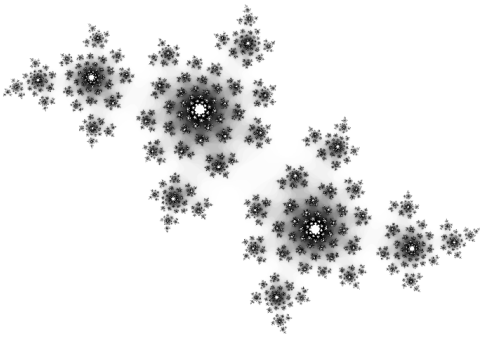
KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

DISTRIBUTED OPTIMIZATION METHODS FOR LARGE SCALE OPTIMAL CONTROL



Attila Kozma

Distributed Optimization Methods for Large Scale Optimal Control

Attila KOZMA

Supervisory Committee:
Prof. Yves Willems, chair
Prof. Moritz Diehl, supervisor
Prof. Stefan Vandewalle
Prof. François Glineur
Prof. Toon van Waterschoot
Prof. Jacek Gondzio
(Univ. of Edinburgh)

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

November 2013

© KU Leuven – Faculty of Engineering Science
Kasteelpark Arenberg 10, bus 2446, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

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

D/2014/7515/24
ISBN 978-94-6018-800-8

Acknowledgements

First, I would like to express my gratitude to my supervisor, Moritz Diehl. Already when he interviewed me he was very enthusiastic which feature he kept for the whole period of my doctoral studies. He has been an endless source of inspiration giving ideas gladly. Even in the busiest periods he managed to find time to have discussions with me, which I respect a lot.

I am also thankful to my committee members, Yves Willems, Stefan Vandewalle, François Glineur, Toon van Waterschoot and Jacek Gondzio for reading my thesis and providing constructive feedback.

I learnt a lot by working together with Carlo Savorgnan, from whom I could get instant help even in my smallest scientific problems. I could also improve my knowledge by cooperating with my phd brothers: Quoc Tran Dinh, Joel Andersson, Mario Zanon, Joris Gillis, Rien Quirynen, Janick Frasch, and my only phd sister Adeleh Mohammadi. I was very happy in the easy-going atmosphere in the office that was created by them. Thanks go to colleagues Sebastien Gros and Emil Klintberg from Chalmers University for the fruitful cooperation.

My mother, Ágnes Ipacs, my grandmother Elemérné Szentpéteri and my sisters Gabriella Kozma, Judit Kozma supported me from the very beginning to do the PhD abroad. They were also there to help in the hard moments, and accepted me warmly in Hungary any time I went home.

I am grateful to Gergő Kováts who came to visit me in Belgium multiple times and helped to establish my social life in Leuven and to Krisztián Veress on whom I could always count. They were both very friendly and accepted me with great pleasure whenever I returned to Hungary.

There have been a lot of people living in Belgium I met and many of them with whom I even got friends with. In particular, Alessandro Casini, who cooked the first italian pasta for me, Sebastian Hanisch who has been always very laid-back and understanding both as a friend and as a flatmate. Alessandro Arditi and

later Marcello Giannini became my italian friends with whom we had lots of fun. The very positive attitude of Koosha Paridel and all that "bakje met kip" we ate together was great pleasure. It has also been great to know the musicians Gergő Fajd and Angelika Zolcsák, with whom we could summon the atmosphere of Hungary. I was very glad to be the neighbour of Keith Magee and Tania Ferreria in Bloemenberggang with whom we always made a great craic. Mario Zanon, Matteo Kirchner and Sebastien Gros were permanent guests in our street making our lives eventful. I enjoyed living with Albert Prinn who listened to every UFO story I found on the internet and with Dorota Iwaszkiewicz-Grześ who always ordered only the best quality holy water. Mark Boyle showed up at the most random times but with a big smile on his face. Santiago Amaro was my last neighbour, he can cook the best tortilla patatina on the Earth. With the help of Cristel Goethals and Csenge Koller, I managed to learn the basics of how to play the violin. I thank Gergely Csány for teaching me some Hungarian folk songs on the violin and for making a fantastic atmoshphere at the Gulasch Festival. Thanks go to Kobe Nys for introducing Lubbeek and its great mustard to me.

"What you get by achieving your goals is not as important as what you become by achieving your goals."

"Amit a cél elérésével kapunk közel sem olyan fontos, mint amivé válunk, amíg azt elérjük."

// Zig Ziglar, American self-help author and speaker. //

Abstract

This thesis aims to develop and implement both nonlinear and linear distributed optimization methods that are applicable, but not restricted to the optimal control of distributed systems. Such systems are typically large scale, thus the well-established centralized solution strategies may be computationally overly expensive or impossible and the application of alternative control algorithms becomes necessary. Moreover, it is often desired that the information on the coupled subsystems inner dynamics is kept local, making the deployment of a centralized optimal control scheme impossible in practice. In such a case, optimization approaches based on some limited exchange of information remain the only possible option.

In the first part of this thesis, we consider nonlinear optimal control problems with distributed structure, for which we design an efficient distributed shooting method resulting in up to 19 times faster integration time when compared to the conventional approaches. The first part reports the testing of the proposed method on two different dynamic optimization problems.

The second part of this thesis investigates linear optimal control problems with quadratic cost functions, i.e. quadratic programs (QP). An overview of dual decomposition based approaches is given, followed by the development of a novel dual decomposition scheme. The proposed method employs second-order derivatives. Experimental results suggest that it requires up to 152 times less local optimization steps than classical first-order schemes.

Furthermore, as a part of this thesis, an open-source QP solver (PyDQP) with 11 different dual decomposition based methods is implemented. A set of large scale distributed quadratic programs is set up and released for benchmarking purposes.

Beknopte samenvatting

Deze thesis handelt over de ontwikkeling en implementatie van lineaire en niet-lineaire gedistribueerde optimalisatietechnieken, bruikbaar voor onder meer optimale controle van gedistribueerde systemen. Zulke systemen zijn vaak grootschalig, waardoor klassieke gecentraliseerde oplossingsstrategieën een zeer grote rekencapaciteit vergen of zelfs onmogelijk worden om toe te passen. Voor zulke systemen zijn alternatieve controle-algoritmes noodzakelijk. Bovendien wordt informatie over de dynamica van de gekoppelde deelsystemen graag lokaal gehouden, wat de implementatie van een gecentraliseerde strategie verhindert. Voor deze situatie vormen de gepresenteerde optimalisatietechnieken op basis van schaarse informatie-overdracht de enige mogelijkheid.

In een eerste deel van de thesis bestuderen we niet-lineaire optimale controle problemen met gedistribueerde structuur, waarvoor we een efficiënte gedistribueerde shooting methode ontwerpen, die leidt tot een oplossingstijd voor integratie, tot 19 maal sneller dan klassieke technieken. In dit eerste deel beschrijven we verder de toepassing van de voorgestelde techniek op twee verschillende dynamische optimalisatie problemen.

Het tweede deel van de thesis gaat in op lineaire optimale controle problemen met kwadratische kostfunctie i.e. kwadratische programma's. Naast een overzicht van gekende methodes gebaseerd op duale decompositie, geven we de ontwikkeling van een nieuwe zulke methode weer die gebruik maakt van tweede-orde afgeleiden. De eerste resultaten geven aan dat het aantal lokale optimalisatie-stappen gereduceerd wordt met een factor tot 152 ten opzichte van de gekende methodes.

Als laatste deel van de thesis volgt de implementatie van een open-source QP solver (PyDQP) met 11 verschillende duale decompositie methodes. Samenhangend is ook een collectie van grootschalige gedistribueerde kwadratische programma's opgebouwd en vrijgegeven, om tot benchmark te dienen.

Abbreviations

AS	Active Set
DMS	Distributed Multiple Shooting
HPV	Hydro Power Valley
IP	Interior Point
IVP	Initial Value Problem
LICQ	Linear Independence Constraint Qualification
NLP	Nonlinear Programming Problem
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
QP	Quadratic Programming
SQP	Sequential Quadratic Programming

Notation

$x, y, z, \ldots$	Column vectors of real numbers
$x(t), y(t), \ldots$	Time dependent column vectors
$A, B, C, \ldots$	Matrices of real numbers
A^T	Transpose of matrix A
$f, g, h, \ldots$	Real vector-vector functions, $\mathbb{R}^n \to \mathbb{R}^m$
$\nabla f(x)$	Gradient function of $f(x)$
$\dot{x}(t)$	Time derivative of $x(t)$
$\mathcal{N}_i$	Neighbours of node i
$\mathcal{N}(x)$	Neighbourhood of vector x
C^n	The function space of n times continuously differentiable real vector-vector functions
$A \succ 0$	Matrix A is positive definite
$A \succeq 0$	Matrix A is positive semidefinite
$\mathbb{R}^n$	The n dimensional vector-space of real numbers
$\mathbb{R}^{n \times m}$	The vector-space of $n \times m$ dimensional matrices
$\subset$	Subset
$\emptyset$	Empty set

Contents

Abstract iii

Contents xi

1 Introduction 1

1.1 Motivation 1

1.2 Contribution of the thesis and overview 2

1.3 Nonlinear Programming 3

1.4 Solution methods for Nonlinear Programming 4

1.5 Convex Quadratic Programming 7

1.6 Optimal control 8

I Large Scale Nonlinear Optimal Control 13

2 Distributed Multiple Shooting (DMS) 15

2.1 Problem statement and contribution 15

2.2 Literature survey 17

2.3 Decomposition in the time domain 18

2.4 Decomposition in the spatial domain 19

2.5 DMS - Parallel simulation, centralized optimization 22

2.6 Adjoint-based Distributed Multiple Shooting 26

3 Optimal control of a Hydro Power Valley 39

3.1 Model description and problem statement 39

3.2 DMS on HPV 50

3.3 Comparison with other distributed methods 52

4 Smoke detection in buildings 57

4.1 Problem description 57

4.2 Numerical results 59

II Distributed Quadratic Programming 63

5 Dual decomposition with first-order methods 65

5.1 Distributed QP formulations 65

5.2 Literature survey 68

5.3 Mathematical background 72

5.4 Quadratic programming with dual decomposition 74

5.5 Methods for strongly convex separable QPs 77

5.5.1 Gradient method with fixed stepsize 77

5.5.2 Global Barzilai-Borwein method 79

5.5.3 Fast gradient method with fixed stepsize 81

5.5.4 Fast gradient method with adaptive stepsize 82

5.5.5 Fast gradient method with adaptive restart 84

5.6 Methods for non-strongly convex QPs 85

5.6.1 Alternating direction method of multipliers 86

5.6.2 Inexact Uzawa method 89

6 Numerical performance of distributed QP methods **93**

6.1 Performance evaluation 93

6.2 A benchmark suite for distributed Quadratic Programming . . 99

7 Dual decomposition with second-order methods **103**

7.1 Introduction 103

7.2 Dual Newton-CG method 107

7.2.1 Non-smooth Newton method in the dual space 107

7.2.2 Numerical Example: Optimal Control of Coupled Masses without State Constraints 114

7.3 Regularized Dual Newton-CG method 116

7.3.1 L_2-relaxation of inequalities 118

7.3.2 Singularity-free dual Newton-CG method 120

7.3.3 Numerical Example: Optimal Control of Connected Masses with State Constraints 123

8 Conclusion **127**

8.1 Summary 127

8.2 Directions for future research 128

A Model parameters of HPV **131**

B Open-source distributed QP solver: PyDQP **135**

B.1 Overview 135

B.2 Algorithmic features 136

B.3 Software design 138

B.4 Tutorial examples 139

Bibliography **141**

List of publications **155**

Chapter 1

Introduction

This chapter motivates the work done in the area of distributed optimization for large scale optimal control problems, gives an overview of this thesis, and provides a brief introduction to optimization theory and optimal control.

1.1 Motivation

Large scale optimization problems and in particular control problems arise naturally in the age of networks and hierarchy. Nowadays, most of the elements of our lives are members or derivatives of one or more networks and inserted in a hierarchy. For instance, one can think of the network of houses in a city, or the network of cities in a country or from an engineering point of view, one can consider the electricity network, the telephone network, the television network, the traffic network, the internet, etc. A simple tool in the kitchen is a result of highly interconnected economical network, where the production is geographically distributed. Also, manufacturing units, power and chemical plants become more and more complex and are composed of multiple subunits which cooperate with each other. The optimal and safe operation of these networks is in the interest of the whole humanity and needs automation and control.

The source of difficulty in the optimization of the above mentioned networks is twofold. First, if they can be described as one big problem, the number of decision variables may be gigantic, which challenges the state-of-the-art optimal control methods with their unsatisfiable memory or CPU need. Second, if these problems cannot be described as a centralized one, which is most often the case, the problem

data exists locally based upon which subunits wish to make decisions leading to possible suboptimality. In such circumstances, methods that use detailed local and limited global information are the only options to keep decision making local.

This thesis addresses these difficulties while respecting the "local decision with local data" principle. By this we mean that we try to avoid centralized decision making and data summary and prefer local decisions via information exchange between the subunits.

1.2 Contribution of the thesis and overview

This thesis is divided into two parts. In Part I, the emphasis is on distributed optimization methods for nonlinear optimal control problems. In this part, we discuss the Distributed Multiple Shooting Method, and present numerical experiments with two different applications. This part includes the following contributions.

- Establishment of a novel massively parallel integration scheme, Distributed Multiple Shooting (DMS), for nonlinear distributed systems to be used in an optimization loop, which employs a time domain and a spatial domain based decomposition in order to speed up the integration process.
- Experimental validation of DMS, showing that the proposed decomposition techniques lead to remarkable speedup in the solution of optimal control problems of distributed systems.

In Part II, the topic of discussion is the distributed solution of large scale convex QPs. This part includes the discussion and comparison of the state-of-the-art QP solution methods based on dual decomposition. Moreover, the development of a novel dual decomposition based algorithm employing second-order derivatives can be found in this part. Furthermore, the implementation of PyDQP, an open-source dual decomposition based QP solver and a benchmark suite of large scale convex quadratic programs are described here. This part has the following contributions.

- Extensive description and comparison of the state-of-the-art distributed convex quadratic programming (QP) approaches based on dual decomposition.
- Design of a novel method for the solution of distributed convex QPs employing second-order derivatives in a dual decomposition framework.
- Implementation of an open-source software supporting several distributed QP solution methods based on dual decomposition.
- Composition of a freely available benchmark suite of distributed convex QPs.

This thesis uses the theory and nomenclature of mathematical programming and optimal control. For this reason, we give a brief introduction to the above topics. For further and more detailed information we refer the reader to standard textbooks in optimization theory [110, 24, 17, 11] and in optimal control [15, 12, 119].

1.3 Nonlinear Programming

In general nonlinear programming (NLP), a minimizer of a scalar function that respects a given set of conditions is sought for. More formally, we want to solve

$$\min_x \ f(x) \tag{1.1a}$$

$$\text{s.t. } g_i(x) = 0, \quad i = 1, \dots, p, \tag{1.1b}$$

$$h_i(x) \geq 0, \quad i = 1, \dots, q. \tag{1.1c}$$

Here, $x \in \mathbb{R}^n$ is referred to as the *optimization variable*, for $f \in C^2$, $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, for $\forall i \in \{1, \dots, p\}$ $g_i \in C^2$, $g_i : \mathbb{R}^n \to \mathbb{R}$ are the *equality constraint* functions, and for $\forall j \in \{1, \dots, q\}$ $h_j \in C^2$, $h_j : \mathbb{R}^n \to \mathbb{R}$ are the *inequality constraint* functions. This optimization problem is called *nonlinear programming* problem.

Now we introduce the optimization terminology that is commonly used throughout this thesis. We define $g(x) := (g_1(x), \dots, g_p(x))^T$, $h(x) := (h_1(x), \dots, h_q(x))^T$, $\nabla g(x) := (\nabla g_1(x), \dots, \nabla g_p(x))$ and $\nabla h(x) := (\nabla h_1(x), \dots, \nabla g_q(x))$.

Definition 1.3.1 (feasible set) *The feasible set $\mathcal{F}$ of (1.1) is defined by*

$$\mathcal{F} := \{x \in \mathbb{R}^n | g(x) = 0 \text{ and } h(x) \geq 0\}. \tag{1.2}$$

Definition 1.3.2 (feasible solution) *A vector $\tilde{x}$ is said to be a feasible solution of (1.1) if $\tilde{x} \in \mathcal{F}$ holds.*

Definition 1.3.3 (neighbourhood) *The ϵ-neighbourhood of $x^* \in \mathbb{R}^n$ is defined by*

$$\mathcal{N}(x^*) = \{y : \|x^* - y\| \leq \epsilon\}. \tag{1.3}$$

Definition 1.3.4 (locally optimal solution) *A feasible solution $x^* \in \mathcal{F} \neq \emptyset$ is a locally optimal solution of (1.1) if there exists a neighbourhood $\mathcal{N}(x^*)$ such that $\forall y \in \mathcal{F} \cap \mathcal{N}(x^*) : f(x^*) \leq f(y)$ holds.*

Definition 1.3.5 (active/inactive constraint) *An inequality constraint $h_i(\tilde{x}) \geq 0$ is an active constraint in point $\tilde{x} \in \mathcal{F}$ if $h_i(\tilde{x}) = 0$ holds, otherwise $h_i(\tilde{x}) \geq 0$ is an inactive constraint.*

Definition 1.3.6 (active set) *The active set of inequality constraints in a point $\tilde{x}$ is the index set $\mathcal{A}(\tilde{x}) \subseteq \{1, \dots q\}$ of all active inequality constraints.*

Definition 1.3.7 (LICQ) *The linear independence constraint qualification (LICQ) holds at $\tilde{x} \in \mathcal{F}$ if all vectors $\nabla g_i(\tilde{x})$ for $i = 1, \dots, p$ and $\nabla h_j(\tilde{x})$ for $j \in \mathcal{A}(\tilde{x})$ are linearly independent.*

In the theory of nonlinear programming and also in practical methods, the following theorem has central importance.

Theorem 1.3.1 (Karush-Kuhn-Tucker optimality conditions) *If $x^* \in \mathcal{F}$ is a locally optimal solution of (1.1) and the LICQ holds at x^*, then there exist Lagrange multipliers $\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^q$ such that*

$$\nabla f(x^*) - \nabla g(x^*)\lambda - \nabla h(x^*)\mu = 0, \tag{1.4a}$$

$$g(x^*) = 0 \quad , \tag{1.4b}$$

$$h(x^*) \geq 0, \tag{1.4c}$$

$$h_i(x^*)\mu_i = 0, \quad i = 1, \dots, q, \tag{1.4d}$$

$$\mu_i \geq 0, \quad i = 1, \dots, q, \tag{1.4e}$$

hold.

We often refer to the residual of (1.4a) as *dual infeasibility*, to the one of (1.4b)-(1.4c) as *primal infeasibility*. The constraints in (1.4c)-(1.4e) having non-smooth right-hand side are called *complementarity conditions*.

Note that this theorem is a necessary condition for optimality and thus having a KKT point (x^*, λ^*, μ^*) that satisfies (1.4) does not imply local optimality in x^*.

1.4 Solution methods for Nonlinear Programming

Since one of the contributions of this thesis is a tailored NLP solver for special problems, we discuss the two most important general purpose approaches to solve

NLPs. We restate the problem that is treated by the methods we discuss in the following,

$$\min_x \ f(x) \tag{1.5a}$$

$$\text{s.t. } g(x) = 0 \tag{1.5b}$$

$$h(x) \geq 0. \tag{1.5c}$$

Sequential Quadratic Programming (SQP)

One possible way to solve (1.5) is to employ sequential quadratic programming, which was first published in [115]. For a good introduction to SQP methods, see [110, Chapter 18]. In each iteration of this approach, a quadratic approximation of the original problem is computed resulting in a quadratic program (QP) that is possibly easier to solve. The optimal solution of this subproblem yields a search direction in the original space. Along this search direction, a scalar stepsize is sought for e.g. by means of a backtracking line-search procedure. This approach is repeated until a KKT point of (1.5) is found up to some accuracy.

Definition 1.4.1 (Lagrange function) *The Lagrange function of (1.5) is defined by*

$$L(x, \lambda, \mu) := f(x) - \lambda^T g(x) - \mu^T h(x), \tag{1.6}$$

where $\lambda \in \mathbb{R}^p$ and $\mu \in \mathbb{R}^q$ are the Lagrange multipliers corresponding to (1.5b) and (1.5c), respectively.

The gradient of the Lagrangian function with respect to x is given by

$$\nabla_x L(x, \lambda, \mu) = \nabla f(x) - \nabla g(x)\lambda - \nabla h(x)\mu, \tag{1.7}$$

and the Lagrangian Hessian is

$$\nabla_x^2 L(x, \lambda, \mu) = \nabla^2 f(x) - \sum_{i=1}^{p} \lambda_i \nabla^2 g_i(x) - \sum_{i=1}^{q} \mu_i \nabla^2 h_i(x). \tag{1.8}$$

In each iteration of the SQP method, an approximation of the original problem is calculated and solved in the actual iterate $x^{(k)}$ of the form

$$\min_{\Delta x} \ \frac{1}{2}\Delta x^T B^{(k)} \Delta x + \nabla f(x^{(k)})^T \Delta x \tag{1.9a}$$

$$\text{s.t. } g(x^{(k)}) + \nabla g(x^{(k)})^T \Delta x = 0 \tag{1.9b}$$

$$h(x^{(k)}) + \nabla h(x^{(k)})^T \Delta x \geq 0. \tag{1.9c}$$

Here, $B^{(k)} \approx \nabla_x^2 L(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$ is the Lagrange Hessian approximation.

The optimal primal solution of (1.9) denoted by Δx_{QP} provides a search direction in the original space of x. The optimal dual solutions of (1.9) are denoted by $\lambda_{\mathrm{QP}}, \mu_{\mathrm{QP}}$. One has to find an appropriate stepsize t in the given direction in order to make the primal and dual steps

$$x^{(k+1)} = x^{(k)} + t\Delta x_{\mathrm{QP}}, \tag{1.10a}$$

$$\lambda^{(k+1)} = (1-t)\lambda^{(k)} + t\lambda_{\mathrm{QP}}, \tag{1.10b}$$

$$\mu^{(k+1)} = (1-t)\mu^{(k)} + t\mu_{\mathrm{QP}}, \tag{1.10c}$$

For this purpose one can use backtracking line-search with the so-called Armijo condition. For an introduction on general globalization techniques, see [110, Chapter 3]

Interior-point method (IP)

The other popular NLP solution method family is the interior-point schemes. For a more detailed discussion of the convex case we refer the reader to the textbook [144]. The main difference between SQP methods and IP methods is in the manner how the inequalities are treated. SQP methods typically rely on underlying QP solvers to determine the correct active set, while IP methods convert the non-smooth complementarity conditions into smooth approximations. We discuss the basics of a primal-dual interior-point method.

We restate the KKT conditions of (1.5)

$$\nabla f(x^*) - \nabla g(x^*)\lambda - \nabla h(x^*)\mu = 0 \tag{1.11a}$$

$$g(x^*) = 0 \tag{1.11b}$$

$$h(x^*) \geq 0 \tag{1.11c}$$

$$h_i(x^*)\mu_i = 0, \quad i = 1, \ldots, q \tag{1.11d}$$

$$\mu_i \geq 0, \quad i = 1, \ldots, q. \tag{1.11e}$$

Since (1.11d)-(1.11e) are non-smooth conditions, (1.11c)-(1.11e) are replaced with a smooth approximation of the form

$$\nabla f(x^*) - \nabla g(x^*)\lambda - \nabla h(x^*)\mu = 0 \tag{1.12a}$$

$$g(x^*) = 0 \tag{1.12b}$$

$$h_i(x^*)\mu_i = \tau. \tag{1.12c}$$

Here, $\tau \in \mathbb{R}$ is called the *barrier parameter*. Observe that once $\tau \to 0$, we obtain a better and better approximation of the complementarity conditions. Note that (1.12) is a root finding problem. In an IP method, the barrier parameter τ is driven to zero, while the root finding subproblems are solved by a Newton method. In each iteration of the Newton method, the following linear system is solved

$$\begin{bmatrix} B^{(k)} & -\nabla g(x^{(k)}) & -\nabla h(x^{(k)}) \\ \nabla g(x^{(k)})^T & 0 & 0 \\ M^{(k)}\nabla h(x^{(k)}) & 0 & h(x^{(k)}) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = \begin{bmatrix} -\nabla_x L(x^{(k)}, \lambda^{(k)}, \mu^{(k)}) \\ -g(x^{(k)}) \\ \tau - M^{(k)}h(x^{(k)}) \end{bmatrix}, \tag{1.13}$$

where $M^{(k)} = \operatorname{diag}(\mu_1^{(k)}, \ldots, \mu_q^{(k)})$. The solution of (1.12) for a fixed τ provides the next primal-dual iterate $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)})$. In the outer iteration, τ is decreased by a factor of $\beta \in (0, 1)$. Under some assumptions, one can show that if $\tau \to 0$ then $x^{(k+1)} \to x^*$.

1.5 Convex Quadratic Programming

A special and very important case of nonlinear programming is convex quadratic programming (QP). The task in this problem class is to find the minimizer of a convex quadratic function that is inside an polyhedral set. More formally, we want to solve

$$\min_x \ \frac{1}{2}x^T Q x + c^T x \tag{1.14a}$$

$$\text{s.t. } Ax + b = 0 \tag{1.14b}$$

$$Cx + d \leq 0, \tag{1.14c}$$

where $x \in \mathbb{R}^n$ is the optimization variable, $0 \preceq Q \in \mathbb{R}^{n\times n}$ is the Hessian matrix, and $c \in \mathbb{R}^n$ is the linear coefficient vector. Furthermore, $A \in \mathbb{R}^{p\times n}$, $b \in \mathbb{R}^p$, $C \in \mathbb{R}^{q\times n}$, and $d \in \mathbb{R}^q$ form the affine equalities and inequalities, respectively. The

KKT condition system for (1.14) simplifies to

$$Qx + c - A^T\lambda + C^T\mu = 0 \tag{1.15a}$$

$$Ax + b = 0 \tag{1.15b}$$

$$Cx + d \leq 0 \tag{1.15c}$$

$$M(Cx + d) = 0 \tag{1.15d}$$

$$\mu \geq 0, \tag{1.15e}$$

where $\lambda \in \mathbb{R}^p$, $\mu \in \mathbb{R}^q$ are Lagrange multipliers corresponding to (1.14b) and (1.14c), respectively, and $M = \operatorname{diag}(\mu_1, \ldots, \mu_q)$. It has to be emphasized that the KKT conditions for convex problems are not only necessary, but also sufficient conditions. Thus, a KKT point of (1.15) is a globally optimal solution of (1.14) and vice versa. Observe that the class of QP problems includes linear programming (LP) once $Q = 0$.

There are two classes of methods which find a solution of (1.15), and several other special purpose approaches. The first family is the one of active set (AS) methods, which try to find the optimal active set by adding and removing inequality constraints following some strategy. Two subproblems with two different active set guesses are very similar, which is heavily exploited in such methods. Active set methods have very good hot-starting capabilities once the problem data change, since an almost optimal active set and possibly matrix factorizations are already available from the previous QP solution. For these reasons, active set methods are particularly suitable for a series of optimal control problems. The second famous family is the interior-point methods. These approaches, as we have seen in the nonlinear case, transform the non-smooth conditions to a smooth approximation, such that in each iteration of the IP method a structured linear system is solved. For a more detailed description about QP methods in general, see [110, Chapter 16].

1.6 Optimal control

In the area of optimal control, the theory and methodology of optimization are used, although these optimization problems typically describe the dynamical behaviour of a certain mechanical, physical or chemical system and possess special structure. The system under consideration has to be controlled in an optimal manner, resulting in e.g. maximal power production, minimal power consumption, minimal time, etc., while respecting operational constraints such as physical limits of actuators, safety

limits, etc. We differentiate between linear and nonlinear optimal control depending on the complexity of the used plant model and control goals.

The terminology of optimal control is frequently used in this thesis, which we introduce now.

Definition 1.6.1 (dynamic variable) *A function $x(t) : \mathbb{R} \to \mathbb{R}^n$ is called a dynamic variable if its value depends on and thus evolves in time.*

Definition 1.6.2 (ordinary differential equation (ODE)) *An equation that gives the time derivative of a dynamic variable in the form $\dot{x}(t) = f(x(t))$ is called an explicit ordinary differential equation (ODE).*

Definition 1.6.3 (initial value problem (IVP)) *An initial value problem is formed by an ODE and an initial value defined by*

$$\dot{x}(t) = f(x(t)) \tag{1.16}$$

$$x(0) = x_0, \tag{1.17}$$

where $x_0 \in \mathbb{R}^n$ is fixed, $T > 0$ is the end time, and the value of $x(T)$ is sought for.

For the direct solution of IVPs, one may use integrators. These subroutines propagate the value of the dynamic variable on the time interval $[0, T]$ following some integration rule such as the Runge-Kutta (RK) scheme or backward differentiation formula (BDF). Integrator schemes have various properties such as fixed/variable stepsize, explicit/implicit methods, presence of precision guarantee or ability to deliver derivatives. In Newton-type optimization algorithms the calculation of derivatives is necessary, which is essentially the sensitivity of the differential equation with respect to the initial value or some system parameters. In order to choose the appropriate integration routine, one has to be aware of the features of the considered system. In this thesis, we pay more attention to algorithmic design and we always assume that an integrator method is available which can deliver reliable and accurate solution along with derivatives. For further reading we refer to the textbooks [71, 72].

Definition 1.6.4 (integrator) *A method is called an integrator once it can deliver the solution $x(t)$ of the IVP*

$$\dot{x}(t) = f(x(t)) \tag{1.18}$$

$$x(0) = x_0, \tag{1.19}$$

at any $t \in [0, T]$.

In particular, we will treat this integrator routine as a differentiable function. We define a static function $F(\cdot)$ depending on the initial value x_0 and some model parameters p, which delivers the value of the dynamic variable at time T, i.e. $F(x_0, p) := x(T)$.

Distributed systems

In this thesis, the focus is on distributed systems, thus we shortly discuss what we mean by this. In general, we call a system distributed if it is formed by connected functional or structural building blocks. We refer to these building blocks as subsystems or agents. The behaviour of each agent depends on the decisions and the evolution of other subsystems. More mathematically speaking, each subsystem is described by some differential equations that depend not only on its own dynamic variables (system states) and parameters (control input variables), but in addition, on state or input variables of some other subsystems.

Definition 1.6.5 (distributed system) *A distributed system formed by N subsystems is defined by*

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t), z_i(t)) \tag{1.20a}$$

$$z_i(t) = g_i(x(t), u(t)), \qquad i = 1, \dots, M \tag{1.20b}$$

Here, $x_i(t) \in \mathbb{R}^{n_{x_i}}, u_i(t) \in \mathbb{R}^{n_{u_i}}, z_i(t) \in \mathbb{R}^{n_{z_i}}$ are the state variable, the control input variable, and the coupling input variable of subsystem i, respectively. We denote the collection of all state variables, control input variables by $x(t)$ and $u(t)$, respectively. The time derivative of $x_i(t)$ depends on the local system states $x_i(t)$, some control input $u_i(t)$ and coupling input $z_i(t)$ given in (1.20a). The coupling input variable $z_i(t)$ of subsystem i may depend (nonlinearly) on state or control input variables of other subsystems, which is characterized by (1.20b).

For instance, given $N = 2$ subsystems, the dynamics of subsystem 2 may depend on the states of subsystem 1, then the dependency can be described by

$$\dot{x}_1(t) = f_1(x_1(t), u_1(t)) \tag{1.21a}$$

$$\dot{x}_2(t) = f_2(x_1(t), x_2(t), u_2(t)). \tag{1.21b}$$

As these are coupled equations, their integration has to take place simultaneously as one system, which is what we want to avoid due to the computation burden this would require.

Another example is when we have $N = 2$ subsystems, the dynamics are completely decoupled, but the control variables are shared. In this case, we have

$$\dot{x}_1(t) = f_1(x_1(t), u(t)) \tag{1.22a}$$

$$\dot{x}_2(t) = f_2(x_2(t), u(t)). \tag{1.22b}$$

Nonlinear optimal control

A nonlinear optimal control problem represents a mathematical model of a real-world dynamic system. The goal is to calculate and inject such a control input signal into a nonlinear system which drives the system to an optimal trajectory, e.g. tracking a power reference, or reaching an economical optimum, e.g. maximal power production, while respecting operational constraints. If the calculated control signal depends on the actual status of the plant, we talk about feedback control or closed-loop control. Otherwise, we operate an open-loop controller.

We consider an optimization problem that has central importance in optimal control theory and practise.

Definition 1.6.6 (nonlinear optimal control problem (OCP)) *A nonlinear optimal control problem on the prediction horizon $[0, T]$ is defined by*

$$\min_{x(\cdot), u(\cdot)} \int_0^T \ell(x(t), u(t))\mathrm{d}t + \overline{\ell}(x(T), u(T)) \tag{1.23a}$$

$$\text{s.t. } \dot{x}(t) = f(x(t), u(t)) \tag{1.23b}$$

$$x(0) = x_0 \tag{1.23c}$$

$$p(x(t), u(t)) \geq 0 \tag{1.23d}$$

$$\underline{x} \leq x(t) \leq \overline{x}, \quad \underline{u} \leq u(t) \leq \overline{u}, \quad t \in [0, T]. \tag{1.23e}$$

Here, $x(t) \in \mathbb{R}^{n_x}$ and $u(t) \in \mathbb{R}^{n_u}$ are dynamic variables; the states and the control inputs of the controlled system, respectively. The first term of the objective function (1.23a) is called the Lagrange term, and the second is the Mayer term. In (1.23b), the system behaviour is described by an explicit ODE, of which initial value is fixed by (1.23c). Note that the ODE right-hand side now depends on an extra dyanamic variable $u(t)$, which is essentially the degree of freedom in this problem. By (1.23d), one can introduce so called path constraints, i.e. nonlinear constraints on the state or control trajectories throughout the whole prediction horizon $[0, T]$. In (1.23e), state and input trajectories can be constrained by simple bounds.

Note that (1.23) is an infinite dimensional optimization problem and the optimization variables are functionals themselves. In order to be able to employ the state-of-the-art methods from nonlinear programming, the problem has to be discretized in time. This way of treating OCPs are often regarded as direct approaches of optimal control, in particular, direct single shooting, direct multiple shooting, and direct collocation. In this thesis, we concentrate on shooting methods, which essentially rely on underlying integrators to carry out the discretization. For further information on direct approaches, we refer to [15, 16, 17].

Linear optimal control

Similarly to nonlinear optimal control, in linear optimal control, we also want to control a system in an optimal manner, while meeting operational conditions. The difference is that in the latter, the system to be controlled either has a simpler, i.e. linear, behaviour or we have a sufficiently good linear model of a nonlinear system. Typically for tracking problems of systems operating close to the steady state, a linear controller is sufficient.

Definition 1.6.7 (linear optimal control problem) *A linear optimal control problem is of the form*

$$\min_{x(\cdot),u(\cdot)} \int_0^T \left[\frac{1}{2}x(t)^T Q x(t) + c^T x(t)\right] + \left[\frac{1}{2}u(t)^T R u(t) + d^T u(t)\right] \mathrm{d}t \tag{1.24a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t) \tag{1.24b}$$

$$x(0) = x_0 \tag{1.24c}$$

$$Cx(t) + Du(t) + e \geq 0 \tag{1.24d}$$

$$\underline{x} \leq x(t) \leq \overline{x}, \quad \underline{u} \leq u(t) \leq \overline{u}, \quad t \in [0, T]. \tag{1.24e}$$

Here, $x(t) \in \mathbb{R}^{n_x}$ and $u(t) \in \mathbb{R}^{n_u}$ denote the state variable and the control input variable, respectively. In (1.24a), the first two terms penalize deviation from a state reference with weighting matrix $Q \succeq 0$, while the last two terms penalize control actions with weighting matrix $R \succeq 0$. The system dynamics are determined by (1.24b) and (1.24c). Since this problem is again an infinite dimensional problem, the state and control trajectories have to be discretized in time. The resulting problem is a structured convex QP.

Part I

Large Scale Nonlinear Optimal Control

Chapter 2

Distributed Multiple Shooting (DMS)

In this chapter, we develop a methodology for the optimal control of distributed systems using shooting methods. A method for solving nonlinear optimal control problems is presented, which efficiently exploits the distributed structure of the system to be controlled. The approach can be considered as the generalization of the *direct multiple shooting* method [20], which enables for parallel simulation on different time intervals. In addition, we propose to introduce parallel simulation of different subsystems. The discussion to be presented here has overlap with the work published in [128, 127, 83].

The optimization problem that we treat in this chapter is stated in Section 2.1. Section 2.2 gives a survey of similar methods that can be found in the literature. Section 2.3 explains the time domain based decomposition, which is followed by the presentation of the spatial decomposition in Section 2.4. We summarize the DMS approach in an algorithmic way, in Section 2.5. The chapter ends with Section 2.6, in which a variant of the DMS scheme is introduced.

2.1 Problem statement and contribution

We are concerned with the optimal control problem (OCP) of a distributed system consisting of N agents influencing each other.

In order to cover all types of interdependency, we introduce two extra dynamic

variables $y_i(t)$ and $z_i(t)$ for each subsystem. On the one hand, we refer to $y_i(t)$ as a coupling output variable, which represents all the information about subsystem i that may be necessary at any other subsystem such as a nonlinear function of $x_i(t)$ or $u_i(t)$. On the other hand, we introduce $z_i(t)$ as coupling input variable, which represents all the information about other subsystems that is needed to integrate the subsystem itself. We shall couple variables $z_i(t)$ and $y_i(t)$ as constraints.

We regard the following nonlinear optimal control problem for distributed systems.

$$\min_{\substack{x(\cdot),u(\cdot),\\ z(\cdot),y(\cdot)}} \sum_{k=1}^{N} \int_0^T \ell_k(x_k(t), u_k(t), z_k(t))\mathrm{d}t + \bar{\ell}_k\left(x_k(T)\right) \tag{2.1a}$$

$$\text{s.t. } \dot{x}_i(t) = f_i(x_i(t), u_i(t), z_i(t)) \tag{2.1b}$$

$$y_i(t) = g_i(x_i(t), u_i(t), z_i(t)) \tag{2.1c}$$

$$x_i(0) = \bar{x}_i^0 \tag{2.1d}$$

$$z_i(t) = \sum_{j=1}^{N} A_{ij} y_j(t) \tag{2.1e}$$

$$p_i(x_i(t), u_i(t)) \geq 0, \quad t \in [0, T],\ i = 1, \ldots, N. \tag{2.1f}$$

Here, $x_i(t) \in \mathbb{R}^{n_{x_i}}$, $u_i(t) \in \mathbb{R}^{n_{u_i}}$, $z_i(t) \in \mathbb{R}^{n_{z_i}}$, $y_i(t) \in \mathbb{R}^{n_{y_i}}$ denote the state variable, control input variable, coupling input variable, and coupling output variable of agent i, respectively. Each agent i has a local nonlinear stage cost function $\ell_i(\cdot)$ and a local nonlinear terminal cost function $\bar{\ell}_i(\cdot)$, see (2.1a). Each subsystem is driven by nonlinear dynamics $f_i(\cdot)$ (2.1b) subject to local state and input constraints (2.1f). At any time $t \in [0, T]$ the system output of agent i is determined by $g_i(\cdot)$ (2.1c). The connection between the agents is ensured by the linear constraint (2.1e).

The classical direct approaches tackle (2.1) as one big system, not considering the distributed nature and the inherent structure of controlled system. The main contribution of this chapter is the design of a special integration method which enables for parallel integration and thus sensitivity generation of different subsystems on different time intervals. Each subsystem might have a tailored integration routine respecting the features of the corresponding subsystem such as stiffness, etc.

2.2 Literature survey

The control of distributed systems has been an active research area [99, 47], especially for model predictive control applied to distributed systems resulting in hierarchical and distributed predictive control approaches [26, 97, 142, 122, 120, 129]. Most of these methods deploy local controllers to the subsystems that respect some coordination and communication scheme. This fact results in typically many more iterations (sublinear rate) needed to calculate an optimal control input (if at all possible) when compared to centralized solution methods.

One way to solve nonlinear optimal control problems is direct multiple shooting [20], which employs an integrator to solve initial value problems on each control interval. The method proposed in this chapter, in addition to the time domain based decomposition, introduces a spatial domain based decomposition, by which all subsystems can be integrated individually and independently. The consistency between the subdomains is restored by a Newton-type method.

Decomposition techniques are often used for the simulation of large differential equations. In particular, the so called waveform relaxation technique is employed to solve implicit differential-algebraic equations [90], spatially discretized partial differential equations (PDE) [52], nonlinear ordinary differential equations with periodic constraints [82]. These schemes decompose the original problem into smaller, but independent subsystems, which are then easier to solve. The consistency between the subsystems is ensured via a Gauss-Seidel type method, which has a linear convergence rate. For further reading, we refer the reader to [141, 54].

In the context of partial differential equations, the domain decomposition methods are very similar to waveform relaxation techniques. However, they make explicit use of the geometry and discretization of the underlying PDE, see the textbooks [118, 133]. The consistency of connecting domains is recovered by a gradient method in [42, 68].

The time-parallel integration of ODEs by the direct multiple shooting method [20] is analyzed in [53]. In this work, the consistency between connecting time intervals is restored by a Newton method.

Since we aim to solve a nonlinear optimal control problem, we need to simulate (and linearize) the subsystems several times. In this case, the application of a Gauss-Seidel type approach would result in slow convergence (or even no convergence), whereas with a Newton method we expect faster practical convergence. Our approach, which we call *distributed multiple shooting* (DMS), can be regarded as a waveform relaxation method, in which the connecting subdomains are made consistent by a Newton method, along with time-parallel integration of each subsystem in the spirit of [20] and [53].

2.3 Decomposition in the time domain

In this section, we discuss how to decompose a general, not necessarily distributed system in the time domain with *direct multiple shooting* [20]. Since this approach does not exploit the distributed structure of the system, we drop the subindex of agents and consider a special case of (2.1) with $N = 1$ in the form of

$$\min_{x,u} \int_0^T \ell(x(t), u(t))\mathrm{d}t + \overline{\ell}\left(x(T)\right) \tag{2.2a}$$

$$\text{s.t. } \dot{x}(t) = f(x(t), u(t)) \tag{2.2b}$$

$$p(x(t), u(t)) \geq 0, \quad t \in [0, T], \tag{2.2c}$$

where $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$ The direct multiple shooting algorithm introduces a time grid of $M + 1$ time points as

$$0 = t^1 < t^2 < \cdots < t^{M+1} = T \tag{2.3}$$

On each shooting interval $[t^i, t^{i+1}]$, a finite discretization of the control input variable $u(t)$ is introduced. To this end, we introduce piecewise constant functions, i.e. for all $i = 1, \ldots, M + 1$, $t \in [t^i, t^{i+1})$

$$u(t) = u^i \in \mathbb{R}^{n_u}. \tag{2.4}$$

On each shooting interval $[t^i, t^{i+1}]$, the state trajectory is discretized by an integration scheme such as the Runge-Kutta methods or the backward differentiation formula. We assume that once an initial state value x^i and control input u^i is given on shooting interval $[t^i, t^{i+1})$, we have an integrator available that can propagate the state trajectory reliably. More precisely, we define $F^i(x^i, u^i) := x(t^{i+1})$ with

$$\dot{x}(t) = f(x(t), u^i) \tag{2.5a}$$

$$x(t^i) = x^i. \tag{2.5b}$$

Note that $F^i(\cdot)$ corresponds to shooting interval $[t^i, t^{i+1})$. In the remainder, we will treat $F^i(\cdot)$ as a smooth function that can deliver first and second order derivatives as well.

Now we describe the resulting NLP in the form of

$$\min_{x,u} \sum_{j=1}^{M} \ell(x^j, u^j) + \overline{\ell}\left(x^{M+1}\right) \tag{2.6a}$$

$$\text{s.t. } x^{i+1} = F^i(x^i, u^i), \tag{2.6b}$$

$$p(x^i, u^i) \geq 0, \quad i = 1, \ldots, M \tag{2.6c}$$

Here, the objective (2.6a) sums up the original objective only in the shooting nodes, the constraint (2.6b) ensures the continuity of the states between shooting intervals, and the inequality (2.6c) imposes the state and control constraints only in the shooting nodes. The solution of (2.6) may take place by using standard NLP solvers, such as IP or SQP methods. In any case, the first and possibly the second order derivatives of $F^i(x^i, u^i)$ are necessary for each $i = 1, \dots, M$. The computation of these derivatives are independent of each other and thus can be parallelized by M processes. This parallization is even more important if we consider that the computation time of solving (2.6) is dominated by the calculation of sensitivities throughout the iterations of the optimization routine.

Essentially, the time-parallel integration exploits the fact that on different shooting intervals the dynamics of the system can be integrated simultaneously. The consistency between connecting time intervals is ensured by a Newton-type method.

2.4 Decomposition in the spatial domain

In this section, we return to our original formulation introduced in (2.1) and in addition to the time domain based decomposition we present a decomposition in the spatial domain. For this purpose, we consider a distributed system where the subsystem boundaries are well-defined. Since the dynamic equations of different agents are coupled together via the variables $z(t)$ and $y(t)$, we need to introduce a finite discretization of these. Although other choices are possible we use orthogonal polynomials to approximate variables $z(t)$ and $y(t)$.

Legendre polynomials

We define *normalized Legendre polynomials* γ_p of order p and γ_q of order q as

$$\int_{-1}^{1} \gamma_p(t)\gamma_q(t)dt = \begin{cases} 1 & \text{if } p \neq q \\ 0 & \text{otherwise} \end{cases}. \tag{2.7}$$

Also, we define the *normalized Legendre basis* of order S as

$$\Gamma_S(t) = [\gamma_0(t), \gamma_1(t), \dots, \gamma_S(t)]^T. \tag{2.8}$$

Note that in the i-th row of $\Gamma_S(t)$ with $i = 1, \dots S+1$ there is a normalized Legendre polynomial of order $i-1$. In order to keep the notation simple, we assume that the degree of the Legendre basis is S in all connections. In practice, it can be chosen differently.

Once we have a dynamic variable $w(t) \in \mathbb{R}^w$ defined in $t \in [t_0, t_f]$, it can be shown that the coefficients may be given by

$$W = \frac{2}{t_f - t_0} \int_{t_0}^{t_f} \Gamma_S(\hat{t}) w(t)^T dt, \tag{2.9}$$

where $W \in \mathbb{R}^{S+1 \times w}$ and $\hat{t}$ is defined by

$$\hat{t} := \frac{2t - t_0 - t_f}{t_f - t_0}. \tag{2.10}$$

Observe that (2.9) involves the calculation of an integral numerically, often referred to as quadrature, based on the value of $w(t)$ at specific time points. By using $\hat{t}$, the Legendre basis is translated and scaled from $[-1, 1]$ to the interval $[t_0, t_f]$, where we want to approximate $w(t)$. Note that the integrand on the right-hand side of (2.9) is a matrix and that this operation ought to be carried out component-wise. To be more precise, if we want to approximate $w(t) \in \mathbb{R}$ on $[t_0, t_f]$ with Legendre polynomials of order S, we have to evaluate $(S + 1) \times w$ quadratures.

In the other way around, if we have a matrix $W \in \mathbb{R}^{(S+1) \times w}$ of Legendre coefficients valid on $[t_0, t_f]$, we can recover the approximated variable $\hat{w}(t) \in \mathbb{R}^w$ up to approximation error at time $t \in [t_0, t_f]$, by using

$$\hat{w}(t) = W^T \Gamma_s(\hat{t}). \tag{2.11}$$

Now we return to the discussion of the spatial decomposition. The core idea of decomposing not only in time, but in space as well is that the coupling variables $y_i(t)$ and $z_j(t)$ are made equal in terms of approximating Legendre polynomials. In other words, not the infinite dimensional variables are made equal, but rather their finite dimensional Legendre polynomial based approximations. In the first step, we apply the time domain based decomposition explained in Section 2.3 with time grid $0 = t^1 < t^2 < \cdots < t^{M+1} = T$, which discretizes the state $x_i(t)$ and the control variables $u_i(t)$. But since the dynamics of subsystem i involves $z_i(t)$, which is coupled to certain other $y_j(t)$ variables, one cannot integrate the subsystems independently. To this end, we approximate these quantities with their Legendre polynomial based approximation. Let Z_i^l, Y_i^l denote the coefficient matrix that approximate the coupling input variable, system output variables of subsystem i on time interval $[t^l, t^{l+1}]$. Since these matrices are finite quantities, we can introduce the simulation of different subsystems independently. The corresponding couplings are ensured in terms of the time polynomials. More formally, we obtain the following

NLP.

$$\min_{\substack{x,u,\\ z,y}} \sum_{k=1}^{N}\sum_{j=1}^{M} \ell_k^j(x_k^j, u_k^j, (Z_k^j)^T\Gamma_S(t^j)) + \ell_k^{M+1}(x_k^{M+1}) \tag{2.12a}$$

$$\text{s.t. } x_i^{l+1} = F_i^l(x_i^l, u_i^l, Z_i^l) \tag{2.12b}$$

$$Y_i^l = G_i^l(x_i^l, u_i^l, Z_i^l) \tag{2.12c}$$

$$x_i^1 = \overline{x}_i^0 \tag{2.12d}$$

$$Z_i^l = \textstyle\sum_{j=1}^{N} \hat{A}_{ij} Y_j^l \tag{2.12e}$$

$$p_i\left(x_i^l, u_i^l, (Z_i^l)^T\Gamma_s(t^l)\right) \geq 0 \tag{2.12f}$$

$$i = 1, \ldots, N \qquad l = 1, \ldots, M$$

Here, the objective function (2.12a) sums up the stage cost of each subsystem in each shooting node. The constraint (2.12b) ensures the continuity of the states of each subsystem. The function $F_i^l(\cdot)$ is a simulator, which integrates subsystem i on shooting interval $[t^l, t^{l+1}]$ with initial state value x_i^l control input u_i^l and coupling input approximation Z_i^l. In (2.12c), $G_i^l(\cdot)$ calculates the approximation of the system output $g_i(\cdot)$ in terms of Legendre polynomial coefficients. The constraint (2.12e) couples the subsystems together in terms of the Legendre polynomial based approximations. The matrices $\hat{A}_{i,j}$ are directly derivable from the original coupling matrices A_{ij}. The state and control input constraints of subsystem i are imposed in the shooting nodes t^l, $l = 1, \ldots, M$ using the corresponding state value x_i^l, control input u_i^l and the approximated coupling input $(Z_i^l)^T\Gamma_s(t^l)$. The quadratures calculating the Legendre approximation can be evaluated along with the dynamics.

It is important to note that when solving (2.12) with an NLP solver, such as IP or SQP methods, the first derivatives, and possibly second derivatives if using exact Lagrange Hessian, of $F_i^l(\cdot)$ and $G_i^l(\cdot)$ need to be calculated. The most powerful feature of DMS is that the evaluation and linearization of these functions is parallelized into $M \times N$ parallel processes. For each shooting interval $[t^l, t^{l+1}]$ and for each subsystem $i = 1, \ldots, N$ there is a reliable integrator that delivers the propagated state values and the necessary sensitivity information. For instance, in Figure 2.1, the parallel integrators of a distributed system with chain-topology is depicted.

Moreover, one can employ tailored integrators for different subsystems having special structure or features.

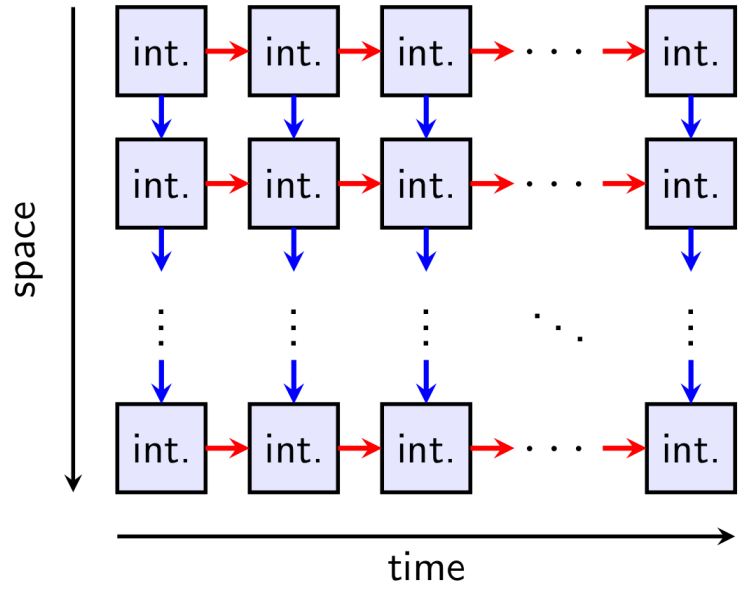


Figure 2.1: The $M \times N$ parallel simulators of a system having a chain-topology. The vertical arrows indicate spatial coupling, while the horizontal arrows suggest time domain based coupling.

2.5 DMS - Parallel simulation, centralized optimization

In this section, we summarize the necessary ingredients, advantages and drawbacks of the DMS method and give an algorithmic description for easy understanding and implementation.

The method of distributed multiple shooting is intended to be used in the optimal control of distributed systems with Newton-type methods, i.e. with SQP and IP approaches. It introduces a time domain based and a spatial domain based decomposition in order to parallelize the linearization process on M time intervals and in N subsystems. The simulators may follow different integration rules, but have to be able to provide sensitivity information. In addition to the $M \cdot N$ simulator processes, one needs an extra coordinator process that is dedicated for optimization. The coordinator sends the first linearization point to the simulator nodes, which simulate their corresponding subsystem on the corresponding time interval and in return they send the linearization back to the coordinator. This carries out

one iteration of the optimization routine, typically solves a large, but structured linear system or a QP problem. After having broadcasted the next linearization point, the iterations continue until the optimality criterion holds. One drawback of DMS is that its optimization phase is centralized and serial, however, this could be parallelized or even distributed. Moreover, one has to transmit possibly dense matrices from the integrator nodes to the coordinator, which may be expensive. In Part II, we consider these disadvantages and investigate possibilities to decentralize the optimization phase.

We give the description of DMS in Algorithm 2.1. For a software implementation

Algorithm 2.1: Distributed Multiple Shooting (DMS)

Problem preparation

1. Introduce time mesh $0 = t_1 \leq \cdots \leq t_{M+1} = T$ and optimization variables x_i^l and u_i^l.
2. Choose the degree of Legendre polynomials S and introduce optimization variables Z_i^l and Y_i^l.
3. Reformulate local dynamic equations f_i for each time interval $[t_j, t_{j+1})$: plug $(Z_i^l)^T \Gamma_S(\hat{t})$ into $z_i(t)$.
4. Extend local dynamic equations with quadratures that calculate local output approximations.

Solution method

1. Evaluate and linearize $\ell_i^l(\cdot), F_i^l(\cdot)$ and $G_i^l(\cdot)$ in the actual linearization point using $M \cdot N$ parallel processes.
2. Collect linearizations at the dedicated coordinator process.
3. Determine the next linearization point using an NLP solver.
4. Communicate the new linearization point to the $M \cdot N$ simulator processes.
5. If convergence is achieved then quit, otherwise go to Step 1.

one requires the following building blocks.

- *Symbolic computer algebra:* A software environment in which one can implement model equations symbolically, see e.g. [4].

- *Simulator:* A reliable solver for ODEs that can deliver first and second order sensitivity information, see e.g. [76].
- *NLP solver:* A Newton-type optimization method that solves general nonlinear programs, see e.g. [109, 143].
- *Parallelization environment:* A framework for parallel execution, communication by message passing, see e.g. [51, 34].

Although DMS can be employed within any Newton-type method, we discuss the special case when a Sequential Quadratic Programming (SQP) method [115] is used as NLP solver. Given the problem in (2.12), in each iteration of the SQP method an approximation of the original problem is calculated resulting in a QP. The solution of this QP provides a search direction in the original variable space. The proper stepsize in the given direction is calculated in terms of a line-search strategy. For the sake of a simple discussion, we assume that Y_i^l and Z_i^l are vectors. This is a mild assumption, since the matrices Y_i^l and Z_i^l can be vectorized and (2.12) can be modified correspondingly. We introduce the Lagrangian function of (2.12) as

$$\begin{aligned}\mathcal{L}(x,u,z,y,\lambda,\sigma,\nu,\eta,\mu) &:= \textstyle\sum_{k=1}^{N}\sum_{j=1}^{M} \ell_k^j(x_k^j,u_k^j,(Z_k^j)^T\Gamma_S(t^j)) + \overline{\ell}_k^{M+1}(x_k^{M+1}) \\ &+ \sum_{i=1}^{N}\sum_{l=1}^{M}\Bigg(\left(\lambda_i^l\right)^T\left(x_i^{l+1} - F_i^l(x_i^l,u_i^l,Z_i^l)\right) + \left(\sigma_i^l\right)^T\left(Y_i^l - G_i^l(x_i^l,u_i^l,Z_i^l)\right) \\ &+ \left(\eta_i^l\right)^T\left(Z_i^l - \textstyle\sum_{j=1}^{N}\hat{A}_{ij}Y_j^l\right) + \left(\mu_i^l\right)^T\left(p_i\left(x_i^l,u_i^l,(Z_i^l)^T\Gamma_s(t^l)\right)\right)\Bigg) \\ &+ \sum_{i=1}^{N}\left(\nu_i^T\left(x_i^1 - \overline{x}_i^0\right)\right), \end{aligned} \tag{2.13}$$

where λ_i^l, σ_i^l, ν_i, η_i^l and μ_i^l are the dual variables corresponding to (2.12b), (2.12c), (2.12d), (2.12e) and (2.12f), respectively. Furthermore, $\lambda = \mathrm{vec}(\lambda_1^1,\ldots,\lambda_N^M)$, $\sigma = \mathrm{vec}(\sigma_1^1,\ldots,\sigma_N^M)$, $\nu = \mathrm{vec}(\nu_1,\ldots,\nu_N)$, $\eta = \mathrm{vec}(\eta_1^1,\ldots,\eta_N^M)$ and $\mu = \mathrm{vec}(\mu_1^1,\ldots,\mu_N^M)$. We denote the Lagrangian gradient and Lagrangian Hessian with respect to the primal variables $v := \mathrm{vec}(x,u,z,y)$ with $\nabla\mathcal{L}$ and $\nabla^2\mathcal{L}$, respectively. In each iteration of the SQP method, a quadratic approximation of the original

problem is calculated that is

$$\min_{\Delta v} \frac{1}{2}\Delta v^T B \Delta v + \nabla \mathcal{L}^T \Delta v \tag{2.14a}$$

$$\text{s.t. } x_i^{l+1} + \Delta x_i^{l+1} = F_i^l + (\nabla F_i^l)^T \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \tag{2.14b}$$

$$Y_i^l + \Delta Y_i^l = G_i^l + (\nabla G_i^l)^T \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \tag{2.14c}$$

$$x_i^1 + \Delta x_i^1 = \overline{x}_i^0 \tag{2.14d}$$

$$Z_i^l + \Delta Z_i^l = \textstyle\sum_{j=1}^N \hat{A}_{ij} Y_j^l + \sum_{j=1}^N \hat{A}_{ij} \Delta Y_j^l \tag{2.14e}$$

$$p_i + \nabla p_i^T \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \geq 0 \tag{2.14f}$$

$$i = 1, \ldots, N \qquad l = 1, \ldots, M.$$

Here, Δv is the search direction that is sought for, $B \approx \nabla^2 \mathcal{L}$ is the Lagrange Hessian approximation. Note that (2.14) is a QP of which solution Δv provides a search direction in the original space of v. Moreover, this QP has a fixed sparsity structure that has to be exploited in the underlying QP solver. Once a search direction Δv is found, a stepsize parameter t is sought for by means of a line-search procedure. The next iterate is calculated as

$$v^{(k+1)} := v^{(k)} + t\Delta v \tag{2.15}$$

This procedure is repeated until a certain convergence criterion is met.

Convergence of DMS

Guaranteeing convergence of an optimization method is always of both theoretical and practical interest. As we have seen it before, in the DMS scheme, we propose the parallelization of the linearization, but the optimization remains centralized. This means that up to the approximation error of the Legendre polynomials and the integration accuracy, we solve the centralized optimal control problem. For this reason, the local convergence rate of the DMS scheme is determined by the optimization routine. Global convergence can be attained by employing standard

line-search or trust-region methods. The standard proofs, which can be found in the optimization literature, hold for DMS with both SQP and IP methods.

2.6 Adjoint-based Distributed Multiple Shooting

In the previous section, we have investigated the DMS method, which introduces time and spatial domain based decomposition in order to provide a massively parallel linearization scheme. The question arise naturally whether we can save even more time in the linearization process. In fact, it is possible to shorten the runtime of sensitivity calculation in DMS, comes at a price, however. In this section, we introduce an inexact approximation of sensitivities in order to decrease the linearization time. Our approach is inspired by [38]. Similar approaches may be found in [74, 80].

For the sake of simple presentation, we assume that Z_i^l is a vector approximating a scalar dynamic variable $z_i(t)$ of order S on $[t^l, t^{l+1}]$. The inexactness that we introduce here can be extended to the general case when Z_i^l is a matrix.

Instead of the full approximation given in (2.14), we consider the following subproblem.

$$\min_{\Delta v} \ \frac{1}{2}\Delta v^T B \Delta v + \nabla \mathcal{L}^T \Delta v \tag{2.16a}$$

$$\text{s.t. } x_i^{l+1} + \Delta x_i^{l+1} = F_i^l + \mathbf{F}_i^l \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \tag{2.16b}$$

$$Y_i^l + \Delta Y_i^l = G_i^l + \mathbf{G}_i^l \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \tag{2.16c}$$

$$x_i^1 + \Delta x_i^1 = \overline{x}_i^0 \tag{2.16d}$$

$$Z_i^l + \Delta Z_i^l = \textstyle\sum_{j=1}^N \hat{A}_{ij} Y_j^l + \sum_{j=1}^N \hat{A}_{ij} \Delta Y_j^l \tag{2.16e}$$

$$p_i + \nabla p_i^T \begin{pmatrix} \Delta x_i^l \\ \Delta u_i^l \\ \Delta Z_i^l \end{pmatrix} \geq 0 \tag{2.16f}$$

$$i = 1, \ldots, N \qquad l = 1, \ldots, M.$$

Note that in (2.16b) and (2.16c), instead of the Jacobian matrices $(\nabla F_i^l)^T$ and $(\nabla G_i^l)^T$, we have $\mathbf{F}_i^l$ and $\mathbf{G}_i^l$. We split the elements of variable $Z_i^l \in \mathbb{R}^{S+1}$ into low-order coefficients and high-order coefficients as $Z_i^l = [\underline{Z}_i^l, \overline{Z}_i^l]$ with $\underline{Z}_i^l \in \mathbb{R}^Q$ and $\overline{Z}_i^l \in \mathbb{R}^{S+1-Q}$. The original exact Jacobians have the following structure

$$(\nabla F_i^l)^T = \left[\frac{\partial F_i^l}{\partial x_i^l} \middle| \frac{\partial F_i^l}{\partial u_i^l} \middle| \frac{\partial F_i^l}{\partial \underline{Z}_i^l} \middle| \frac{\partial F_i^l}{\partial \overline{Z}_i^l}\right]. \tag{2.17}$$

The adjoint based DMS disregards the derivatives with respect to the high-order coefficients of the coupling input, namely uses the following approximations

$$(\nabla F_i^l)^T \approx \mathbf{F}_i^l = \left[\frac{\partial F_i^l}{\partial x_i^l} \middle| \frac{\partial F_i^l}{\partial u_i^l} \middle| \frac{\partial F_i^l}{\partial \underline{Z}_i^l} \middle| 0\right], \tag{2.18a}$$

$$(\nabla G_i^l)^T \approx \mathbf{G}_i^l = \left[\frac{\partial G_i^l}{\partial x_i^l} \middle| \frac{\partial G_i^l}{\partial u_i^l} \middle| \frac{\partial G_i^l}{\partial \underline{Z}_i^l} \middle| 0\right]. \tag{2.18b}$$

Note that this approximation can be generalized to each dynamic coupling input variable, i.e. when Z_i^l is a matrix of polynomial coefficients. The benefit of adjoint-based DMS is that the integrators need to calculate sensitivities in less directions, which results in faster computation. The inexactness introduced in (2.18), not considering local inequalities, is equivalent to an inexact Newton method, where the Newton matrix is inexact, but the right-hand side is exact. In our context, this means that the gradient of the Lagrangian function as the linear term in (2.16) has to be evaluated exactly. This involves the calculation of the product of the actual dual variables and the constraint Jacobians, which is essentially one adjoint derivative computation. This fact gives the name of our scheme. Thus, we might save time in the integration if the degree of the Legendre polynomials is sufficiently large and if an efficient adjoint differentiation is available in the integrator.

Linear convergence of ADMS

The adjoint-based DMS method converges locally to a solution with linear rate if the introduced inexactness is sufficiently small. We prove this statement formally by borrowing results from Kungurtsev [87].

We assume that the vectors $y_i(t)$ and $z_i(t)$ are represented as a linear combination of some functional basis, such as the normalized Legendre polynomials (see [128, 127]), the coefficients for subsystem i and shooting interval l being y_i^l and z_i^l, i.e., $y_i(t) = \sum_{j=1}^{Q} [y_i^l]_j p_j(t - t_l)$, for $t \in [t_l, t_{l+1}]$, with $p_j(t)$ being a polynomial in the functional basis of dimension Q.

Continuity across shooting intervals is enforced by the constraints and the discretized optimal control problem becomes, with $i = 1, ..., N$ and $l = 1, ..., M$,

$$\begin{aligned} \min_{\substack{x_i^l, y_i^l, \\ z_i^l, u_i^l}} \quad & \sum_{k=1}^{N} \sum_{j=1}^{M} l_k^j(x_k^j, u_k^j, z_k^j) \\ \text{s.t.} \quad & x_i^{l+1} = f_i^l(x_i^l, u_i^l, z_i^l), \\ & y_i^l = g_i^l(x_i^l, u_i^l, z_i^l), \\ & x_i^1 = \bar{x}_i^0, \\ & z_i^l = \textstyle\sum_{j=1}^{M} A_{ij} y_j^l, \\ & p_i(x_i^l, u_i^l) \geq 0. \end{aligned}$$

In order to present the problem in a way that demonstrates its basic structure, we collect the discretized variables into a set of vectors v, and w. Denoting v as the vector composed of all the z_i^l stacked together (i.e., the first component of v is $[z_1^1]_1$, the second is $[z_1^1]_2$, etc.), and w as containing all of the u_i^l, x_i^l and y_i^l, we write the discretized nonlinear program in the more compact form,

$$\begin{aligned} \min_{v,w} \quad & f(v, w) \\ \text{s.t.} \quad & 0 = c(v, w), \\ & v = Aw, \\ & p(v, w) \geq 0. \end{aligned} \tag{2.19}$$

We denote the Lagrangian function $L(v, w, y, \mu) = f(v, w) - c(v, w)^T y^1 - \begin{pmatrix} -I & A \end{pmatrix}^T y^2 - p(v, w)^T \mu$ with y^1, y^2 the multipliers corresponding to the equality constraints and μ the inequalities.

A standard SQP method would recursively solve the subproblem, denoting the subscript k to indicate evaluation at an iteration (v_k, w_k, y_k, μ_k),

$$\begin{aligned} \min_{\Delta v, \Delta w} \quad & \frac{1}{2} \begin{pmatrix} \Delta v \\ \Delta w \end{pmatrix}^T \nabla^2_{v,w} L_k \begin{pmatrix} \Delta v \\ \Delta w \end{pmatrix} + \nabla_v f_k^T \Delta v + \nabla_w f_k^T \Delta w \\ \text{s.t.} \quad & 0 = c(v_k, w_k) + \nabla_v c_k^T \Delta v + \nabla_w c_k^T \Delta w, \\ & 0 = A w_k - v_k + A \Delta w - \Delta v, \\ & 0 = p_k + \nabla_v p_k^T \Delta v + \nabla_w p_k^T \Delta w. \end{aligned} \tag{2.20}$$

For applying adjoint-based distributed multiple shooting (ADMS), we expect that some of the coefficients of the polynomials approximating the coupling inputs $z_i(t)$ contribute little to the dynamics of the system, i.e.,

$$\frac{\partial c}{\partial v_j} \approx 0,$$

for many j. In particular, we expect the dynamics of the system relative to z_i^l to be approximated well by the higher order modes of the functional decomposition.

We denote the principal and discarded components of v by $\bar{v}$ and $\widetilde{v}$, respectively, where $\bar{v} = QQ^T v$ for some orthogonal basis matrix Q, and $\widetilde{v} = v - \bar{v}$. Without losing generality, we may maintain the notation of the functions and write $c(\bar{v}, \widetilde{v}, w) = c(\bar{v} + \widetilde{v}, w)$. In the quadratic program (2.20), we discard $\nabla_{\widetilde{v}} c$, and all of the blocks in $\nabla^2 L$ involving $\widetilde{v}$ (e.g., $\nabla^2_{\widetilde{v},v} L$, $\nabla^2_{\widetilde{v},\widetilde{v}} L$, etc.). Note that the term $\nabla_{\widetilde{v}} c^T y$, where y are the appropriate Lagrange multipliers, needed to calculate the KKT conditions can still be available by calculating adjoint derivatives of $\nabla_{\widetilde{v}} c$. This is the source of the term "adjoint-based distributed multiple shooting'. It is a specific implementation of general adjoint-based SQP [19, 39].

In addition, several other matrices are approximated, rather than calculated exactly. We denote the approximate Jacobian of $c(v, w)$ by $J(v, w) \approx \nabla c(v, w)^T$ and use subscripts to denote the matrix composed of the columns corresponding to these variables, e.g., $J_v(v, w) \approx \nabla_v c(v, w)^T$. The Hessian approximation will be denoted as $\bar{H} \approx \nabla^2_{\bar{v},w} L$. In order to provide for more compact notation, we shall let $x = (v, w)$ and $\bar{x} = (\bar{v}, w)$.

In sum, we approximate the Karush-Kuhn-Tucker (KKT) matrix appearing in the subproblem, with the subscript $_A$ denoting the rows of p corresponding to the active constraints at iteration k,

$$\begin{pmatrix} \frac{\partial^2 L}{\partial \bar{v}^2} & \frac{\partial^2 L}{\partial \bar{v} \partial \widetilde{v}} & \frac{\partial^2 L}{\partial \bar{v} \partial w} & \frac{\partial c}{\partial \bar{v}} & -I & \frac{\partial p_A}{\partial \bar{v}} \\ \frac{\partial^2 L}{\partial \widetilde{v} \partial \bar{v}} & \frac{\partial^2 L}{\partial \widetilde{v}^2} & \frac{\partial^2 L}{\partial \widetilde{v} \partial w} & \frac{\partial c}{\partial \widetilde{v}} & -I & \frac{\partial p_A}{\partial \widetilde{v}} \\ \frac{\partial^2 L}{\partial \bar{v} \partial w} & \frac{\partial^2 L}{\partial w \partial \widetilde{v}} & \frac{\partial^2 L}{\partial w^2} & \frac{\partial c}{\partial w} & A^T & \frac{\partial p_A}{\partial w} \\ (\frac{\partial c}{\partial \bar{v}})^T & (\frac{\partial c}{\partial \widetilde{v}})^T & (\frac{\partial c}{\partial w})^T & 0 & 0 & 0 \\ -I & -I & A & 0 & 0 & 0 \\ (\frac{\partial p_A}{\partial \bar{v}})^T & (\frac{\partial p_A}{\partial \widetilde{v}})^T & (\frac{\partial p_A}{\partial w})^T & 0 & 0 & 0 \end{pmatrix},$$

as

$$\begin{pmatrix} \bar{H}_{\bar{v}\bar{v}} & 0 & \bar{H}_{w\bar{v}} & J_{\bar{v}}{}^T & -I & \frac{\partial p_A}{\partial \bar{v}} \\ 0 & 0 & 0 & 0 & -I & \frac{\partial p_A}{\partial \widetilde{v}} \\ \bar{H}_{w\bar{v}} & 0 & \bar{H}_{ww} & J_w^T & A^T & \frac{\partial p_A}{\partial w} \\ J_{\bar{v}} & 0 & J_w & 0 & 0 & 0 \\ -I & -I & A & 0 & 0 & 0 \\ (\frac{\partial p_A}{\partial \bar{v}})^T & (\frac{\partial p_A}{\partial \widetilde{v}})^T & (\frac{\partial p_A}{\partial w})^T & 0 & 0 & 0 \end{pmatrix}.$$

The DMS subproblem may be written as,

$$\begin{aligned} \min_{\Delta x, \Delta y, \Delta \mu} \quad & \nabla L(x, y, \mu)^T \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \mu \end{pmatrix} + \Delta \bar{x}^T \bar{H} \Delta \bar{x} \\ \text{s.t.} \quad & c(x_k) + J_{\bar{x}}(x_k) \Delta \bar{x} = 0 \\ & v_k + \Delta v = A w_k + A \Delta w, \\ & p(x_k) + \nabla_x p(x_k)^T \Delta x \geq 0. \end{aligned} \tag{2.21}$$

By using the constraint $\Delta v = A\Delta w + Aw_k - v_k$, we can replace all instances of $\Delta \widetilde{v}$ with the corresponding components of $A\Delta w + Aw_k - v_k$, and then $\widetilde{v}$ disappears from the QP entirely.

In an SQP method, the multipliers that correspond to the solution of each QP problem are used to estimate the multipliers of the nonlinear problem. In order to perform these iterations in a well-defined procedure, a QP solver requires the form of the problem to include the gradient of the objective as opposed to the Lagrangian, in order to solve for the multipliers as opposed to the change in the multipliers. Thus, in practice the subproblem would be rewritten as,

$$\begin{aligned} \min_{\Delta x} \quad & \nabla f(x)^T \Delta x + \Delta \bar{x}^T \bar{H} \Delta \bar{x} - y_1^T (\nabla_x c^T - J_w - J_{\bar{v}}) \Delta x \\ \text{s.t.} \quad & c(x_k) + J_{\bar{x}}(x_k) \Delta \bar{x} = 0 \\ & \Delta v = A \Delta w + A w_k - v_k, \\ & p(x_k) + \nabla_x p(x_k)^T \Delta x \geq 0. \end{aligned} \tag{2.22}$$

Note that because we use exact derivatives for the inequalities, we do not need to include the corresponding additional terms for them. The adjoint offset terms in the objective can be calculated by a single adjoint sweep and an inner product, and so the equality constraint gradients are only required in adjoint form, and so forming this subproblem does not require full exact derivatives $\nabla_{\widetilde{v}} c(x)$.

Consider the nonlinear program in the form of a generalized equation,

$$\Phi(z) + N(z) \ni 0, \tag{2.23}$$

where Φ is a smooth mapping and N is a multifunction (a set-valued mapping) limiting the solutions to those satisfying complementarity and dual nonnegativity. A generic SQP method solves problems of the form,

$$\Phi(z_k) + \Phi'(z_k)(z - z_k) + N(z) \ni 0. \tag{2.24}$$

We shall define two conditions in the context of ADMS (2.19). We use the subscript notation $\nabla p(v, w)_+^T$ to denote the rows of $\nabla p(v, w)^T$ corresponding to active constraints with an associated positive multiplier, i.e., rows corresponding to $\mathcal{A}_+(x^*, y^*, \mu^*) = \{i : \mu_i^* > 0\}$, and $\nabla p(v, w)_0^T$ for $\mathcal{A}_0(x^*, y^*, \mu^*) = \{i : \mu_i^* = 0\}$.

Definition 2.6.1 (SMFCQ) *The strict Mangasarian-Fromovitz constraint qualification holds at x^* if the matrix,*

$$\tilde{M} = \begin{pmatrix} \nabla_v c(x^*)^T & \nabla_w c(x^*)^T \\ -I & A \\ \nabla_v p_+(x^*)^T & \nabla_w p_+(x^*)^T. \end{pmatrix},$$

has rank $l + |\mathcal{A}_+(x^*, y^*, \mu^*)|$ and there exists a p such that,

$$\tilde{M}p = \begin{pmatrix} 0 \\ 0 \\ r \end{pmatrix},$$

where r is any vector satisfying $r > 0$.

Note that the SMFCQ is equivalent to the MFCQ and the uniqueness of the multiplier.

Definition 2.6.2 (SOSC) *The second-order sufficiency condition holds at x^* if $p^T \nabla^2_{xx} H(x^*, y^*, \mu^*)p > 0$ for all p satisfying $\tilde{M}p = 0$ and*

$$\begin{pmatrix} \nabla_v p_0(x^*)^T & \nabla_w p_0(x^*)^T \end{pmatrix} p \geq 0. \tag{2.25}$$

The sharpest result in the literature states that the generic SQP locally converges from a starting point (x_0, y_0, μ_0) sufficiently close to a a primal-dual solution (x^*, y^*, μ^*) satisfying the assumptions of the SMFCQ and the SOSC. The rate of convergence is superlinear, and quadratic if Φ is Lipschitz continuous. This result is because the SMFCQ and the SOSC conditions imply semistability and hemistability of the generalized equation. For details, see Bonnans [22] and Izmailov and Solodov [79].

An inexact SQP can be described as recursive solutions to the system,

$$\Phi(z_k) + \Phi'(z_k)(z - z_k) + \omega_k + N(z) \ni 0, \tag{2.26}$$

with some perturbation ω_k. Izmailov and Solodov [79] show that superlinear convergence of SQP is maintained under a number of perturbations. Diehl et al. [38] shows linear, and under additional assumptions, superlinear convergence under specific conditions of the inverse of the resulting subproblem adjoint KKT matrices. The convergence result that is presented in the following is a hybrid of the two, in analyzing subproblems with less tight approximations than done in Izmailov and Solodov [79], but in a more general formulation and under weaker problem assumptions than in Diehl et al. [38].

Convergence. We write the distributed multiple shooting QP as a combination of two perturbations. In particular, the subproblem wherein everything is exact except for the discarding of the $\widetilde{v}$ components,

$$\begin{aligned} \min_{\Delta x} \quad & \nabla f_k^T \Delta \bar{x} + \Delta \bar{x}^T \nabla^2_{\bar{x}\bar{x}} L_k \Delta \bar{x} - y_k^{1T} \nabla_{\widetilde{v}} c_k^T \Delta x \\ \text{s.t.} \quad & c(x_k) + \nabla_{\bar{x}} c(x_k)^T \Delta \bar{x} = 0 \\ & v_k + \Delta v = A(w_k + \Delta w), \\ & p(x_k) + \nabla_x p(x_k)^T \Delta x \geq 0, \end{aligned} \tag{2.27}$$

will be, in compact notation, described as the solution to,

$$\Phi(z_k) + \Psi(z_k)(z_{k+1} - z_k) + N(z_{k+1}) \ni 0. \tag{2.28}$$

The fully adjoint subproblem, (2.21), will be described in generalized equation form as,

$$\Phi(z_k) + \bar{\Psi}(z_k)(z_{k+1} - z_k) + N(z_{k+1}) \ni 0. \tag{2.29}$$

We state the following assumption. As discussed before, these are the weakest conditions for which a conventional SQP method has been proven to be superlinearly convergent.

Assumption 1 *There exists a primal dual pair $z^* = (x^*, y^*)$ that satisfies the strict Mangasarian-Fromovitz Constraint qualification (SMFCQ) and the second order sufficiency conditions (SOSC).*

In addition, we assume that discarding the derivatives with respect to the coupling variable v does not affect the geometric structure of the problem insofar as changing the regularity.

Assumption 2 *Assumption 1 holds, and furthermore, $M =$*

$$\begin{pmatrix} \nabla_{\bar{v}} c(x^*)^T & 0 & \nabla_w c(x^*)^T \\ -I & 0 & A_1 \\ 0 & -I & A_2 \\ \nabla_{\bar{v}} p_+(x^*)^T & \nabla_{\widetilde{v}} p_+(x^*)^T & \nabla_w p_+(x^*). \end{pmatrix},$$

where A_1 and A_2 are matrices composed of the rows corresponding to $\bar{v}$ and $\widetilde{v}$, respectively, has rank $l + |\mathcal{A}_+(x^*, y^*, \mu^*)|$ and there exists a p such that,

$$Mp = \begin{pmatrix} 0 \\ 0 \\ 0 \\ r \end{pmatrix},$$

where r is any vector satisfying $r > 0$.

In addition, $p^T \nabla^2_{\bar{v},w} H(x^*, y^*, \mu^*) p > 0$ for all p satisfying, $Mp = 0$ and

$$\begin{pmatrix} \nabla_{\bar{v}} p_0(x^*)^T & \nabla_{\widetilde{v}} p_0(x^*)^T & \nabla_w p_0(x^*)^T \end{pmatrix} p \geq 0.$$

Notice that $J_{\widetilde{v}}$ does not appear in M and the $\widetilde{v}$ components of the Hessian are not taken into account.

We will assume that the adjoint matrix approximations are sufficiently precise.

Assumption 3 *The inexact matrix approximations J_k and $\widehat{H}_k$ satisfy,*

$$\|(J_{\bar{v}}(x_k) - \nabla_{\bar{v}} c(x_k)^T) p_2\| \leq \gamma_1 \|p_2\|,$$
$$\|(J_w(x_k) - \nabla_w c(x_k)^T) p_2\| \leq \gamma_1 \|p_2\|,$$

and

$$\|(\bar{H}_k - \nabla^2_{\bar{x}\bar{x}} L(x_k)) d\| \leq \gamma_1 \|d\|,$$

for any $p_1 \in \mathbb{R}^{n_1}$, $p_2 \in \mathbb{R}^{n_2}$ and $d \in \mathbb{R}^{n_1+n_2+n_3+n_4}$, with γ_1 a small scalar constant satisfying $0 < \gamma_1 < 1$.

Note that this is equivalent to,

$$\|(\Psi_k - \bar{\Psi}_k) \Delta z\| \leq \gamma_1 \|\Delta z\|.$$

We formalize the idea that the derivatives with respect to $\widetilde{v}$ are unimportant for the dynamics of the problem in the following form,

Assumption 4

$$\|(\Psi_k - \Phi'(z_k)) \Delta z\| \leq \bar{\gamma}_2 \|\Phi'(z_k) \Delta z\|,$$

for some scalar $\bar{\gamma}_2$ satisfying $0 < \bar{\gamma}_2 < 1$. Note that by continuity of Φ, this implies that,

$$\|(\Psi_k - \Phi'(z_k)) \Delta z\| \leq \gamma_2 \|\Delta z\|.$$

We assume the following property of the adjoint Hessian approximations,

Assumption 5 *H_k is positive definite along the null-space of $\begin{pmatrix} J_{\bar{v}} & 0 & J_w \end{pmatrix}$ for all k.*

This is not a particularly strong assumption, as it is typical to provide Hessian approximations that are uniformly positive definite.

Finally, we assume that all full KKT matrices are invertible with a uniformly bounded inverse.

Assumption 6 *The matrix* $M_k =$

$$\begin{pmatrix} J_{\bar{v}}(x_k) & 0 & J_w(x_k) \\ -I & 0 & A_1 \\ 0 & -I & A_2 \\ \nabla_{x_k} p_{\bar{\mathcal{A}}(k)}(x_k)^T & \nabla_{\tilde{v}} p_{\bar{\mathcal{A}}(k)}(x_k)^T & \nabla_w p_{\bar{\mathcal{A}}(k)}(x_k)^T. \end{pmatrix},$$

is invertible for all k *with a uniformly bounded inverse, where* $\bar{\mathcal{A}}(k)$ *is the active set at* k.

We begin the proof by citing a result from the literature that will be useful for the convergence theory. This gives an upper and lower bound for the distance to the nearest KKT point relative to the residual for the optimality conditions.

Define the optimality residual $\eta(x, y)$ to be,

$$\eta(x, y, \mu) \equiv \left\| \begin{pmatrix} \nabla_x L(x, y, \mu) \\ c(x) \\ v - Aw \\ p(x)^- \\ p(x)^T \mu \end{pmatrix} \right\|$$

Lemma 2.6.1 *If the second-order sufficiency condition for optimality holds at* $z^* = (x^*, y^*, \mu^*)$, *then for* $z = (x, y, \mu)$ *sufficiently close to* z^*, *it holds that,*

$$C_1 \|z - z^*\| \leq \eta(z) \leq C_2 \|z - z^*)\|.$$

Proof *See, e.g., Wright [145, Theorem 3.1].*

■

Lemma 2.6.2 *There exists some* $\mathcal{B}_1$ *such that, if* $z_k \in \mathcal{B}_1$, *the solution* $\widehat{z}_{k+1}$ *to subproblem* (2.27) *satisfies* $\widehat{z}_{k+1} \to 0$ *as* $z_k \to z^*$ *and, in particular,*

$$\|\widehat{z}_{k+1} - z_k\| \leq O(\|z_k - z^*\|). \tag{2.30}$$

Proof *Consider the subproblem* (2.27) *at the base point* $z_k = (x^*, y^*)$. *Clearly,* $(\Delta x, \Delta y, \Delta \mu) = 0$ *is a solution to this subproblem. Furthermore, Assumption 2 states that the SMFCQ and the SOSC holds for* (2.27) *at* $z_k = (x^*, y^*)$. *Now consider that a different base point* z_k *is a perturbation of* (2.27). *For* (x_k, y_k) *sufficiently close to* (x^*, y^*), *by Robinson [124, Theorem 3.1], there exists a solution to* (2.27) *and by Robinson [124, Corollary 1], and Arutyunov and Izmailov [5, Theorem 3.1], it holds that* $\|z_{k+1} - z_k\| \to 0$ *as* $z_k \to z^*$ *as well as the upper Lipschitz estimate* (2.30).

■

Lemma 2.6.3 *There exists some $\mathcal{B}_2 \subseteq \mathcal{B}_1$ such that, if $z_k \in \mathcal{B}_2$, the solution z_{k+1} to subproblem (2.21) satisfies,*

$$\|z_{k+1} - z_k\| \leq O(\|z_k - z^*\|). \tag{2.31}$$

Proof *Consider the solution to the system,*

$$M_k \delta z = \begin{pmatrix} \widehat{H}_k - \nabla^2_{\bar{x}\bar{x}} L(z_k) \\ J_{\bar{x}}(z_k) - \nabla_{\bar{x}} c(z_k) \\ 0 \\ 0 \end{pmatrix} \widehat{\Delta z},$$

where $\widehat{\Delta z}$ is a solution to (2.27) at z_k and M_k is defined in Assumption 6. It is clear that if the multipliers corresponding to the inequality constraints remain nonnegative, i.e., $\mu + \widehat{\Delta\mu} + \delta\mu \geq 0$, then $\widehat{\Delta z} + \delta z$ solves problem (2.21). In this case, it holds that, by Lemma 2.6.2,

$$\|\widehat{\Delta z} + \delta z\| \leq \|\widehat{\Delta z}\| + \gamma_1 \|M_k^{-1}\| \|\widehat{\Delta z}\| \leq O(\|z_k - z^*\|.$$

Otherwise, if $\mu + \widehat{\Delta\mu} + \delta\mu$ has a negative component, then consider the following procedure. By Assumption 5, H_k is positive definite along the null-space of $\begin{pmatrix} J_{\bar{v}} & 0 & J_w \end{pmatrix}$, e.g., along values for the variables Δw that satisfy the equality constraints. Therefore, we may write the subproblem (2.21) as,

$$\begin{aligned} \min_{\Delta x \in \mathcal{N}} \quad & \nabla f(x_k)^T \Delta x - y^{1T}(\nabla c(x_k) - J_w^T - J_{\bar{v}}{}^T)\Delta x \\ & + \Delta\bar{x}^T \nabla^2_{\bar{x}\bar{x}} L(x_k, y_k, \mu_k) \Delta\bar{x} \\ \text{s.t.} \quad & v_k + \Delta v = A(w_k + \Delta w), \\ & p(x_k) + \nabla_x p(x_k)^T \Delta x \geq 0, \end{aligned} \tag{2.32}$$

where $\mathcal{N}$ is the null-space of the equality constraint matrix. As this subproblem now has a positive-definite Hessian on the variable subspace, i.e., we can perform a change of variables to solve for Δx in $\mathcal{N}$, we may take the dual of (2.32) to obtain a subproblem with no duality gap. This QP has a unique solution since the SOSC trivially holds since the Hessian is positive definite. Now, consider the dual constraint $\mu + \Delta\mu \geq 0$ to be a perturbation of a subproblem with the dual constraint $\mu + \Delta\mu \geq \mu + \widehat{\Delta\mu} + \delta\mu$. Since the dual constraints are no longer blocking, this QP has the solution $\widehat{\Delta z} + \delta z$. Thus, for the original z_k sufficiently close to z^*, since upper-Lipschitz continuity holds for perturbations of problems satisfying the SOSC (see, i.e., Hager and Gowda [70, Lemma 2]), we have that the solution

to (2.32) *satisfies,*

$$\begin{aligned}\|\Delta z\| &\leq \|\Delta z - \widehat{\Delta z} - \delta z\| + \|\widehat{\Delta z} + \delta z\| \\ &\leq O(\|\min(0, \mu + \widehat{\Delta\mu} + \delta\mu)\|) + O(\|z_k - z^*\|) \\ &\leq O(\|z_k - z^*\|).\end{aligned}$$

■

Theorem 2.6.1 *For sufficiently small γ_1 and γ_2, there exists some $\mathcal{B}_3 \subseteq \mathcal{B}_2$ such that, if $z_k \in \mathcal{B}_3$, the sequence of iterates generated by recursive iterations of obtaining the solution of problem* (2.21) *converges to z^* and the rate of convergence is linear.*

Proof *From Taylor's theorem, the optimality conditions of* (2.21)*, Assumptions 3 and 4, and Lemma 2.6.3, we know that*

$$\begin{aligned}\|c(x_{k+1})\| &\leq \|c(x_k) + \nabla c(x_k)^T(x_{k+1} - x_k)\| + o(\|x_{k+1} - x_k\|) \\ &\leq \|c(x_k) + J_{u,\bar{v}}(\bar{x}_{k+1} - \bar{x}_k)\| \\ &\quad + \|(\nabla c(x_k)^T - J_{u,\bar{v}})(x_{k+1} - x_k)\| + o(\|x_{k+1} - x_k\|) \\ &\leq (\gamma_1 + \gamma_2)O(\|z_k - z^*\|).\end{aligned}$$

Similarly,

$$\begin{aligned}\|\nabla_x L(x_{k+1}, y_{k+1})\| &\leq ||\nabla_x L(x_k, y_k) + \nabla^2{}_{xx} L(x_k, y_k))(x_{k+1} - x_k) \\ &\quad + \nabla_x c(x_k)(y_{k+1} - y_k)|| + o(\|x_{k+1} - x_k\|) \\ &\leq \|\nabla_x L(x_k, y_k) + \bar{H}(x_{k+1} - x_k) + J^T(y_{k+1} - y_k)\| \\ &\quad + \|(\bar{H}_k - \nabla^2{}_{xx} L(x_k, y_k))(x_{k+1} - x_k)\| \\ &\quad + \|(\nabla c - J^T)(y_{k+1} - y_k)\| + o(\|x_{k+1} - x_k\|) \\ &\leq \|(\bar{H}_k - \nabla^2{}_{xx} L(x_k, y_k))(x_{k+1} - x_k)\| \\ &\quad + \|(\nabla c - J^T)(y_{k+1} - y_k)\| + o(\|x_{k+1} - x_k\|) \\ &\leq (\gamma_1 + \gamma_2)O(\|z_{k+1} - z_k\|) \leq \gamma_2 O(\|z_k - z^*\|).\end{aligned}$$

Given that Assumption 6 implies that $\{\mu_k\}$ *is bounded, we also have that,*

$$\begin{aligned}\|p(x_{k+1})^T\mu_{k+1}\| &\leq \|(p(x_k) + \nabla_x p(x_k)(x_{k+1} - x_k))^T\mu_{k+1}\| \\ &\quad + \|\mu_{k+1}\|o(\|x_{k+1} - x_k\|) \\ &\leq o(\|x_{k+1} - x_k\|).\end{aligned}$$

All this implies that, by Lemma 2.6.1,

$$\|z_{k+1} - z^*\| \leq C_4\eta(z_{k+1}) \leq (\gamma_1 + \gamma_2)C_5\|z_k - z^*\|.$$

Thus, for small enough γ_1 *and* γ_2*, the QP subproblem* (2.27) *generates a sequence of iterates successively contracting to* z^**, i.e.,*

$$\|z_{k+1} - z^*\| \leq \kappa\|z_k - z^*\|, \tag{2.33}$$

with κ *satisfying* $0 < \kappa < 1$*. Convergence follows from, e.g., Brouwer's fixed point theorem. A linear rate of convergence follows from* (2.33)*.*

■

Chapter 3

Optimal control of a Hydro Power Valley

In this chapter, we present a case study of the DMS scheme on a benchmark example, the Hydro Power Valley (HPV). It is inspired by an existing hydro power plant, which consists of several interconnected subsystems, owned and operated by Électricité de France. After describing the prediction model, we present numerical experiments, which demonstrate the practical features of DMS. The discussion here follows the results published in [128, 127, 96], and the model description was borrowed from [126], which was developed in the framework of the Hierarchical and Distributed Model Predictive Control (HD-MPC) FP7 project.

3.1 Model description and problem statement

In this section, we discuss the structure and model of the HPV. The HPV is formed by three lakes L_1, L_2, L_3, six river reaches $R_1, \ldots, R_6$ that are equipped with dams $D_1, \ldots, D_6$ generating electricity. Lake 1 is connected to Lake 2 with a single duct U_1 and to Reach 1 with a duct equipped with a turbine and a pump C_1. Lake 1 is also connected to Reach 2 with a duct equipped with a turbine T_1. Lake 2 is connected to Reach 4 with a duct equipped with a pump and a turbine C_2 and to Reach 5 with a duct equipped with a turbine T_2. In the beginning of Reach 1, there is a constant water inflow q_{in} and all lakes collect rain water. See Figure 3.1 for the topology and interconnection.

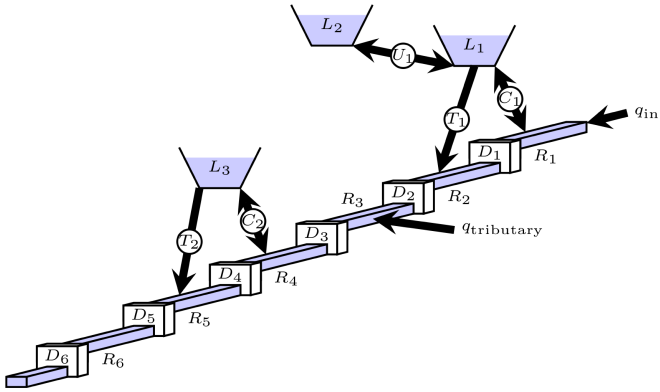


Figure 3.1: The topology of the Hydro Power Valley.

In the following, we shall provide a model for all the subsystems developed by Dr Carlo Savorgnan in [126].

To simplify the system modeling we make the following assumptions:

- the ducts are connected at the bottom of the lakes (or to the bottom of the river bed);
- the cross section of the reaches and of the lakes is rectangular;
- the width of the reaches varies linearly along the reaches;
- the river bed slope is constant along every reach.

Reach model

The model of the reaches is based on the one-dimensional Saint Venant partial differential equation:

$$\begin{cases} \dfrac{\partial q(t,z)}{\partial z} + \dfrac{\partial s(t,z)}{\partial t} = 0 \\ \dfrac{1}{g}\dfrac{\partial}{\partial t}\left(\dfrac{q(t,z)}{s(t,z)}\right) + \dfrac{1}{2g}\dfrac{\partial}{\partial z}\left(\dfrac{q^2(t,z)}{s^2(t,z)}\right) + \dfrac{\partial h(t,z)}{\partial z} + I_{\text{f}}(t,z) - I_0(z) = 0 \end{cases} \tag{3.1}$$

The two equations in (3.1) express the mass and momentum balance. The variables represent the following quantities:

- z is the spatial variable which increases along the flow main direction;
- $q(t, z)$ is the river flow (or discharge) at time t and space coordinate z;
- $s(t, z)$ is the wetted surface;
- $h(t, z)$ is the water level w.r.t. the river bed;
- g is the gravitational acceleration;
- $I_f(t, z)$ is the friction slope;
- $I_0(z)$ is the river bed slope.

Assuming that the cross section of the river is rectangular we can write the following equations.

$$s(t, z) = w(z)h(t, z) \tag{3.2a}$$

$$I_f(t, z) = \frac{q(t, z)^2 \left(w(z) + 2h(t, z)\right)^{4/3}}{k_{\text{str}}^2 \left(w(z)h(t, z)\right)^{10/3}} \tag{3.2b}$$

where $w(z)$ is the river width and k_{str} is the Gauckler-Manning-Strickler coefficient

To take into account lateral inflows, the first equation in (3.1) which expresses the mass balance can be modified as follows

$$\frac{\partial q(t, z)}{\partial z} + \frac{\partial s(t, z)}{\partial t} = q_{\text{l}}(z), \tag{3.3}$$

where $q_{\text{l}}(z)$ is the lateral inflow per space unit.

Discretization of the reach model

The partial differential equation (3.1) can be converted into an ordinary differential equation with the method of lines. Divide the reach into N cells of length dz and denote by $q_i(t)$ the value of the discharge in the middle of the cell i and by $h_i(t)$ the value of the water level at the beginning of cell i, see Figure 3.2. Furthermore, h_{N+1} represents the water level at the end of the reach.

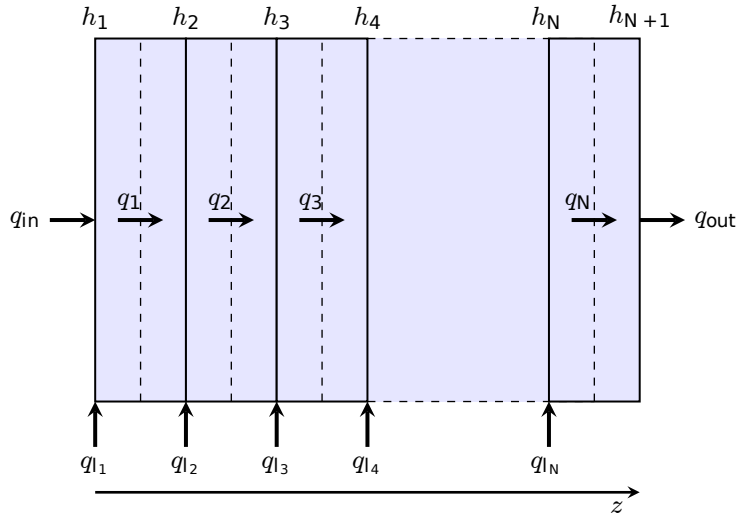


Figure 3.2: The spatial discretization scheme of the Saint Venant equations.

Denoting by $q_{\text{in}}(t)$ and $q_{\text{out}}(t)$ the water inflow at the beginning of the reach and the water outflow at the end of the reach, we obtain the following set of ordinary differential equations (time dependencies are omitted)

$$
\begin{cases}
\dfrac{\partial h_1}{\partial t} &= -\dfrac{1}{w_1}\dfrac{q_1 - q_{\text{in}} - q_{l_1}}{dz/2} \\
\dfrac{\partial q_1}{\partial t} &= \dfrac{q_1}{w_1 h_1}\dfrac{q_{l_1}}{dz/2} - \dfrac{2q_1}{w_1 h_1}\dfrac{q_1 - q_{\text{in}}}{dz/2} + \left[\dfrac{1}{w_1}\left(\dfrac{q_1}{h_1}\right)^2 - g w_1 h_1\right]\dfrac{h_2 - h_1}{dz} + \\
& \quad + g w_1 h_1 I_0 - g w_1 h_1 \left[\dfrac{q_1^2 \left(w_1 + 2h_1\right)^{4/3}}{k_{\text{str}}^2 \left(w_1 h_1\right)^{10/3}}\right]
\end{cases}
$$

$$
\begin{cases}
\dfrac{\partial h_i}{\partial t} &= -\dfrac{1}{w_i}\dfrac{q_i - q_{i-1} - q_{l_i}}{dz} \\
\dfrac{\partial q_i}{\partial t} &= \dfrac{q_i}{w_i h_i}\dfrac{q_{l_i}}{dz} - \dfrac{2q_i}{w_i h_i}\dfrac{q_i - q_{i-1}}{dz} + \left[\dfrac{1}{w_i}\left(\dfrac{q_i}{h_i}\right)^2 - g w_i h_i\right]\dfrac{h_{i+1} - h_i}{dz} + \\
& \quad + g w_i h_i I_0 - g w_i h_i \left[\dfrac{q_i^2 \left(w_i + 2h_i\right)^{4/3}}{k_{\text{str}}^2 \left(w_i h_i\right)^{10/3}}\right]
\end{cases}
$$

$$
\frac{\partial h_{N+1}}{\partial t} = -\frac{1}{w_{N+1}}\frac{q_{\text{out}} - q_N}{dz/2} \tag{3.4}
$$

where $i = 2, \ldots, N$, w_i represents the river width at the beginning of cell i, w_{N+1}

represents the river width at the end of the reach and q_{l_i} is the total lateral inflow of cell i. The river bed slope I_0 is assumed to be constant. Since the width of the reaches changes linearly, the values of w_1 and w_{N+1} are provided in the model data while

$$w_i = w_1 + \frac{(i-1)(w_{N+1} - w_1)}{N}. \tag{3.5}$$

Lake model

Denote by $q_{\text{in}}(t)$ and $q_{\text{out}}(t)$ the water inflow and outflow of the lake under consideration, respectively. The volume of water inside the lake varies accordingly to the following equation

$$\frac{\partial v(t)}{\partial t} = q_{\text{in}}(t) - q_{\text{out}}(t). \tag{3.6}$$

Since the cross section of the lake is assumed to be rectangular, (3.6) can be equivalently expressed as

$$\frac{\partial h(t)}{\partial t} = \frac{q_{\text{in}}(t) - q_{\text{out}}(t)}{S}, \tag{3.7}$$

where $h(t)$ is the water level and S is the lake surface area.

Duct model

The flow inside the duct U_1 can be modeled using Bernoulli's law. Assuming the duct section is much smaller than the lake surface, the flow from lake L_1 to lake L_2 can be expressed as

$$q_{U_1}(t) = S_{U_1} \operatorname{sign}(h_{L_2}(t) - h_{L_1}(t) + h_{U_1}) \sqrt{2g|h_{L_2}(t) - h_{L_1}(t) + h_{U_1}|}, \tag{3.8}$$

where h_{L_1} and h_{L_2} are the water levels for lakes L_1 and L_2, h_{U_1} is the height difference of the duct, S_{U_1} is the section of the duct and g is the gravitational acceleration.

Denoting $x = h_{L_2}(t) - h_{L_1}(t) + h_{U_1}$, equation (3.8) can be written as $S_{U_1}\sqrt{2g}\operatorname{sign}(x)\sqrt{|x|}$. The function $\operatorname{sign}(x)\sqrt{|x|}$ is not differentiable for $x = 0$. The following approximation can be used to make the function $q_{U_1}(t)$ differentiable

$$\operatorname{sign}(x)\sqrt{|x|} \approx \frac{x}{(x^2 + \epsilon^4)^{1/4}}.$$

Notice that for $\epsilon = 0$ the two functions are equivalent, while keeping ϵ small we obtain a good approximation ($\frac{1}{\epsilon}$ corresponds to the derivative of the approximation at $x = 0$).

Turbine model

For every turbine we assume that we can control directly the turbine discharge. The power produced is given by the following equation

$$p_{\mathrm{t}}(t) = k_{\mathrm{t}} q_{\mathrm{t}}(t) \Delta h_{\mathrm{t}}(t), \tag{3.9}$$

where k_{t} is the turbine coefficient, $q_{\mathrm{t}}(t)$ is the turbine discharge and $\Delta h_{\mathrm{t}}(t)$ is the turbine head.

Pump model

Pumps can be modeled similarly to turbines. The power absorbed by a pump is given by

$$p_{\mathrm{p}}(t) = k_{\mathrm{p}} q_{\mathrm{p}}(t) \Delta h_{\mathrm{p}}(t), \tag{3.10}$$

where k_{p} is the pump coefficient, $q_{\mathrm{p}}(t)$ is the pump discharge and $\Delta h_{\mathrm{p}}(t)$ is the pump head.

Modelling of ducts equipped with a turbine and a pump

The ducts C_1 and C_2 are equipped with a pump and a turbine and therefore we can use equations (3.9) and (3.10) to express the amount of power generated or absorbed. However, the turbines and the pumps cannot function together and this should be expressed in the optimal control problems formulated using the hydro power plant. Depending on the OCP formulation and the method used to solve the problem different models can be used. We use the so-called double flow model that we detail in the sequel.

Denote by $q_{C_1}(t)$ the flow through duct C_1. In a discontinuous model, one would have

- $q_{C_1}(t) \geq 0$ when C_1 functions as a turbine;
- $q_{C_1}(t) < 0$ when C_1 functions as a pump.

Notice that by using this convention we do not need to express explicitly that C_1 can function as a turbine or a pump alternatively. The power produced can be expressed as

$$p_{C_1}(t) = k_{C_1}(q_{C_1}(t)) q_{C_1}(t) \Delta h_{C_1}(t), \tag{3.11}$$

where $\Delta h_{C_1}(t)$ it the duct head which depends on the water level in lake L_1 and reach R_1 and

$$k_{C_1}(q_{C_1}(t)) = \begin{cases} k_{\mathrm{t}_{C_1}} & \text{when } q_{C_1}(t) \geq 0 \\ k_{\mathrm{p}_{C_1}} & \text{when } q_{C_1}(t) < 0 \end{cases}, \tag{3.12}$$

($k_{\mathrm{t}_{C_1}}$ is the turbine coefficient and $k_{\mathrm{p}_{C_1}}$ is the pump coefficient). The flow in C_1 is limited:

$$q_{C_1}(t) \in \left[-q_{C_{1\,p,\max}}, -q_{C_{1\,p,\min}}\right] \cup \left[q_{C_{1\,t,\min}}, q_{C_{1\,t,\max}}\right], \tag{3.13}$$

where the values $q_{C_{1\,p,\max}}$, $q_{C_{1\,p,\min}}$, $q_{C_{1\,t,\min}}$ and $q_{C_{1\,t,\max}}$ are positive (the subscript t stands for turbine, while p stands for pump).

Equation (3.12) and the constraint (3.13) make the model of the C_1 discontinuous and therefore not suitable for many control techniques.

The double flow model can be obtained by introducing two separate variables to express the flow in C_1

- $q_{C_{1\,p}}(t)$: flow when C_1 is functioning as a pump;
- $q_{C_{1\,t}}(t)$: flow when C_1 is functioning as a turbine.

Assuming these new variables are both positive we can write

$$q_{C_1}(t) = q_{C_{1\,t}}(t) - q_{C_{1\,p}}(t) \tag{3.14}$$

and

$$p_{C_1}(t) = (k_{\mathrm{t}_{C_1}} q_{C_{1\,t}}(t) - k_{\mathrm{p}_{C_1}} q_{C_{1\,p}}(t))\Delta h_{C_1}(t). \tag{3.15}$$

The constraint (3.13) can be rewritten in terms of $q_{C_{1\,p}}(t)$ and $q_{C_{1\,t}}(t)$

$$q_{C_{1\,p}}(t) \in \left[q_{C_{1\,p,\min}}, q_{C_{1\,p,\max}}\right] \tag{3.16}$$

$$q_{C_{1\,t}}(t) \in \left[q_{C_{1\,t,\min}}, q_{C_{1\,t,\max}}\right]. \tag{3.17}$$

Subsystem partition

The system is partitioned into 8 subsystems.

Subsystem 1 ($L_1 + L_2 + U_1 + T_1 + C_1$)

Subsystem 1 is composed of lakes L_1 and L_2 and ducts U_1, T_1 and C_1. Duct C_1 can function as a pump or a turbine.

Define the following quantities:

- $h_{L_1}(t)$ is the water level in lake L_1;
- $h_{L_2}(t)$ is the water level in lake L_2;
- $q_{L_1}(t)$ is the water inflow for L_1 which takes into account rain, small tributaries, etc.;
- $q_{L_2}(t)$ is the water inflow for L_2 which takes into account rain, small tributaries, etc.;
- $q_{T_1}(t)$ is the water discharge going to turbine T_1 (control variable);
- $q_{C_1}(t)$ is the water discharge going through the duct C_1 (control variable);
- h_{T_1} is the height difference of duct T_1;
- h_{C_1} is the height difference of duct C_1;
- $h_{R_2,T_1}(t)$ is the water level in R_2 at the outflow point of duct T_1;
- $h_{R_1,C_1}(t)$ is the water level in R_1 at the outflow point of duct C_1;
- $k_{\mathrm{t}_{T_1}}$ is the turbine coefficient of T_1;
- $k_{\mathrm{t}_{C_1}}$ is the turbine coefficient of C_1;
- $k_{\mathrm{p}_{C_1}}$ is the pump coefficient of C_1
- $p_{S_1}(t)$ is the power produced by subsystem 1.

The equations governing the subsystem behavior can be derived using the equations illustrated in the previous section and setting

- lake L_1

$$q_{\mathrm{in}}(t) = q_{L_1}(t) + q_{U_1}(t)$$
$$q_{\mathrm{out}}(t) = q_{T_1}(t) + q_{C_1}(t)$$

- lake L_2

$$q_{\mathrm{in}}(t) = q_{L_2}(t)$$
$$q_{\mathrm{out}}(t) = q_{U_1}(t)$$

- turbine T_1
$$\Delta h_{\mathrm{t}}(t) = h_{T_1} + h_{L_1}(t) - h_{R_2,T_1}(t)$$
- combined turbine/pump C_1
$$\Delta h_{C_1}(t) = h_{C_1} + h_{L_1}(t) - h_{R_1,C_1}(t).$$

The variables of subsystem 1 are subject to the following constraints

$$\begin{array}{l} h_{L_{1\min}} \leq h_{L_1}(t) \leq h_{L_{1\max}} \\ h_{L_{2\min}} \leq h_{L_2}(t) \leq h_{L_{2\max}} \\ q_{T_{1\min}} \leq q_{T_1}(t) \leq q_{T_{1\max}} \\ q_{C_1}(t) \in \left[-q_{C_{1\mathrm{t,max}}}, -q_{C_{1\mathrm{t,min}}}\right] \cup \left[q_{C_{1\mathrm{p,min}}}, q_{C_{1\mathrm{p,max}}}\right] \end{array}$$

Subsystem 2 (L_3 + T_2 + C_2)

Subsystem 2 is composed of lake L_3 and ducts T_2 and C_2.

Define the following quantities:

- $h_{L_3}(t)$ is the water level in lake L_3;
- $q_{L_3}(t)$ is the water inflow for L_3 which takes into account rain, small tributaries, etc.;
- $q_{T_2}(t)$ is the water discharge going to turbine T_2 (control variable);
- $q_{C_2}(t)$ is the water discharge going through the duct C_2. $q_{C_2}(t)$ is positive when C_2 functions as a pump (control variable);
- h_{T_2} is the height difference of duct T_2;
- h_{C_2} is the height difference of duct C_2;
- $h_{R_5,T_2}(t)$ is the water level in R_5 at the outflow point of duct T_2;
- $h_{R_4,C_2}(t)$ is the water level in R_4 at the outflow point of duct C_2;
- $k_{\mathrm{t}_{T_2}}$ is the turbine coefficient of T_2;
- $k_{\mathrm{t}_{C_2}}$ is the turbine coefficient of C_2;
- $k_{\mathrm{p}_{C_2}}$ is the pump coefficient of C_2;
- $p_{S_2}(t)$ is the power produced by subsystem 2.

The equations governing the subsystem behavior can be derived using equations (3.7)–(3.10) and setting

- lake L_3

$$q_{\text{in}}(t) = q_{L_3}(t)$$
$$q_{\text{out}}(t) = q_{T_2}(t) + q_{C_2}(t)$$

- turbine T_2

$$\Delta h_{\text{t}}(t) = h_{T_2} + h_{L_3}(t) - h_{R_5,T_2}(t)$$

- combined turbine/pump C_2

$$\Delta h_{C_2}(t) = h_{C_2} + h_{L_3}(t) - h_{R_4,C_2}(t).$$

The variables of subsystem 2 are subject to the following constraints

$$h_{L_{3\min}} \leq h_{L_3}(t) \leq h_{L_{3\max}}$$
$$q_{T_{2\min}} \leq q_{T_2}(t) \leq q_{T_{2\max}}$$
$$q_{C_2}(t) \in \left[-q_{C_{2\text{t},\max}}, -q_{C_{2\text{t},\min}}\right] \cup \left[q_{C_{2\text{p},\min}}, q_{C_{2\text{p},\max}},\right]$$

Subsystem 3 (R_1 + D_1), 4 (R_2 + D_2), 5 (R_3 + D_3), 6 (R_4 + D_4), 7 (R_5 + D_5), 8 (R_6 + D_6)

Subsystems 3, 4, 5, 6, 7, and 8 are composed by a reach and dam. Figure 3.3 represents the structure of the dams. All the flow going through the dams is used by the turbine to produce electricity. The head of the turbines inside the dams can be expressed as the difference of the water level before and after the dam. Since the water level after dam D_6 is not part of the model we consider it constant ($h_{D_{6\text{out}}}$). The constraints on the subsystem variables are

- subsystem 3

$$h_{R_{1\min}} \leq h_{R_1}(t) \leq h_{R_{1\max}}$$
$$q_{D_{1\min}} \leq q_{D_1}(t) \leq q_{D_{1\max}}$$

  where $h_{R_1}(t)$ is the water level at the end of reach R_1 and $q_{D_1}(t)$ is the dam discharge which goes to the turbine (the control variable);

- subsystem 4

$$h_{R_{2\min}} \leq h_{R_2}(t) \leq h_{R_{2\max}}$$
$$q_{D_{2\min}} \leq q_{D_2}(t) \leq q_{D_{2\max}}$$

  where $h_{R_2}(t)$ is the water level at the end of reach R_2 and $q_{D_2}(t)$ is the dam discharge which goes to the turbine (the control variable);

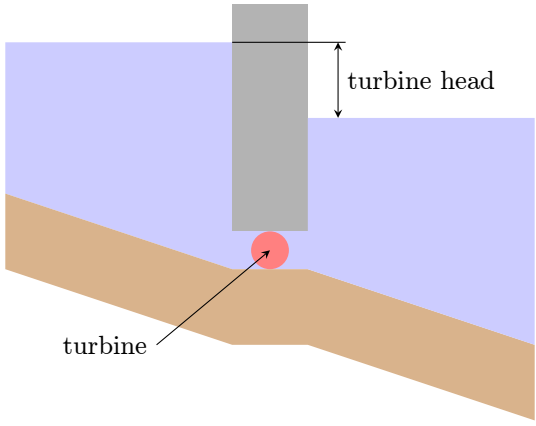


Figure 3.3: Dam configuration.

- subsystem 5

$$h_{R_{3\min}} \leq h_{R_3}(t) \leq h_{R_{3\max}}$$
$$q_{D_{3\min}} \leq q_{D_3}(t) \leq q_{D_{3\max}}$$

where $h_{R_3}(t)$ is the water level at the end of reach R_1 and $q_{D_3}(t)$ is the dam discharge which goes to the turbine (the control variable);

- subsystem 6

$$h_{R_{4\min}} \leq h_{R_4}(t) \leq h_{R_{4\max}}$$
$$q_{D_{4\min}} \leq q_{D_4}(t) \leq q_{D_{4\max}}$$

where $h_{R_4}(t)$ is the water level at the end of reach R_4 and $q_{D_4}(t)$ is the dam discharge which goes to the turbine (the control variable);

- subsystem 7

$$h_{R_{5\min}} \leq h_{R_5}(t) \leq h_{R_{5\max}}$$
$$q_{D_{5\min}} \leq q_{D_5}(t) \leq q_{D_{5\max}}$$

where $h_{R_5}(t)$ is the water level at the end of reach R_5 and $q_{D_5}(t)$ is the dam discharge which goes to the turbine (the control variable);

- subsystem 8

$$h_{R_{6\min}} \leq h_{R_6}(t) \leq h_{R_{6\max}}$$
$$q_{D_{6\min}} \leq q_{D_6}(t) \leq q_{D_{6\max}}$$

where $h_{R_6}(t)$ is the water level at the end of reach R_6 and $q_{D_6}(t)$ is the dam discharge which goes to the turbine (the control variable).

Optimal control of HPV

In the optimal control of the HPV, our goal is primarily to track a predefined power reference, and secondarily to track the steady state of the system, while respecting operational constraints such as water levels, bounds on inputs, we consider a prediction horizon of [0, 24 h]. The control intervals coincide with the shooting intervals of 30 mins. For each coupling variable we use a polynomial of degree 9. After applying the discretization procedure of DMS, we obtain the following NLP.

$$\min_{\substack{x,u,\\ z,y}} \sum_{i=1}^{8}\sum_{j=1}^{48} \frac{1}{2}\left\|x_i^j - \tilde{x}_i^j\right\|_2^2 + \gamma\left\|KY_i^j - p_i^j\right\|_1 \tag{3.18a}$$

$$\text{s.t. } x_i^{l+1} = F(x_i^l, u_i^l, Z_i^l) \tag{3.18b}$$

$$Y_i^l = G(x_i^l, u_i^l, Z_i^l) \tag{3.18c}$$

$$x_i^1 = \overline{x}_i^0 \tag{3.18d}$$

$$Z_i^l = \sum_{j=1}^{8} A_{ij} Y_j^l \tag{3.18e}$$

$$\underline{x}_i^l \le x_i^l \le \overline{x}_i^l, \quad \underline{u}_i^l \le u_i^l \le \overline{u}_i^l \tag{3.18f}$$

$$i = 1, \ldots, 8, \quad l = 1, \ldots, 48 \tag{3.18g}$$

Here, the objective function (3.18a) consists of two terms. The L_2-term penalizes the deviations from the steady state $\tilde{x}_i^j$, while the L_1-term forces the power production to be close to the predefined reference p_i^j. The continuity of the states along the time domain is ensured by (3.18b), while the spatial coupling between the subsystems are imposed by (3.18e). The number of variables within each subsystem is presented in Table 3.1.1. The non-smoothness of the objective is handled by using slack variables resulting in a smooth formulation.

The exact model parameters used for the implementation are reported in Appendix A.

3.2 DMS on HPV

In this section, we demonstrate that the method of distributed multiple shooting leads to a remarkable speed-up in the integration and sensitivity generation and thus inside an optimization loop when applied to the Hydro Power Valley.

Table 3.1.1: The number of state variables x, control variables u, coupling input variables z, output variables y of each each subsystem in HPV.

Subsystem no.	x	u	z	y
1	2	3	12	3
2	1	3	11	3
3	41	1	7	11
4	41	1	11	16
5	41	1	11	11
6	41	1	11	16
7	41	1	11	16
8	41	1	6	6
SUM	249	12	80	82

The runtime of DMS within an SQP method is dominated by two operations, first, the integration and the sensitivity generation of the system, second, the centralized solution of the QP subproblems. With a single shooting approach, which does not use parallelization, the integration of the whole system along with sensitivities takes 452.24s. If we use direct multiple shooting, the integration together with sensitivity generation of the whole system can be paralellized into 48 processes, one shooting interval each. This takes 37.71s of runtime, which is 12 times faster than the single shooting approach. If, in addition to the time domain based decomposition, we apply the spatial decomposition, the simulation time decreases to 1.92s using 384 parallel processes. This is a remarkable speed-up of a factor 235 compared to the single shooting scheme, and a factor of 19 compared to the multiple shooting approach.

We should also stress that once we use time-parallel DMS, i.e. the spatial decomposition is present, but runs serially, the runtime is $1.92\text{s} \cdot 8 = 15.36\text{s}$, which is more than 2 times faster than the multiple shooting approach.

The centralized, but thread-based QP solution with the multiple shooting approach takes 8.67 seconds, while the QP suproblem of DMS is solved in 8.01 seconds. The reason for the difference is that the DMS approach results in a QP that is sparser allowing faster sparse linear algebraic operations inside the QP solver.

The optimal solution of (3.18) was found in 35 SQP iterations. The power production of the prediction model versus the power reference is depicted in Figure 3.4. The power requirements can be perfectly fulfilled.

Some of the optimal water level evolution of each subsystem is shown in Figure 3.5.

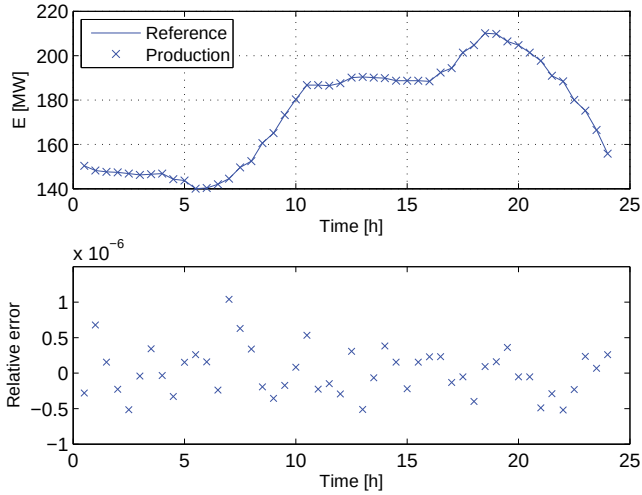


Figure 3.4: Comparison of the produced and the reference power in the solution of the optimal control problem.

Table 3.3.1: List of distributed control methods investigated in [96].

Scheme	Name	Citation
DMS	Distributed Multiple Shooting	[128]
DMPC-BAN	DMPC Based on Agent Negotiation	[95]
DAPG	Distributed Accelerated Proximal Gradient	[40]
S-DMPC	Sensitivity-driven DMPC	[130]
GT-DMPC	Bargaining Approach for DMPC	[139]

3.3 Comparison with other distributed methods

In this section, we compare DMS with other state-of-the-art distributed control methods, in particular, the results of [96] are presented here. In Table 3.3.1, the list of the compared methods are reported along with references to their origin.

Table 3.3.2 summarizes the features of the compared distributed predictive control methods. In particular, the controller design, which can be top-down, i.e. the centralized optimal control problem is decomposed into smaller partitions, or bottom-up, i.e. local controllers make decisions, while cooperating with each other.

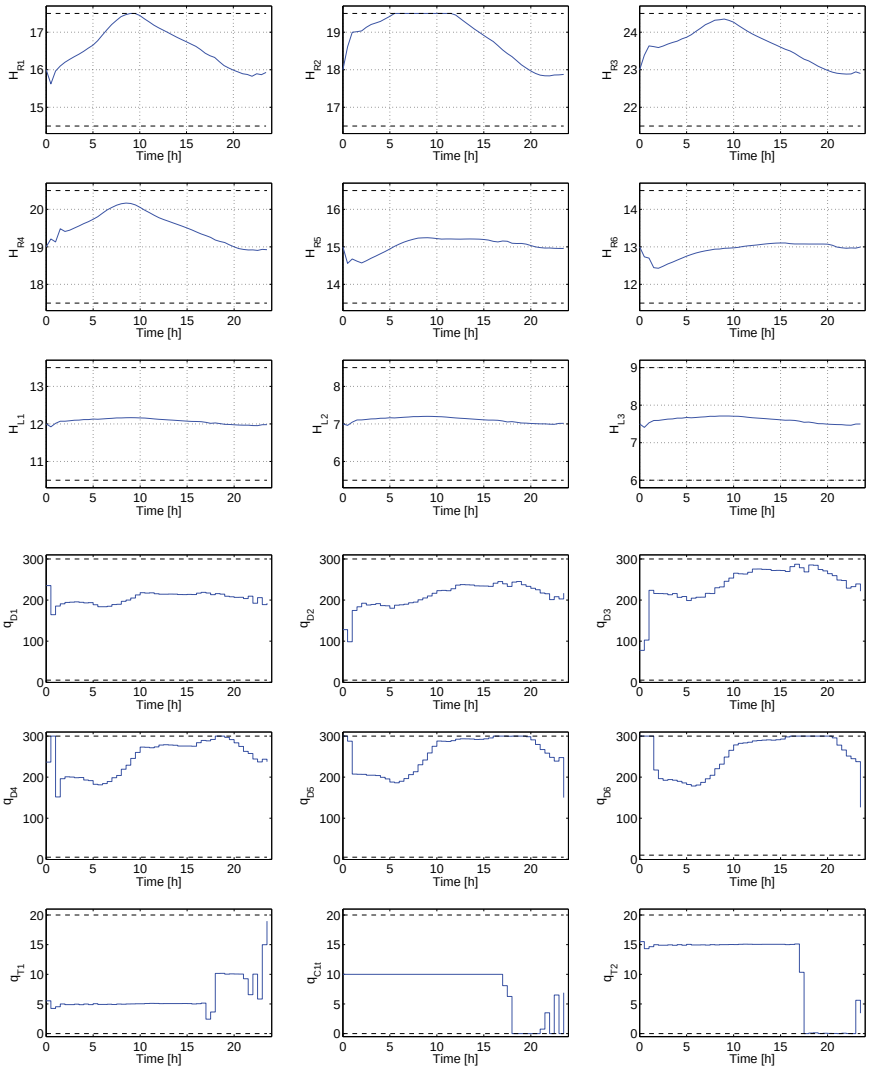


Figure 3.5: The optimal evolution of water levels (top three lines) and the optimal control inputs (bottom three lines).

Table 3.3.2: Feature summary of different distributed predictive control methods considered in [96].

Scheme	Design	Model type	Architecture	Computation	Optimality
DMS	Top-down	Nonlinear	Hierarchical	Iterative	Yes
DMPC-BAN	Bottom-up	Linear	Distributed	Iterative	No
DAPG	Top-down	Linear	Distributed	Iterative	Yes
S-DMPC	Top-down	Linear	Hierarchical	Iterative	Yes
GT-DMPC	Bottom-up	Linear	Distributed	Non-iterative	No

Moreover, the model class and architecture of the different methods are reported. Finally, the computation method and whether convergence to the centralized optimum is attained or not.

In Figure 3.6, the predicted power production given by the different distributed approaches is depicted. The DMS scheme tracks the power reference almost perfectly, being the best performing method in terms of tracking error. This outstanding result is reasonable due to the fact that essentially the centralized optimal control problem is solved, but with a massively parallel integration and linearization routine.

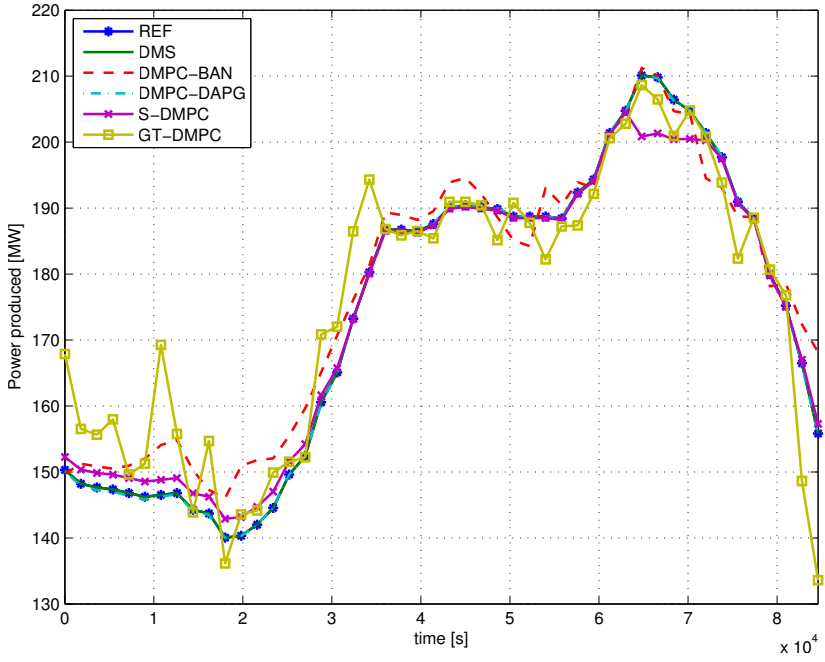


Figure 3.6: Predicted power production with different distributed schemes.

Chapter 4

Smoke detection in buildings

In this chapter, we present another application of DMS. We solve an estimation problem of determining the source of smoke based upon measurements within a building. First, we introduce the dynamic optimization problem and then numerical results are presented. The discussion here follows the results published in [83].

4.1 Problem description

We regard the estimation problem of smoke source detection inside a building. The rooms are connected to each other with doors and regarded as separate subsystems. We assume that smoke spreads according to the PDE

$$\frac{\partial \psi(t,x)}{\partial t} = D\Delta\psi(t,x) - v^f \nabla\psi + u(t,x) + h(\psi(t,x)), \tag{4.1}$$

where $t \in \mathbb{R}$, $x \in \mathbb{R}^2$ is the time and space variables, respectively. Furthermore, $\psi(t,x)$ denotes the smoke concentration in spatial point x and at time t, D is the diffusion coefficient and v^f describes a fixed source-free flow field. The term $u(t,x)$ represents the smoke concentration in space point x and time t, while $h(\psi(t,x))$ is a nonlinear reaction term defined by $h(\psi(t,x)) := -\gamma\psi(t,x)^2$. The walls insulate the rooms with the boundary condition

$$\frac{\partial \psi(t,x)}{\partial n} = 0, \tag{4.2}$$

where n denotes the normal of the boundary. In the first step, we discretize in the spatial domain using the method of lines. In each room, we introduce a mesh of

finite volumes, each of which is denoted by $x_{i,j}(t)$ and the corresponding smoke emission $u_{i,j}(t)$ with $i = 1, \dots, I$ and $j = 1, \dots, J$. The Laplace operator on ψ is approximated by central differences as

$$\Delta\psi(t, x_{i,j}(t)) \approx -4x_{i,j}(t) + x_{i-1,j}(t) + x_{i,j+1}(t) + x_{i+1,j}(t) + x_{i,j-1}(t), \quad (4.3)$$

while the gradient operator on ψ is approximated by means of upwind finite differences. In the following, $x_i(t)$ will denote the aggregated spatial variables of room i with $i = 1, \dots, N$. The smoke evolution in each room can be described by a nonlinear continuous-time ODE of form

$$\dot{x}_i(t) = A_i x_i(t) + u_i(t) - \gamma x_i(t)^T x_i(t) + \sum_{j \in \mathcal{N}(i)} S_{i,j} x_j, \quad (4.4a)$$

$$x_i(0) = 0, \quad (4.4b)$$

where A_i describes the flow field and the diffusion of the smoke and $S_{i,j}$ selects the states of room j that have effect on room i.

We summarize the dynamic estimation problem as

$$\min_{x(\cdot), u(\cdot)} \int_0^T \sum_{k=1}^{N} \|C_k x_k(t) - \mu_k(t)\|_2^2 + \alpha \|u_k(t)\|_1 \mathrm{d}t \quad (4.5a)$$

$$\dot{x}_i(t) = f_i(x(t), u_i(t)) \quad (4.5b)$$

$$x_i(t) = 0 \quad (4.5c)$$

$$u_i(t) \geq 0, \qquad i = 1, \dots, N, \qquad t \in [0, T]. \quad (4.5d)$$

Here, $\mu_k(t)$ denotes the measurements taken in room k in the time interval $[0, T]$, the matrix C_k selects the corresponding state variables, $\alpha > 0$, and $f(\cdot)$ summarizes (4.4a). The first term in (4.5a) penalizes deviations from the measurements, while the second term forces u_k to be sparse. Note that the number of finite volumes, i.e. the dimension of $x_k(t)$ is larger than the dimension of the measurements μ_k, thus the problem is ill-posed. Since it is assumed that the smoke originates from a couple of finite volumes that are close to each other, the L_1 regularization makes the problem tractable and provides useful results.

In the next step, we use the time and space domain based decomposition of the DMS method. We introduce a time grid of $0 = t_1 < \cdots < t_{M+1} = T$ and the spatial decomposition is in between rooms. The dependency between the subsystems is essentially a state-dependency located at the doors connecting two rooms. After

discretization, we obtain the following NLP

$$\min_{x,u,y,z} \sum_{j=1}^{M+1} \sum_{k=1}^{N} \|C_k^j x_k^j - \mu_k^j\|_2^2 + \alpha \|u_k^j\|_1 \tag{4.6a}$$

$$\text{s.t. } x_i^{l+1} = F_i^l(x_i^l, u_i^l, z_i^l) \tag{4.6b}$$

$$y_i^l = G_i^l(x_i^l, u_i^l, z_i^l) \tag{4.6c}$$

$$z_i^l = \sum_{k=1}^{N} A_{ik} y_k^l \tag{4.6d}$$

$$u_i^l \geq 0, \qquad i = 1, \ldots, N,\ l = 1, \ldots, M. \tag{4.6e}$$

Here, y_i^l represents the states of subsystem i on shooting interval $[t^l, t^{l+1}]$ that are necessary for other subsystems. Whereas z_i^l collects all the states of neighbouring subsystems that are necessary for subsystem i.

Note that when solving the NLP (4.6), the evaluation and linearization of functions $F_i^l(\cdot)$ and $G_i^l(\cdot)$ takes place in $N \times M$ processes. Moreover, since this problem originates from a PDE, the derivatives are all structurally non-zero, i.e. the smoke concentration in a particular finite volume depends on all other volumes inside the room.

4.2 Numerical results

In this section, we present an instance of the smoke source detection problem. We consider two rooms connected to each other via a door consisting of 3 finite volumes. Each room is spatially discretized with 19×19 cells and in each room we place 16 sensors, see Figure 4.1. Each shooting interval has a length of 1s, while the whole estimation horizon is 10 seconds. The measurements $\mu(t)$ were generated based upon simulation, to which we added 1% Gaussian noise, see Figure 4.2, which depicts the smoke evolution at two different time points.

First, we compare the runtime for integration and sensitivity generation with different approaches. With the single shooting approach, the rooms are treated as one system and the calculation of sensitivities needs 452 seconds. The multiple shooting method calculates sensitivities on different shooting intervals in parallel, but yet the distributed structure of the problem is not exploited. The runtime for this approach is 80 seconds, which is 5 times faster than the single shooting scheme. If we use the time and spatial domain based decomposition of the DMS method,

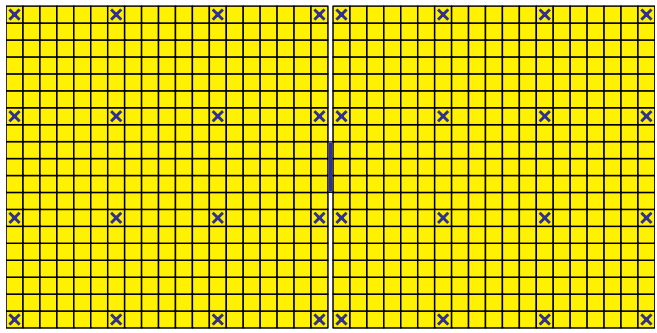


Figure 4.1: The topology of two connected rooms with the location of the smoke sensors.

the runtime of the sensitivity calculation is 16 seconds, which is 5 times faster than multiple shooting and 28 times faster than the single shooting approach. If we turn the parallelization off in the spatial domain, but keep the distributed structure of the NLP, the linearization takes $16 \cdot 2 = 32$ seconds, which is 2 times faster than the multiple shooting approach. Moreover, if we execute DMS fully serially, the runtime is $16 \cdot 2 \cdot 10 = 320$, which is 1.4 times faster than the single shooting scheme. This fact again experimentally proves that the DMS method leads to remarkable speed-up in terms of the sensitivity computation time already in a serial implementation. The explanation for this phenomenon is that in a spatially distributed setup, there is no direct dependency between the states of different subsystems, and thus less forward derivatives have to be calculated inside the integrator. Whereas with single shooting or even multiple shooting this dependency is kept, which results in more forward sensitivities to be calculated and denser Jacobians.

The resulting QP with the single shooting approach is solved in 3.4 seconds, while with multiple shooting it is solved in 35.8 seconds. With DMS, this reduces to 23.8 seconds, which can be explained with the faster linear algebra operations due to the larger number of structural zeros in the Jacobians. Our experiments suggest that the solution quality is the same with all methods, only computation costs differ. We have plotted the sparsity pattern of one QP subproblem with the DMS scheme in Figure 4.3. Here we can see the dense blocks of each room on each shooting interval, together with the identity matrices that couples different shooting intervals together. The spatial coupling is in the bottom of the figure.

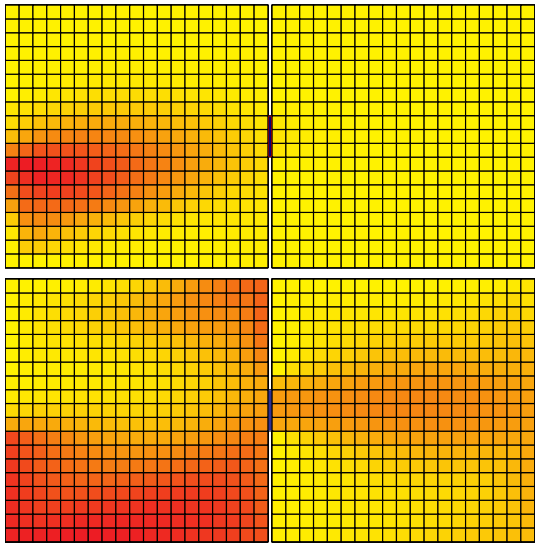


Figure 4.2: The simulation of the smoke evolution at two different time points.

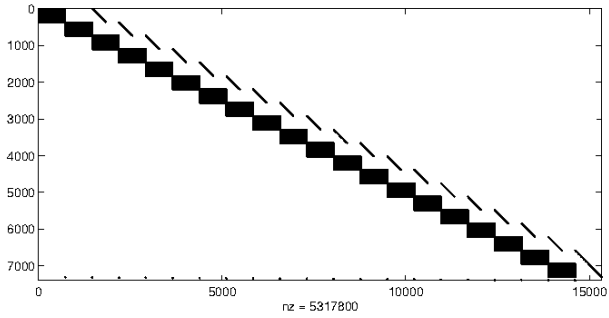


Figure 4.3: Sparsity pattern of the constraint Jacobian of the smoke source detection problem using DMS.

We have solved (4.6) with an SQP method taking full steps. After 4 SQP iterations, a locally optimal solution was found, and the smoke source was found correctly in time. The location of the detected source is mostly correct and is adjacent to the original source. , see Figure 4.4.

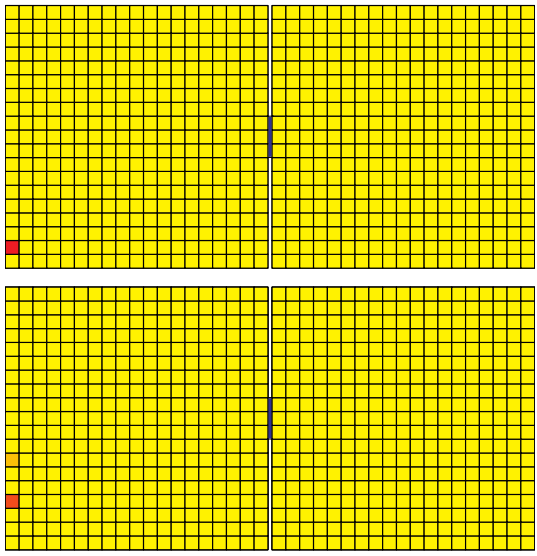


Figure 4.4: The detected location of smoke sources at different times given by the DMS method.

Part II

Distributed Quadratic Programming

Chapter 5

Dual decomposition with first-order methods

In the previous part of this thesis, we have dealt with nonlinear optimal control problems. We have seen that quadratic programs often arise as subproblems, in particular in an SQP framework. This part of the thesis thus investigates the distributed solution of convex quadratic programs.

This chapter is concerned with distributed algorithms for solving convex quadratic programming problems with dual decomposition using only first-order derivatives. In Section 5.1, different distributed quadratic programming formulations are discussed. Section 5.2 provides a survey about distributed QP approaches. A summary of basic mathematical tools used in convex optimization is given in Section 5.3. In Section 5.4, the methodology of dual decomposition is reported and the mathematical features of the resulting dual problem are discussed. Section 5.5 describes existing first-order methods for solving strongly convex QPs with dual decomposition. Here, we pay special attention to characterize the discussed methods in terms of worst case convergence rate and communication requirements. In Section 5.6, we present methods for solving (not necessarily strongly) convex QPs with dual decomposition. The discussion here presents results published in [84].

5.1 Distributed QP formulations

In this section, we describe two different, yet equivalent formulations for distributed QPs. They both have advantages and drawbacks, which we will discuss in the

sequel.

First, we discuss the *matrix-based distributed QP formulation*. We regard the following convex QP formed by M subproblems or agents

$$\min_{x} \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{5.1a}$$

$$\text{s.t.} \sum_{k=1}^{M} A_k x_k = b \tag{5.1b}$$

$$x_k \in \mathcal{X}_k, \qquad k = 1, \ldots, M. \tag{5.1c}$$

Here, $x_i \in \mathbb{R}^{n_i},\ i = 1, \ldots, M$ are the optimization variables, $x^T = [x_1^T, \ldots, x_M^T]$, $0 \preceq H_i \in \mathbb{R}^{n_i \times n_i}$, $c_i \in \mathbb{R}^{n_i}$, $A_i \in \mathbb{R}^{p \times n_i}$, $b \in \mathbb{R}^p$, and $\mathcal{X}_i$ is a polyhedral set defined by $\mathcal{X}_i := \{x_i \in \mathbb{R}^{n_i} | D_i x_i \leq e_i\}$. Note that the objective function (5.1a) and the constraint (5.1c) are decoupled and the different variables of different subproblems are coupled together via (5.1b). A QP having mixed terms in the objective or in the local inequalities can always be transformed into the form of (5.1) by introducing extra variables.

Example 5.1.1 *Given the following QP with mixed terms in the objective,*

$$\min_{x_1, x_2} \frac{1}{2} x_1^T H_1 x_1 + \frac{1}{2} x_2^T H_2 x_2 + \frac{1}{2} x_1^T K x_2, \tag{5.2}$$

we can introduce copies $\bar{x}_1$, $\bar{x}_2$ of x_1, x_2, respectively, which are made equal with coupling constraints as

$$\min_{x_1, \bar{x}_2, x_2, \bar{x}_1} \frac{1}{2} x_1^T H_1 x_1 + \frac{1}{4} x_1^T K \bar{x}_2 + \frac{1}{2} x_2^T H_2 x_2 + \frac{1}{4} \bar{x}_1^T K x_2 \tag{5.3}$$

$$\text{s.t. } x_1 = \bar{x}_1 \tag{5.4}$$

$$x_2 = \bar{x}_2. \tag{5.5}$$

Note that the objective function can be partitioned with respect to $\{x_1, \bar{x}_2\}$ and $\{\bar{x}_1, x_2\}$.

In the matrix-based formulation, we make the following assumption on the sparsity structure of A_i. For all $i = 1, \ldots, M$,

- most of the rows of A_i are zeros,
- if row j of A_i has non-zero elements then there exists k such that row j of A_k has non-zero elements.

The coupling between subproblems are very typical and determine the sparsity pattern of A. Subproblems can share one or more variables. A weighted sum of some variables of each subsystem can also be represented as such a constraint. For example, assume that we have $M = 3$ subproblems, and $A = [A_1|A_2|A_3]$ is defined by

$$A_1 = \begin{bmatrix} \dots & 1 & 0 & 0 & \dots \\ \dots & 0 & 1 & 0 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}, \quad A_2 = \begin{bmatrix} \dots & -1 & 0 & 0 & \dots \\ \dots & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}, \quad A_3 = \begin{bmatrix} \dots & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & -1 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}. \tag{5.6}$$

The first row of A ensures that subproblem 1 and 2 share a variable. The second row ensures that subproblem 1 and 3 share a variable. While the third row sums up some variables of each subsystem.

The matrix-based formulation is useful for easy description of optimization theory and algorithms. However, it is not directly clear which subproblems are coupled with the others and thus talking about information exchange between agents becomes ambiguous.

We refer to the second formulation as *graph-based distributed QP*. It is in the form

$$\min_x \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{5.7a}$$

$$\text{s.t. } A_{i,j} x_i = B_{i,j} x_j \quad (i,j) \in E \tag{5.7b}$$

$$x_l \in \mathcal{X}_l, \quad l = 1, \dots, M. \tag{5.7c}$$

Here, $x_i \in \mathbb{R}^{n_i},\ i = 1, \dots, M$ are the optimization variables, $x^T = [x_1^T, \dots, x_M^T]$, $0 \preceq H_i \in \mathbb{R}^{n_i \times n_i}$, $c_i \in \mathbb{R}^{n_i}$, $A_{i,j} \in \mathbb{R}^{v_{i,j} \times n_i}$ and $B_{i,j} \in \mathbb{R}^{v_{i,j} \times n_j}$ are selector matrices by which certain variables of subproblem i are coupled together with some variables of subproblem j in constraint (5.7b). The topology of the coupled subproblems is described by the graph $\mathcal{G} = (V, E)$ with vertex set $V = \{1, \dots, M\}$ and edge set E. Moreover, $\mathcal{X}_l$ is a polyhedral set defined by $\mathcal{X}_i := \{x_i \in \mathbb{R}^{n_i} | D_i x_i \leq e_i\}$. With the graph-based formulation, one can easily describe operations that involve information exchange between subproblems with the help of $\mathcal{G}$. In fact, in subproblem i it is sufficient to store only the indices of neighbouring subproblems, instead of the whole dependency graph. However, formal derivation of optimization theory becomes more involved due to multiple indices and involvement of the dependency graph $\mathcal{G}$.

The conversion from the graph-based to the matrix-based formulation is straightforward. One has to loop over all the connections $(i, j) \in E$, and insert matrices $A_{i,j}$ and $-B_{i,j}$ after the existing rows of A_i and A_j, while setting the corresponding rows of b to zero. The other way around is somewhat more complex. In the first step, one has to eliminate the non-zero elements of b by adding an extra variable to exactly one subproblem that contributes to the corresponding rows. In the next step, one has to make sure that each row of matrix $A = [A_1, \ldots, A_M]$ involves only two subproblems. This can be reached by adding extra variables to subproblems that are coupled to more than one other so that all coupling variables correspond to a certain other problem partition. Now that the connections are only in between two partitions, this can be expressed with a graph-based coupling with appropriate weights.

5.2 Literature survey

In this section, first we summarize the most commonly used centralized convex QP methods, since in distributed optimization it is often necessary to solve subproblems with a centralized solver. Second, the most widespread and recent convex distributed QP approaches are cited here.

Centralized Convex Quadratic Programming

General convex quadratic programming problems can be solved by various centralized methods. Choosing the most suitable method for a certain problem depends on several factors. The dimensions, the inherent structure, in particular the sparsity pattern of the problem, the context in which the problem has to be solved, e.g. parametric QP. In the following, we collect the most widespread techniques for centralized convex quadratic programming and emphasize their features.

a) *Active-set methods*
One possible way to treat convex QPs is with *active-set methods* (AS methods). These approaches try to find the optimal active set of constraints by means of inexpensive iterations. Although their complexity in the worst-case is exponential in the number of inequalities, they appear to be very efficient in solving a sequence of similar problems due to their hot-starting capabilities. Such a solver is often employed both in model predictive control (MPC) and nonlinear model predictive control (NMPC) applications. In a dual decomposition framework, as it is discussed in the next section, such a method is preferred, since in between the dual iterations, only the linear term of each QP changes. For large scale

problems, in particular QPs with many inequalities, active-set methods might require way too many iterations to find the optimal solution when cold-started.

We have knowledge of the following implementations.

- Dense: QuadProg [63], qpOASES [45], QPOPT [58]
- Sparse: [18], QPA [66], CPLEX [77], QPBLUR [94], Gurobi[2]

b) *Interior-point methods*
Another commonly used family for convex quadratic programming is the *interior-point methods* (IP methods) (for a good summary, see [144]). These approaches are typically intended for large scale problems, since the worst-case complexity for the number of iterations is a polynomial of the number of inequalities. IP methods relax the inequality constraints with a barrier penalty term and approach to an optimal solution by making steps in Newton directions for barrier subproblems. Thus, one iteration of an IP method is typically more expensive than one of an AS method. IP methods have the disadvantage that they are difficult to hotstart, unlike AS methods. In the literature one can find several methods, each for solving sparse problems, e.g. QPSOL [59], LOQO [140], OOQP [57], MOSEK [3], HOPDM [10], QPB [66], CPLEX [77], Gurobi [2].

c) *Gradient-based methods*
These approaches rely only on evaluating the objective gradient and a cheap projection step onto the feasible set. Calculating the objective gradient necessitates the computation of a matrix-vector product and a vector-vector product. Since these methods do not utilize second-order derivatives, their worst-case convergence rate is typically linear or sublinear depending on the problem properties. For instance, in [112], a conjugate gradient method is proposed for sparse bound constrained QPs. The *fast gradient methods* [106] maintain an extra so-called momentum term to accelerate the gradient method. They can both solve convex QPs with "simple enough" constraints, e.g. orthant, box, simplex. In [36], the author proposes a hybdrid method, a projected gradient step if the active set changes, otherwise a conjugate gradient step.

d) *Other methods*
In [32], Cryer proposed a method for solving convex QPs on the nonnegative orthant by solving the KKT system with a Gauss-Seidel type approach, which is often referred to as successive overrelaxation (SOR).

Distributed Convex Quadratic Programming

Distributed methods are of interest once the centralized optimization problem is too large to be solved centrally due to memory or computation capacity limits. It may

also occur that the whole optimization problem cannot be summarized centrally, becasue different parts of the problem are known only by different subsystems. For instance, connecting segments of a power grid are aware of their own objective, operational constraints and how they are connected to other subsystems, but have limited or no knowledge of their detailed operation.

In distributed optimization, two levels of distribution can be considered. First, the underlying linear algebraic operations, such as linear system solvers [69], matrix multiplication [28], etc., are implemented in a distributed fashion in order to obtain speedup. These numerical algorithms usually operate on a computer with multiple cores, but share the physical memory. Second, the higher level distribution, when the focus is on decomposing the original large problem in a way so that the solution of several smaller dimensional problems are necessary, which are typically easier than the original problem itself. The distinct parts of the problems exist in the distributed memory, e.g. even in geographically separated locations. Here, the communication costs have to be taken into account, for instance, prefer information exchange of float vectors to matrices depending on the number of non-zeros, and minimize the efforts taken by a central coordinator.

In the following, we review how optimization methods can be used in a distributed fashion. In particular, we differentiate between primal and dual decomposition methods. Primal decomposition methods operate in the primal space and thus the dual variables are not involved in the optimization process, unlike dual decomposition methods, which typically relax some coupling constraints by some Lagrange multipliers.

a) *Primal decomposition methods*

The *gradient method* (GM) [27] is suitable for distributed execution, since it requires only the evaluation of the objective gradient function, which is a matrix-vector product and a vector-vector product in the context of unconstrained convex quadratic functions. For this class of functions the worst case convergence rate is sublinear, whereas for the strongly convex case the rate is linear [106, p. 70]. The GM is easy to understand and implement, however, determining a proper stepsize may be difficult and expensive in a distributed context due to the computation complexity of a Lipschitz constant. Simple constraints can be handled by a projection step, which is very difficult to distribute.

The *fast gradient method* [105], since it requires only the objective gradient evaluations, is also a possible distributed method for unconstrained or simply constrained strictly convex QPs with a projection step. For strictly convex QP the worst-case convergence rate is linear, but faster than the gradient method. The stepsize control mechanism may be difficult.

In [108], Nesterov proposed a random coordinate descent method (RCDM) and its accelerated variant for convex problems, for which the computation of the full objective gradient is very costly. Instead, the current iterate is updated by using a random slice of the gradient. The worst-case convergence rate of the expected objective value is proven to be linear for strongly convex functions. Simple constraints are also incorporated. The stepsize is determined by a backtracking process based on gradient slices.

The *Jacobi algorithm* minimizes a convex objective function along several (set of) directions simultaneously and the new iterate is given by the found directional minimizers. In a stricly convex QP context, it boils down to a scaled gradient method and the worst-case convergence rate is linear [78, pp. 61-65]. Simple constraints can be treated by projection into the feasible set.

Iterative methods for large linear systems such as *conjugate gradient (CG) method* [131] and its variants, are directly applicable to solving unconstrained or equality constrained strictly convex QPs. Since these methods rely only on matrix-vector products, they are well suited for distributed optimization, although several global communication steps are necessary. It has been shown that the CG method theoretically converges to the exact solution in at most n steps [110, p. 114], where n is the dimension of optimization variable. However, in practice this may be different due to the accumulation of rounding errors. Moreover, assume $n = 10^6$, the same number of iterations is not tractable in practice. Simple constraints can be handled by a projection step.

b) *Dual decomposition methods*

These approaches are applicable to strongly convex problems with decomposable objective and linear coupling in between the subproblems, and thus for strictly convex QPs as well. The coupling equality constraints are dualized by moving them into the objective with a Lagrange weight. In the dual problem the optimal dual variables are the optimal solution of an unconstrained problem. See e.g. [13, pp. 229-231], [91] and the references therein. There are different strategies in this framework, depending on how the optimal dual variables are found. For instance, *coordinate ascent method* [31], *dual ascent* (i.e. a gradient method in the dual space) [89, 135, 61], also fast gradient method [123, 101, 60]. A non-strictly convex objective function renders the dual problem non-differentiable, for which a *dual subgradient method* [132, 88, 104] might be used. The most important drawback of gradient- or subgradient-based dual decomposition methods is that they provably cannot be faster than sublinear in the worst case [106, p. 61]. In [65], a primal-dual interior-point is proposed, which does not require formation of large linear systems, but relies on a preconditioned iterative method to solve the primal-dual Newton system.

Non strongly convex separable problems can be treated by the *proximal minimization algorithm* [13, p. 232], which instead of the original convex problem

solves a series of strictly convex subproblems. The same idea is used in a dual decomposition framework for convex decomposable problems in [102], where the optimal dual variables are found by a fast gradient method. Furthermore, a similar smoothing technique is used for convex problems along with dual decomposition in Chapter 7 of [134].

The typical sublinear rate of gradient based methods can be overcome by employing second-order derivatives in the dual space. In [44, 48], the authors make use of Hessian information in order to accelerate the optimization of dual variables. A Newton method coupled with direct linear algebra is used in a dual decomposition framework in [103], in which the inequalities are treated by barrier penalty functions.

The *method of multipliers* [75] utilizes the augmented Lagrangian function, which is minimized over the primal variables followed by an update of Lagrange multipliers and penalty parameter. The approach is proven to converge to a local optimum of a nonlinear programming problem with at least linear rate under regular assumptions [14, pp. 116-117]. However, the method of multipliers cannot distribute the optimization over the primal variables. If the original problem is convex, this limitation can be overcome by using the Jacobi algorithm (or its similar variant the Gauss-Seidel method). An extreme version of this approach is the *alternating method of multipliers* (ADMM) [23], which takes only one step of the Gauss-Seidel algorithm in the primal space followed by the update of Lagrange multipliers. Its accelerated version [64] finds the dual optimizers with an optimal rate.

5.3 Mathematical background

In this section, the nomenclature of convex optimization is introduced. For further and more detailed descriptions we refer the reader to optimization textbooks such as [24, 110].

Definition 5.3.1 (Lipschitz continuity) *A function $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is called Lipschitz continuous with constant $L \in \mathbb{R}$ if for $\forall x, y \in D$*

$$|f(x) - f(y)| \leq \|x - y\| L \tag{5.8}$$

holds.

Definition 5.3.2 (convex function) *A function $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is called convex if for $\forall x, y \in D$ and $\forall \alpha \in [0, 1]$*

$$f(\alpha x + (1 - \alpha) y) \leq \alpha f(x) + (1 - \alpha) f(y) \tag{5.9}$$

holds.

Definition 5.3.3 (concave function) *A $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is called concave if $-f(\cdot)$ is convex.*

Definition 5.3.4 (strictly convex function) *A function $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is called strictly convex if for $\forall x, y \in D$ and $\forall \alpha \in (0, 1)$*

$$f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha) f(y) \tag{5.10}$$

holds.

Definition 5.3.5 (strongly convex function) *A differentiable function $f : D \subset \mathbb{R}^n \to \mathbb{R}$ is called strongly convex with strong convexity parameter $\mu > 0$ if for $\forall x, y \in D$*

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq \|x - y\|^2 \mu \tag{5.11}$$

holds.

Definition 5.3.6 (convergence rates) *Assume that $\lambda^{(k)} \to \lambda^*$ as $k \to \infty$. The series $\lambda^{(k)}$ converges*

- *sublinearly if*

$$\|\lambda^{(k)} - \lambda^*\| \leq c \frac{1}{k^\alpha}, \text{ with } \alpha \in (0, \infty], \ c \in \mathbb{R} \tag{5.12a}$$

- *linearly if*

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha^k, \text{ with } \alpha \in [0, 1), \ c \in \mathbb{R} \tag{5.12b}$$

- *superlinearly if*

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha_{(k)}^k, \text{ with } \alpha_{(k)} \to 0, \ c \in \mathbb{R} \tag{5.12c}$$

- *quadratically if*

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha^{2^k}, \text{ with } \alpha \in [0, 1), \ c \in \mathbb{R}. \tag{5.12d}$$

From an optimization point of view, the sublinear convergence rate means very poor performance, the linear convergence rate has sometimes acceptable performance depending on the value of α. The superlinear and quadratic convergence rates are desirable.

5.4 Quadratic programming with dual decomposition

In this section, we discuss the concept of dual decomposition, which introduces a two-level optimization problem in order to decompose a distributed convex QP, which we restate in the graph-based form as

$$\min_{x} \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{5.13a}$$

$$\text{s.t. } A_{i,j} x_i = B_{i,j} x_j \quad (i,j) \in E \tag{5.13b}$$

$$x_l \in \mathcal{X}_l, \quad l = 1, \ldots, M. \tag{5.13c}$$

The low-level optimization problems, which provide gradients to the high-level, are solved in parallel, while in the high-level problem the updates of the dual variables are carried out and propagated to the low-level. We consider several optimization methods for the high-level problem that make use of only gradients and give their convergence properties. We describe how the methods relate to each other, explain their parallel or distributed nature and clarify the communication requirements.

In the sequel, we restrict our attention to a subset of problem (5.13), namely where for all $i \in \{1, \ldots, M\}$ the matrix $H_i \succ 0$, i.e. H_i is positive definite. The methods considered here rely on the dualization of constraint (5.13b), which results in the (partial) *Lagrangian function* defined by

$$L(x, \lambda) := \sum_{i=1}^{M} L_i(x_i, \lambda_{\mathcal{N}_i}) := \sum_{i=1}^{M} \left(\frac{1}{2} x_i^T H_i x_i + c_i^T x_i + \right.$$

$$\left. \sum_{(i,k) \in E} \lambda_{i,k}^T A_{i,k} x_i - \sum_{(k,i) \in E} \lambda_{k,i}^T B_{k,i} x_i \right), \tag{5.14}$$

where for $\forall (i,j) \in E$, $\lambda_{i,j} \in \mathbb{R}^{v_{i,j}}$ are the dual variables corresponding to (5.13b). We denote the collection of all the dual variables with λ, let

$$\mathcal{N}_i := \{j | (i,j) \in E \text{ or } (j,i) \in E\}, \tag{5.15}$$

and let $\lambda_{\mathcal{N}_i}$ denote the concatenation of all dual multipliers $\lambda_{j,k}$ which influence suproblem i. Using strong duality one can introduce the *Lagrange dual function* $d(\lambda)$ as

$$d(\lambda) := \min_{x \in \mathcal{X}} L(x, \lambda) = \sum_{i=1}^{M} \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}) \tag{5.16}$$

and the unconstrained dual concave optimization problem as

$$\max_{\lambda} d(\lambda). \tag{5.17}$$

Note that the evaluation of $d(\lambda)$ may be divided into solving M independent inequality constrained parametric QPs of form

$$\min_{x_i \in \mathcal{X}_i} \frac{1}{2} x_i^T H_i x_i + \left(c_i^T + \sum_{(i,k)\in E} \lambda_{i,k}^T A_{i,k} - \sum_{(k,i)\in E} \lambda_{k,i}^T B_{k,i} \right) x_i \tag{5.18}$$

which provides space for parallelized or even distributed methods. In the following, we will refer to (5.18) as the i-th subproblem. Notice that by using the Lagrangian function, subproblem i depens only linearly on λ, thus it is beneficial to use a QP solver with hotstarting capabilities, such as an online active set strategy [43].

Features of the dual function

In the following lemmas, we analyze the dual function from an optimization point of view.

Lemma 5.4.1 *The dual function $d(\lambda)$ defined in (5.16) is concave and continuously differentiable.*

Proof *Since the Lagrange function $L(x, \lambda)$ is affine in λ and $d(\lambda)$ is the minimum of affine functions indexed by x, the dual function is concave. The continuity of $d(\lambda)$ is implied by being concave. The differentiability of the dual function depends on whether there exists a bijection between the optimal primal and dual variables. In our case, this bijection is ensured by having a strictly convex primal objective. Indeed, we apply Danskin's theorem [35] to the dual function; since $\min_{x\in\mathcal{X}} L(x, \lambda)$ is attained at a unique point x^* for a fixed λ, $d(\cdot)$ is differentiable at λ with $\nabla d(\lambda) = \nabla_\lambda L(x^*, \lambda)$. In [46], the author shows that $x^*(\lambda) := \min_{x\in\mathcal{X}_i} L(x, \lambda)$ is a continuous function in λ, which implies that $\nabla d(\lambda)$ is continuous. We note that a (non strictly) convex primal problem would render the dual function non-differentiable.*

■

The results of this lemma allows for the application of first-order methods in the dual space to find the optimal dual variables. The dual gradient for any λ in direction $\lambda_{i,j}$ is given by

$$\nabla_{\lambda_{i,j}} d(\lambda) = A_{i,j} x_i^*(\lambda_{\mathcal{N}_i}) - B_{i,j} x_j^*(\lambda_{\mathcal{N}_j}). \tag{5.19}$$

where $x_i^*(\lambda_{\mathcal{N}_i})$ is the optimal solution of subproblem i. Note that the calculation of this slice of the dual gradient involves the solution of only subproblem i and j. In a distributed optimization context, if subproblems i and j are solved on different nodes, only a local exchange of contributions to (5.19) is necessary to calculate the corresponding slice of the dual gradient. This fact makes first-order methods particularly suited for distributed optimization.

Lemma 5.4.2 (Lipschitz continuity of the dual gradient) *The dual gradient function $\nabla d(\lambda)$ is Lipschitz continuous.*

Proof *In [146], the author shows that $x^*(\lambda)$ is a piecewise affine function of λ. In order to show that $d(\lambda)$ is Lipschitz continuous, we have to find the affine piece where this function changes with the highest steepness. Indeed, the eigenvalue of the coefficient matrix with the largest magnitude of all affine pieces provides a maximal rate of change.*

■

In the view of Lemma 5.4.2 the (concave) dual function may be bounded below by a quadratic function having curvature L.

It can be shown that the dual function $d(\lambda)$ has zero curvature in certain directions (see proof in Lemma 7.1.1), which implies that $-d(\lambda)$ is not strongly convex.

The methods discussed in this chapter are based upon the general scheme presented in Algorithm 5.1.

Algorithm 5.1: Dual decomposition framework

Input : $\lambda^{(0)}$
1 **while** *no convergence* **do**
2 **foreach** $i = 1, \ldots, M$ **do**
3 $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}^{(k)})$
4 **end**
5 Calculate $\nabla d(\lambda^{(k)})$
6 $\lambda^{(k+1)} = \lambda^{(k)} + \mathcal{M}\left(\nabla d(\lambda^{(0)}), \ldots, \nabla d(\lambda^{(k)})\right)$
7 $k := k + 1$
8 **end**
Output : $\lambda^{(k)}, x^{(k)}$

Here $\mathcal{M} : \mathbb{R}^{p \times (k+1)} \to \mathbb{R}^p$. In Steps 2-4, M parametric quadratic programs have to be solved in parallel, which provides gradient information to the dual space. In Step 6, the next dual iterate $\lambda^{(k+1)}$ is calculated by mapping the previous gradients to a correction term.

There exists an important theoretical result from Y. Nesterov [106, p. 61], who showed that if a method implements $\mathcal{M}$ as a linear combination operator, i.e. $\mathcal{M}\left(\nabla d(\lambda^{(0)}), \ldots, \nabla d(\lambda^{(k)})\right) \in \operatorname{span}\left(\nabla d(\lambda^{(0)}), \ldots, \nabla d(\lambda^{(k)})\right)$ then the convergence cannot be fast in the beginning. More precisely, for any method following a scheme

$$\lambda^{(k+1)} \in \lambda^{(k)} + \operatorname{span}\left(\nabla f(\lambda^{(0)}), \ldots, \nabla f(\lambda^{(k)})\right), \tag{5.20}$$

and for any $k: \ 1 \leq k \leq \frac{1}{2}(p-1)$ and $\lambda^{(0)}$ there exists a concave, differentiable function $f(\lambda) : \mathbb{R}^p \to \mathbb{R}$ with L-Lipschitz continuous gradient such that

$$f^* - f(\lambda^{(k)}) \geq \frac{3L\|\lambda^{(0)} - \lambda^*\|}{32(k+1)^2}, \tag{5.21a}$$

$$\|\lambda^{(k)} - \lambda^*\|^2 \geq \frac{1}{8}\|\lambda^{(0)} - \lambda^*\|^2. \tag{5.21b}$$

In other words, the existence of function $f(\cdot)$ renders the worst-case convergence rate of any first-order method sublinear in the beginning. However, this does not mean that with practical applications one might not observe faster convergence. We will see that different realizations of $\mathcal{M}$ lead to different convergence behaviour.

5.5 Methods for strongly convex separable QPs

In this section, we present methods that employ the Lagrangian function directly. These schemes can cope only with strongly convex QPs, i.e. the original problem has to be positive definite. Non-strongly convex problems can be transformed into strongly convex ones by adding a small diagonal regularization to the Hessian matrix. This implies that the regularized solution may slightly differ from the solution of the original problem.

5.5.1 Gradient method with fixed stepsize

The *gradient method*, also referred to as *steepest ascent* method, is one of the oldest first-order methods proposed first by Cauchy in 1847 [27]. It uses only the latest gradient to obtain a search direction.

$$\lambda^{(k+1)} := \lambda^{(k)} + t^{(k)} \nabla d(\lambda^{(k)}) \tag{5.22}$$

It can be shown (see [106, pp. 69-70] for details) that the optimal stepsize is $t^{(k)} = \frac{1}{L}$, where L is the Lipschitz constant of the gradient function. The resulting

method as a dual decomposition approach is often referred to as *dual ascent*, which we report in Algorithm 5.2.

Algorithm 5.2: Gradient method with fixed stepsize. (GM)

Input : $\lambda^{(0)}, L$
1 $k := 0$
2 **while** *no convergence* **do**
3 $\quad x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}^{(k)})$
4 $\quad$ **foreach** $j \in \mathcal{N}_i$ **do**
5 $\quad\quad$ Send $x_i^{(k)}$ to node j
6 $\quad\quad$ Receive $x_j^{(k)}$ from node j
7 $\quad\quad$ **if** $(i,j) \in E$ **then**
8 $\quad\quad\quad \nabla_{\lambda_{i,j}} d = A_{i,j} x_i^{(k)} + B_{i,j} x_j^{(k)}$
9 $\quad\quad\quad \lambda_{i,j}^{(k+1)} := \lambda_{i,j}^{(k)} + \frac{1}{L} \nabla_{\lambda_{i,j}} d$
10 $\quad\quad$ **else**
11 $\quad\quad\quad \nabla_{\lambda_{j,i}} d = A_{j,i} x_j^{(k)} + B_{j,i} x_i^{(k)}$
12 $\quad\quad\quad \lambda_{j,i}^{(k+1)} := \lambda_{j,i}^{(k)} + \frac{1}{L} \nabla_{\lambda_{j,i}} d$
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 $\quad k := k + 1$
16 **end**
Output : $\lambda_{\mathcal{N}_i}^{(k)},\ x_i^{(k)}$

Convergence: We can see that if the function $d(\lambda)$ has high curvature, i.e. the Lipschitz constant L gets large, the method takes very short steps in Step 7, while if it has moderately flat curvature it takes longer steps. The convergence rate of the algorithm is sublinear [106, pp. 70] that is

$$d^* - d(\lambda^{(k)}) \leq \frac{2L\|\lambda^{(0)} - \lambda^*\|^2}{k+4}. \tag{5.23}$$

Since this is not proportional to the lower bound given in (5.21a), this method is not optimal in the sense of Nesterov [106, p. 71].

Communication: Algorithm 5.2 can be implemented in a fully distributed fashion. The i-th subproblem is solved on node i in Step 3, and its primal solution is communicated only to the neighbours in Steps 5 and 6. The dual variables are

updated with a steepest ascent step in Steps 7-13. In essence, in every iteration each subproblem sends its contribution to the dual gradient to other subsystems that are coupled to it and makes the step locally. In this way, only local communication of vectors between coupled subproblems is necessary and no central coordination is needed. However, the Lipschitz constant L must be globally available a priori.

Note that calculating the Lipschitz constant L might be hard or even impossible in practice. This motivates us to use adaptive stepsizes or equivalently to approximate the Lipschitz constant based on local information. In the literature, one can find many different approaches on how to do this (for a good summary see [148] and the references therein).

5.5.2 Global Barzilai-Borwein method

We consider the adaptive gradient method from [7] that is often regarded as *Barzilai-Borwein update rule* and reads as

$$y^{(k)} := \nabla d(\lambda^{(k+1)}) - \nabla d(\lambda^{(k)}), \tag{5.24a}$$

$$s^{(k)} := \lambda^{(k+1)} - \lambda^{(k)}, \tag{5.24b}$$

$$t^{(k)} := \frac{(s^{(k)})^T s^{(k)}}{(s^{(k)})^T y^{(k)}} \ . \tag{5.24c}$$

This way of calculating the stepsize does not involve the calculation of the Lipschitz constant. A variant of the method for nonlinear convex functions was published in [121] and is presented in Algorithm 5.3 in a centralized manner. Note that in Algorithm 5.3, there is no explicit need for a Lipschitz constant and the stepsize is determined by a non-monotone line-search subprocedure, see the while loop in Line 14.

Convergence: In [7], the authors prove superlinear asymptotic convergence for two-dimensional strongly convex quadratic functions. Moreover, R-linear convergence was shown for general strongly convex quadratics in [33]. The general convex quadratic case was investigated in [49].

Communication: As a consequence of the line-search procedure, the update of the dual variables has to be coordinated from a dedicated process, while the primal variables x_i are updated in parallel in the coordinated processes, see Lines 9 and 17. These processes compute their contribution to the dual objective and dual gradient functions, which have to be transmitted to the coordinator process, which updates

Algorithm 5.3: Global Barzilai-Borwein method. (GBB)

Input : $\lambda^{(0)},\ M \geq 0,\ \gamma \in (0,1),\ \delta > 0,\ \sigma \in (0,1),\ 0 < \epsilon << 1$

$k := 0$
$\alpha_0 = 1$
while *no convergence* **do**
  if $\alpha_k \leq \epsilon$ or $\alpha_k \geq \frac{1}{\epsilon}$ **then**
    $\alpha_k := \delta$
  end
  if $k == 0$ **then**
    foreach $i = 1, \ldots, M$ **do**
      $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda^{(k)})$
    end
    $\nabla d(\lambda^{(k)}) := \sum_{(j,l)\in E} A_{j,l} x_j^{(k)} - B_{j,l} x_l^{(k)}$
  end
  $t := \frac{1}{\alpha_k}$
  while *True* **do**
    $\lambda^{(k+1)} := \lambda^{(k)} + t \nabla d(\lambda^{(k)})$
    foreach $i = 1, \ldots, M$ **do**
      $x_i^{(k+1)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda^{(k+1)})$
    end
    $d(\lambda^{(k+1)}) := L(x^{(k+1)}, \lambda^{(k+1)})$
    if $-d\left(\lambda^{(k+1)}\right) \leq \max_{0 \leq j \leq \min(k,M)} (-d(\lambda^{(k-j)})) - \gamma t \|\nabla d(\lambda^{(k)})\|_2^2$ **then**
      break
    else
      $t := \sigma t$
    end
  end
  $\nabla d(\lambda^{(k+1)}) := \sum_{(j,l)\in E} A_{j,l} x_j^{(k+1)} - B_{j,l} x_l^{(k+1)}$
  $y^{(k)} := \nabla d(\lambda^{(k+1)}) - \nabla d(\lambda^{(k)})$
  $s^{(k)} := \lambda^{(k+1)} - \lambda^{(k)},$
  $\alpha_{k+1} := \dfrac{(s^{(k)})^T y_k}{\|s^{(k)}\|_2^2}$
  $k := k + 1$
end

Output : $\lambda^{(k)},\ x^{(k)}$

the dual variables and propagates them to the coordinated processes. In our case, the price of calculating $d(\lambda)$ and $\nabla d(\lambda)$ is the same, namely solving M QPs in parallel. While the cost of the coordination is proportional to the logarithm of node number.

5.5.3 Fast gradient method with fixed stepsize

One can introduce another type of acceleration without a line-search procedure but fixed stepsize. The family of *fast gradient method*s was introduced in [105]. We describe a simplified optimal method for convex differentiable functions with L-Lipschitz gradient from [106, pp. 80] in Algorithm 5.4 for distributed execution. The fast gradient method maintain a sequence $\tilde{\lambda}^{(k)}$, which is often referred to as

Algorithm 5.4: Fast gradient method with constant stepsize (FG)

Input : $\lambda^{(0)},\ L,\ \alpha_0 \in (0,1)$
$\tilde{\lambda}^{(0)} := \lambda^{(0)}$
$k := 0$
while *no convergence* **do**
  $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \tilde{\lambda}^{(k)}_{\mathcal{N}_i})$
  Compute $\alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2$
  $\beta_k := \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
  foreach $j \in \mathcal{N}_i$ **do**
    Send $x_i^{(k)}$ to node j
    Receive $x_j^{(k)}$ from node j
    if $(i,j) \in E$ **then**
      $\nabla_{\lambda_{i,j}} d := A_{i,j} x_i^{(k)} - B_{i,j} x_j^{(k)}$
      $\lambda_{i,j} := \tilde{\lambda}_{i,j}^{(k)} + \frac{1}{L} \nabla_{\lambda_{i,j}} d$
      $\tilde{\lambda}_{i,j}^{(k)} := \lambda_{i,j}^{(k+1)} + \beta_k \left(\lambda_{i,j}^{(k+1)} - \lambda_{i,j}^{(k)} \right)$
    else
      $\nabla_{\lambda_{j,i}} d := A_{j,i} x_j^{(k)} - B_{j,i} x_i^{(k)}$
      $\lambda_{j,i} := \tilde{\lambda}_{j,i}^{(k)} + \frac{1}{L} \nabla_{\lambda_{j,i}} d$
      $\tilde{\lambda}_{j,i}^{(k)} := \lambda_{j,i}^{(k+1)} + \beta_k \left(\lambda_{j,i}^{(k+1)} - \lambda_{j,i}^{(k)} \right)$
    end
  end
  $k := k+1$
end
Output : $\lambda^{(k)},\ x^{(k)}$

the momentum term. Indeed, the momentum term tries to give an approximation of the next iterate by extrapolation. Note that since α_k and β_k do not depend on the problem data, they can be computed offline.

Convergence: The convergence rate of Algorithm 5.4 is optimal, yet sublinear, i.e.

$$d^* - d(\lambda) \leq \frac{4L\|\lambda^{(0)} - \lambda^*\|^2}{(k+2)^2}. \tag{5.25}$$

The optimality of the method refers to the fact that one can give lower and upper estimates on $d^* - d(\lambda^{(k)})$ that are proportional. In this case, they are both in the order of $\frac{1}{k^2}$.

Communication: One can implement Algorithm 5.4 in a fully distributed fashion, since for the update of dual variables only gradient vectors of the coupled subproblems are necessary. Thus, local communication of vectors and knowledge of the Lipschitz constant system-wide is necessary.

5.5.4 Fast gradient method with adaptive stepsize

Finding the tightest Lipschitz constant, or even an approximation might be again a difficult problem, thus we discuss an adaptive variant of Algorithm 5.4 from [81], with a line-search strategy from [8] in Algorithm 5.5. Here, the Lipschitz constant is adjusted in each iteration by a line-search procedure. A similar approach is proposed in [107] for problems with composite objective function. A Lipschitz constant that is too large may lead to very conservative steps, while a Lipschitz constant that is too small may lead to divergence of the algorithm. This method does not require the knowledge of the Lipschitz constant L. Instead, an approximation $L^{(k)} \leq L$ is maintained based upon local information. In the inner while loop from Line 8 on, we increase $L^{(k)}$ repeatedly until it fulfils the Lipschitz property in points $\tilde{\lambda}^{(k)}$, λ. In this approach, the decrease of $L^{(k)}$ is not allowed and the convexity parameter is not approximated as it is known to be zero.

Algorithm 5.5: Fast gradient method with adaptive Lipschitz (FG-AL)

Input : $\lambda^{(0)},\ L^{(k)},\ \rho > 1,\ \alpha_0 \in (0,1)$
$\tilde{\lambda}^{(0)} := \lambda^{(0)}$
$k := 0$
while *no convergence* **do**
  foreach $i = 1, \dots, n$ **do**
    $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \tilde{\lambda}^{(k)})$
  end
  $\nabla \tilde{d} := \sum_{(j,l)\in E} A_{j,l} x_j - B_{j,l} x_l$
  while *True* **do**
    $\lambda := \tilde{\lambda}^{(k)} + \frac{1}{L^{(k)}} \nabla \tilde{d}$
    foreach $i = 1, \dots, n$ **do**
      $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda)$
    end
    $\nabla d := \sum_{(j,l)\in E} A_{j,l} x_j - B_{j,l} x_l$
    if $|(\tilde{\lambda}^{(k)} - \lambda)^T (\nabla \tilde{d} - \nabla d)| \leq \frac{1}{2} L^{(k)} \|\lambda - \tilde{\lambda}^{(k)}\|_2^2$ **then**
      break
    else
      $L^{(k)} := \rho L^{(k)}$
    end
  end
  $\lambda^{(k+1)} := \lambda$
  Compute $\alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2$
  $\beta_k := \frac{\alpha_k (1 - \alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$
  $\tilde{\lambda}^{(k+1)} := \lambda^{(k+1)} + \beta_k \left(\lambda^{(k+1)} - \lambda^{(k)}\right)$
  $k := k + 1$
end
Output : $\lambda^{(k)}$

Convergence: If we choose $L^{(k)} := L$ the convergence speed of Algorithm 5.5 is optimal sublinear, i.e.

$$d(\lambda^*) - d(\lambda^{(k)}) \leq \left(\frac{4L}{(2\sqrt{L} + k\sqrt{\gamma_0})^2} \right) \left(d(\lambda^*) - d(\lambda^{(0)}) + \frac{\gamma_0}{2} \|\lambda^{(0)} - \lambda^*\|_2^2 \right), \tag{5.26}$$

with some $\gamma_0 \in \mathbb{R}$. L is the Lipschitz constant of $\nabla d(\lambda)$. The addition of the line-search based Lipschitz approximation does not affect the asymptotic convergence, thus FG-AL is an optimal method.

Communication: Since $L^{(k)}$ is adjusted with a line-search procedure, central coordination is necessary. The coordinated processes calculate their contribution to the dual gradient in parallel, see Steps 5 and 11. Then the contributions are sent to the coordinator, which governs the stepsize procedure accordingly.

5.5.5 Fast gradient method with adaptive restart

A recent approach from [111] tries to avoid the increase of the objective in fast gradient methods by a simple heuristic. Whenever

$$-\nabla d(\tilde{\lambda}^{(k)})^T \left(\lambda^{(k+1)} - \lambda^{(k)}\right) > 0 \tag{5.27}$$

holds, $\tilde{\lambda}^{(k+1)}$ is reset to $\lambda^{(k+1)}$. The method itself is given in Algorithm 5.6.

Algorithm 5.6: Fast gradient method with adaptive restart (FG-AR)

Input : $\lambda^{(0)},\ L,\ \alpha_0 \in (0,1)$
1 $\tilde{\lambda}^{(k)} := \lambda^{(0)}$
2 $k := 0$
3 **while** *no convergence* **do**
4 **foreach** $i = 1, \ldots, n$ **do**
5 $x_i^{(k)} := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \tilde{\lambda}^{(k)})$
6 **end**
7 $\nabla d := \sum_{(j,l)\in E} A_{j,l} x_j^{(k)} - B_{j,l} x_l^{(k)}$
8 $\lambda^{(k+1)} := \tilde{\lambda}^{(k)} + \frac{1}{L}\nabla d$
9 Compute $\alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2$
10 $\beta_k := \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
11 $\tilde{\lambda}^{(k+1)} := \lambda^{(k+1)} + \beta_k \left(\lambda^{(k+1)} - \lambda^{(k)}\right)$
12 **if** $-\nabla d^T \left(\lambda^{(k+1)} - \lambda^{(k)}\right) > 0$ **then**
13 $\tilde{\lambda}^{(k+1)} := \lambda^{(k+1)}$
14 $\alpha_{k+1} := 1$
15 **end**
16 $k := k + 1$
17 **end**
Output : $\lambda^{(k)}$

Convergence: The authors show that for strongly convex quadratic functions (optimal) linear convergence can be attained. In other words, the optimal linear

convergence is not affected by the restart mechanism. For our problem, we expect optimal sublinear convergence rate.

Communication: The heuristic restart procedure has to be implemented with centralized coordination, since we have to check the sign of an inner product of two vectors in Step 12. However, the gradient computation may take place in parallel in Line 5. Again, the knowledge of the Lipschitz constant L is necessary in all subproblems.

5.6 Methods for non-strongly convex QPs

In this section, we consider methods for non-strongly convex problems. We restate the considered graph-based distributed QP.

$$\min_{x} \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{5.28a}$$

$$\text{s.t. } A_{i,j} x_i = B_{i,j} x_j \quad (i,j) \in E \tag{5.28b}$$

$$x_l \in \mathcal{X}_l, \quad l = 1, \dots, M. \tag{5.28c}$$

For each $i \in \{1, \dots, M\}$, $H_i \succeq 0$ holds. The two methods presented here have their origin in the method of multipliers that was initially suggested by Hestenes [75] and make use of the (partial) *augmented Lagrangian function* defined as

$$L_\rho(x, \lambda) := \left(\sum_{i=1}^{M} \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \right) + \sum_{(j,k) \in E} \lambda_{j,k}^T \left(A_{j,k} x_j + B_{j,k} x_k \right) \tag{5.29}$$
$$+ \frac{\rho}{2} \sum_{(j,k) \in E} \left\| A_{j,k} x_j - B_{j,k} x_k \right\|_2^2,$$

where ρ is often called the *penalty parameter*. This function (compared to the partial Lagrangian function defined in (5.14)) has an extra quadratic term that introduces curvature to the quadratic subproblems, which results in a differentiable dual function. The corresponding dual function $d_\rho(\lambda)$ is concave and defined by

$$d_\rho(\lambda) := \min_{x \in \mathcal{X}} L_\rho(x, \lambda). \tag{5.30}$$

The *method of multipliers* (MOM) [14] in the first step minimizes the augmented Lagrangian function, and in the second step updates the dual variables with a

steepest ascent method. More precisely, it follows

$$x^{(k+1)} := \arg\min_{x\in\mathcal{X}} L_\rho(x, \lambda^{(k)}) \tag{5.31a}$$

$$\lambda^{(k+1)} := \lambda^{(k)} + \rho\nabla_\lambda L_\rho(x^{(k)}, \lambda^{(k)}) \tag{5.31b}$$

$$\rho := \gamma \cdot \rho, \tag{5.31c}$$

where $\gamma > 1$. It has been shown that MOM converges to a solution even for non-convex problems with at least linear rate assuming that the penalty parameter is big enough and some constraint qualification hold. The main drawback of the augmented Lagrangian is that the evaluation of $d_\rho(\lambda)$ cannot be paralellized anymore due to the added quadratic term. This limitation can be overcome by utilizing a nonlinear Gauss-Seidel algorithm for solving the subproblem in (5.31a) assuming convexity. An extreme version of this approach is, when only one step of the nonlinear Gauss-Seidel algorithm is taken followed by the update of dual multipliers and penalty parameter, is called the alternating direction of multipliers. We consider this scheme in the sequel.

5.6.1 Alternating direction method of multipliers

The *alternating direction method of multipliers* (ADMM) is an increasingly popular method originating from [62, 50] and employing the augmented Lagrangian. A recent summary was published in [23]. In order to make the augmented Lagrangian function separable, we introduce slack variables $a_{i,j} \in \mathbb{R}^{v_{i,j}}$, $b_{i,j} \in \mathbb{R}^{v_{i,j}}$, $(i,j) \in E$ and solve an equivalent problem of the form

$$\min_{x,a,b} \sum_{i=1}^{M} \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \tag{5.32a}$$

$$\text{s.t. } A_{j,k}x_j - a_{j,k} = 0 \tag{5.32b}$$

$$B_{j,k}x_k - b_{j,k} = 0 \tag{5.32c}$$

$$x_i \in \mathcal{X}_i \ , \quad a_{j,k} = b_{j,k} \tag{5.32d}$$

$$i = 1, \ldots, M, \quad (j,k) \in E. \tag{5.32e}$$

The corresponding augmented Lagrangian function reads as

$$L_\rho(x,a,b,\lambda,\mu) := \sum_{i=1}^{M} L_{\rho,i}(x_i,a,b,\lambda,\mu) := \sum_{i=1}^{M} \Bigg(x_i^T H_i x_i + c_i^T x_i$$

$$+ \sum_{(i,k)\in E} \lambda_{i,k}^T \left(A_{i,k}x_i - a_{i,k}\right) + \sum_{(k,i)\in E} \mu_{k,i}^T \left(B_{k,i}x_i - b_{k,i}\right)$$

$$+ \frac{\rho}{2} \sum_{(i,k)\in E} \|A_{i,k}x_i - a_{i,k}\|_2^2 + \frac{\rho}{2} \sum_{(k,i)\in E} \|B_{k,i}x_i - b_{k,i}\|_2^2 \Bigg), \tag{5.33}$$

where a, b, λ, μ collect variables $a_{i,j}, b_{i,j}, \lambda_{i,j}, \mu_{i,j}$, $(i,j) \in E$, respectively. Moreover, $\lambda_{i,j}$ and $\mu_{i,j}$ are the Lagrange multipliers corresponding to (5.32b) and (5.32c), respectively. Note that the augmented Lagrangian function in (5.33) is separable along variables x_i, $i = 1, \ldots, M$. However, the variables a and b couple them together. Now, we divide the primal variables into two subsets $\{x\}$ and $\{a, b\}$ and carry out one step of the nonlinear Gauss-Seidel algorithm on these subsets of variables. More precisely, in each iteration, an update of primal variables x, a, b and dual variables λ, μ is carried out as

$$x_i^{(k+1)} := \arg\min_{x_i \in \mathcal{X}_i} L_{\rho,i}\left(x_i, a^{(k)}, b^{(k)}, \lambda^{(k)}, \mu^{(k)}\right) \tag{5.34a}$$

$$\begin{pmatrix} a \\ b \end{pmatrix}^{(k+1)} := \begin{pmatrix} \arg\min\limits_{a,b} & L_\rho\left(x^{(k+1)}, a, b, \lambda^{(k)}, \mu^{(k)}\right) \\ \text{s.t.} & a_{i,j} = b_{i,j}, \quad (i,j) \in E \end{pmatrix} \tag{5.34b}$$

$$\lambda_{i,j}^{(k+1)} := \lambda_{i,j}^{(k)} + \rho\left(A_{i,j}x_i^{(k+1)} - a_{i,j}^{(k+1)}\right) \tag{5.34c}$$

$$\mu_{i,j}^{(k+1)} := \mu_{i,j}^{(k)} + \rho\left(B_{i,j}x_j^{(k+1)} - b_{i,j}^{(k+1)}\right) \tag{5.34d}$$

Note that since $L_\rho(\cdot)$ is separable, (5.34a) is essentially solving M QPs in parallel. In (5.34b), we have to solve an equality constrained QP over variables a and b. Note that this QP is separable with respect to each coupling equation, i.e. for each $(i,j) \in E$ after neglecting constant terms in the objective, we have to solve

$$\min_{a_{i,j}, b_{i,j}} \quad \frac{1}{2} a_{i,j}^T (\rho \mathbf{I}) a_{i,j} - \left(\rho x_i^T A_{i,j}^T + \lambda_{i,j}^T\right) a_{i,j} +$$

$$\frac{1}{2} b_{i,j}^T (\rho \mathbf{I}) b_{i,j} - \left(\rho x_j^T B_{i,j}^T + \mu_{i,j}^T\right) b_{i,j} \tag{5.35a}$$

$$\text{s.t. } a_{i,j} = b_{i,j} \tag{5.35b}$$

The solution can be found explicitly by solving the KKT optimality conditions of form

$$\begin{bmatrix} \rho\mathbf{I} & 0 & \mathbf{I} \\ 0 & \rho\mathbf{I} & -\mathbf{I} \\ \mathbf{I} & -\mathbf{I} & 0 \end{bmatrix} \begin{bmatrix} a_{i,j}^* \\ b_{i,j}^* \\ \nu_{i,j}^* \end{bmatrix} = \begin{bmatrix} \rho A_{i,j} x_i + \lambda_{i,j} \\ \rho B_{i,j} x_j + \mu_{i,j} \\ 0 \end{bmatrix}, \tag{5.36}$$

where $\nu_{i,j}$ denotes the Lagrange multipliers corresponding to (5.35b). The optimal primal variables are then computed by

$$a_{i,j}^{(k+1)} = b_{i,j}^{(k+1)} = \frac{A_{i,j} x_i^{(k+1)} + B_{i,j} x_j^{(k+1)}}{2} + \frac{\lambda_{i,j}^{(k)} + \mu_{i,j}^{(k)}}{2\rho} \tag{5.37a}$$

We summarize ADMM in Algorithm 5.7 for distributed execution on node i. Observe that after the first iteration, $a_{i,j} = b_{i,j}$ holds and thus one of them may be eliminated.

Convergence: The convergence rate of ADMM for convex problems in the worst case was shown to be $O\left(\frac{1}{k}\right)$ in [73]. More precisely, assuming that at least one of the subproblems is solved exactly and the penalty parameter is sufficiently big, we have

$$\|w^{(k)} - w^{(k+1)}\|_H^2 \leq \frac{1}{k+1} \|w^{(0)} - w^*\|_H^2, \tag{5.38}$$

with $w = col(x, a, b, \lambda, \mu)$ and some matrix $H \succeq 0$. Moreover, if (5.32a) is strongly convex global linear convergence is guaranteed [37], i.e.

$$\|w^{(k+1)} - w^*\|_G^2 \leq \frac{1}{1+\delta} \|w^{(k)} - w^*\|_G^2, \tag{5.39}$$

where $\delta > 0$ and for some $G \succeq 0$. In practise, ADMM is often sensitive to the choice of the penalty parameter ρ. A too low penalty parameter might lead to divergence, while a too high one may cause numerical difficulties.

Communication: ADMM may be implemented in a fully distributed fashion. Communication is only necessary when the actual primal iterates x_i are sent to direct neighbours and vice versa.

Algorithm 5.7: Alternating Direction Method of Multipliers (ADMM)

Input : $x_i^{(0)},\ a_{i,j}^{(0)},\ b_{i,j}^{(0)}, \lambda_{i,j}^{(0)},\ \mu_{i,j}^{(0)},\ \rho$

$k := 0$

while *no convergence* **do**

  $x_i^{(k+1)} := \arg\min_{x_i \in \mathcal{X}_i} L_{\rho,i}(x_i, a^{(k)}, b^{(k)}, \lambda^{(k)}, \mu^{(k)})$

  foreach $(i,j) \in E$ **do**

    Send $x_i^{(k+1)}$ to node j

    Receive $x_j^{(k+1)}$ from node j

    $a_{i,j}^{k+1} := b_{i,j}^{k+1} := \frac{1}{2}\left(A_{i,j}x_i^{(k+1)} + B_{i,j}x_j^{(k+1)}\right) + \frac{1}{2\rho}\left(\lambda_{i,j}^{(k)} + \mu_{i,j}^{(k)}\right)$

    $\lambda_{i,j}^{(k+1)} := \lambda_{i,j}^{(k)} + \rho\left(A_{i,j}x_i^{(k+1)} - a_{i,j}^{(k+1)}\right)$

    $\mu_{i,j}^{(k+1)} := \mu_{i,j}^{(k)} + \rho\left(B_{i,j}x_j^{(k+1)} - b_{i,j}^{(k+1)}\right)$

  end

  foreach $(j,i) \in E$ **do**

    Receive $x_j^{(k+1)}$ from node j

    Send $x_i^{(k+1)}$ to node j

    $a_{j,i}^{k+1} := b_{j,i}^{k+1} := \frac{1}{2}\left(A_{j,i}x_j^{(k+1)} + B_{j,i}x_i^{(k+1)}\right) + \frac{1}{2\rho}\left(\lambda_{j,i}^{(k)} + \mu_{j,i}^{(k)}\right)$

    $\lambda_{j,i}^{(k+1)} := \lambda_{j,i}^{(k)} + \rho\left(A_{j,i}x_j^{(k+1)} - a_{j,i}^{(k+1)}\right)$

    $\mu_{j,i}^{(k+1)} := \mu_{j,i}^{(k)} + \rho\left(B_{j,i}x_i^{(k+1)} - b_{j,i}^{(k+1)}\right)$

  end

  $k := k + 1$

end

Output : $x^{(k)}, z^{(k)}, \lambda^{(k)}$

5.6.2 Inexact Uzawa method

The *inexact Uzawa method* [147] employs the augmented Lagrangian function (5.33) also and is closely related to ADMM. However, the augmented Lagrangian is extended with an extra term, which penalizes big changes in the variable x_i. With this extra term the quadratic subproblems become strictly convex. The method

proceeds as

$$x_i^{(k+1)} := \arg\min_{x_i \in \mathcal{X}_i} L_{\rho,i}\left(x_i, a^{(k)}, b^{(k)}, \lambda^{(k)}, \mu^{(k)}\right) + \frac{1}{2}\|x_i - x_i^{(k)}\|_{D_i}^2 \quad (5.40a)$$

$$\begin{pmatrix} a \\ b \end{pmatrix}^{(k+1)} := \begin{pmatrix} \arg\min\limits_{a,b} & L_\rho\left(x^{(k+1)}, a, b, \lambda^{(k)}, \mu^{(k)}\right) \\ \text{s.t.} & a_{i,j} = b_{i,j}, \quad (i,j) \in E \end{pmatrix} \quad (5.40b)$$

$$\lambda_{i,j}^{(k+1)} := \lambda_{i,j}^{(k)} + \rho\left(A_{i,j}x_i^{(k+1)} - a_{i,j}^{(k+1)}\right) \quad (5.40c)$$

$$\mu_{i,j}^{(k+1)} := \mu_{i,j}^{(k)} + \rho\left(B_{i,j}x_j^{(k+1)} - b_{i,j}^{(k+1)}\right) \quad (5.40d)$$

where $D_i \succeq 0$. Note that once $D = 0$, the resulting algorithm is ADMM. Similarly to ADMM, in (5.40a) we have to solve M QPs in parallel if D_i is chosen to be block-diagonal, which we assume. However, if $D_i \succ 0$ holds, the QP subproblems become strictly convex, which is not necessarily the case with ADMM. The QP in (5.40b) can be solved again explicitly with local communications.

Convergence: The convergence properties of the Inexact Uzawa method are the same as the ones of ADMM, we refer to [73] for more details.

Communication: The communication scheme can be again distributed. The reasoning presented at ADMM holds for IUM as well.

We summarize the discussed methods in Table 5.6.1. In the next chapter, we present experimental results on the practical convergence behaviour of the discussed methods.

Algorithm 5.8: Inexact Uzawa Method (IUM)

Input : $x_i^{(0)},\ a_{i,j}^{(0)},\ b_{i,j}^{(0)}, \lambda_{i,j}^{(0)},\ \mu_{i,j}^{(0)},\ \rho$

$k := 0$

while *no convergence* **do**

  $x_i^{(k+1)} := \arg\min_{x_i \in \mathcal{X}_i} L_{\rho,i}(x_i, a^{(k)}, b^{(k)}, \lambda^{(k)}, \mu^{(k)}) + \frac{1}{2}\|x_i - x_i^{(k)}\|_D^2$

  foreach $(i,j) \in E$ **do**

    Send $x_i^{(k+1)}$ to node j

    Receive $x_j^{(k+1)}$ from node j

    $a_{i,j}^{k+1} := b_{i,j}^{k+1} := \frac{1}{2}\left(A_{i,j}x_i^{(k+1)} + B_{i,j}x_j^{(k+1)}\right) + \frac{1}{2\rho}\left(\lambda_{i,j}^{(k)} + \mu_{i,j}^{(k)}\right)$

    $\lambda_{i,j}^{(k+1)} := \lambda_{i,j}^{(k)} + \rho\left(A_{i,j}x_i^{(k+1)} - a_{i,j}^{(k+1)}\right)$

    $\mu_{i,j}^{(k+1)} := \mu_{i,j}^{(k)} + \rho\left(B_{i,j}x_j^{(k+1)} - b_{i,j}^{(k+1)}\right)$

  end

  foreach $(j,i) \in E$ **do**

    Receive $x_j^{(k+1)}$ from node j

    Send $x_i^{(k+1)}$ to node j

    $a_{j,i}^{k+1} := b_{j,i}^{k+1} := \frac{1}{2}\left(A_{j,i}x_j^{(k+1)} + B_{j,i}x_i^{(k+1)}\right) + \frac{1}{2\rho}\left(\lambda_{j,i}^{(k)} + \mu_{j,i}^{(k)}\right)$

    $\lambda_{j,i}^{(k+1)} := \lambda_{j,i}^{(k)} + \rho\left(A_{j,i}x_j^{(k+1)} - a_{j,i}^{(k+1)}\right)$

    $\mu_{j,i}^{(k+1)} := \mu_{j,i}^{(k)} + \rho\left(B_{j,i}x_i^{(k+1)} - b_{j,i}^{(k+1)}\right)$

  end

  $k := k + 1$

end

Output : $x^{(k)}, z^{(k)}, \lambda^{(k)}$

Table 5.6.1: List of implemented methods. The meaning of the columns are "Abbr": short abbreviation, "Name": a longer name, "Hessian": definiteness of Hessian, "Comm": nature of communication, "Lip.": if the knowledge of the Lipschitz constant of the dual gradient function is necessary.

Abbr.	Name	Hessian	Comm.	Lip.
GM	Gradient method	$H_i \succ 0$	local	Yes
GBB	Global Barzilai-Borwein method	$H_i \succ 0$	global	No
FG	Fast gradient method	$H_i \succ 0$	local	Yes
FG-AL	Fast gradient method with adaptive Lipschitz	$H_i \succ 0$	global	No
FG-AR	Fast gradient method with adaptive restart	$H_i \succ 0$	local	Yes
ADMM	Alternating direction of multipliers method	$H_i \succeq 0$	local	No
IUM	Inexact Uzawa method	$H_i \succeq 0$	local	No

Chapter 6

Numerical performance of distributed QP methods

This chapter is concerned with the performance evaluation of different distributed QP methods discussed in the previous chapter. In Section 6.1, an extensive performance evaluation of dual decomposition methods using only first-order derivatives is given. In Section 6.2, we introduce a set of benchmark quadratic programming problems for distributed optimization. Here, we present results published in [84, 67].

6.1 Performance evaluation

In this section, we present the practical convergence properties of methods considered in the previous sections. We report our comparisons based on the number of QP solutions necessary to fulfill a well-defined convergence criterion. Let $\mathcal{E}^{(k)} := \| \sum_{(i,j)\in E} A_{i,j} x_i^{(k)} - B_{i,j} x_j^{(k)} \|_\infty$, $\epsilon > 0$. We consider the primal infeasibility of the coupling constraints

$$\frac{\mathcal{E}^{(k)}}{\max\left(1, \mathcal{E}^{(0)}\right)} \leq \epsilon, \tag{6.1}$$

as a measure to terminate the benchmarked algorithms. Since dual decomposition methods relax the coupling constraints, the measure of their violation can be used as a sufficient condition. We always assume that the actual primal iterate is obtained reliably depending on the actual dual iterate. Once the optimal dual iterate is found, up to some accuracy, the actual primal iterate is an optimal solution of the non-dualized problem.

In our performance evaluation, we follow the methodology of *performance profiles* proposed in [41]. Given a set of problems $\mathcal{P}$ with $|\mathcal{P}| = n_p$ elements and a set of methods $\mathcal{M}$ with $|\mathcal{M}| = n_m$ elements, we define the quantity $K_{p,m}$ as the number of QP solutions to reach convergence in terms of (6.1) with problem $p \in \mathcal{P}$ and method $m \in \mathcal{M}$. The *performance ratio* $r_{p,m}$ of a fixed problem p and a fixed method m shows how m performs on problem p when compared to the best performance on the same problem, i.e.

$$r_{p,m} := \frac{K_{p,m}}{\min\{K_{p,\hat{m}} | \hat{m} \in \mathcal{M}\}} \geq 1. \tag{6.2}$$

We assume that for all $p \in \mathcal{P}$ and $m \in \mathcal{M}$ there is $K_{\max} \geq K_{p,m}$ and $K_{\max} = K_{p,m}$ holds if and only if method m cannot solve problem p. The value of $K_{\max}$ has no effect on the performance evaluation. We define the function $\rho_m(\tau) : \mathbb{R} \to [0, 1]$ as

$$\rho_m(\tau) := \frac{1}{n_p} \left| \{ p \in \mathcal{P} | r_{p,m} \leq \tau \} \right| \tag{6.3}$$

The function $\rho_m(\tau)$ shows the empirical performance ratio being smaller than τ for method m on the problem set $\mathcal{P}$. In other words, the ratio of problems solved by method m in not more iterations than τ times the iterations of the best method on the same problem.

The experiments were carried out with our own open-source solver called PyDQP, which is presented in Appendix B. We have used test problems from the test suite presented in the next section.

Comparison based upon inaccurate solutions

First-order methods often have difficulties with converging to a highly accurate solution due to the lack of quadratic convergence. For this reason, we evaluate our comparison in two different scenarios. In the first scenario, we request for a solution of accuracy $\epsilon = 10^{-3}$ only.

We have plotted $\rho_m(\tau)$ on $\tau \in [0, 10]$ for all discussed methods in Figure 6.1 to discuss their the overall quality. From this plot, we can conlude that IUM has the most wins. It has a performance measure of 61.56 % to be the best solver. In other words, in 61.56 % of all test cases, IUM managed to solve the problem in the fewest dual iterations. The second best is ADMM with 44.44 %. The third best is FG-AR, which is the winner in 38.89 % of the cases. With FG this number is 11.11 %. With GM, FG-AL and GBB, the performance measure is 0, meaning that they have never performed the best on any problems. If we chose being within a factor of $\tau = 3$ of the best solver then FG-AR becomes competitive and solves the 55.56 % of the problems. It can also be seen that methods FG-AR, GBB, FG and

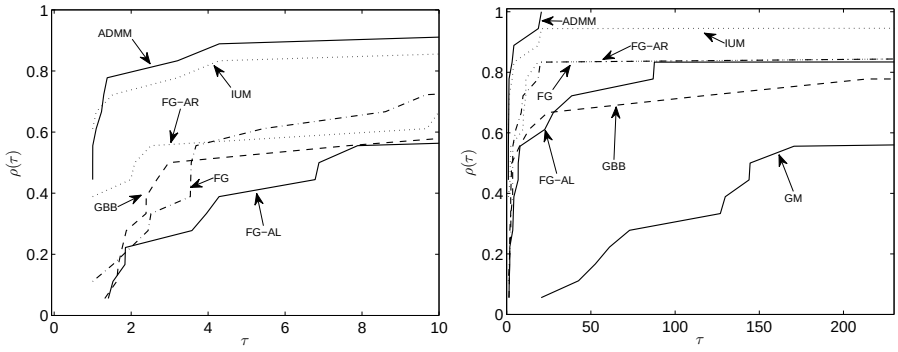


Figure 6.1: The performance measures of different methods on $\tau \in [0, 10]$ (left) and on $\tau \in [0, 230]$ (right) to reach a tolerance of $\epsilon = 10^{-3}$.

FG-AR solve above 50 % of the problems at most with a factor 4 of the best solver. In other words, allowing 4 times more QP solutions than the best method results in solving the 50 % of the problems with these methods. In the interval $\tau \in [0, 10]$ GM cannot be even seen. This demonstrates the poor performance of GM. If we consider $\tau = 21$, GM can solve only 5.56 % of the problems.

Comparison based upon highly accurate solutions

We expect that due to the sublinear convergence behaviour of the discussed approaches, a highly accurate solution requires many more iterations. Figure 6.2 shows the performance comparison where we have set a high accuracy of $\epsilon = 10^{-5}$ in the stopping criterion. In this scenario, ADMM is the best solver with performance measure 50 %. IUM and FG-AR have solved 27.78 % of the problems as best methods. This is a remarkable decrease compared to the performance measure with low accurate solutions. FG-AL is the best method with a measure of 5.56 %. FG-AL, GM and GBB have never been the best methods on our test suite. If we regard a factor $\tau > 1$ of the best solver, IUM becomes competitive very quickly and also FG-AR performs fairly well.

In order to obtain better insight into how these methods behave, we have plotted the primal infeasibility defined in (6.1) versus the iteration counter on a log-log scale for some problems. In Figure 6.3 the evolution of the primal infeasibility is depicted on problem `AUG2DC-5`.

It is clear that for this problem FG-AR is the winner, while GM needs almost two order of magnitude more iterations to reach the same accuracy. ADMM and IUM

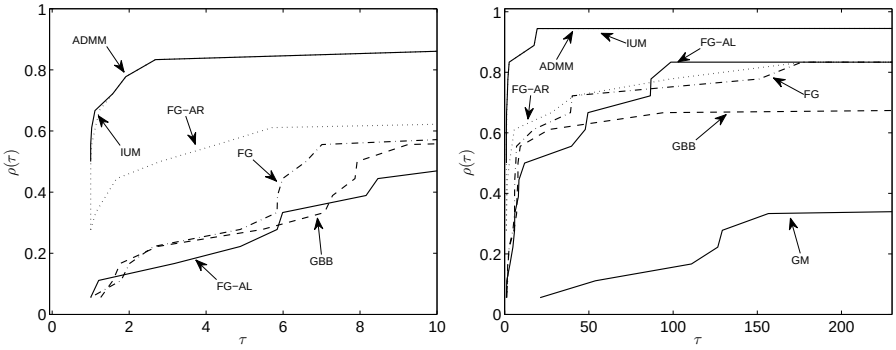


Figure 6.2: The permormance measures of different methods on $\tau \in [0, 10]$ (left) and on $\tau \in [0, 230]$ (right) to reach a tolerance of $\epsilon = 10^{-5}$.

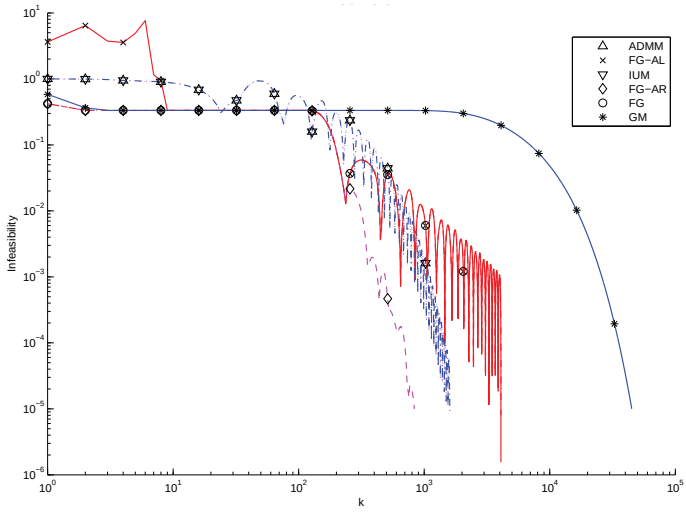


Figure 6.3: The primal infeasibility versus the iteration number of different methods on problem `AUG2DC-5`.

are also competitive.

In Figure 6.4, the same converge measure is shown on problem `DSL1-PT-8`. Here, ADMM and IUM coincide, the fast gradient methods are competitive and GM performs the worst. It is interesting to notice that all the methods seem to have difficulties at a certain tolerance value and require many iterations to obtain a

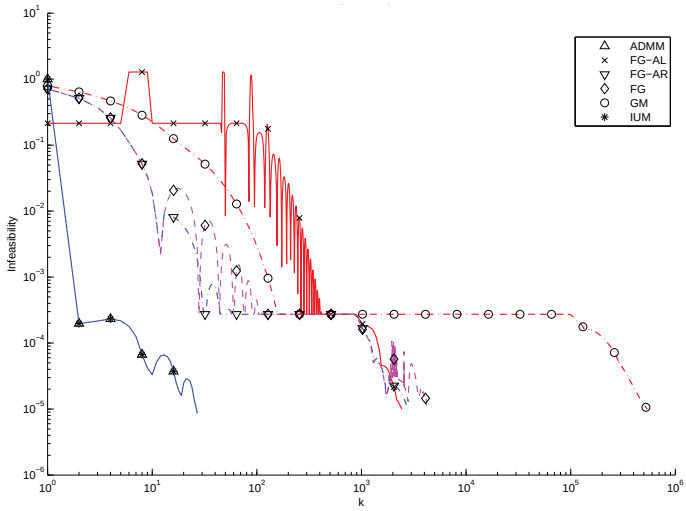


Figure 6.4: The primal infeasibility versus the iteration number of different methods on problem `DSL1-PT-8`.

highly accurate solution. Figure 6.5 depicts the primal infeasibility on problem `SMOKE-AG-66`.

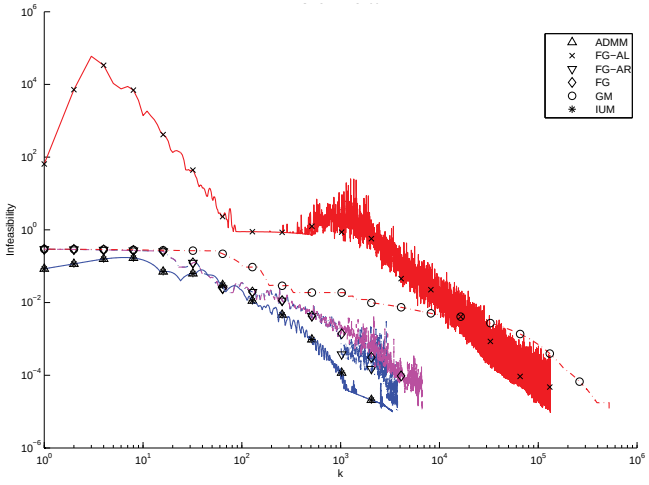


Figure 6.5: The primal infeasibility versus the iteration number of different methods on problem `SMOKE-AG-66`.

Table 6.1.1: Comparison of the computation time of distributed and centralized solvers. Each number is a ratio of one iteration of a distributed QP method and the solution time with a centralized interior point solver.

Problem	GM	FG	FG-AL	FG-AR	GBB	ADMM	IUM
AUG2DC-5	1.30	1.31	1.32	1.38	1.38	5.70	5.57
AUG2DCQP-5	0.20	0.23	0.20	0.21	0.21	1.69	1.71
CONT-100-2	0.13	0.14	0.13	0.14	0.14	5.45	5.52
CONT-200-8	0.08	0.09	0.10	0.08	0.08	1.21	1.48
AUG2D-reg-5	3.07	2.96	3.02	3.05	3.22	11.41	12.05
AUG2DQP-reg-5	0.42	0.46	0.42	0.44	0.42	2.82	2.80
CONT-101-reg-2	0.35	0.33	0.36	0.35	0.33	7.30	7.40
DTOC3-reg-5	3.53	3.30	3.43	3.25	3.48	6.24	5.60
HPV-sys-reg-8	0.05	0.05	0.04	0.04	0.05	1.05	1.06
JG1-8	0.52	0.27	0.15	0.28	0.39	1.11	1.66
JG2-4	0.12	0.14	0.05	0.12	0.14	1.55	1.45
JG3-11	0.09	0.11	0.10	0.11	0.12	0.98	1.02
UBH1-reg-18	0.86	0.69	1.07	1.26	1.30	2.42	2.43
DSL1-PT-8	5.68	5.52	5.27	5.59	5.51	1385.16	1246.13
DSL2-PT-12	3.14	3.19	3.24	3.19	3.12	2747.01	2602.32
SMOKE-AGd-24	0.01	0.01	0.01	0.01	0.01	0.46	0.52
SMOKE-AGd-66	0.01	0.01	0.01	0.01	0.01	0.26	0.26
2nd-order-chain	0.07	0.07	0.07	0.07	0.08	0.24	0.25

This time ADMM and IUM are the winners, FG and FG-AR are competitive. FG-AL had some difficulties in the beginning due to its adaptivity. GM seems to be stable, but very slow.

Comparison of runtime with centralized solution

As we have seen before, the number of necessary QP solutions are often in the order of 10^3 or even more. One consequence is that the runtime to solve these problems in a distributed fashion is very large. We have solved all problems also centrally with `CPLEX` [77], a sparsity exploiting interior-point algorithm, to get an idea how competitive the distributed methods can be. This approach turns out to be extremely efficient. We have managed to solve each problem in maximum 12 seconds. CPLEX uses thread-based parallel linear algebra within the interior-point algorithm.

We have carried out our experiments on an `x86-64` computer with 24 Intel Xeon 2.5 GHz cores equipped with 64377 MB shared memory. In Table 6.1.1, the ratio between the runtime of 1 iteration of each distributed method and the solution time of the centralized solver can be found. The smaller ratio means less runtime per iteration. For instance, with GM on problem SMOKE-AGd-24 this ratio is 0.01 meaning that 1 iteration of GM takes 1 % of the time that CPLEX needs to solve the complete problem. Another example is ADMM on DSL2-PT-12, where

this ratio is 2747.01. Essentially, each iteration of ADMM costs about 2700 times more than solving the whole problem centrally. We conclude that all distributed approaches considered here are not competitive with centralized structure-exploiting solvers.

6.2 A benchmark suite for distributed Quadratic Programming

This section introduces a freely available benchmark suite [1] including several large scale quadratic programs for testing and research purposes. Each problem is stored in the sparse matrix-based form, which we restate here.

$$\min_{x} \sum_{i=1}^{M} x_i^T H_i x_i + c_i^T x_i \tag{6.4a}$$

$$\text{s.t.} \sum_{i=1}^{M} A_i x_i = b \tag{6.4b}$$

$$A_i^{\text{eq}} x_i = b_i^{\text{eq}} \qquad i = 1, \ldots, M \tag{6.4c}$$

$$x_i^{\text{lb}} \leq x_i \leq x_i^{\text{ub}} \qquad i = 1, \ldots, M \tag{6.4d}$$

Each benchmark is stored in one or many `.mat` files depending on their size. These files contain the following fields.

- `H`: A list of the local primal Hessian matrices of length M.
- `c`: A list of the local linear terms of length M.
- `A`: A list of sparse coupling matrices of length M.
- `b`: The right-hand side vector of the coupling constraints.
- `Aeq`: A list of local equation matrices of length M.
- `beq`: A list of local equation right-hand sides of length M.
- `lbx`: A list of local lower bounds of length M.
- `ubx`: A list of local upper bounds of length M.

We have chosen the binary format of `.mat` files since it is accessible from many programming languages such as Matlab, C/C++ [1], and Python [2] .

Several QPs were originally not positive definite, especially those originating from an NLP, which were made convex by adding extra curvature by a diagonal regularization of form

$$\hat{H}_i := H_i + \sigma \mathbf{I} \tag{6.5}$$

with a suitable $\sigma > 0$.

List of benchmarks

In the sequel, the full list of benchmark problems is given together with their dimension and mentioning their origin. The benchmarks are classified into three different groups, see Table 6.2.1.

Group 1

The first group of benchmarks include problems from the convex quadratic programming test suite of Maros and Mészáros [98]. We can find problems `AUG2DC-5`, `AUG2DC-400`, `AUG2DCQP-5`, `AUG2DCQP-400`, `CONT-100-2`, `CONT-100-250`, `CONT-200-8`, `CONT-200-300`, `AUG2D-5`, `AUG2D-400`, `AUG2DQP-5`, `AUG2DQP-400`, `CONT-101-2`, `CONT-101-250`, `CONT-300-9`, `CONT-300-120`, `DTOC3-5`, `DTOC3-150`, `UBH1-18` and `UBH1-100`, which are all academic and have block-diagonal Hessian matrix. The separation of variables took place artificially and has no underlying meaning.

Group 2

The problems in this group are application oriented, partitioned artificially, however. It includes `OBAV-JG1-8`, `OBAV-JG2-4`, `OBAV-JG3-11`, `DSL1-PT-8`, `DSL1-PT-8`, `DSL1-PT-222`, `DSL2-PT-12`, `DSL2-PT-392`, `RS-MB-50`, and `RS-MB-100`.

The first three benchmarks originate from a nonlinear robust optimal control problem for obstacle avoidance [3] .

[1] See `http://www.mathworks.nl/help/matlab/matlab_external/custom-applications-to-read-and-write-mat-files.html` for more detail.

[2] See `http://docs.scipy.org/doc/scipy/reference/io.html` for more detail.

[3] See the talk of J. Gillis at *BeNeLux Meeting on Systems and Control* in Heijen, entitled "Lyapunov based design of robust linear-feedback for time-optimal periodic quad- copter motion", 2012

Table 6.2.1: List of benchmark problems. n: number of variables, M: number of subproblems, #eq: number of equalities, #ineq: number of inequalities, σ: regularization parameter, .

Name	n	M	**#eq**	**#ineq**	σ	Citation
			Group 1			
AUG2DC-5	20200	5	10000	0	0	[98]
AUG2DC-400	20200	400	10000	0	0	[98]
AUG2DCQP-5	20200	5	10000	40400	0	[98]
AUG2DCQP-400	20200	400	10000	40400	0	[98]
CONT-100-2	10197	2	9801	20394	0	[98]
CONT-100-250	10197	250	9801	20394	0	[98]
CONT-200-8	40397	8	39601	80794	0	[98]
CONT-200-300	40397	300	39601	80794	0	[98]
AUG2D-5	20200	5	10000	0	10^{-1}	[98]
AUG2D-400	20200	400	10000	0	10^{-1}	[98]
AUG2DQP-5	20200	5	10000	20200	10^{+0}	[98]
AUG2DQP-400	20200	400	10000	20200	10^{+0}	[98]
CONT-101-2	10197	2	10098	20394	10^{-2}	[98]
CONT-101-250	10197	250	10098	20394	10^{-2}	[98]
CONT-201-4	40397	4	40198	80794	1E-2	[98]
CONT-201-150	40397	150	40198	80794	1E-2	[98]
CONT-300-9	90597	9	90298	181194	10^{-3}	[98]
CONT-300-120	90597	120	90298	181194	10^{-3}	[98]
DTOC3-5	14999	5	9998	2	10^{-5}	[98]
DTOC3-150	14999	150	9998	2	10^{-5}	[98]
UBH1-18	18009	18	12000	12030	10^{-4}	[98]
UBH1-100	18009	100	12000	12030	10^{-4}	[98]
			Group 2			
OBAV-JG1-8	1288	8	1205	87	10^{-9}	[84]
OBAV-JG2-4	5788	4	5701	95	10^{-9}	[84]
OBAV-JG3-11	5789	11	5701	97	10^{-9}	[84]
DSL1-PT-8	10976	8	1569	21952	0.5	[84]
DSL1-PT-222	10976	222	1569	21952	0.5	[84]
DSL2-PT-12	41292	12	6883	82584	0.5	[84]
DSL2-PT-392	41292	392	6883	82584	0.5	[84]
RS-MB-50	9749	50	19409	9734	10	[25]
RS-MB-100	48314	100	48254	96314	0	[25]
			Group 3			
OPF-EK-14bus-21	7644	21	1456	7770	0.1	[100]
HPV-sys-8	27812	8	26976	2518	10^{-3}	[126]
HPV-full-384	27812	384	26976	2518	10^{-3}	[126]
2nd-ord-ch	117760	128	79360	235520	0	[30]
CONNMASS-AK-320	1540	320	1170	2340	10^{-4}	[85]
CONNMASS-AK-1050	5100	1050	3960	7920	10^{-4}	[85]
SMOKEd-AK-201	151250	201	75440	148091	10^{-4}	[83]
SMOKEd-AK-1351	1000180	1351	509220	1021969	10^{-4}	[83]
SMOKEc-AG-16	41850	16	20250	43200	10^{-4}	[83]
SMOKEd-AG-24	5730	24	2826	5808	10^{-4}	[83]
SMOKEd-AG-66	17406	66	9420	15972	10^{-4}	[83]

The benchmarks whose name start with DSL are instances of a problem related to sparse vectoring resource allocation for improving the performance of digital subscriber line (DSL) broadband access networks [137, 136]. Here, the subproblems summarize the optimization problem of a number of DSL agents.

Benchmarks RS-MB-50, RS-MB-100 originate from an optimal control problem of the river system Demer equipped with reservoirs for flood prevention. We refer to [25, p. 125] for further reading.

Group 3

In the last group of problems with natural partitioning are included thus the problem slices correspond to physical units. Problems OPF-EK-14bus-21, HPV-sys-8, HPV-full-384, 2nd-ord-ch, CONNMASS-AK-320, CONNMASS-AK-1050, SMOKEd-AK-201, SMOKEd-AK-1351, SMOKEc-AG-16, SMOKEd-AG-24 and SMOKEd-AG-66 can be found.

The QP OPF-EK-14bus-21 originates from a nonlinear security constrained power flow optimization problem [100], in which the subproblems correspond to different power production scenarios.

Problems HPV-sys-8, HPV-full-384 originate from a nonlinear optimal control problem of a hydro power plant described in Chapter 3 . The goal is on the one hand to track a prescribed power reference, on the other hand to track the steady state of the system, while respecting operational constraints.

Problems whose name start with SMOKE are different instances of the dynamic estimation problem presented in Chapter 4. Given a set of smoke sensors in a building consisting of connected rooms, based upon the measurements the origin of the smoke is sought for. In particular, SMOKEd-AK-201 is an instance with 10 connected rooms on 20 time intervals, SMOKEd-AK-1351 with 10 rooms on 135 time intervals.

Visit http://pydqp.sourceforge.net/ in order to obtain the benchmarks.

Chapter 7

Dual decomposition with second-order methods

In this chapter, we deal with dual decomposition, but in addition to the first derivative of the dual function we make use of some (inexact) curvature information. We carry out our investigations in the hope of obtaining faster practical convergence than the typically sublinear first-order methods. However, obtaining curvature information comes at a price: we have to introduce global communication operations in order to broadcast information along all nodes. Section 7.1 provides an introduction to dual decomposition with second-order derivatives. Throughout Section 7.2, the proposed second-order method is described and analysed, followed by a numerical example. Section 7.3 discusses a relaxation method with which local inequalities can be treated in a more efficient way and so accelerating the speed of practical convergence. This chapter presents the results published in [85, 86].

7.1 Introduction

Second-order derivatives have been used in the context of dual decomposition in [103]. The authors propose a dual decomposition framework for convex nonlinear programs, in which the linear coupling constraints are dualized and the local inequality constraints are treated by barrier penalties. In such a way, the dual function becomes smooth, which enables us to use a classical Newton method in the dual space.

Employing a non-smooth variant of Newton's method has been proposed to treat

non-smooth root finding problems. In [113], the authors propose to solve QPs arising from MPC with a non-smooth Newton method inside an active set framework. The KKT conditions of the QP are reformulated such that the root of a piecewise affine function has to be found. The Newton system is solved by direct factorization, for which factors from previous iterations are reused. In the line search procedure, a piecewise quadratic function is employed to attain locally quadratic convergence. However, the proposed algorithm is not in a distributed context.

A non-smooth Newton method is applied in [55, 56] for the solution of NLPs originating from optimal control problems. Therein, the KKT system is reformulated as a non-smooth system of nonlinear equations, which is then solved by a non-smooth variant of Newton's method. The author shows that the convergence is locally superlinear, and may be even quadratic under some assumptions, see [117, 116] for a detailed convergence analysis.

In [44], a non-smooth Newton method is proposed to find the optimal dual variables in a dual decomposition approach for the solution of MPC subproblems. The proposed algorithm applies a direct factorization of the Newton system while the dual gradient and Hessian contributions are calculated in parallel. The connection topology of the coupled subproblems is assumed to be chain-topology, i.e. each subproblem is connected to at most two other subproblems.

In our framework, we apply a non-smooth version of Newton's method in order to find the optimal multipliers. However, since we design an algorithm for large scale problems, the Newton system is solved by an iterative method, and thus no formation of large linear systems is necessary. Furthermore, the topology of the connecting subproblems may be arbitrary.

We are concerned with a distributed quadratic program being in the graph-based form, which reads as

$$\min_{x} \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{7.1a}$$

$$\text{s.t. } A_{i,j} x_i = B_{i,j} x_j \quad (i,j) \in E \tag{7.1b}$$

$$x_l \in \mathcal{X}_l, \quad l = 1, \dots, M. \tag{7.1c}$$

In the remainder of this chapter we always assume that $H \succ 0$. The dual function is defined by

$$d(\lambda) := -\min_{x \in \mathcal{X}} L(x, \lambda) = -\sum_{i=1}^{M} \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}) \tag{7.2}$$

Lemma 7.1.1 (curvature of the dual function) *The dual function $d(\lambda)$ is piece-wise twice differentiable.*

Proof *The proof is constructive, we also give an algorithmic description how to calculate the Hessian corresponding to a certain active set of constraints. Depending on λ, different constraints in (7.2) may get active or inactive and to each active set corresponds a well-defined Hessian $\nabla^2 d(\lambda)$. We assume that there exists an $\epsilon > 0$ neighbourhood of a fixed λ in which the active constraint set of inequalities $D_i x_i \leq e_i,\ i = 1, \ldots, M$ does not change. In order to keep the proof simple, we summarize $d(\lambda)$ in the compact form*

$$d(\lambda) = -\begin{pmatrix} \min_x & \frac{1}{2}x^T H x + c^T x + \lambda^T C x \\ \text{s.t.} & Dx \leq e \end{pmatrix}. \tag{7.3}$$

Let $\mathcal{A}$ report the active set of constraints denoted by $D_{\mathcal{A}} x = e_{\mathcal{A}}$. The primal solution $x^*(\lambda)$ of (7.3) can be obtained by solving the linear system

$$\begin{bmatrix} H & -D_{\mathcal{A}} \\ -D_{\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} x \\ \mu \end{bmatrix} = -\begin{bmatrix} c + C^T \lambda \\ -e_{\mathcal{A}} \end{bmatrix}, \tag{7.4}$$

where μ denotes the dual variables corresponding to the active constraints assembled in set $\mathcal{A}$. In order to derive the dual Hessian corresponding to the actual active set $\mathcal{A}$, we exploit the fact that a feasible primal solution $x(\lambda)$ of (7.3) can be written [110, pp. 431-432] as

$$x(\lambda) = Y(D_{\mathcal{A}} Y)^{-1} e_{\mathcal{A}} + N_{\mathcal{A}} \tilde{x}(\lambda) \tag{7.5}$$

where $[Y|N_{\mathcal{A}}] \in \mathbb{R}^{n \times n}$ is a nonsingular matrix, $N_{\mathcal{A}}$ is the nullspace basis of $D_{\mathcal{A}}$ and $\tilde{x}(\lambda)$ is any vector. After plugging this into (7.3), we obtain an unconstrained optimization problem over $\tilde{x}(\lambda)$ of which solution reads as

$$\tilde{x}^*(\lambda) = -(N_{\mathcal{A}}^T H N_{\mathcal{A}})^{-1} \left(N_{\mathcal{A}}^T H Y (D_{\mathcal{A}} Y)^{-1} e_{\mathcal{A}} + N_{\mathcal{A}}^T c + N_{\mathcal{A}}^T C^T \lambda \right) \tag{7.6}$$

Note that $x^*(\lambda)$ is a linear expression of λ and thus $d(\lambda)$ is a quadratic function of λ. The optimal primal solution is obtained by plugging this into (7.5), giving

$$\begin{aligned} x^*(\lambda) =& Y(D_{\mathcal{A}} Y)^{-1} e_{\mathcal{A}} \\ & - N_{\mathcal{A}} (N_{\mathcal{A}}^T H N_{\mathcal{A}})^{-1} \left(N_{\mathcal{A}}^T H Y (D_{\mathcal{A}} Y)^{-1} e_{\mathcal{A}} + N_{\mathcal{A}}^T c + N_{\mathcal{A}}^T C^T \lambda \right) \end{aligned} \tag{7.7}$$

Now, the dual Hessian reads as

$$\nabla^2 d(\lambda) = -C \frac{\partial x^*(\lambda)}{\partial \lambda} = C N_{\mathcal{A}} \left(N_{\mathcal{A}}^T H N_{\mathcal{A}} \right)^{-1} N_{\mathcal{A}}^T C^T. \tag{7.8}$$

■

The dual Hessian may be singular as the following corollary states.

Corollary 7.1.1 (flat directions in the dual function) *The Hessian of the dual function $\nabla^2 d(\lambda)$ may have zero eigenvalue(s).*

Indeed, in view of (7.8), the dual Hessian is singular once $CN_{\mathcal{A}}$ is rank deficient. In the context of the the original problem, this situation occurs once the coupling constraints C together with the active constraints $D_{\mathcal{A}}$ become linearly dependent at intermediate iterations of a dual decomposition method.

In the following simple example, we demonstrate how the dual Hessian may become singular.

Example 7.1.1 *We regard the following quadratic program*

$$\begin{aligned} \min_{x,y} \quad & \frac{1}{2}x^2 + \frac{1}{2}y^2 \\ \text{s.t.} \quad & -x \leq a, \quad y \leq a, \quad x = y, \end{aligned}$$

and its corresponding dual function

$$d(\lambda) = -\begin{pmatrix} \min_{x,y} & \frac{1}{2}x^2 + \frac{1}{2}y^2 + \lambda(x-y) \\ \text{s.t.} & -x \leq a, \quad y \leq a \end{pmatrix}. \tag{7.9}$$

It can be easily verified that

$$d(\lambda) = \begin{cases} 2\lambda a - a^2 & \text{for } \lambda \leq a, \\ \lambda^2 & \text{for } \lambda > a, \end{cases} \tag{7.10}$$

and

$$\nabla^2 d(\lambda) = \begin{cases} 0 & \text{for} \quad \text{for } \lambda \leq a, \\ 2 & \text{for} \quad \text{for } \lambda > a, \end{cases} \tag{7.11}$$

such that the dual Hessian is singular for $\lambda > a$. This situation yields the activation of both inequality constraints in (7.9), hence fixing the residual of the coupling constraints $x - y = 0$. In such a case, the residual cannot be improved by any dual step that retains the current active set, hence yielding the singularity of the dual Hessian. It is worth observing that the optimal λ is zero in this example, such that the singular Hessian occurs when the dual variable is far enough from its optimal value.

■

One possible way to calculate the dual Hessian $\nabla^2 d(\lambda)$ for a fixed λ is to use the result of (7.8). This approach is even cheaper once the underlying QP-solver is an active-set solver of the nullspace type, such as [43]. This way, the reduced Hessian $N_{\mathcal{A}}^T H N_{\mathcal{A}}$ and its decomposition are readily available.

7.2 Dual Newton-CG method

In this section, we describe a novel algorithm that solves distributed QPs with dual decomposition using both first- and second-order derivatives. We develop an inexact Newton method to be used in the dual space coupled with an iterative linear solver. We aim to establish an approach with nicer practical convergence properties than many plain gradient based method. Furthermore, we do not want to rely on the direct solution and transmission of large linear systems, but rather trust matrix-vector products, instead.

The optimal dual variables corresponding to (7.1b) can be found as the solution of the unconstrained dual optimization problem defined by

$$\min_{\lambda} d(\lambda). \tag{7.12}$$

Our method is inspired by [93] and proposes to use curvature information to solve (7.12). The proposed approach consists of three essential ingredients. First, a *non-smooth Newton method* [117, 56] minimizes the approximation of $d(\lambda)$ in the actual iterate $\lambda^{(k)}$ using the gradient and Hessian information. Second, the linear system, i.e., Newton system, is solved inexactly by a conjugate gradient (CG) method [131]. Third, a line-search procedure tunes the stepsize in order to obtain sufficient decrease in the objective function. In the following, we discuss these building blocks in more detail.

7.2.1 Non-smooth Newton method in the dual space

The classical Newton method, when applied to a smooth unconstrained problem, calculates a quadratic model of the original problem of form

$$\Delta\lambda := \arg\min_{p} m(p) := \arg\min_{p} d(\lambda^{(k)}) + \nabla d(\lambda^{(k)})^T p + \frac{1}{2} p^T \nabla^2 d(\lambda^{(k)}) p, \tag{7.13}$$

which yields a search direction in the space of λ. Note that this is equivalent to solving the linear system

$$\nabla^2 d(\lambda^{(k)}) p = -\nabla d(\lambda^{(k)}). \tag{7.14}$$

Since the dual function is only piecewise twice differentiable, we use the non-smooth variant of Newton's method. The non-smooth Newton method minimizes

$$\Delta\lambda := \arg\min_p m(p) := \arg\min_p d(\lambda^{(k)}) + \nabla d(\lambda^{(k)})^T p + \frac{1}{2} p^T H^{(k)} p, \quad (7.15)$$

where $H^{(k)} \in \partial(\nabla d(\lambda))$ is the generalized Jacobian of $\nabla d(\lambda)$ in the sense of Clarke [29]. The dual variables are then updated in terms of

$$\lambda^{(k+1)} := \lambda^{(k)} + t^{(k)} \Delta\lambda, \quad (7.16)$$

where $t^{(k)}$ is a properly chosen stepsize. This is chosen by means of a backtracking line-search procedure with Armijo condition. A candidate stepsize $t^{(k)}$ is accepted once

$$d(\lambda^{(k+1)}) \leq d(\lambda^{(k)}) + \gamma t^{(k)} \nabla d(\lambda^{(k)})^T \Delta\lambda \quad (7.17)$$

holds, otherwise we shorten the candidate stepsize by setting $t^{(k)} := \beta t^{(k)}$ with $0 < \beta < 1$. It was shown in [116] that the non-smooth variant of the Newton method converges locally superlinearly if

1. all $H \in \partial(\nabla d(\lambda^*))$ are non-singular,
2. $\nabla d(\cdot)$ is locally Lipschitz and semi-smooth in λ^*.

Assuming that some constraint qualification, e.g. LICQ, holds at the primal solution, one can show that the optimal dual multipliers are unique, $d(\cdot)$ is strictly convex in λ^*, and that $\nabla^2 d(\lambda^*)$ exists and is non-singular. Moreover, $\nabla d()$ is Lipschitz continuous everywhere, and smooth in λ^*. In fact, once the correct active set is found, the non-smooth Newton method boils down to a classical Newton method and thus we expect locally quadratic convergence in practice.

We summarize the non-smooth Newton method in Algorithm 7.1.

This algorithm consists of three essential parts. First, the calculation of first- and second-order derivatives, see Step 4. Second, the distributed solution of the Newton system, see Step 5, which takes place by iterative linear algebra. Third, the line-search procedure, see Steps 6-7, choses an appropriate stepsize. In the following, we discuss each component in more detail. Throughout the design of the algorithm we aim to establish an approach that

- converges to the centralized solution,

Algorithm 7.1: Dual non-smooth Newton-CG method (prototype)

Input : $\lambda^{(0)}$
$k := 0$
while *no convergence* **do**
  $\lambda = \lambda^{(k)}$
  Calculate $\nabla^2 d(\lambda)$ and $\nabla d(\lambda)$ in a distributed manner.
  Solve linear system $\nabla^2 d(\lambda)\Delta\lambda = -\nabla d(\lambda)$ in a distributed manner.
  Adjust stepsize t, such that $d(\lambda + t\Delta\lambda)$ leads to sufficient decrease.
  $\lambda^{(k+1)} := \lambda + t\Delta\lambda$
  $k := k + 1$
end
Output : $\lambda^{(k)}$

- is faster than the dual gradient in practice.
- does not form large linear systems,
- avoids communication of matrices and prefers communication of vectors locally,
- communicates only scalars globally.

Calculation of derivatives in a distributed manner

The first important ingredient in our algorithm is the calculation of first- and second-order derivatives. The gradient of the dual function is computable in slices, i.e. for each connection $(i,j) \in E$ the corresponding dual gradient slice is given by

$$\frac{\partial d(\lambda)}{\partial \lambda_{i,j}} = -A_{i,j} x_i^*(\lambda_{\mathcal{N}_i}) + B_{i,j} x_j^*(\lambda_{\mathcal{N}_j}). \tag{7.18}$$

This quantity has to be calculated on both nodes i and j by transmitting the local primal solution to the connecting subproblem, i.e. subproblem i sends $-A_{i,j}x_i^*$ to node j, whereas node j sends $B_{i,j}x_j^*(\lambda_{\mathcal{N}_j})$ to the i-th process.

The dual Hessian has well defined structure characterized by the interconnections between the subproblems. We regard the blocks that have to be calculated on node i. In the row of each $\lambda_{i,j}$, $(i,j) \in E$, the following blocks are nonzero.

$$\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda_{k,l}} = \begin{cases} -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{i,j}} + B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{i,j}} & \text{if } (k,l) = (i,j), \\ -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}} & \text{else if } k = i \text{ or } l = i, \\ B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{k,l}} & \text{else if } k = j \text{ or } l = j. \end{cases} \tag{7.19}$$

Note that each block is computable either directly or via communication with direct neighbours, however, we avoid the transmission of matrices. The iterative linear algebra necessitates the calculation of Hessian times a vector products, which can be computed as

$$\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda} p = \sum_{(k,l) \in \mathcal{N}_i} -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}} p_{k,l} + \sum_{(k,l) \in \mathcal{N}_j} B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{k,l}} p_{k,l}. \quad (7.20)$$

Note that the rows of $\nabla^2 d(\lambda) p$ corresponding to $\lambda_{i,j}$ can be computed on node i by local communication with node j and only vectors are transmitted locally. The solution sensitivities $\frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}}$ can be computed using the nullspace approach given in (7.8).

Inexact solution of the Newton system

The second ingredient in our method is the iterative solution of the Newton system. The whole algorithm is designed so that no direct formation of large linear systems are necessary, nor matrices need to be communicated. Instead, we rely only on matrix-vector products.

We consider the solution of the linear system

$$\nabla^2 d(\lambda^{(k)}) \Delta\lambda = -\nabla d(\lambda^{(k)}). \quad (7.21)$$

We propose a distributed and modified version of the *conjugate gradient method* [131] that we summarize in Algorithm 7.2. The modification involves the introduction of two safeguards; one for checking the definiteness of the dual Hessian, and the other for detecting non-conjugate directions.

We present this algorithm in a distributed fashion, each node $u = 1, \ldots, N$ executes an instance of Algorithm 7.2. In general, subproblem u maintains an iterate of all dual variables that has influence on its optimal primal solution. More precisely, subproblem u optimizes all dual variables $\lambda_{i,j}$ with $(i,j) \in \mathcal{N}_u$. In Steps 1-4, the residual of the corresponding rows of the Newton system is calculated. Note that this operation involves communication to direct neighbours. Inside the CG loop, in Steps 6-8 a slice of the Hessian times the actual conjugate vector product is calculated by communicating to direct neighbours. In Step 9, three inner products of vectors are computed. Since the slices of vectors $p^{(k)}$, $s^{(k)}$ and $r^{(k)}$ exist on different nodes, their contribution has to be collected and broadcasted to all nodes. This operation is often referred to as *global summation* [92], which necessitates either a

Algorithm 7.2: Conjugate Gradient method on node u

Input : $\Delta\lambda_{\mathcal{N}_u}^{(0)}$, $k_{\max}$, ϵ_1, ϵ_2, ϵ_3,

foreach $(i,j) \in \mathcal{N}_u$ **do**
  $r_{i,j}^{(0)} := \sum_{(l,m)\in\mathcal{N}_u} \frac{\partial^2 d(\lambda^{(k)})}{\partial\lambda_{i,j}\partial\lambda_{l,m}} \Delta\lambda_{l,m}^{(0)} + \frac{\partial d(\lambda^{(k)})}{\partial\lambda_{i,j}}$ `// local communication`
  $p_{i,j}^{(0)} := -r_{i,j}^{(0)}$ `// local update`
end
for $k = 0, \ldots, k_{\max}$ **do**
  foreach $(i,j) \in \mathcal{N}_u$ **do**
    $s_{i,j}^{(k)} := \sum_{(l,m)\in\mathcal{N}_u} \frac{\partial^2 d(\lambda)}{\partial\lambda_{i,j}\partial\lambda_{l,m}} p_{l,m}^{(k)}$ `// local communication`
  end
  $\begin{bmatrix}\nu^{(k)}\\ \mu^{(k)}\\ \tau^{(k)}\end{bmatrix} := \sum_{(l,m)\in E} \begin{bmatrix}(p_{l,m}^{(k)})^T s_{l,m}^{(k)}\\ (r_{l,m}^{(k)})^T s_{l,m}^{(k)}\\ (r_{l,m}^{(k)})^T r_{l,m}^{(k)}\end{bmatrix}$ `// global summation`
  if $\nu^{(k)} < \epsilon_1$ **then**
    if $k = 0$ **then return** $p_{i,j}^{(0)}$ for all $(i,j) \in \mathcal{N}_u$
    else return $\Delta\lambda_{i,j}^{(k)}$ for all $(i,j) \in \mathcal{N}_u$
  end
  $\alpha^{(k)} := \tau^{(k)}/\nu^{(k)}$ `// local update`
  foreach $(i,j) \in \mathcal{N}_u$ **do**
    $\Delta\lambda_{i,j}^{(k+1)} := \Delta\lambda_{i,j}^{(k)} + \alpha^{(k)} p_{i,j}^{(k)}$ `// local update`
    $r_{i,j}^{(k+1)} := r_{i,j}^{(k)} + \alpha^{(k)} s_{i,j}^{(k)}$ `// local update`
  end
  if $\frac{\tau^{(k)} + \alpha^{(k)}\mu^{(k)}}{\tau^{(k)}} > \epsilon_2$ **then** $\beta^{(k+1)} := 0$
  else $\beta^{(k+1)} := \frac{1}{\tau^{(k)}} \sum_{(l,m)\in E} (r_{l,m}^{(k+1)})^T r_{l,m}^{(k+1)}$ `// global summation`
  foreach $(i,j) \in \mathcal{N}_u$ **do**
    $p_{i,j}^{(k+1)} := -r_{i,j}^{(k+1)} + \beta^{(k+1)} p_{i,j}^{(k)}$ `// local update`
  end
  if $\sqrt{\tau^{(k)}} < \epsilon_3$ **then return** $\Delta\lambda_{i,j}^{(k)}$ for all $(i,j) \in \mathcal{N}_u$
end
return $\Delta\lambda_{i,j}^{(k)}$ for all $(i,j) \in \mathcal{N}_u$

Output : $\Delta\lambda_{i,j}^{(k)}$ for all $(i,j) \in \mathcal{N}_u$

coordinator process or communication to nodes, which are not direct neighbours. In Steps 10-13, the definiteness of the actual dual Hessian in the direction of $p^{(k)}$ is checked. Since the dual Hessian is positive semidefinite, the CG method might hit singular directions. In this case, we return with the steepest descent direction, see Step 11, or with the most recent solution of the Newton system, see Step 12. Furthermore, in Steps 15-18, local updates of the next solution iterate $\Delta\lambda^{(k+1)}$ and residual vector $r^{(k+1)}$ is carried out. In Steps 19-20, the conjugacy property of $p^{(k)}$ and $p^{(k+1)}$ is checked. Once this property does not hold due to rounding and numerical errors, we proceed with a steepest descent direction, otherwise we need to carry out a global summation with vector $r^{(k+1)}$. In Steps 21-23, the next conjugate directions are computed by local updates. Finally, in Step 24, the residual of the current iterate is measured as a convergence criterion. It has to be understood that Algorithm 7.2, in the best case returns with a Newton direction, an exact solution of the Newton system. In the worst case, the returned solution is a steepest decent direction. In all other cases, a search direction between the steepest descent direction and the Newton direction is returned. For this reason, this algorithm can be considered as an inexact Newton method.

Line search strategy with backtracking

The third ingredient in our dual Newton-CG framework is the line search strategy. One has to make sure that in each Newton iteration the dual function value decreases with a certain amount. We use a distributed Armijo type line search strategy with backtracking. We denote the approximate solution of the Newton system given by the CG procedure with $\Delta\lambda$. A candidate iterate $\lambda^{(k+1)}$ is accepted if

$$d(\lambda^{(k+1)}) \leq d(\lambda^{(k)}) + \gamma t^{(k)} \nabla d(\lambda^{(k)})^T \Delta\lambda \tag{7.22}$$

holds. Otherwise we set $t^{(k)} := \beta t^{(k)}$ with some $\beta \in (0,1)$. Note that the local contributions to $d(\lambda^{(k)})$ are already available on different nodes, similarly the corresponding slices of $\nabla d(\lambda^{(k)})$. The line search procedure is summarized in Algorithm 7.3, which is executed by all nodes $u = 1, \ldots, N$

In Step 1, the dual function is evaluated by a global summation operation. Note that the QP subproblems have been already solved at the beginning of the Newton step, thus this is only a communication step. In Step 2, a directional derivative of the dual function is computed by a global summation operation. Inside the line search loop, in Steps 5-7, the local update of corresponding dual variables is carried out. In Step 8, the dual function is evaluated at the candidate iterate that is solving

Algorithm 7.3: Line search procedure of dual Newton-CG method

Input : $\Delta\lambda_{\mathcal{N}_u}$, $\beta \in (0,1)$, $\gamma \in (0, \frac{1}{2})$

1 $d := d(\lambda)$ // global summation

2 $v := \sum_{(i,j)\in E} \left(A_{i,j}x_i^*(\lambda_{\mathcal{N}_i}) - B_{i,j}x_j^*(\lambda_{\mathcal{N}_j})\right)^T \Delta\lambda_{i,j}$ // global summation

3 $t := 1$

4 **while** *true* **do**

5 | **foreach** $(i,j) \in \mathcal{N}_u$ **do**

6 | | $\lambda_{i,j}^+ := \lambda_{i,j}^{(k)} + t\Delta\lambda_{i,j}$ // local update

7 | **end**

8 | $d^+ := -\sum_{k=1}^{N} \min_{x_k \in \mathcal{X}_k} L_k(x_k, \lambda_{\mathcal{N}_i}^+)$ // global summation

9 | **if** $d^+ \leq d + \gamma t v$ **then** break

10 | **else** $t := \beta t$

11 **end**

12 **return** t

Output : t

M QPs locally in parallel followed by a global summation operation. In Steps 9 and 10, the Armijo condition is checked locally, i.e. whether the actual stepsize leads to sufficient decrease. Now, having presented all ingredients of our algorithm we revise our prototype and present the dual non-smooth Newton method in Algorithm 7.4, which is executed on each node $u = 1, \ldots, N$.

Algorithm 7.4: Dual non-smooth Newton-CG method (revisited)

Input : u, $\lambda_{\mathcal{N}_u}^{(0)}$, $k_{\max}$, ϵ_1, ϵ_2, ϵ_3, ϵ_4,

1 $k := 0$

2 Solve local QP with $\lambda_{\mathcal{N}_u}$

3 **while** *no convergence* **do**

4 | Calculate local contribution to $\nabla^2 d(\lambda^{(k)})$ and $\nabla d(\lambda^{(k)})$

5 | $\Delta\lambda_{\mathcal{N}_u} := \text{ConjugateGradient}(\Delta\lambda_{\mathcal{N}_u}^{(k)}, \epsilon_1, \epsilon_3, \epsilon_3)$

6 | $t^{(k)} := \text{LineSearch}(\Delta_{\mathcal{N}_u}, \beta, \gamma)$

7 | $\lambda_{\mathcal{N}_u}^{(k+1)} := \lambda_{\mathcal{N}_u}^{(k)} + t^{(k)}\Delta\lambda_{\mathcal{N}_u}$

8 **end**

Output : $\lambda^{(k)}$

Note that the algorithmic framework consists of the loop of the Newton method, into which the loop of the CG method and the line search procedure are nested. Moreover, the solution of local QPs in Step 2 is out of the main loop. The reason is that in the following iterations the most recent solutions and derivative contributions will be already available computed by the line search procedure.

Analysis of computational costs

We give an analysis of the different fractions of the computational costs. Our method is executed by M processes in a distributed memory system. The method starts with a QP solution, which is cold-started, i.e. no factorizations or active set can be used to initialize the QP solver. In the next step, we jump into the Newton loop and proceed with the calculation of derivatives. The computation of the gradient requires one matrix-vector product and one transmission of a vector on each node with respect to each connection starting from or ending in node i. The local dual Hessian contribution with a nullspace-based QP solver costs about the computation time that is needed to invert the reduced Hessian and a couple of matrix-matrix products. Once a black box QP solver is used, an extra cost has to be taken into account due to the calculation of the nullspace of the active constraints locally. The next step is the CG procedure. Each iteration of this loop locally costs the computation time of a matrix-vector product, plus the transmission of this to direct neighbours. This should be added to the global cost of two global summation operation. If the nodes of the cluster are organized into a tree structure, e.g. binary tree, the cost of global summation operation is proportional to $log(M)$ [114]. Once the Newton system is solved exactly or inexactly, the line search loop follows. This procedure starts with two global summation of scalars, while inside the loop a local QP solution and another global summation takes place. The result of the last line search trial is reused in the next Newton iteration.

7.2.2 Numerical Example: Optimal Control of Coupled Masses without State Constraints

It should be emphasized that at this stage state constraints are not present in order to avoid the realization of a singular dual Hessian, which renders our approach into a gradient method resulting in very slow convergence. We discuss the phenomenon and provide remedy in the next chapter.

We have tested the dual Newton-CG approach on an academic optimal control problem. We consider a linear model for a chain of connected masses in one dimension. Each mass $n \in \{1, \ldots, N\}$ is described by its position $p_n \in \mathbb{R}$ and its velocity $v_n \in \mathbb{R}$ and may be controlled via a force F_n. For the sake of simplicity,

we directly present the equations obtained by using distributed multiple shooting method, i.e., after spatial and time discretization. We discretize in time by one step of an explicit Euler method. Let us denote the length of time intervals and the number of time intervals by Δt and M, respectively. The discrete-time system dynamics read as

$$\begin{bmatrix} p_n^{(m+1)} \\ v_n^{(m+1)} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & & & \\ -2\Delta t & 1 & \Delta t & \Delta t & \Delta t \end{bmatrix} \begin{bmatrix} p_n^{(m)} \\ v_n^{(m)} \\ z_{n,n-1}^{(m)} \\ z_{n,n+1}^{(m)} \\ F_n^{(m)} \end{bmatrix}. \tag{7.23}$$

Here, $m = 1, \ldots, M-1$, $n = 1, \ldots, N$ and the variables $z_{n,n-1}^{(m)}$ and $z_{n,n+1}^{(m)}$ belong to the masspoint n on time interval m and represent a finite discretization, e.g. polynomial coefficients, of the positions of masspoints $n-1$ and $n+1$, respectively. The spatial coupling between the mass points is ensured by

$$z_{n,n+1}^{(m)} = p_{n+1}^{(m)} \quad n = 1, \ldots, N-1 \tag{7.24}$$

$$z_{n,n-1}^{(m)} = p_{n-1}^{(m)} \quad n = 2, \ldots, N \tag{7.25}$$

The objective function penalizes the deviation from the steady state 0 in an L_2-norm. The bounds on the actuated forces are given by

$$\underline{F} \leq F_n^{(m)} \leq \overline{F}, \quad m = 1, \ldots, M,\ n = 1, \ldots, N, \tag{7.26}$$

and the first and last mass points are fixed

$$p_1^m = p_{\text{first}},\ p_n^m = p_{\text{last}},\ m = 1, \ldots, M \tag{7.27}$$

We introduce for all $n = 1, \ldots, N$, and $m = 1, \ldots, M$

$$y_{(m-1)N+n}^T := (p_n^{(m)}, v_n^{(m)}, z_{n,n-1}^{(m)}, z_{n,n+1}^{(m)}, F_n^{(m)})^T, \tag{7.28}$$

$S := NM$ and summarize the discretized optimal control problem as a convex QP of the form

$$\min_{y_1, \ldots y_S} \frac{1}{2} \sum_{k=1}^{S} y_k^T R_k y_k \tag{7.29a}$$

$$\text{s.t } C_{i,j} y_i = D_{i,j} y_j,\ (i,j) \in \overline{E} \tag{7.29b}$$

$$y_l \in \mathcal{Y}_l,\ l = 1, \ldots, S. \tag{7.29c}$$

Here, $(i,j) \in \overline{E}$ if and only if $j = i + N$ or $(j = i + 1$ and $j \mod N \neq 0)$. Moreover, $C_{i,j}$ and $D_{i,j}$ are directly computable from (7.23)-(7.25) and $\mathcal{Y}_l$ can be derived from (7.26). Note that in (7.29), only the control input variables are constrained, the state variables are not bounded.

We have solved an instance of (7.29) with $m = 20$ mass points and $n = 15$ time intervals resulting in a decomposable QP with 321 subproblems. The full QP has a total of 1540 primal variables, 1170 equalities, which is the dimension of the dual space as well, and 2340 inequalities. In the dual Newton-CG framework, we have set the maximal number of line search trials to 30, and $\beta = 0.6$. In each Newton iteration, the number of CG iterations was limited by 500. The tolerances of the CG procedure were, $\epsilon_1 = 10^{-20}$, $\epsilon_2 = 10^{-1}$, $\epsilon_3 = \min\left(10^{-2}, \sqrt{\|\nabla d(\lambda^{(k)})\|}\right)\|\nabla d(\lambda^{(k)})\|$. As a stopping criterion, we have used $\frac{\|\nabla d(\lambda^k)\|}{\|\nabla d(\lambda^0)\|} < \epsilon$ to measure the infeasibility of the primal problem.

The iterations of the dual Newton-CG method are shown in Table 7.2.1. The dual solution is found in 43 Newton iterations. The maximum number of CG loops is only reached once in the 42-nd Newton iteration. In the third column, we show the number of active constraints and whether it has changed since the previous iteration. In the fourth column, the residual of the Newton system is shown, while in the fifth the number of line-search trials is given. We can see that in the first 38 Newton iterations the correct active set is not yet found, but from the 39-th iteration on, the dual is pure quadratic without primal active set changes.

7.3 Regularized Dual Newton-CG method

As we have seen in the previous section, the dual function may become singular, which is triggered by linearly dependent inequalities in some extreme dual iterates. In such a situation, the conjugate gradient method breaks down and essentially functions as a gradient method resulting in very many steps and line search trials. In this section, we develop tools to overcome this phenomenon, we introduce a relaxation method based on L_2 penalties that in fact regularizes the dual function. Via the L_2 relaxation, we show both theoretically and experimentally that singular regions can be avoided.

Table 7.2.1: Iterations of the dual Newton-CG method. The columns are iteration counter, norm of the dual gradient, number of active constraints, number of CG iterations, residual of the Newton system, number of line search trials and the stepsize.

It.	du.grad.	#AC	#CGit	CG-res	#LS	stepsize
1	2.040E-02	86C	58	4.126E-02	1	1.000E+00
2	2.706E-02	102C	96	1.054E-02	1	1.000E+00
3	1.692E-02	109C	97	9.591E-02	1	1.000E+00
4	1.432E-02	114C	215	2.825E-02	1	1.000E+00
5	2.025E-02	115C	49	2.265E-01	2	6.000E-01
6	2.022E-02	115	44	1.147E-02	2	6.000E-01
7	2.218E-02	116	143	1.587E-02	4	2.160E-01
8	2.091E-02	116C	107	3.604E-01	4	2.160E-01
9	1.832E-02	116C	57	4.922E-02	3	3.600E-01
10	9.954E-03	117C	76	4.221E-02	4	2.160E-01
11	2.006E-02	117	47	1.419E-02	4	2.160E-01
12	8.735E-03	117	62	6.483E-01	6	7.776E-02
13	1.216E-02	117C	38	2.665E-01	5	1.296E-01
14	8.565E-03	117	51	2.324E-01	6	7.776E-02
15	1.883E-02	117	38	2.889E-01	6	7.776E-02
16	8.237E-03	117	63	1.056E-02	8	2.799E-02
17	1.178E-02	117	38	2.582E-01	8	2.799E-02
18	8.203E-03	118	44	2.230E-02	9	1.680E-02
19	1.172E-02	117	38	2.606E-01	9	1.680E-02
20	7.533E-03	116	51	2.143E-01	10	1.008E-02
21	1.169E-02	115	8	5.980E-02	4	2.160E-01
22	1.245E-02	114	14	1.206E-01	5	1.296E-01
23	9.630E-04	115C	9	1.161E-02	1	1.000E+00
24	1.974E-02	118C	323	4.001E-03	1	1.000E+00
25	2.105E-02	119	92	1.863E-02	5	1.296E-01
26	4.093E-03	119	44	2.783E-02	4	2.160E-01
27	7.044E-04	119	14	5.918E-02	1	1.000E+00
28	1.971E-02	119C	129	7.217E-03	2	6.000E-01
29	4.335E-03	120	38	2.420E-02	4	2.160E-01
30	7.382E-04	120	14	2.360E-02	1	1.000E+00
31	1.039E-02	119C	127	5.116E-03	6	7.776E-02
32	6.456E-04	120C	3	1.798E-02	1	1.000E+00
33	7.199E-03	119C	127	4.364E-03	8	2.799E-02
34	6.003E-04	120C	3	3.237E-02	1	1.000E+00
35	6.841E-03	118	138	4.180E-03	8	2.799E-02
36	7.169E-04	119C	3	5.727E-02	1	1.000E+00
37	7.912E-03	118C	120	2.006E-03	8	2.799E-02
38	5.071E-04	119C	4	1.024E-02	1	1.000E+00
39	2.666E-03	117	127	2.138E-03	10	1.008E-02
40	2.833E-04	117	12	1.051E-02	1	1.000E+00
41	1.255E-05	117	316	1.129E-03	1	1.000E+00
42	1.577E-05	117	500F	2.570E-05	1	1.000E+00
43	1.935E-07	117	109	1.252E-05	1	1.000E+00

7.3.1 L_2-relaxation of inequalities

We restate the original problem that we treat as

$$\min_x \sum_{k=1}^{M} \frac{1}{2} x_k^T H_k x_k + c_k^T x_k \tag{7.30a}$$

$$\text{s.t. } A_{i,j} x_i = B_{i,j} x_j \quad (i,j) \in E \tag{7.30b}$$

$$D_l x_l \leq e_l, \quad l = 1, \ldots, M. \tag{7.30c}$$

The dual function is defined by

$$d(\lambda) := -\min_{x \in \mathcal{X}} L(x, \lambda) = \sum_{i=1}^{M} \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}). \tag{7.31}$$

Now, we introduce a *modified dual function*, which uses penalties, instead of the hard local inequality constraints. More precisely, we regard

$$d_\gamma(\lambda) := -\sum_{i=1}^{N} \begin{pmatrix} \min_{x_i} & L_i(x_i, \lambda_{\mathcal{N}_i}) + \frac{\gamma}{2} \|s_i\|_2^2 \\ \text{s.t.} & D_i x_i - e_i \leq s_i \end{pmatrix}. \tag{7.32}$$

Here, $\gamma \in \mathbb{R}$ is the weight of the L_2 penalty of the constraints, and $s_i \in \mathbb{R}^{w_i}$ is a slack variable. Note that once $\gamma = 0$, the local inequalities are completely relaxed and can be dropped, whereas if $\gamma \to \infty$ the local inequalities are becoming more and more important, i.e. less and less relaxed. Moreover, in the limit, the modified dual $d_\infty(\lambda)$ is equivalent to $d(\lambda)$. Observe that the positivity of the slack variables does not need to be enforced. Indeed, it can be easily verified that inequality constraints of the form $s_i \geq 0$ included in (7.32) would never be strictly active, and can therefore be discarded.

The L_2 relaxation proposed in (7.32) offers a remedy to render the dual Hessian non-singular. The following lemma establishes the effect of the relaxation.

Lemma 7.3.1 *Assuming that $H_i \succ 0$, for any λ and $\gamma > 0$, the Hessian of the modified dual function $d_\gamma(\lambda)$ is non-singular.*

Proof *We write the modified dual problem in the compact form*

$$d_\gamma(\lambda) = -\begin{pmatrix} \min_x & \frac{1}{2} x^T H x + c^T x + \frac{\gamma}{2} s^T s + \lambda^T C x \\ \text{s.t.} & Dx - e \leq s \end{pmatrix}.$$

The dual Hessian reads

$$\nabla^2 d_\gamma(\lambda) = -C\frac{\partial x}{\partial \lambda}, \tag{7.33}$$

where the sensitivity $\frac{\partial x}{\partial \lambda}$ *is the solution of the linear system*

$$\begin{bmatrix} H & D_{\mathcal{A}}^T & 0 \\ D_{\mathcal{A}} & 0 & -I_{\mathcal{A}} \\ 0 & -I_{\mathcal{A}}^T & \gamma I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \lambda} \\ \frac{\partial \mu_A}{\partial \lambda} \\ \frac{\partial s}{\partial \lambda} \end{bmatrix} = - \begin{bmatrix} C^T \\ 0 \\ 0 \end{bmatrix}. \tag{7.34}$$

Here, $I_{\mathcal{A}}$ *is a full row rank matrix with single unitary entries at the row indices reported by* $\mathcal{A}$*. We use the fact that* $I_{\mathcal{A}} I_{\mathcal{A}}^T = I$ *to eliminate the last row in* (7.34) *and obtain*

$$\begin{bmatrix} H & D_{\mathcal{A}}^T \\ D_{\mathcal{A}} & -\gamma^{-1} I \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \lambda} \\ \frac{\partial \mu_A}{\partial \lambda} \end{bmatrix} = - \begin{bmatrix} C^T \\ 0 \end{bmatrix}, \tag{7.35}$$

This structure, so that the bottom-right corner has a diagonal regularization term, is typical in interior-point methods, by which the spectral properties of the KKT matrix are improved [125, 65]. The solution for the primal sensitivities reads as

$$\frac{\partial x}{\partial \lambda} = -\left(H + \gamma D_{\mathcal{A}}^T D_{\mathcal{A}}\right)^{-1} C^T. \tag{7.36}$$

The dual Hessian therefore reads as

$$\nabla_\lambda^2 d_\gamma(\lambda) = C\left(H + \gamma D_{\mathcal{A}}^T D_{\mathcal{A}}\right)^{-1} C^T, \tag{7.37}$$

which is non-singular since C *is full row rank.*

■

We demonstrate next the effect of the softened constraints on a simple example.

Example 7.3.1 *We consider the following quadratic program with relaxed inequality*

$$\begin{aligned} \min_{x,y} \quad & \frac{1}{2}x^2 + \frac{1}{2}y^2 + \frac{\gamma}{2}(s_1^2 + s_2^2) \\ \text{s.t.} \quad & -x - s_1 \le a, \quad y - s_2 \le a, \quad x = y, \end{aligned}$$

and its corresponding modified dual function

$$d_\gamma(\lambda) = -\begin{pmatrix} \min_{x,y} & \frac{1}{2}x^2 + \frac{1}{2}y^2 + \frac{\gamma}{2}(s_1^2 + s_2^2) + \lambda(x - y) \\ \text{s.t.} & -x - s_1 \le a, \quad y - s_2 \le a, \end{pmatrix}.$$

It can be easily verified that

$$d_\gamma(\lambda) = \begin{cases} \frac{(a-\lambda)^2}{\gamma+1} - a(a-2\lambda) & \text{for } \lambda \leq a, \\ \lambda^2 & \text{for } \lambda > a, \end{cases} \tag{7.38}$$

and

$$\nabla^2_\lambda d\,(\lambda) = \begin{cases} \frac{2}{\gamma+1} & \text{for} \quad \lambda \leq a, \\ 2 & \text{for} \quad \lambda > a. \end{cases}$$

We can see that for any λ the modified dual function has positive curvature.

■

7.3.2 Singularity-free dual Newton-CG method

The optimal dual variables for a fixed γ can be found as the solution of the unconstrained dual optimization problem defined by

$$\min_\lambda d_\gamma(\lambda). \tag{7.39}$$

Now we revise the Algorithm 7.4 presented in the previous chapter. The main modification we introduce is the update of the penalty parameter γ in each Newton loop. In each iteration of the non-smooth Newton method, a second-order model of $d_\gamma(\lambda)$ in $\lambda^{(k)}$ is minimized that is

$$m(p) = d_\gamma(\lambda^{(k)}) + \nabla d_\gamma(\lambda^{(k)})^T p + \frac{1}{2} p^T \nabla^2 d_\gamma(\lambda^{(k)}) p. \tag{7.40}$$

Solving $p^{(k)} := \arg\min_p m(p)$ yields a descent direction in the space of λ and can be obtained as the solution of the linear system

$$\nabla^2 d_\gamma(\lambda^{(k)}) p + \nabla d_\gamma(\lambda^{(k)}) = 0. \tag{7.41}$$

This Newton system, in the view of Lemma 7.3.1, is non-singular and can be solved inexactly by a conjugate gradient (CG) method [131].

In Algorithm 7.5, we have summarized the most important steps of the proposed approach. Observe that termination takes place once the optimal dual multipliers are found and the relaxed constrains become tight, up to accuracy ϵ_1 and ϵ_2, respectively. The user-given parameter ϵ_1 controls the tolerance of the dual gradient, while ϵ_2 is used to check whether the slack variables are sufficiently small. Both quantities need to be zero to obtain an optimal primal-dual solution. In the following, we discuss each step in more detail.

Algorithm 7.5: Singularity-free Dual Newton-CG method

Input : $\lambda^{(0)}$, $\gamma > 0$, $\tau > 1$, $\epsilon_1 > 0$, $\epsilon_2 > 0$
while $\|\nabla d_\gamma(\lambda)\| > \epsilon_1$ *or* $\|s\| > \epsilon_2$ **do**
  Compute $\nabla d_\gamma(\lambda)$ and $\nabla^2 d_\gamma(\lambda)$ in a distributed fashion.
  Solve $\nabla^2 d_\gamma(\lambda^{(k)})p + \nabla d_\gamma(\lambda^{(k)}) = 0$ with CG.
  Find proper stepsize t.
  $\lambda^{(k+1)} := \lambda^{(k)} + tp$
  $\gamma := \tau \cdot \gamma$
  $k := k + 1$
end
return $\lambda^{(k)}$

Calculation of derivatives in a distributed manner

Since the derivatives of the modified dual function are different from the original dual function, we provide a description of how to calculate these derivatives in a distributed manner.

The dual function $d_\gamma(\lambda)$ is continuously differentiable [13, p. 100] and its derivative in the direction of $\lambda_{i,j}$ is given by

$$\frac{\partial d_\gamma(\lambda)}{\partial \lambda_{i,j}}^T = -A_{i,j} x_i^*(\lambda_{\mathcal{N}_i}) + B_{i,j} x_j^*(\lambda_{\mathcal{N}_j}), \tag{7.42}$$

where $x_i^*(\lambda_{\mathcal{N}_i}) := \arg\min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_{\mathcal{N}_i}) + \frac{\gamma}{2} s_i^2$. It should be understood that only subproblems i and j need to be solved in order to calculate the dual gradient in the direction of $\lambda_{i,j}$. Moreover, this slice of the gradient can be computed in subproblems i and j by a simple local exchange of contributions. This is one of the reasons why first order methods are often used in a dual decomposition framework.

We obtain $\nabla^2 d_\gamma(\lambda)$ via differentiating (7.42). It is important to observe that the dual Hessian has a well-defined sparsity structure, which is essentially determined by the interconnections of subsystems. The block corresponding to variables $\lambda_{i,j}$, $(i,j) \in E$ can be written as

$$\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda_{i,j}} = -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{i,j}} + B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{i,j}}. \tag{7.43a}$$

The off-diagonal blocks corresponding to the row of $\lambda_{i,j}$ and the column of $\lambda_{k,l},\ (i,j) \neq (k,l)$ are given by

$$\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda_{k,l}} =$$

$$= \begin{cases} -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}} & \text{if} (k,l) \in \mathcal{N}_i \setminus \mathcal{N}_j, \\ B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{k,l}} & \text{if} (k,l) \in \mathcal{N}_j \setminus \mathcal{N}_i, \\ 0 & \text{otherwise.} \end{cases} \tag{7.43b}$$

In other words, the second derivatives with respect to $\lambda_{i,j}$ and $\lambda_{k,l}$ are nonzero if and only if edge (k,l) is connected to the node i or j.

Now, let us compute the sensitivity of the solution in subproblem i with respect to all influential dual variables, i.e. $\frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{\mathcal{N}_i}}$. To this end, we follow the procedure presented in Lemma 7.3.1. Let $\mathcal{A}_i$ report the active set at the actual optimal solution for problem i. For notational convenience, we introduce matrix C_i such that

$$\lambda_{\mathcal{N}_i} C_i x_i := \sum_{(i,j) \in E} \lambda_{i,j}^T A_{i,j} x_i - \sum_{(j,i) \in E} \lambda_{j,i}^T B_{j,i} x_i. \tag{7.44}$$

The optimal primal-dual solution of subproblem i is the solution of

$$\begin{bmatrix} H_i & 0 & D_{i,\mathcal{A}_i}^T \\ 0 & \gamma I & -I_{\mathcal{A}} \\ D_{i,\mathcal{A}_i} & -I_{\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} x_i^*(\lambda_{\mathcal{N}_i}) \\ s_i^* \\ \mu_i^* \end{bmatrix} = - \begin{bmatrix} c_i + C_i^T \lambda_{\mathcal{N}_i} \\ 0 \\ -e_{i,\mathcal{A}_i} \end{bmatrix}, \tag{7.45}$$

where μ_i^* is the optimal dual variable corresponding to the constraints reported by $\mathcal{A}_i$. It can be easily verified that

$$\frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{\mathcal{N}_i}} = \left(\gamma D_{i,\mathcal{A}_i}^T D_{i,\mathcal{A}_i} + H_i\right)^{-1} C_i^T. \tag{7.46}$$

Note that the cost of this operation is dominated by the matrix inversion, which is carried out locally in each subproblem. Also, the local contributions to (7.43) are directly computable.

Since the Newton system is solved via a CG procedure, the product of the Hessian with a vector is of major interest. We consider first rows in the dual Hessian corresponding to $\lambda_{i,j}$ that is

$$\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda} p = \sum_{(k,l) \in \mathcal{N}_i} -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}} p_{k,l} + \sum_{(k,l) \in \mathcal{N}_j} B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{k,l}} p_{k,l}. \tag{7.47}$$

Here, $p \in \mathbb{R}^v$, $p^T = [p_{i_1,j_1}^T, \ldots, p_{i_Q,j_Q}^T]$ and the order $(i_1, j_1), \ldots, (i_Q, j_Q)$ corresponds to the one defined in λ. It should be noted that the vector slice $\frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda} p$ can be calculated on both nodes i and j via only local exchange of contributions to the sum in (7.47). In fact, in each CG iteration, a local exchange of vectors, i.e. a matrix-vector product, takes place in between subproblem i and j for each $(i, j) \in E$.

We use (7.47) to calculate the quadratic form $p^T \frac{\partial^2 d(\lambda)}{\partial \lambda^2} p$, i.e.

$$p^T \frac{\partial^2 d(\lambda)}{\partial \lambda^2} p = \sum_{(i,j)\in E} p_{i,j}^T \frac{\partial^2 d(\lambda)}{\partial \lambda_{i,j} \partial \lambda} p =$$

$$= \sum_{(i,j)\in E} p_{i,j}^T \left[\sum_{(k,l)\in \mathcal{N}_i} -A_{i,j} \frac{\partial x_i^*(\lambda_{\mathcal{N}_i})}{\partial \lambda_{k,l}} p_{k,l} + \sum_{(k,l)\in \mathcal{N}_j} B_{i,j} \frac{\partial x_j^*(\lambda_{\mathcal{N}_j})}{\partial \lambda_{k,l}} p_{k,l} \right]. \tag{7.48}$$

Note that in (7.48) a summation over all $(i, j) \in E$ takes place. Each term of this sum can be calculated via local communication as mentioned earlier. However, the calculation of the sum itself requires a global operation, such that all local contributions are collected and the result is broadcasted. This operation is often referred to as a *global summation* [92].

Based upon the previous discussion the residual $\nabla^2 d(\lambda) p + \nabla d(\lambda)$ is computable in slices with respect to $\lambda_{i,j}$ and is available in subproblems i and j.

The conjugate gradient method and line-search procedure are kept unchanged, we refer to Algorithms 7.2 and 7.3.

7.3.3 Numerical Example: Optimal Control of Connected Masses with State Constraints

The numerical experiments are carried out on an academic optimal control problem. We consider a linear model of a unidimensional chain consisting of connected masses. We use the same model as given in Subsection 7.2.2, to which we added state constraints.

We generated an instance of (7.29) with $m = 20$ mass points and $n = 15$ time intervals resulting in a decomposable QP with 320 subproblems. The full QP has a total of 1540 primal variables, 1170 dual variables, and 2340 inequality constraints.

The proposed stopping criterion was $\|\nabla d_\gamma(\lambda)\|_\infty < 10^{-5}$ and $\|s\|_\infty < 10^{-5}$. The convergence of our algorithm was tested with many different initial guesses, including

Table 7.3.1: Solution statistics with different initial guesses.

Init. guess	Newt. it.	QP sol.	CG it.	RFG
random	46	92	1261	14045
$\mathbf{1}$	46	92	1171	9877
$2 \cdot \mathbf{1}$	46	92	1183	10151
$3 \cdot \mathbf{1}$	46	92	1189	11619

random initialization. In each test case, convergence was attained in 46 Newton iterations taking only full steps. See Table 7.3.1, where the number of Newton iterations, the number of local QP solutions per subproblem, and overall number of CG iterations is reported for a couple of test cases. In the last column, one can observe the number of dual iterations made by a state-of-the-art restarted fast gradient method (RFG) [111], applied in the dual space. This number also shows the necessary number of local QP solutions per node. It was also observed that in all test cases, the original method without the relaxation technique was never successful and always got stuck with a certain active set due the occurrence of the singular dual Hessian.

We show one test case, where the initial guess was chosen randomly, and plotted the evolution of $\|d_\gamma(\lambda)\|$ and $\|s\|$ with the proposed method versus $\|d(\lambda)\|$ given by the approach with hard constraints in Figure 7.1.

We conclude that for this particular example, the number of iterations and thus the number of local QP solutions is two orders of magnitude smaller than with a first order method. Moreover, the use of L_2 penalty on the constraints accelerates the practical convergence speed of the dual Newton-CG approach.

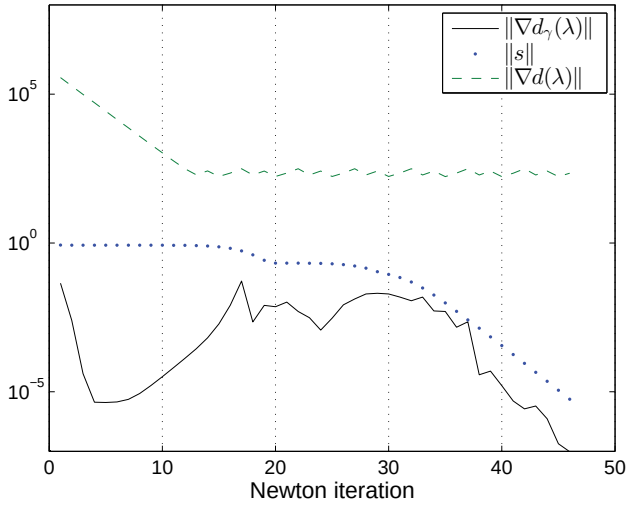


Figure 7.1: Illustration of the relaxation-based approach (plain line) versus the original method (dashed line), initialized at a random point. The dotted line displays the convergence of the slack variables, resulting from gradually increasing the γ parameter over the dual iterations. The two approaches start with different dual gradient values since the relaxed problem is different from the original one.

Chapter 8

Conclusion

This chapter concludes the thesis with a summary and with mentioning some possible directions to follow as a future research project.

8.1 Summary

In this thesis, we investigated distributed optimization methods for the optimal control of large scale distributed systems.

In Part I, the topic of discussion was nonlinear optimal control. In Chapter 2, we have established a massively parallel simulation scheme called distributed multiple shooting (DMS) to be used in optimization loops. This scheme takes into account the distributed structure of the problem by decomposing both the time and the spatial domains. Via these decomposition methods the independent integration and sensitivity generation of different subsystems is made possible. The consistency between the time and spatial domains are recovered by an efficient Newton-type method leading to fast local convergence rate. However, the optimization phase of the proposed scheme is centralized, which necessitates a coordinator process to which several vectors and matrices are transmitted and so introducing communication costs.

In Chapters 3 and 4, we applied the DMS scheme inside an SQP method to two different dynamic optimization problems, where the experiments suggest that the novel approach can lead up to 19 times faster integration and sensitivity calculation. The problem of centralized optimization within the DMS method was addressed in Part II. The author of this thesis has contributed to the publications [128, 127] by

implementing and executing numerical experiments, to [83] by model development and numerical results.

In Part II, the topic of interest was linear optimal control, i.e. the solution of quadratic programs in a distributed fashion. In Chapter 5, existing dual decomposition methods using only first-order derivatives were discussed such as gradient method, fast gradient method, alternating method of multipliers (ADMM), etc. These methods were extensively tested and compared with each other in Chapter 6. The experiments here suggest that the ADMM and the inexact Uzawa methods are the best distributed QP approaches in terms of necessary local QP solutions, while the restarted variant of the fast gradient method is competitive. However, often all methods need thousands of dual iterations to achieve sufficient accuracy. In the same chapter, a set of large scale benchmark quadratic programs was set up and made freely available. The author of this thesis contributed to the literature survey, software development and numerical experiments in the publication [84], while he has got help in proofreading from his co-authors.

We attempted to overcome the slow convergence properties of first-order methods in Chapter 7. Here, a novel approach, dual Newton-CG method was established, which improves the often experienced sublinear convergence rate of classical dual decomposition schemes by utilizing second-order derivatives. A Newton method coupled with an iterative linear algebra is proposed to find the optimal dual multipliers. One drawback of the proposed method is that it cannot be fully distributed, central coordination is necessary. Communication of vectors takes place locally, while global operations on scalars are carried out. Moreover, it has been observed that in the presence of local inequalities the failure of the CG method may occur rendering the dual Newton-CG method a pure gradient method. In order to avoid this situation, a singularity-free Newton-CG method was proposed being able to treat local inequalities efficiently and leading to fast local convergence with high accuracy. The algorithm design was carried out by the author of this thesis in [85], while co-authors provided help in setting up the numerical experiments. Most tasks in [86] were shared between the main and the co-authors.

Appendix B presents PyDQP, the open-source software for distributed convex quadratic programming implementing 11 different dual decomposition schemes.

8.2 Directions for future research

In the context of distributed nonlinear optimal control, the investigation of asynchronous linearizations [138, 21] is an open topic. In such an approach, which could be regarded as an asynchronous DMS scheme, the different subsystems may linearize themselves with different frequency depending on the speed of their

dyanamics. Moreover, the automatic treatment of infeasible linearizations and smart regularization would have practical importance.

In distributed quadratic programming, and in particular with our approach, a preconditioning scheme [6, 9] based on local communication should be established in order to improve the performance of the iterative linear algebra. Furthermore, an adaptive routine for updating the penalty parameter in the singularity-free approach needs to be designed.

Dual decomposition methods can be applied to general convex nonlinear problems and thus the generalization of the dual Newton-CG scheme to this problem class is an interesting research topic.

In our dicussion, the ADMM method used a simple gradient update of the dual variables, this could be exchanged by a more efficient approach such as a restarted fast gradient method or even a second-order update of the multipliers [14].

Appendix A

Model parameters of HPV

In the following table we can find the data borrowed from [126] that is necessary to implement the HPV model. The data are grouped per subsystem. General data can be found at the end of the table.

Symbol	Description	Unit of measure	Value
	Subsystem 1		
S_{L_1}	Surface lake L_1	m^2	10×10^3
S_{L_2}	Surface lake L_2	m^2	5×10^3
$h_{L_{1\min}}$	Minimum lake water level L_1	m	10.5
$h_{L_{1\max}}$	Maximum lake water level L_1	m	13.5
$h_{L_{2\min}}$	Minimum lake water level L_2	m	5.5
$h_{L_{2\max}}$	Maximum lake water level L_2	m	8.5
$q_{\text{in}_{L_1}}$	Water inflow lake L_1	$m^3\ s^{-1}$	5
$q_{\text{in}_{L_2}}$	Water inflow lake L_2	$m^3\ s^{-1}$	5
h_{U_1}	Height difference duct U_1	m	5
S_{U_1}	Cross section duct U_1	m^2	6
$k_{t_{T_1}}$	Turbine coefficient T_1	$J\ m^{-4}$	8000
$k_{t_{C_1}}$	Turbine coefficient C_1	$J\ m^{-4}$	8000
$k_{p_{C_1}}$	Pump coefficient C_1	$J\ m^{-4}$	14000
h_{T_1}	Height difference duct T_1	m	223
h_{C_1}	Height difference duct C_1	m	200
$q_{T_{1\min}}$	Minimum turbine flow T_1	$m^3\ s^{-1}$	0
$q_{T_{1\max}}$	Maximum turbine flow T_1	$m^3\ s^{-1}$	20
$q_{C_{1\text{t,min}}}$	Minimum flow in C_1 in turbine mode	$m^3\ s^{-1}$	0
$q_{C_{1\text{t,max}}}$	Maximum flow in C_1 in turbine mode	$m^3\ s^{-1}$	10
$q_{C_{1\text{p,min}}}$	Minimum flow in C_1 in pump mode	$m^3\ s^{-1}$	0
$q_{C_{1\text{p,max}}}$	Maximum flow in C_1 in pump mode0.0	$m^3\ s^{-1}$	5
	Subsystem 2		

continues...

...continued

Symbol	Description	Unit of measure	Value
S_{L_3}	Surface lake L_3	m^2	10×10^3
$h_{L_{3\min}}$	Minimum lake water level L_3	m	6
$h_{L_{3\max}}$	Maximum lake water level L_3	m	9
$q_{\text{in}_{L_3}}$	Water inflow lake L_3	$m^3\ s^{-1}$	10
$k_{t_{T_2}}$	Turbine coefficient turbine T_2	J m^{-4}	8000
$k_{t_{C_2}}$	Turbine coefficient C_2	J m^{-4}	8000
$k_{p_{C_2}}$	Pump coefficient C_2	J m^{-4}	14000
h_{T_2}	Height difference duct C_2	m	233
h_{C_2}	Height difference duct T_2	m	250
$q_{T_{2\min}}$	Minimum turbine flow T_2	$m^3\ s^{-1}$	0
$q_{T_{2\max}}$	Maximum turbine flow T_2	$m^3\ s^{-1}$	20
$q_{C_{2\text{t,min}}}$	Minimum flow in C_2 in turbine mode	$m^3\ s^{-1}$	0
$q_{C_{2\text{t,max}}}$	Maximum flow in C_2 in turbine mode	$m^3\ s^{-1}$	10
$q_{C_{2\text{p,min}}}$	Minimum flow in C_2 in pump mode	$m^3\ s^{-1}$	0
$q_{C_{2\text{p,max}}}$	Maximum flow in C_2 in pump mode	$m^3\ s^{-1}$	5
	Subsystem 3		
$I_{0_{R_1}}$	Bed slope reach R_1	-	0.0025
L_{R_1}	Length reach R_1	m	10000
$w_{1_{R_1}}$	Width beginning reach R_1	m	30
$w_{N_{R_1}}$	Width end reach R_1	m	50
N_{R_1}	Number of cells used in the discretizartion of reach R_1	-	20
$k_{\text{t}_{D_1}}$	Turbine constant dam D_1	J m^{-4}	8000
$h_{R_{1\min}}$	Minimum water level at dam D_1	m	14.5
$h_{R_{1\max}}$	Maximum water level at dam D_1	m	17.5
$q_{D_{1\min}}$	Minimum turbine flow dam D_1	$m^3 s^{-1}$	5
$q_{D_{1\max}}$	Maximum turbine flow dam D_1	$m^3\ s^{-1}$	300
q_{in}	Water inflow reach R_1	$m^3 s^{-1}$	200
L_{C_1}	Distance from the beginning of the reach to duct C_1	m	5000
	Subsystem 4		
$I_{0_{R_2}}$	Bed slope of reach R_2	-	0.0015
L_{R_2}	Length reach R_2	m	8000
$w_{1_{R_2}}$	Width beginning reach R_2	m	40
$w_{N_{R_2}}$	Width end reach R_2	m	45
N_{R_2}	Number of cells used in the discretization of reach R_2	-	20
$k_{\text{t}_{D_2}}$	Turbine constant dam D_2	J m^{-4}	8000
$h_{R_{2\min}}$	Minimum water level at dam D_2	m	16.5
$h_{R_{2\max}}$	Maximum water level at dam D_2	m	19.5
$q_{D_{2\min}}$	Minimum turbine flow dam D_2	$m^3 s^{-1}$	5
$q_{D_{2\max}}$	Maximum turbine flow dam D_2	$m^3\ s^{-1}$	300
L_{T_1}	Distance from the beginning of the reach to duct T_1	m	4000
	Subsystem 5		
$I_{0_{R_3}}$	Bed slope reach R_3	-	0.002

continues...

...continued

Symbol	Description	Unit of measure	Value
L_{R_3}	Length reach R_3	m	6000
$w_{1_{R_3}}$	Width beginning reach R_3	m	40
$w_{N_{R_3}}$	Width end reach R_3	m	55
N_{R_3}	Number of cells used in the discretizartion of reach R_3	-	20
$k_{t_{D_3}}$	Turbine constant dam D_3	J m^{-4}	8000
$h_{R_{3\min}}$	Minimum water level at dam D_3	m	21.5
$h_{R_{3\max}}$	Maximum water level at dam D_3	m	24.5
$q_{D_{3\min}}$	Minimum turbine flow dam D_3	m^{3}s^{-1}	5
$q_{D_{3\max}}$	Maximum turbine flow dam D_3	m^3 s^{-1}	300
$L_{\text{tributary}}$	Distance from the beginning of the reach to tributary inflow point	m	3000
$q_{\text{tributary}}$	Tributary inflow	m^3 s^{-1}	30
	Subsystem 6		
$I_{0_{R_4}}$	Bed slope of reach R_2	-	0.0015
L_{R_4}	Length reach R_2	m	8000
$w_{1_{R_4}}$	Width beginning reach R_2	m	55
$w_{N_{R_4}}$	Width end reach R_2	m	45
N_{R_4}	Number of cells used in the discretizartion of reach R_4	-	20
$k_{t_{D_4}}$	Turbine constant dam D_4	J m^{-4}	8000
$h_{R_{4\min}}$	Minimum water level at dam D_4	m	17.5
$h_{R_{4\max}}$	Maximum water level at dam D_4	m	20.5
$q_{D_{4\min}}$	Minimum turbine flow dam D_4	m^{3}s^{-1}	5
$q_{D_{4\max}}$	Maximum turbine flow dam D_4	m^3 s^{-1}	300
L_{C_2}	Distance from the beginning of the reach to duct C_2	m	4000
	Subsystem 7		
$I_{0_{R_5}}$	Bed slope reach R_5	-	0.0025
L_{R_5}	Length reach R_5	m	6000
$w_{1_{R_5}}$	Width beginning reach R_5	m	50
$w_{N_{R_5}}$	Width end reach R_5	m	60
N_{R_5}	Number of cells used in the discretizartion of reach R_5	-	20
$k_{t_{D_5}}$	Turbine constant dam D_5	J m^{-4}	8000
$h_{R_{5\min}}$	Minimum water level at dam D_5	m	13.5
$h_{R_{5\max}}$	Maximum water level at dam D_5	m	16.5
$q_{D_{5\min}}$	Minimum turbine flow dam D_5	m^{3}s^{-1}	5
$q_{D_{5\max}}$	Maximum turbine flow dam D_5	m^3 s^{-1}	300
L_{T_2}	Distance from the beginning of the reach to duct T_2	m	3000
	Subsystem 8		
$I_{0_{R_6}}$	Bed slope of reach R_6	-	0.002
L_{R_6}	Length reach R_6	m	8000
$w_{1_{R_6}}$	Width beginning reach R_6	m	60
$w_{N_{R_6}}$	Width end reach R_6	m	80

continues...

...continued

Symbol	Description	Unit of measure	Value
N_{R_6}	Number of cells used in the discretizartion of reach R_6	-	20
$k_{\mathrm{t}_{D_6}}$	Turbine constant dam D_6	J m^{-4}	8000
$h_{R_{6\min}}$	Minimum water level at dam D_6	m	11.5
$h_{R_{6\max}}$	Maximum water level at dam D_6	m	14.5
$q_{D_{6\min}}$	Minimum turbine flow dam D_6	$\mathrm{m}^3\mathrm{s}^{-1}$	10
$q_{D_{6\max}}$	Maximum turbine flow dam D_6	$\mathrm{m}^3\ \mathrm{s}^{-1}$	300
$h_{D_{6\mathrm{out}}}$	Water level after dam D_6	m	2
	General data		
k_{str}	Gauckler-Manning-Strickler coefficient	$\mathrm{m}^{1/3}\ \mathrm{s}^{-1}$	30
g	Gravitational acceleration constant	m s^{-2}	9.81

Appendix B

Open-source distributed QP solver: PyDQP

This chapter is devoted to the presentation of the open-source QP solver, PyDQP, which implements 11 different QP solution methods based on the dual decomposition scheme. The source code of the sofware is made freely available and can be downloaded from the homepage http://pydqp.sourceforge.net/.

B.1 Overview

This section provides a short overview of PyDQP. The software provides solution methods to optimize decomposable convex quadratic programs on computer clusters

with distributed memory. The QP to be solved has to be of the form

$$\min_{x} \sum_{i=1}^{M} x_i^T H_i x_i + c_i^T x_i \tag{B.1a}$$

$$\sum_{i=1}^{M} A_i x_i = b \tag{B.1b}$$

$$A_j^{\text{eq}} x_j = b_j^{\text{eq}} \tag{B.1c}$$

$$D_j x_j \leq e_j \tag{B.1d}$$

$$\underline{x}_j \leq x \leq \overline{x}_j, \quad j = 1, \dots M, \tag{B.1e}$$

where $H_i \succ 0$.

PyDQP itself is implemented in Python and consists of the following building blocks

- *Parallelization environment*: PyDQP is essentially $M + 1$ processes running in parallel on a distributed memory system, where M processes solve small convex QPs, while a dedicated process finds the optimal dual multipliers and coordinates the rest. The coordinator process communicates with the QP solver processes via `OpenMPI` [51], an open-source message passage interface using `mpi4py` [34], a Python interface to MPI.
- *Local QP solver*: In a dual decomposition framework, several local smaller QPs have to be solved. The subproblems between two dual iterations may be similar and for this reason an active-set QP solver should be used with hot-starting capabilities, such as `qpOASES` [43], `CPLEX` (simplex) [77]. By default, the latter is used, but this can be easily exchanged with other methods.

B.2 Algorithmic features

This section shortly describes the different algorithmic features of PyDQP that are available for distributed QP solutions.

Solution methods

PyDQP supports 11 different dual decomposition methods using either only first-order or second-order derivatives.

- Gradient method: The optimal dual variables are found by a fixed stepsize gradient method.
- Gradient method with adaptive multi-stepsize: The dual gradient is adaptively scaled in each direction.
- Fast gradient method: A fixed stepsize accelerated gradient method is optimizing the dual function.
- Fast gradient with adaptive Lipschitz: An accelerated gradient method with adaptive stepsize applied in the dual space.
- Fast gradient with adaptive restart: Restarted fixed stepsize accelerated gradient method applied in the dual space.
- Global Barzilai-Borwein method: Variable stepsize method based on local Lipschitz approximation.
- Nonlinear CG method: The dual variables are optimized with a nonlinear conjugate gradient method.
- ADMM: An alternating method minimizing the augmented Lagrangian function, the multipliers are updated with a gradient method.
- Inexact Uzawa method: The same as ADMM, but with strictly convex subproblems.
- Dual Newton-CG: The dual multipliers are found by a Newton method, which is coupled with a conjugate gradient method.
- Singularity-free Newton-CG: Same as Newton-CG, but with L_2-relaxed inequalities.

For more detailed discussion of the solution methods, we refer to Chapters 5 and 7 of this thesis.

Lipschitz constant

Several solution methods require a Lipschitz constant in order to attain convergence. This constant can be given by the user directly or if nothing was given one is calculated in a distributed fashion by disregarding local inequalities.

Parallel and sequential reading of input

PyDQP is designed to be executed on parallel clusters with distributed memory systems. In such an environment, the reading of input file(s) takes place in parallel by default since it is less probable that the local memory gets full. One can easily use PyDQP on a shared memory system as well once the appropriate MPI libraries are available. In this case, in order to maintain the presumably small size memory, sequential reading of input is recommended, which can be chosen by a command line option.

Single and multiple input

Problems that need relatively small ($<$2GB) disk space can be stored in a single `.mat` file, whereas with larger problems, the partitioning of the data is necessary in order to respect file system limits. Each partition corresponds to one subproblem in the QP and is read by exactly one MPI process. The default option is single file input, while the multiple file input can be chosen by command line option.

B.3 Software design

This section discusses the structure and the building blocks of PyDQP. The PyDQP distribution consists of the following Python files

- `PyDQP.py`: The main file of PyDQP handling the command line parameters and basic error checks.
- `master.py`: Implements the `Master` class, which is instantiated by the coordinator process computing the dual step, checking convergence criterion, etc.
- `slave.py`: The `Slave` class is implemented here, which is instantiated on each QP solver process
- `cplexinterface.py`: A class providing access to the `CPLEX` QP solver using sparse matrices from the `scipy.sparse` package.
- `termcolor.py`: Package for fancy printing to the output.

Command-line arguments

One can get an exhaustive list of options by passing "--help" or "-h" to PyDQP.py. This generates the following output.

```
Usage: ./PyDQP.py [options] FILENAME

Options:
  -h, --help            show this help message and exit
  -d DUAL_ITER, --dual-iteration=DUAL_ITER
                        number of maximal dual iterations [default: 300000]
  -L LIPSCHITZ, --lipschitz=LIPSCHITZ
                        Lipschitz constant to be used (calculated if none is
                        given), interpreted as penalty parameter with ADMM and
                        IUM
  -m METHOD, --method=METHOD
                        chooses the dual method to be used [default: 0]
  -S, --sequential      Reads the input file sequentially (recommended for
                        shared memory), by default reading is parallel.
  --help-method         prints the list of available dual methods
  -f FEAS_EPSILON, --feas-epsilon=FEAS_EPSILON
                        Tolerance of primal infeasibility [default: 0.0001]
```

We can get the list of the available-methods by passing "-help-method", which prints the following list.

```
Possible methods to be given with -m NUM:
    0 : gradient method
    1 : gradient method with adaptive multi-stepsize
    2 : fast gradient method
    3 : fast gradient with adaptive Lipschitz
    4 : fast gradient with adaptive restart
    5 : global Barzilai-Borwein method
    6 : nonlinear conjugate gradient (CG) method
    7 : alternating direction method of multipliers (ADMM)
    8 : fine-tuned fast gradient method
    9 : inexact Uzawa method
   10 : decomposition algorithm with two dual steps
   11 : CPLEX
```

B.4 Tutorial examples

This section presents examples how to use PyDQP. Assume that we want to solve a problem called AUG2DC-5.mat, which consists of 5 coupled subproblems. We give the following command.

```
$ mpirun -np 6 PyDQP.py AUG2DC-5.mat
```

The `mpirun` binary is shipped with the `Open MPI` distribution, which initializes and coordinates the MPI environment in a cluster environment. [1] We request for $5+1=6$ MPI processes each of which executes `PyDQP.py` with the input filename as first argument. After the calculation of a Lipschitz constant, the gradient method, as the default dual scheme, is run until one of the convergence criteria holds.

Entering

```
$ mpirun -np 6 PyDQP.py -m 2 -L 8.5 -d 1000 -S AUG2DC-5.mat
```

executes a fast gradient method with a Lipschitz constant 8.5 allowing a maximum number of 1000 dual iterations and reading the input file sequentially.

The last command

```
$ mpirun -np 6 PyDQP.py -m 7 -L 1E2 -d 150 -f 1E-5 AUG2DC-5.mat
```

executes an ADMM method with penalty parameter 10^2 allowing only 150 dual iterations and requesting a tolerance of 10^{-5}.

[1] In a cluster environment there is often an extra layer of a scheduler, which schedules the different jobs of the cluster users.

Bibliography

[1] PyDQP Homepage. http://pydqp.sourceforge.net.

[2] *Gurobi optimizer reference manual*, 2012.

[3] Andersen, E. D., Roos, C., and Terlaky, T. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming 95*, 2 (2003), 249–277.

[4] Andersson, J., Åkesson, J., and Diehl, M. CasADi – A symbolic package for automatic differentiation and optimal control. In *Recent Advances in Algorithmic Differentiation* (Berlin, 2012), S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds., Lecture Notes in Computational Science and Engineering, Springer.

[5] Arutyunov, A., and Izmailov, A. Sensitivity analysis of abnormal cone-constrained optimization problems. *Computational Mathematics and Mathematical Physics 44*, 4 (2004), 552–574.

[6] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Der, H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* Philadalphia: Society for Industrial and Applied Mathematics. Also available as postscript file on http://www.netlib.orgtemplatesTemplates.html, 1994.

[7] Barzilai, J., and Borwein, J. M. Two-Point Step Size Gradient Methods. *IMA J Numer Anal 8*, 1 (Jan. 1988), 141–148.

[8] Becker, S., Candès, E., and Grant, M. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation 3* (2011), 165–218.

[9] Benzi, M. Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of Computational Physics 182* (2002), 418–477.

[10] Bergamaschi, L., Gondzio, J., and Zilli, G. Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and Applications 28*, 2 (2004), 149–171.

[11] Bertsekas, D. *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[12] Bertsekas, D. *Dynamic Programming and Optimal Control*, 3rd ed., vol. 2. Athena Scientific, 2007.

[13] Bertsekas, D., and Tsitsiklis, J. N. *Parallel and distributed computation: Numerical methods*. Prentice Hall, 1989.

[14] Bertsekas, D. P. *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. Athena Scientific, 1996.

[15] Betts, J. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, Philadelphia, 2001.

[16] Biegler, L. An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing 46* (2007), 1043–1053.

[17] Biegler, L. T. *Nonlinear Programming*. MOS-SIAM Series on Optimization. SIAM, 2010.

[18] Björck, r. A direct method for sparse least squares problems with lower and upper bounds. *Numerische Mathematik 54*, 1 (1988), 19–32.

[19] Bock, H., Diehl, M., Kostina, E., and Schlöder, J. Constrained Optimal Feedback Control of Systems Governed by Large Differential Algebraic Equations. In *Real-Time and Online PDE-Constrained Optimization*, L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, Eds. SIAM, 2007, pp. 3–22.

[20] Bock, H., and Plitt, K. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest* (1984), Pergamon Press, pp. 242–247.

[21] Bojańczyk, A. Optimal asynchronous newton method for the solution of nonlinear equations. *J. ACM 31*, 4 (1984), 792–803.

[22] Bonnans, J. Local Analysis of Newton-Type Methods for Variational Inequalities and Nonlinear Programming. *Appl. Math. Optim 29* (1994), 161–186.

[23] Boyd, S., Parikh, N., Chu, E., and Peleato, B. Distributed optimization and statistics via alternating direction method of multipliers. *Foundations and Trends in Machine Learning 3*, 1 (2011), 1–122.

[24] Boyd, S., and Vandenberghe, L. *Convex Optimization*. University Press, Cambridge, 2004.

[25] Breckpot, M. *Flood control of river systems with Model Predictive Control*. PhD thesis, KU Leuven, 2013.

[26] Camponogara, E., Jia, D., Krogh, B., and Talukdar, S. Distributed model predictive control. *Control Systems Magazine 22*, 1 (2002), 44–52.

[27] Cauchy, A. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris 25*, 1847 (1847), 536–538.

[28] Choi, J., Dongarra, J., Ostrouchov, S., Petitet, A., Walker, D., and Whaley, R. A proposal for a set of parallel basic linear algebra subprograms. In *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*, J. Dongarra, K. Madsen, and J. Waśniewski, Eds., vol. 1041 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1996, pp. 107–114.

[29] Clarke, F. *Optimization and Nonsmooth Analysis*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1990.

[30] Conte, C., Summers, T., Zeilinger, M., Morari, M., and Jones, C. Computational Aspects of Distributed Optimization in Model Predictive Control. In *Conference on Decision and Control (CDC)* (Maui, HI, USA, Dec. 2012).

[31] Cottle, R. W., Duvall, S. G., and Zikan, K. A lagrangean relaxation algorithm for the constrained matrix problem. *Naval Research Logistics Quarterly 33*, 1 (1986), 55–76.

[32] Cryer, C. The solution of a quadratic programming problem using systematic overrelaxation. *SIAM Journal on Control 9*, 3 (1971), 385–392.

[33] Dai, Y., and Liao, L. R-linear convergence of the barzilai and borwein gradient method. *IMA Journal of Numerical Analysis 22*, 1 (2002), 1–10.

[34] Dalcín, L., Paz, R., Storti, M., and D'Elía, J. MPI for Python: Performance improvements and MPI-2 extensions. *J. Parallel Distrib. Comput. 68*, 5 (May 2008), 655–662.

[35] Danskin, J. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics 14*, 4 (1966), 641–664.

[36] Dembo, R. S., and Tulowitzki, U. On the minimization of quadratic functions subject to box constraints. Tech. rep., Yale University, Department of Computer Science, 1984.

[37] Deng, W., and Yin, W. On the global and linear convergence of the generalized alternating direction method of multipliers. Tech. rep., Rice University CAAM, 2012. TR12-14.

[38] Diehl, M., Walther, A., Bock, H., and Kostina, E. An adjoint-based SQP algorithm with quasi-Newton Jacobian updates for inequality constrained optimization. *Optim. Methods Softw. 25*, 4 (2010), 531–552.

[39] Diehl, M., Walther, A., Bock, H. G., and Kostina, E. An adjoint-based SQP algorithm with quasi-Newton Jacobian updates for inequality constrained optimization. *Optimization Methods and Software 25* (2010), 531–552.

[40] Doan, M. D., Keviczky, T., and De Schutter, B. Application of distributed and hierarchical model predictive control in hydro power valleys. In *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference* (Leeuwenhorst, Netherlands, 2012), vol. 4, pp. 375–383.

[41] Dolan, E., and Moré, J. Benchmarking optimization software with performance profiles. *Math. Program. 91* (2002), 201–213.

[42] Du, Q., and Gunzburger, M. A gradient method approach to optimization-based multidisciplinary simulations and nonoverlapping domain decomposition algorithms. *SIAM Journal on Numerical Analysis 37*, 5 (2000), 1513–1541.

[43] Ferreau, H., Kirches, C., Potschka, A., Bock, H., and Diehl, M. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation* (2013). (under review).

[44] Ferreau, H., Kozma, A., and Diehl, M. A parallel active-set strategy to solve sparse parametric quadratic programs arising in MPC. In *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands* (2012).

[45] Ferreau, H. J., Bock, H. G., and Diehl, M. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control 18*, 8 (2008), 816–830.

[46] Fiacco, A. *Introduction to sensitivity and stability analysis in nonlinear programming*. Academic Press, New York, 1983.

[47] Findeisen, W., Bailey, F. N., Malinowski, K., Tatjewski, P., and Wozniak, A. *Control and Coordination in Hierarchical Systems*. Wiley, 1980.

[48] Frasch, J. V., Vukov, M., Ferreau, H., and Diehl, M. A dual Newton strategy for the efficient solution of sparse quadratic programs arising in SQP-based nonlinear MPC, 2013. Optimization Online 3972.

[49] Friedlander, A., Martinez, J., and Raydan, M. A new method for large-scale box constrained convex quadratic minimization problems. *Optimization Methods and Software 5* (1995), 57–74.

[50] Gabay, D., and Mercier, B. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications 2*, 1 (1976), 17 – 40.

[51] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting* (Budapest, Hungary, 2004), pp. 97–104.

[52] Gander, M. J., and Stuart, A. M. Space-time continuous analysis of waveform relaxation for the heat equation. *SIAM Journal on Scientific Computing 19* (1997), 2014–2031.

[53] Gander, M. J., and Vandewalle, S. Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput. 29* (2007), 556–578.

[54] Geiser, J. *Decomposition Methods for Differential Equations*. Chapman & Hall/CRC, 2009.

[55] Gerdts, M. A nonsmooth newton's method for control-state constrained optimal control problems. In *5th Vienna International Conference on Mathematical Modelling/Workshop on Scientific Computing in Electronic Engineering of the 2006 International Conference on Computational Science/Structural Dynamical Systems: Computational Aspects* (2008), vol. 79, pp. 925 – 936.

[56] Gerdts, M., and Kunkel, M. A nonsmooth Newton's method for discretized optimal control problems with state and control constraints. *Journal of Industrial and Management Optimization 4* (2008), 247–270.

[57] Gertz, E., and Wright, S. Object-Oriented Software for Quadratic Programming. *ACM Transactions on Mathematical Software 29*, 1 (2003), 58–81.

[58] Gill, P., Murray, W., and Saunders, M. *User's Guide For QPOPT 1.0: A Fortran Package For Quadratic Programming*, 1995.

[59] Gill, P., Murray, W., Saunders, M., and Wright, M. *User's guide for SOL/QPSOL: a Fortran package for quadratic programming*, vol. SOL 83-7 of *Technical Report*. Stanford University, Systems Optimization Laboratory, Department of Operations Research, 1983.

[60] Giselsson, P., Doan, M. D., Keviczky, T., Schutter, B. D., and Rantzer, A. Accelerated gradient methods and dual decomposition in distributed model predictive control. *Automatica 49*, 3 (2013), 829 – 833.

[61] Giselsson, P., and Rantzer, A. Distributed model predictive control with suboptimality and stability guarantees. In *Decision and Control (CDC), 2010 49th IEEE Conference on* (2010), pp. 7272–7277.

[62] Glowinski, R., and Marrocco, A. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Rev. Française Automat. Informat. Recherche Opérationnelle, RAIRO Analyse Numérique 9*, R-2 (1975), 41–76.

[63] Goldfarb, D., and Idnani, A. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming 27* (1983), 1–33.

[64] Goldstein, T., O´Donoghue, B., and Setzer, S. Fast alternating direction optimization methods. Tech. report., Department of Mathematics, University of California, Los Angeles, USA, May 2012.

[65] Gondzio, J. Matrix-free interior point method. *Computational Optimization and Applications* (2010), 1–24.

[66] Gould, N., Orban, D., and Toint, P. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software 29*, 4 (2004), 353–372.

[67] Grünig, A. Distributed large-scale convex quadratic programming via dual decomposition. Master's thesis, KU Leuven, 2013.

[68] Gunzburger, M., and Kwon Lee, H. An optimization-based domain decomposition method for the navier-stokes equations. *SIAM Journal on Numerical Analysis 37*, 5 (2000), 1455–1480.

[69] Gupta, A., Koric, S., and George, T. Sparse matrix factorization on massively parallel computers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 1:1–1:12.

[70] Hager, W. W., and Gowda, M. S. Stability in the presence of degeneracy and error estimation. *Mathematical Programming 85*, 1 (1999), 181–192.

[71] Hairer, E., Nørsett, S., and Wanner, G. *Solving Ordinary Differential Equations I*, 2nd ed. Springer Series in Computational Mathematics. Springer, Berlin, 1993.

[72] Hairer, E., and Wanner, G. *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*, 2nd ed. Springer, Berlin Heidelberg, 1991.

[73] He, B., and Yuan, X. On non-ergodic convergence rate of Douglas-Rachford alternating direction method of multipliers. Tech. rep., 2012.

[74] Heinkenschloss, M., and Vicente, L. Analysis of Inexact Trust-Region SQP Algorithms. *SIAM Journal on Optimization 12*, 2 (2001), 283–302.

[75] Hestenes, M. R. Multiplier and gradient methods. *Journal of Optimization Theory and Applications 4* (1969), 303–320.

[76] Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., and Woodward, C. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software 31* (2005), 363–396.

[77] IBM Corp. *IBM ILOG CPLEX V12.1, User's Manual for CPLEX*, 2009.

[78] Isaacson, E. *Analysis of numerical methods*. Courier Dover Publications, 1994.

[79] Izmailov, A., and Solodov, M. Inexact Josephy-Newton framework for generalized equations and its applications to local analysis of Newtonian methods for constrained optimization. *Comput. Optim. Appl. 46* (2010), 347–368.

[80] Jaeger, H., and Sachs, E. Global convergence of inexact reduced SQP methods. *Optimization Methods and Software 7*, 2 (1997), 83–110.

[81] Jensen, T., Jørgensen, J., Hansen, P., and Jensen, S. Implementation of an optimal first-order method for strongly convex total variation regularization. *BIT Numerical Mathematics 52* (2012), 329–356.

[82] Jiang, Y.-L. Periodic waveform relaxation solutions of nonlinear dynamic equations. *Applied Mathematics and Computation 135*, 2–3 (2003), 219 – 226.

[83] Kozma, A., Andersson, J., Savorgnan, C., and Diehl, M. Distributed Multiple Shooting for Optimal Control of Large Interconnected Systems. In *Proceedings of the International Symposium on Advanced Control of Chemical Processes* (2012).

[84] Kozma, A., Conte, C., and Diehl, M. Benchmarking large scale distributed convex quadratic programming algorithms. *Optimization Methods & Software* (2013). (accepted for publication).

[85] Kozma, A., Frasch, J. V., and Diehl, M. A Distributed Method for Convex Quadratic Programming Problems Arising in Optimal Control of Distributed Systems. In *Proceedings of the 52nd Conference on Decision and Control (CDC)* (2013).

[86] Kozma, A., Klintberg, E., Gros, S., and Diehl, M. An improved distributed dual Newton-CG method for convex quadratic programming problems. In *American Control Conference* (2014). (submitted for publication).

[87] Kungurtsev, V., Kozma, A., and Diehl, M. Linear convergence of distributed multiple shooting. In *European Control Conference* (2014). (submitted for publication).

[88] Larsson, T., Patriksson, M., and Strömberg, A.-B. Ergodic, primal convergence in dual subgradient schemes for convex programming. *Mathematical Programming 86*, 2 (1999), 283–312.

[89] Lasdon, L. S. *Optimization theory for Large Systems*. Dover, 1970.

[90] Lelarasmee, E. *The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits: Theory and Applications*. PhD thesis, EECS Department, University of California, Berkeley, 1982.

[91] Lemaréchal, C. Lagrangian relaxation. In *Computational Combinatorial Optimization*, M. Jünger and D. Naddef, Eds., vol. 2241 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 112–156.

[92] Lewis, J., and van de Geijn, R. Distributed memory matrix-vector multiplication and conjugate gradient algorithms. In *Supercomputing '93. Proceedings* (nov. 1993), pp. 484 – 492.

[93] Li, W., and Swetits, J. A new algorithm for solving strictly convex quadratic programs. *SIAM Journal of Optimization 7*, 3 (1997), 595–619.

[94] Maes, C. *A Regularized Active-Set Method for Sparse Convex Quadratic Programming*. PhD thesis, Stanford University, 2010.

[95] Maestre, J., de la Peña, D. M., Camacho, E., and Alamo, T. Distributed model predictive control based on agent negotiation. *Journal of Process Control 21*, 5 (2011), 685 – 697. Special Issue on Hierarchical and Distributed Model Predictive Control.

[96] Maestre, J. M., Ridao, M. A., Kozma, A., Savorgnan, C., Diehl, M., Doan, M. D., Sadowska, A., Keviczky, T., Schutter, B. D., Scheu, H., Marquardt, W., Valencia, F., and Espinosa, J. A comparison of distributed MPC schemes on a hydro power plant benchmark. *Optimal Control Applications and Methods* (2013). (submitted for publication).

[97] Magni, L., and Scattolini, R. Stabilizing decentralized model predictive control of nonlinear systems. *Automatica 42* (2006), 1231–1236.

[98] Maros, I., and Mészáros, C. A repository of convex quadratic programming problems. *Optimization Methods and Software 11* (1999), 431–449.

[99] Mesarovic, M., Macko, D., and Takahara, Y. *Theory of Hierarchical, Multilevel Systems*. Academic Press, 1970.

[100] Monticelli, A., Pereira, M. V. F., and Granville, S. Security-constrained optimal power flow with post-contingency corrective rescheduling. *Power Systems, IEEE Transactions on 2*, 1 (1987), 175–180.

[101] Necoara, I., Nedelcu, V., and Dumitrache, I. Parallel and distributed optimization methods for estimation and control in networks. *Journal of Process Control 21*, 5 (2011), 756 – 766. Special Issue on Hierarchical and Distributed Model Predictive Control.

[102] Necoara, I., and Suykens, J. Applications of a smoothing technique to decomposition in convex optimization. *IEEE Trans. Automatic control 53*, 11 (2008), 2674–2679.

[103] Necoara, I., and Suykens, J. Interior-point Lagrangian decomposition method for separable convex optimization. *J. Optim. Theory and Appl. 143*, 3 (2009), 567–588.

[104] Nedić, A., and Ozdaglar, A. Approximate primal solutions and rate analysis for dual subgradient methods. *SIAM Journal on Optimization 19*, 4 (2009), 1757–1780.

[105] Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. *Doklady AN SSSR 269*, translated as Soviet Math. Dokl. (1983), 543–547.

[106] Nesterov, Y. *Introductory lectures on convex optimization: a basic course*, vol. 87 of *Applied Optimization*. Kluwer Academic Publishers, 2004.

[107] Nesterov, Y. Gradient methods for minimizing composite objective function. *CORE Discussion paper 76* (2007).

[108] Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization 22*, 2 (2012), 341–362.

[109] Nikolayzik, T., Büskens, C., and Gerdts, M. Nonlinear large-scale Optimization with WORHP. In *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference* (2010).

[110] Nocedal, J., and Wright, S. *Numerical Optimization*, 2 ed. Springer Series in Operations Research and Financial Engineering. Springer, 2006.

[111] O´Donoghue, B., and Candès, E. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics* (2013), 1–18.

[112] O'Leary, D. P. A generalized conjugate gradient algorithm for solving a class of quadratic programming problems. *Linear Algebra and its Applications 34*, 0 (1980), 371 – 399.

[113] Patrinos, P., Sopasakis, P., and Sarimveis, H. A global piecewise smooth newton method for fast large-scale model predictive control. *Automatica 47* (2011), 2016–2022.

[114] Pješivac-Grbović, J., Angskun, T., Bosilca, G., Fagg, G. E., Gabriel, E., and Dongarra, J. J. Performance analysis of MPI collective operations. *Cluster Computing 10*, 2 (2007), 127–143.

[115] Powell, M. Algorithms for nonlinear constraints that use Lagrangian functions. *Mathematical Programming 14*, 3 (1978), 224–248.

[116] Qi, L. Convergence analysis of some algorithms for solving nonsmooth equations. *Mathematics of Operations Research 18*, 1 (1993), pp. 227–244.

[117] Qi, L., and Sun, J. A nonsmooth version of Newton's method. *Mathematical Programming 58* (1993), 353–367.

[118] Quarteroni, A., and Valli, A. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, 1999.

[119] Rawlings, J., and Mayne, D. *Model Predictive Control: Theory and Design*. Nob Hill, 2009.

[120] Rawlings, J., and Stewart, B. Coordinating multiple optimization-based controllers: New opportunities and challenges. *Journal of Process Control 18* (2008), 839–845.

[121] Raydan, M. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization 7*, 1 (1997), 26–33.

[122] Richards, A., and How, J. Robust distributed model predictive control. *International Journal of Control 80*, 9 (2007), 1517–1531.

[123] Richter, S., Morari, M., and Jones, C. Towards computational complexity certification for constrained MPC based on Lagrange Relaxation and the fast gradient method. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)* (2011), pp. 5223 –5229.

[124] Robinson, S. M. Stability theory for systems of inequalities, Part II: Differentiable nonlinear systems. *SIAM Journal on Numerical Analysis 13*, 4 (1976), 497–513.

[125] Saunders, M. A. Cholesky-based methods for sparse least squares: the benefits of regularization. In *Linear and nonlinear conjugate gradient-related methods (Seattle, WA, 1995)*. SIAM, Philadelphia, PA, 1996, pp. 92–100.

[126] Savorgnan, C., and Diehl, M. Control benchmark of a hydro power plant. Tech. report, Optimization in Engineering Center, KU Leuven,, 2010.

[127] Savorgnan, C., Kozma, A., Andersson, J., and Diehl, M. Adjoint-Based Distributed Multiple Shooting for Large-Scale Systems. In *18th IFAC World Congress* (2011), vol. 18.

[128] Savorgnan, C., Romani, C., Kozma, A., and Diehl, M. Multiple shooting for distributed systems with applications in hydro electricity production. *Journal of Process Control 21* (2011), 738–745.

[129] Scattolini, R. Architectures fior distributed and hierarchical model predictive control. *Journal of Process Control 19* (2009), 723–731.

[130] Scheu, H., and Marquardt, W. Sensitivity-based coordination in distributed model predictive control. *Journal of Process Control 21*, 5 (2011), 715 – 728. Special Issue on Hierarchical and Distributed Model Predictive Control.

[131] Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[132] Shor, N. Z., Kiwiel, K. C., and Ruszcayǹski, A. *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[133] Smith, B., Bjorstad, P., and Gropp, W. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 2004.

[134] Tran-Dinh, Q. *Sequential Convex Programming and Decomposition Approaches for Nonlinear Optimization*. Phd thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, November 2012.

[135] Tseng, P. Dual ascent methods for problems with strictly convex costs and linear constraints: A unified approach. *SIAM Journal on Control and Optimization 28*, 1 (1990), 214–242.

[136] Tsiaflakis, P., and Moonen, M. A flexible and real-time constrained controller for sparse linear zero-forcing based dsl vectoring. In *IEEE International Conference on Communications* (2013).

[137] Tsiaflakis, P., Vangorp, J., Moonen, M., Verlinden, J., and Ysebaert, G. Partial crosstalk cancellation in a multi-user xdsl environment. In *IEEE International Conference on Communications, ICC '06.* (2006), vol. 7, pp. 3264–3269.

[138] Tsitsiklis, J., Bertsekas, D., and Athans, M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control 31* (Sep 1986), 803–812.

[139] Valencia, F., Espinosa, J. J., De Schutter, B., and Stanková, K. Feasible-cooperation distributed model predictive control scheme based on game theory. In *Proceedings of the 18th IFAC World Congress* (2010).

[140] Vanderbei, R. J. Loqo: An interior point code for quadratic programming. *Optimization methods and software 11*, 1-4 (1999), 451–484.

[141] Vandewalle, S., and Piessens, R. Numerical experiments with nonlinear multigrid waveform relaxation on a parallel processor. *Applied Numerical Mathematics 8*, 2 (1991), 149 – 161.

[142] Venkat, A. *Distributed Model Predictive Control: Theory and Applications*. PhD thesis, University of Wisconsin-Madison, 2006.

[143] Wächter, A., and Biegler, L. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming 106*, 1 (2006), 25–57.

[144] Wright, S. *Primal-Dual Interior-Point Methods*. SIAM Publications, Philadelphia, 1997.

[145] Wright, S. J. An algorithm for degenerate nonlinear programming with rapid local convergence. *SIAM Journal on Optimization 15*, 3 (2005), 673–696.

[146] Zafiriou, E. Robust model predictive Control of processes with hard constraints. *Computers & Chemical Engineering 14*, 4–5 (1990), 359–371.

[147] Zhang, X., Burger, M., and Osher, S. A unified primal-dual algorithm framework based on bregman iteration. *J. Sci. Comput. 46*, 1 (Jan. 2011), 20–46.

[148] Zhou, B., Gao, L., and Dai, Y. Gradient methods with adaptive step-sizes. *Computational Optimization and Applications 35* (2006), 69–86.

List of publications

Book chapters

1. Kozma, A., Savorgnan, C., and Diehl, M. Distributed multiple shooting for large scale nonlinear systems. In *Distributed Model Predictive Control Made Easy*, J. M. Maestre and R. R. Negenborn, Eds., vol. 69 of *Intelligent Systems, Control and Automation: Science and Engineering*. Springer Netherlands, 2014, pp. 327–340.

Conference proceedings

1. H.J. Ferreau, A. Kozma, and M. Diehl. A parallel active-set strategy to solve sparse parametric quadratic programs arising in MPC. In *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012.

2. A. Kozma, J. Andersson, C. Savorgnan, and M. Diehl. Distributed Multiple Shooting for Optimal Control of Large Interconnected Systems. In *Proceedings of the International Symposium on Advanced Control of Chemical Processes*, 2012.

3. A. Kozma, J. V. Frasch, and M. Diehl. A Distributed Method for Convex Quadratic Programming Problems Arising in Optimal Control of Distributed Systems. In *Proceedings of the 52nd Conference on Decision and Control (CDC)*, 2013.

4. C. Savorgnan, A. Kozma, J. Andersson, and M. Diehl. Adjoint-Based Distributed Multiple Shooting for Large-Scale Systems. In *18th IFAC World Congress*, volume 18, 2011.

Journal papers

1. C. Savorgnan, C. Romani, A. Kozma, and M. Diehl. Multiple shooting for distributed systems with applications in hydro electricity production. *Journal of Process Control*, 21:738–745, 2011.

2. A. Kozma, C. Conte, and M. Diehl. Benchmarking large scale distributed convex quadratic programming algorithms. *Optimization Methods & Software*, 2013. (accepted for publication).

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
STADIUS CENTER
Kasteelpark Arenberg 10, bus 2446
B-3001 Heverlee
Attila.Kozma@esat.kuleuven.be
http://www.esat.kuleuven.be