeisen and F. Allgöwer. Computational Delay in Nonlinear Model Predictive Control. Proc. Int. Symp. Adv. Control of Chemical Processes, ADCHEM, 2003.

[86] R. Fletcher. A new approach to variable metric algorithms. *Computer J.*, 13:317–322, 1970.

[87] R. Fletcher. *Practical Methods of Optimization*. Wiley, Chichester, 2nd edition, 1987.

[88] R. Fletcher. Resolving degeneracy in quadratic programming. *Annals of Operations Research*, 46–47:307–334, 1993.

[89] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[90] R. Franke and E. Arnold. HQP Webpage. http://hqp.sourceforge.net, 2008.

[91] G.F. Franklin, J.D. Powell, and A. Emami-Naeini. *Feedback control of dynamic systems*. Prentice Hall, 6 edition, 2009.

[92] M.P. Friedlander and P. Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18:1326–1350, 2007.

[93] L. Di Gaspero. QuadProg++ homepage. http://quadprog.sourceforge.net, 2010.

[94] E.M. Gertz and S.J. Wright. Object-Oriented Software for Quadratic Programming. *ACM Transactions on Mathematical Software*, 29(1):58–81, 2003.

[95] T. Geyer, G.A. Beccuti, G. Papafotiou, and M. Morari. Model predictive direct torque control of permanent magnet synchronous motors. In *Proceedings of the IEEE Energy Conversion Congress and Exposition (ECCE), Atlanta, GA*, 2010.

[96] P. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for NPSOL 5.0: A fortran package for nonlinear programming. Technical report sol 86-6, Stanford University, 2001.

[97] P.E. Gill, N.I.M. Gould, W. Murray, M.A. Saunders, and M.H. Wright. A Weighted Gram-Schmidt Method for Convex Quadratic Programming. *Mathematical Programming*, 30:176–195, 1984.

[98] P.E. Gill and W. Murray. Numerically Stable Methods for Quadratic Programming. *Mathematical Programming*, 14:349–372, 1978.

[99] P.E. Gill, W. Murray, and M.A. Saunders. *User's Guide For QPOPT 1.0: A Fortran Package For Quadratic Programming*, 1995.

[100] P.E. Gill, W. Murray, and M.A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Technical report, Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[101] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints. *ACM Transactions on Mathematical Software*, 10(3):282–298, 1984.

[102] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[103] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. Inertia-Controlling Methods for General Quadratic Programming. *SIAM Review*, 33(1):1–36, 1991.

[104] P.E. Gill, W. Murray, and M.H. Wright. *Practical optimization*. Academic Press, London, 1999.

[105] W.J. Givens. Numerical computation of the characteristic values of a real symmetric matrix. Technical Report 1574, Oak Ridge National Laboratory, 1954.

[106] T. Glad and H. Johnson. A method for state and control constrained linear-quadratic control problems. In *Proceedings of the 9th IFAC World Congress, Budapest, Hungary*, pages 1583–1587, 1984.

[107] D. Goldfarb. A family of variable metric methods derived by variational means. *Maths. Comp.*, 17:739–764, 1970.

[108] D. Goldfarb. Matrix Factorizations in Optimization of Nonlinear Functions Subject to Linear Constraints. *Mathematical Programming*, 10:1–31, 1975.

[109] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.

[110] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[111] N.I.M. Gould, D. Orban, and P.L. Toint. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372, 2004.

[112] M. Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2004.

[113] M. Grant and S. Boyd. CVX webage. http://cvxr.com/cvx, 2011.

[114] A. Griewank. On Automatic Differentiation. In *Mathematical Programming: Recent Developments and Applications*. Kluwer Academic Publishers, Dordrecht, Boston, London, 1989.

[115] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation.* Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.

[116] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++. Technical report, Technical University of Dresden, Institute of Scientific Computing and Institute of Geometry, 1999. Updated version of the paper published in *ACM Trans. Math. Software* 22, 1996, 131–167.

[117] A. Griewank and Ph.L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137, 1982.

[118] A. Griewank and A. Walther. On Constrained Optimization by Adjoint based quasi-Newton Methods. *Optimization Methods and Software*, 17:869 – 889, 2002.

[119] J. Guddat, F. Guerra Vasquez, and H.T. Jongen. *Parametric Optimization: Singularities, Pathfollowing and Jumps.* Teubner, Stuttgart, 1990.

[120] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I.* Springer Series in Computational Mathematics. Springer, Berlin, 2nd edition, 1993.

[121] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems.* Springer Series in Computational Mathematics. Springer, Berlin, 2nd edition, 1996.

[122] S. P. Han. A Globally Convergent Method for Nonlinear Programming. *JOTA*, 22:297–310, 1977.

[123] A. Helbig, O. Abel, and W. Marquardt. Model Predictive Control for On-line Optimization of Semi-batch Reactors. In *Proc. Amer. Contr. Conf.*, pages 1695–1699, Philadelphia, 1998.

[124] M.R. Hestenes. *Calculus of variations and optimal control theory.* Wiley, New York, 1966.

[125] N.J. Higham. *Accuracy and stability of numerical algorithms.* SIAM, 2nd edition, 2002.

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

[127] B. Houska and H.J. Ferreau. ACADO Toolkit User's Manual. http://www.acadotoolkit.org, 2009–2011.

[128] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[129] B. Houska, H.J. Ferreau, and M. Diehl. An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 2011 (in print).

[130] M. Hovd and R.D. Braatz. Handling state and output constraints in MPC using time-dependent weights. In *Proceedings of the American Control Conference, Arlington, VA*, 2001.

[131] S. Hovland. Soft constraints in explicit model predictive control. Master's thesis, Trondheim University, 2004.

[132] M. Huschenbett, D. Alberer, G. Beshouri, and M. Richter. Emission reduction & stability improvement by predictive model based predictive control of legacy gas engines. In *Proceedings of the Gas Machinery Conference 2007, Dallas, Texas*, 2007.

[133] A. Ilzhoefer, B. Houska, and M. Diehl. Nonlinear MPC of kites under varying wind conditions for a new class of large scale wind power generators. *International Journal of Robust and Nonlinear Control*, 17(17):1590–1599, 2007.

[134] Free Software Foundation Inc. GNU Lesser General Public License. http://www.gnu.org/copyleft/lesser.html, 2007–2011.

[135] The MathWorks Inc. *Real-Time Workshop for Use with SIMULINK, User's Guide*, 1999.

[136] T.A. Johansen and A. Grancharova. Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Trans. Automatic Control*, 48:810–815, 2003.

[137] C.N. Jones and M. Morari. Approximate explicit MPC using bilevel optimization. In *European Control Conference*, Budapest, Hungary, 2009.

[138] C.N. Jones and M. Morari. Polytopic approximation of explicit model predictive controllers. *IEEE Transactions on Automatic Control*, 55(11):2542–2553, 2010.

[139] H. Jonson. *A Newton-Method for Solving Non-Linear Optimal Control Problems with General Constraints*. PhD thesis, Linköpping University, 1983.

[140] J.B. Jorgensen, J.B. Rawlings, and S.B. Jorgensen. Numerical methods for large-scale moving horizon estimation and control. In *Proceedings of Int. Symposium on Dynamics and Control Process Systems (DYCOPS)*, 2004.

[141] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[142] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[143] S.S. Keerthi and E.G. Gilbert. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. *Journal of Optimization Theory and Applications*, 57(2):265–293, 1988.

[144] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[145] E.C. Kerrigan. *Robust Constraint Satisfaction: Invariant Sets and Predictive Control*. PhD thesis, University of Cambridge, UK, 2000.

[146] C. Kirches, H.G. Bock, J.P. Schlöder, and S. Sager. Block-structured quadratic programming for the direct multiple shooting method for optimal control. *Optimization Methods and Software*, 26(2):239–257, 2010.

[147] C. Kirches, L. Wirsching, S. Sager, and H.G Bock. Efficient numerics for nonlinear model predictive control. In M. Diehl, Francois F. Glineur, and E. Jarlebring W. Michiels, editors, *Recent Advances in Optimization and its Applications in Engineering*, pages 339–357. Springer, 2010.

[148] K.-U. Klatt and S. Engell. Rührkesselreaktor mit Parallel- und Folgereaktion. In S. Engell, editor, *Nichtlineare Regelung – Methoden, Werkzeuge, Anwendungen. VDI-Berichte Nr. 1026*, pages 101–108. VDI-Verlag, Düsseldorf, 1993.

[149] V. Klee and G.J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972.

[150] P. Krämer-Eis and H.G. Bock. Numerical Treatment of State and Control Constraints in the Computation of Feedback Laws for Nonlinear Control Problems. In P. Deuflhard et al., editor, *Large Scale Scientific Computing*, pages 287–306. Birkhäuser, Basel Boston Berlin, 1987.

[151] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, 1951. University of California Press.

[152] M. Kvasnica, P. Grieder, M. Baotic, and F.J. Christophersen. *Multi-Parametric Toolbox*, 2006.

[153] M. Kvasnica, I. Rauova, and F. Miroslav. Automatic code generation for real-time implementation of model predictive control. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design, Yokohama, Japan*, 2010.

[154] J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *Proceedings of the ACM/IEEE SC 2006 Conference*, pages 50–66, 2006.

[155] D.B. Leineweber. *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, volume 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. VDI Verlag, Düsseldorf, 1999.

[156] D.B. Leineweber, I. Bauer, H.G. Bock, and J.P. Schlöder. An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization. Part I: Theoretical Aspects. *Computers and Chemical Engineering*, 27:157–166, 2003.

[157] D.B. Leineweber, I. Bauer, A.A.S. Schäfer, H.G. Bock, and J.P. Schlöder. An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization (Parts I and II). *Computers and Chemical Engineering*, 27:157–174, 2003.

[158] W. Li and J. Swetits. A new algorithm for solving strictly convex quadratic programs. *SIAM Journal of Optimization*, 7(3):595–619, 1997.

[159] W.C. Li and L.T. Biegler. Multistep, Newton-Type Control Strategies for Constrained Nonlinear Processes. *Chem. Eng. Res. Des.*, 67:562–577, 1989.

[160] W.C. Li and L.T. Biegler. Newton-Type Controllers for Constrained Nonlinear Processes with Uncertainty. *Industrial and Engineering Chemistry Research*, 29:1647–1657, 1990.

[161] L. Ljung. *System identification: Theory for the User*. Prentice Hall, Upper Saddle River, N.J., 1999.

[162] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[163] F. Logist, B. Houska, M. Diehl, and J. Van Impe. Fast Pareto set generation for nonlinear optimal control problems with multiple objectives. *Structural and Multidisciplinary Optimization*, 42(4):591–603, 2010.

[164] M.L. Loyd. Crosswind Kite Power. *Journal of Energy*, 4(3):106–111, July 1980.

[165] J.M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[166] C.M. Maes. *A Regularized Acitve-Set Method for Sparse Convex Quadratic Programming*. PhD thesis, Stanford University, 2010.

[167] I. Maros and C. Meszaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11:431–449, 1999.

[168] J. Mattingley and S. Boyd. CVXGEN webage. http://cvxgen.com, 2008–2011.

[169] J. Mattingley and S. Boyd. *Convex Optimization in Signal Processing and Communications*, chapter Automatic Code Generation for Real-Time Convex Optimization. Cambridge University Press, 2009.

[170] D. Q. Mayne and S. Rakovic. Optimal Control of Constrained Piecewise Affine Discrete-Time Systems. *Computational Optimization and Applications*, 25:167–191, 2003.

[171] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: stability and optimality. *Automatica*, 26(6):789–814, 2000.

[172] S. Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[173] A. M'hamdi, A. Helbig, O. Abel, and W. Marquardt. Newton-type Receding Horizon Control and State Estimation. In *Proc. 13rd IFAC World Congress*, pages 121–126, San Francisco, 1996.

[174] M. Morari. Approximate explicit model predictive control. Talk at International Workshop on Assessment and Future Directions of Nonlinear Model Predictive Control, Pavia, Italy, September 2008.

[175] A. Morger. Synthesis, design and control of a tendon-driven robot platform for vestibular stimulation during sleep. Master's thesis, ETH Zurich, 2010.

[176] Y. Nesterov. *Introductory lectures on convex optimization: a basic course*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, 2003.

[177] Y. Nesterov and A. Nemirovski. *Interior-point Polynomial Algorithms in Convex Programming*. Society for Industrial Mathematics, 1994.

[178] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2 edition, 2006.

[179] S.J. Norquay, A. Palazoglu, and J.A. Romagnoli. Application of Wiener Model Predictive Control (WMPC) to a pH Neutralization Experiment. *IEEE Transactions on Control Systems Technology*, 7(4):437–445, 1999.

[180] T. Ohtsuka. A Continuation/GMRES Method for Fast Computation of Nonlinear Receding Horizon Control. *Automatica*, 40(4):563–574, 2004.

[181] T. Ohtsuka and A. Kodama. Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers*, 38(7):617–623, 2002.

[182] T. Oohori and A. Ohuchi. An efficient implementation of Karmarkar's algorithm for large sparse linear programs. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,*, 1988.

[183] Tomlab Optimization. PROPT: Matlab Optimal Control Software (ODE,DAE). http://tomdyn.com, 2009–2011.

[184] P. Ortner, R. Bergmann, H.J. Ferreau, and L. del Re. Nonlinear Model Predictive Control of a Diesel Engine Airpath. In *IFAC Workshop on Control Applications of Optimisation, Agora, Finland*, 2009.

[185] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal of Numerical Analysis*, 12(4):617–629, 1975.

[186] G. Pannocchia, J.B. Rawlings, D.Q. Mayne, and W. Marquardt. On computing solutions to the continuous time constrained linear quadratic regulator. *IEEE Transactions of Automatic Control*, 55(9):2192–2198, 2010.

[187] G. Pannocchia, J.B. Rawlings, and S.J. Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43:852–860, 2007.

[188] G. Pannocchia, J.B. Rawlings, and S.J. Wright. Partial enumeration MPC: Robust stability results and application to an unstable CSTR. In *Proceedings of the 9th International Symposium on Dynamics and Control of Process Systems, Leuven, Belgium*, 2010.

[189] P. Patrinos, P. Sopasakis, and H. Sarimveis. A global piecewise smooth newton method for fast large-scale model predictive control. Technical report, National Technical University of Athens, 2010.

[190] L.R. Petzold. Solver DASSL. http://pitagora.dm.uniba.it/∼testset/solvers/-dassl.php, June 1991.

[191] K.J. Plitt. Ein superlinear konvergentes Mehrzielverfahren zur direkten Berechnung beschränkter optimaler Steuerungen. Master's thesis, Universität Bonn, 1981.

[192] L.S. Pontryagin, V.G. Boltyanski, R.V. Gamkrelidze, and E.F. Miscenko. *The Mathematical Theory of Optimal Processes*. Wiley, Chichester, 1962.

[193] A. Potschka, C. Kirches, H.G. Bock, and J.P. Schlöder. Reliable solution of convex quadratic programs with parametric active set methods. Technical report, Interdisciplinary Center for Scientific Computing, Heidelberg University, Im Neuenheimer Feld 368, 69120 Heidelberg, GERMANY, November 2010.

[194] M.J.D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G.A. Watson, editor, *Numerical Analysis, Dundee 1977*, volume 630 of *Lecture Notes in Mathematics*, Berlin, 1978. Springer.

[195] S.J. Qin and T.A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[196] C.V. Rao, S.J. Wright, and J.B. Rawlings. Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications*, 99:723–757, 1998.

[197] G. Rauter, J. v. Zitzewitz, A. Duschau-Wicke, H. Vallery, and R. Riener. A tendon-based parallel robot applied to motor learning in sports. In *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics 2010, Japan*, 2010.

[198] J.B. Rawlings. Optimal dynamic operation of chemical processes: Assessment of the last 20 years and current research opportunities. Talk at K.U. Leuven, Belgium, June 2009.

[199] S. Richter, C.N. Jones, and M. Morari. Real-time input-constrained MPC using fast gradient methods. In *Proceedings of the IEEE Conference on Decision and Control, Shanghai, China*, 2009.

[200] S. Richter, S. Mariéthoz, and M. Morari. High-Speed Online MPC Based on a Fast Gradient Method Applied to Power Converter Control. In *Proceedings of the American Control Conference*, pages 4737–4743, Baltimore, MD, USA, 2010.

[201] S.M. Robinson. Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. *Mathematical Programming*, 7:1–16, 1974.

[202] R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877–898, 1976.

[203] L.A. Rodriguez and A. Sideris. An active set method for constrained linear quadratic optimal control. In *American Control Conference (ACC), 2010*, pages 5197–5202, 2010.

[204] A. Romanenko and L. Santos. *Assessment and Future Directions of Nonlinear Model Predictive Control*, volume 358 of *Lecture Notes in Control and Information Sciences*, chapter A Nonlinear Model Predictive Control Framework as Free Software: Outlook and Progress Report, pages 229–238. Springer, 2007.

[205] R.W.H. Sargent and G.R. Sullivan. The development of an efficient optimal control package. In J. Stoer, editor, *Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2*, Heidelberg, 1978. Springer.

[206] K. Schittkowski. QLD: A FORTRAN code for quadratic programming, users guide. Universität Bayreuth, 1986.

[207] J.P. Schlöder. *Numerische Methoden zur Behandlung hochdimensionaler Aufgaben der Parameteridentifizierung*, volume 187 of *Bonner Mathematische Schriften*. Universität Bonn, Bonn, 1988.

[208] P.O.M. Scokaert and J.B. Rawlings. Feasibility Issues in Linear Model Predictive Control. *AIChE Journal*, 45(8):1649–1659, 1999.

[209] M. Seeger. Low rank updates for the cholesky decomposition. Technical report, University of California at Berkeley, 2008.

[210] H. Seguchi and T. Ohtsuka. Nonlinear Receding Horizon Control of an Underactuated Hovercraft. *International Journal of Robust and Nonlinear Control*, 13(3–4):381–398, 2003.

[211] A. Shahzad, E.C. Kerrigan, and G.A. Constantinides. Preconditioners for inexact interior point methods for predictive control. In *American Control Conference (ACC), 2010*, pages 5714–5719, 2010.

[212] A. Shahzad, E.C. Kerrigan, and G.A. Constantinides. A warm-start interior-point method for predictive control. Technical report, Imperial College London, 2010.

[213] R. Shamir. Probabilistic analysis in linear programming. *Statistical Science*, 8(1):57–64, 1993.

[214] D.F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Maths. Comp.*, 24:647–656, 1970.

[215] Y. Shimizu, T. Ohtsuka, and M. Diehl. A Real-Time Algorithm for Nonlinear Receding Horizon Control Using Multiple Shooting and Continuation/Krylov Method. *International Journal of Robust and Nonlinear Control*, 19:919–936, 2009.

[216] L.L. Simon, Z.K. Nagy, and K. Hungerbuehler. *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Swelling Constrained Control of an Industrial Batch Reactor Using a Dedicated NMPC Environment: OptCon, pages 531–539. Springer, 2009.

[217] R.D. Skeel. Iterative refinement implies numerical stability for gaussian elimination. *Mathematics of Computation*, 35(151):817–832, 1980.

[218] IEEE Computer Society. *IEEE Standard 1063 for Software User Documentation*, 2001.

[219] J. Spjøtvold, E.C. Kerrigan, C.N. Jones, T.A. Johansen, and P. Tøndel. Conjectures on an algorithm for convex parametric quadratic programs. Technical report, Department of Engineering, University of Cambridge, 2004.

[220] M.C. Steinbach. A structured interior point SQP method for nonlinear optimal control problems. In R. Bulirsch and D. Kraft, editors, *Computation Optimal Control*, pages 213–222, Basel Boston Berlin, 1994. Birkhäuser.

[221] M. Sznaier and M.J. Damborg. Suboptimal control of linear systems with state and control inequality constraints. In *Proceedings of the 26th IEEE conference on decision and control, Los Angeles*, pages 761–762, 1987.

[222] G. Takács and B. Rohal'-Ilkiv. MPC in active vibration control of lightly damped structures. *Control Engineering Practice*, 2011 (submitted).

[223] G. Takács and B. Rohal'-Ilkiv. *Predictive Vibration Control: Efficient constrained MPC vibration control for lightly damped mechanical systems*. Springer, 2011 (in print).

[224] M.J. Tenny, S.J. Wright, and J.B. Rawlings. Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming. *Computational Optimization and Applications*, 28(1):87–121, April 2004.

[225] P. Tøndel, T.A. Johansen, and A. Bemporad. An Algorithm for Multi-Parametric Quadratic Programming and Explicit MPC Solutions. *Automatica*, 39:489–497, 2003.

[226] P. Tøndel, T.A. Johansen, and A. Bemporad. Computation and Approximation of Piecewise Affine Control Laws via Binary Search Trees. *Automatica*, 39:945–950, 2003.

[227] T.H. Tsang, D.M. Himmelblau, and T.F. Edgar. Optimal control via collocation and non-linear programming. *International Journal on Control*, 21:763–768, 1975.

[228] J. Vada, O. Slupphaug, and T.A. Johansen. Efficient infeasibility handling in linear MPC subject to prioritised constraints. In *Proceeding of the European Control Conference, Karlsruhe, Germany*, 1999.

[229] J. Vada, O. Slupphaug, and T.A. Johansen. Optimal Prioritized Infeasibility Handling in Model Predictive Control: Parametric Preemptive Multiobjective Linear Programming Approach. *Journal of Optimization Theory and Applications*, 109(2):385–413, 2001.

[230] L. Van den Broeck. *Time optimal control of mechatronic systems through embedded optimization*. PhD thesis, K.U. Leuven, july 2011.

[231] L. Van den Broeck, M. Diehl, and J. Swevers. Time Optimal MPC for mechatronic applications. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 8040–8045, Shanghai, China, 2009.

[232] L. Van den Broeck, J. Swevers, and M. Diehl. Performant design of an input shaping prefilter via embedded optimization. In *Proceedings of the 2009 American Control Conference*, pages 166–171, St-Louis, USA, 2009.

[233] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices I. Linear Systems*. John Hopkins University Press, Baltimore, USA, 2007.

[234] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[235] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, 2002.

[236] A. Wächter and L.T. Biegler. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, 106(1):25–57, 2006.

[237] X. Wang. Resolution of Ties in Parametric Quadratic Programming. Master's thesis, University of Waterloo, Ontario, Canada, 2004.

[238] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.

[239] Y. Wang, C.N. Jones, and J.M. Maciejowski. Efficient point location via subdivision walking with application to explicit MPC. In *European Control Conference*, Kos, Greece, 2007.

[240] T. Wiese. Constraint handling in predictive control. Bachelor thesis, Technische Universität München, 2009.

[241] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Claredon Press, Oxford, 1965.

[242] A. Wills. QPC homepage. http://sigpromu.org/quadprog, 2006–2011.

[243] A. Wills, D. Bates, A.J. Fleming, B. Ninness, and S.O. Reza Moheimani. Model predictive control applied to constraint handling in active noise and vibration control. *IEEE Transactions on Control Systems Technology*, 16(1):3–12, 2008.

[244] A.G. Wills, D. Bates, A.J. Fleming, B. Ninness, and S.O.R. Moheimani. Application of MPC to an active structure using sampling rates up to 25kHz. 44th IEEE Conference on Decision and Control and European Control Conference ECC'05, Seville, 2005.

[245] R.B. Wilson. *A simplicial algorithm for concave programming*. PhD thesis, Harvard University, 1963.

[246] L. Wirsching. An SQP Algorithm with Inexact Derivatives for a Direct Multiple Shooting Method for Optimal Control Problems. Master's thesis, University of Heidelberg, 2006.

[247] L. Wirsching, H.J. Ferreau, H.G. Bock, and M. Diehl. An Online Active Set Strategy for Fast Adjoint Based Nonlinear Model Predictive Control. In *Preprints of the 7th Symposium on Nonlinear Control Systems (NOLCOS), Pretoria*, 2007.

[248] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27:382–398, 1959.

[249] P. Wolfe. A duality theorem for non-linear programming. *Quarterly of Applied Mathematics*, 19:239–244, 1961.

[250] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM Publications, Philadelphia, 1997.

[251] S.J. Wright. Efficient methods for structured quadratic programs. Talk at OPTEC Workshop on Large Scale Convex Quadratic Programming – Algorithms, Software, and Applications, Leuven, Belgium, November 2010.

[252] E.A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

[253] E. Zafiriou. Robust model predictive Control of processes with hard constraints. *Computers & Chemical Engineering*, 14(4–5):359–371, 1990.

[254] V. M. Zavala and L.T. Biegler. The Advanced Step NMPC Controller: Optimality, Stability and Robustness. *Automatica*, 45:86–93, 2009.

[255] V.M. Zavala, C.D. Laird, and L.T. Biegler. Fast solvers and rigorous models: can both be accomodated in NMPC? In *Proceedings of the IFAC Workshop on Nonlinear Model Predictive Control for Fast Systems, Grenoble*, 2006.

[256] M.N. Zeilinger, C.N. Jones, and M. Morari. Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization. In *Proceedings of the IEEE Conference on Decision and Control, Cancun, Mexico*, 2008.

[257] J. Zhao, M. Diehl, R. Longman, H.G. Bock, and J.P. Schlöder. Nonlinear Model Predictive Control of Robots Using Real-Time Optimization. In *Proceedings of the AIAA/AAS Astrodynamics Conference*, Providence, RI, August 2004.

List of Publications

Journal Papers:

1. H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock M. Diehl. *qpOASES: A parametric active set strategy for quadratic programming.* ACM Transactions on Mathematical Software, 2011 (to be submitted).

2. B. Houska, H.J. Ferreau, M. Diehl. *An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range.* Automatica, 2011 (in print).

3. B. Houska, H.J. Ferreau, M. Diehl. *ACADO Toolkit – An Open-Source Framework for Automatic Control and Dynamic Optimization.* Optimal Control Methods and Application, 32 (3), pp. 298–312, 2011.

4. H.J. Ferreau, H.G. Bock, M. Diehl. *An online active set strategy to overcome the limitations of explicit MPC.* International Journal of Robust and Nonlinear Control, 18 (8), pp. 816–830, 2008.

5. H.J. Ferreau, P. Ortner, P. Langthaler, L. del Re, M. Diehl. *Predictive Control of a Real-World Diesel Engine using an Extended Online Active Set Strategy.* Annual Reviews in Control, 31 (2), pp. 293–301, 2007.

Book Chapters:

1. H.J. Ferreau, B. Houska, T. Kraus, M. Diehl. *Numerical Methods for Embedded Optimisation and their Implementation within the ACADO Toolkit.* In: 7th Conference – Computer Methods and Systems (CMS'09); R. Tadeusiewicz, A. Ligeza, W. Mitkowski, M. Szymkat (eds.), pp. 13–29, Oprogramowanie Naukowo-Techniczne, 2009.

2. M. Diehl, H.J. Ferreau, N. Haverbeke. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation.* In: Nonlinear Model

Predictive Control; L. Magni, M.D. Raimondo, F. Allgöwer (eds.), pp. 391–417, 2009.

Conference Papers:

1. H.J. Ferreau, B. Houska, K. Geebelen, M. Diehl. *Real-Time Control of a Kite-Carousel using an Auto-Generated Nonlinear MPC Algorithm.* Proceedings of the IFAC World Congress, Milan, Italy, 2011.

2. B. Defraene, T. van Waterschoot, H.J. Ferreau, M. Diehl, M. Moonen. *Perception-based clipping of audio signals.* Proceedings of the 18th European Signal Processing Conference, pp. 517–521, Aalborg, Denmark, 2010.

3. D. Dimitrov, P.-B. Wieber, O. Stasse, H.J Ferreau, H. Diedam. *An Optimized Linear Model Predictive Control Solver for Online Walking Motion Generation.* Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1171–1176, Kobe, Japan, 2009.

4. D. Alberer, H. Kirchsteiger, L. del Re, H.J. Ferreau, M. Diehl. *Receding horizon optimal control of Wiener systems by application of an asymmetric cost function.* Proceedings of the IFAC Workshop on Control Applications of Optimisation, Agora, Finland, 2009.

5. P. Ortner, R. Bergmann, H.J. Ferreau, L. del Re. *Nonlinear Model Predictive Control of a Diesel Engine Airpath.* Proceedings of the IFAC Workshop on Control Applications of Optimisation, Agora, Finland, 2009.

6. D. Dimitrov, P.-B. Wieber, H.J. Ferreau, M. Diehl. *On the Implementation of Model Predictive Control for On-line Walking Pattern Generation.* Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2685–2690, Pasadena, CA, 2008.

7. L. Wirsching, H.J. Ferreau, H.G. Bock, M. Diehl. *An Online Active Set Strategy for Fast Adjoint Based Nonlinear Model Predictive Control.* Preprints of the 7th Symposium on Nonlinear Control Systems (NOLCOS), pp. 164–169, Pretoria, South Africa, 2007.

8. H.J. Ferreau, G. Lorini, M. Diehl. *Fast Nonlinear Model Predictive Control of Gasoline Engines.* Proceedings of the IEEE International Conference on Control Applications, pp. 2754–2759, Munich, Germany, 2006.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty Engineering.
Department of Electrical Engineering
Research group SCD
Kasteelpark Arenberg 10 (bus 2446), B-3001 Leuven