Fast Optimization Methods for Model Predictive Control via Parallelization and Sparsity Exploitation

Haoyang Deng

Department of Systems Science, Graduate School of Informatics

Kyoto University

A thesis submitted for the degree of

Doctor of Philosophy in Informatics

2020

Supervisor:

Prof. Toshiyuki Ohtsuka

Examination committee:

Prof. Toshiyuki Ohtsuka

Prof. Manabu Kano

Prof. Yoshito Ohta

Abstract

Model predictive control (MPC) has been widely applied to the process industry in the past decades, and it has kept evolving and showing great potential in robotics, automobile, power electronics, aerospace, and various industries. The major challenge of MPC comes from solving the arising optimization problems at every sampling instant. This thesis explores novel numerical optimization methods that enable the use of MPC to control applications with high sampling rates, complex and large-scale dynamics, and resource-limited hardware.

For linear MPC, we present a combined first- and second-order method, which requires only first-order derivatives of value functions and incorporates fixed second-order information. The convergence is guaranteed under the framework of the majorization-minimization principle. Numerical experiments indicate that the proposed method can obtain a moderately accurate solution with a small number of cheap iterations.

For nonlinear MPC (NMPC), we present two fast optimization methods: a parallel Newton-type method and a Jacobi-type method. The parallel method splits the NMPC problem into subproblems along the prediction horizon so that these subproblems can be computed simultaneously, and only a very limited communication is conducted between these subproblems. The parallel method is not only highly parallelizable but shows a superlinear rate of convergence. An efficient implementation of the parallel method tailored to multi-core processors is presented. The implementation shows a significant speedup in computation time with respect to other state-of-the-art toolkits. The Jacobi method is designed for large-scale NMPC problems, which are generally sparse. The upper- and lower-layer sparsities arising in the NMPC problem are exploited in the Jacobi method. In particular, we concentrate on systems governed by partial differential equations, and a speedup of two orders of magnitude is observed in the numerical example.

Acknowledgements

PhD study is a journey full of adventure, passion, loneliness, challenge and uncertainty. Looking back on the past four years, I consider myself extremely fortunate to have the opportunity to pursue my PhD degree on such an interesting topic. I owe a debt of gratitude to many people for helping me to be where I am today. This thesis would never have been able to finish without the help I have gotten along the journey.

My first and greatest thanks goes to my supervisor, Prof. Toshiyuki Ohtsuka, who has guided me with great patience through the wonderful world of research. Thank him for his continuous support and encouragement and offering me great freedom to explore research topics and ideas. His experience and vast knowledge helped shape my research. My examination committee members Prof. Ohtsuka, Prof. Kano (also my co-supervisor) and Prof. Ohta deserve my special thanks for their valuable advices and suggestions. In addition, I would like to thank Prof. Maestre for the inspiring discussions and collaborations.

I also thank the Ohtsuka lab's members and stuffs for creating an inspiring and comfortable working environment. I specially thank Dr. Okajima, Ms. Suzuki, Mr. Iori, Mr. Jingyu, Mr. Yuhan and Mr. Kangyu for the delightful conversations and discussions. I would like to thank all my friends for the enjoyable activities and memorable experience that make my life colorful in Kyoto.

I would also like to thank the Yamaoka Scholarship Foundation and JSPS KAKENHI for the financial support of my PhD study.

Last but not least, I want to thank my parents and girlfriend Hui for their unconditional love and support through all the discouragements of my life. The concern and help from my family, friends and colleagues during the outbreak of COVID-19 are gratefully acknowledged.

Contents

Notation

For vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, a matrix $A \in \mathbb{R}^{n\times n}$, and a differentiable function $f(x, y) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$, we have the following notation:

Notation	Meaning
$x_{(i)}$	i-th component of x
x^k	k-th iteration of x
x^*	Optimal value of x
$\lvert x\rvert$	Element-wise absolute value of x
x^T	Vector transpose of x
$\lVert x\rVert$	Euclidean norm of x (ℓ_2 norm of x)
$\lVert x\rVert_1$	$\sum_{i=1}^n \lvert x_{(i)}\rvert$ (ℓ_1 norm of x)
$\lVert x\rVert_A$	$\sqrt{x^T Ax}$ (weighted norm of x)
A^T	Matrix transpose of A
$A > 0$	A is positive-definite
$A \geq 0$	A is positive-semidefinite
$\lVert A\rVert$	Frobenius norm of A
$\rho(A)$	Spectral radius of A
$f_{(i)}(x, y)$	i-th component of $f(x, y)$
$\nabla_x f$	Jacobian matrix of $f(x, y)$ with respect to x ($\nabla_x f \in \mathbb{R}^{p\times n}$)
$\nabla^2_{xx} f$	Second-order derivative of $f(x, y)$ ($\nabla^2_{xx} f \in \mathbb{R}^{p\times n\times n}$)
$\nabla^2_{xy} f$	Second-order derivative of $f(x, y)$ ($\nabla^2_{xy} f \in \mathbb{R}^{p\times n\times m}$)
$\mathbb{R}_{\geq 0}$	Set of non-negative real numbers
I, I_n	Identity matrix, identity matrix with n rows and columns
0, 0_n	Zero or zero matrix, zero matrix with n rows and columns
$\odot$	Element-wise product
$\oslash$	Element-wise division

Chapter 1

Introduction

1.1 Motivation

In the recent years, industrial systems are becoming more and more complex and sophisticated, which brings great challenges to the control methods in order to deal with these complexities, such as multiple variables, nonlinearities, constraints, and uncertainties. Among the emerging advanced control methods, model predictive control (MPC) has demonstrated its ability and potential in the complex industrial applications. In the process industry, MPC has been widely used in the areas of refining, petrochemicals, chemicals, paper making, etc. (see, e.g., Qin and Badgwell (2003)). The automobile industry has reported the first mass production (Bemporad, Bernardini, Long, & Verdejo, 2018) of MPC for engine torque control. The first use of MPC in outpatient wearable artificial pancreas can be found in Del Favero et al. (2014). Moreover, MPC has found applications in autonomous driving, robotics, aerospace, etc. One of the major reasons for the success of MPC is that the MPC problem formulation is unified in the optimization framework, which is powerful enough to handle the complexities. The inherent constraint handling ability of MPC allows safety guarantees and close operations to constraints and hence increases profit. The solution details of the underlying optimization problem are hidden from the users such that the MPC controller can be easily designed and implemented. Moreover, there are several extra reasons for MPC to be successful in multiple industries:

- It does not require deep mathematical knowledge and is easy to understand.

- Its design and implementation procedures are systematic.

- It is easy to tune, maintain, and upgrade.

MPC is an optimization-based control method that optimizes the future behavior of a system by finding a sequence of the optimal control inputs along the prediction horizon, and only the first control input is implemented. In the early stages of MPC, there are two classical formulations: dynamic matrix control (DMC) (Cutler & Ramaker, 1980) and generalized predictive control (GPC) (Clarke, Mohtadi, & Tuffs, 1987). In DMC, a quadratic cost is minimized along the prediction horizon and the future behavior is predicted based on non-parametric step response models, where the open-loop stability of the system is assumed. To tackle the limitations on the system, GPC uses parameterized prediction models, such as auto-regressive and moving-average models (Clarke et al., 1987) and state-space models (Ordys & Clarke, 1993). Note that DMC and GPC introduce no constraints and formulate unconstrained least-square problems so that the sequence of the optimal control inputs can be explicitly calculated. Although DMC and GPC deliver excellent control performance for unconstrained optimal control problems, they cannot handle constraints, e.g., the physical limitations of actuators. Garcia and Morshedi (1986) addressed this problem by integrating constraint handling in DMC, which forms the modern MPC problem structures, i.e., with costs, dynamical models, and constraints. Specifically, MPC problems with quadratic costs, linear dynamical models, and linear constraints, such as DMC, GPC and constrained DMC, are called linear MPC, which has been widely used (see Qin and Badgwell (2003) for the industrial application survey).

Since MPC is formulated in the optimization framework, it offers high degrees of freedom to customize the cost function, dynamical model, and constraints. The generalization of MPC, or nonlinear MPC (NMPC), deals explicitly with nonlinear dynamics and constraints. Furthermore, non-quadratic costs, e.g., the economic cost that directly maximizes the profit, can be imposed. Due to the flexibility of NMPC, NMPC is significantly more powerful than linear MPC and meanwhile, more computationally expensive. Compared with linear MPC, the computational difficulties of NMPC come from the following aspects:

- Linear MPC problems are essentially quadratic programs (QPs), while NMPC problems are nonlinear programs (NLPs), which are generally nonconvex. Extra steps, e.g., line search, need to be made to handle the nonconvexity such that the convergence to at least a local minimum can be guaranteed.

- When a continuous-time nonlinear dynamical system is discretized, there is always some amount of discretization error. Different discretization methods need to be considered in different contexts.

- The nonlinear functions and their Jacobian and Hessian matrices need to be evaluated along the prediction horizon at every sampling instant. Moreover, Hessian approximation techniques need to be applied when the exact Hessian matrices are difficult to obtain.
- The behavior of a nonlinear system is difficult to predict and evaluate unless a long prediction horizon is chosen, which leads to a large-scale optimization problem. Moreover, for NMPC problems with long prediction horizons, simulation of the system dynamics might diverge easily for unstable systems.

With these difficulties, real-time optimization for NMPC is much more challenging than linear MPC. Although real-time optimization for both linear MPC and NMPC has been achieved on high performance computers for most of the applications, there is no end to find more efficient and numerically robust solution methods to broaden the range of application of MPC to low-cost hardware and fast-sampled, complicated, and large-scale systems.

The aim of this thesis is to provide fast optimization methods that push the boundaries of real-time optimization for linear MPC and NMPC.

1.2 Overview of real-time optimization methods for MPC

Since linear MPC and NMPC problems are essentially QPs and NLPs, respectively, off-the-shelf solvers for general optimization problems can in principle be used. However, general-purpose solvers do not exploit the particular structure of MPC, and thus are not efficient. Considerable efforts and progress have been made toward the real-time optimization methods for MPC in recent years. The optimization methods tailored to MPC can be generally categorized into explicit and iterative methods as shown in Fig. 1.1.

Using the fact that the underlying QP in linear MPC is parametric to the initial state and the optimal control input is an affine function of the initial state (see, e.g., Bemporad, Morari, Dua, and Pistikopoulos (2002)), the so-called explicit linear MPC (Bemporad et al., 2002) solves the underlying QP offline and stores the affine functions in different regions in a look-up table. As for online computing, only linear functions have to be evaluated, which makes explicit linear MPC easy-to-implement, reliable, and efficient. However, explicit linear MPC is limited to small-scale problems since its memory requirement as well as the computation time grows exponentially

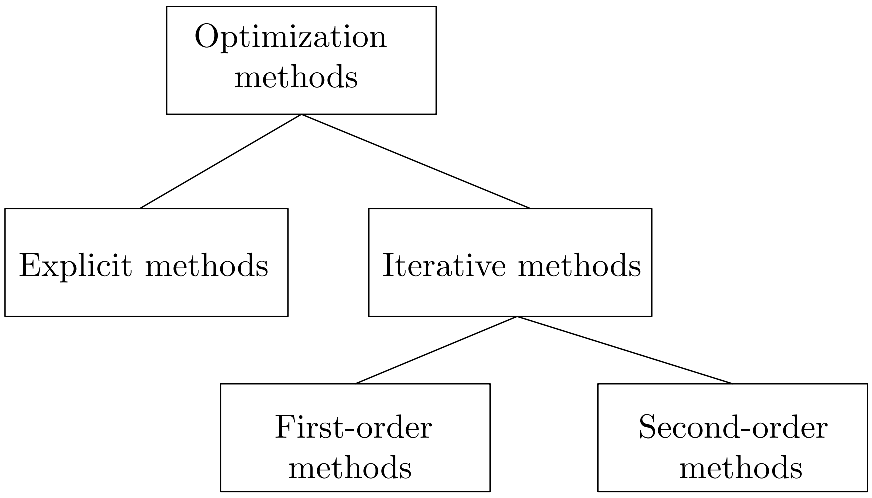


Figure 1.1: Classification of optimization methods for MPC.

with the number of inequality constraints in the worst case. To address such limitations, suboptimal methods, such as merging regions (Geyer, Torrisi, & Morari, 2008), keeping only a subset of critical regions (Pannocchia, Rawlings, & Wright, 2007), and relaxing optimality (Bemporad & Filippi, 2003), can reduce the complexity of explicit linear MPC. An excellent survey on explicit linear MPC and its approximation can be found in Alessio and Bemporad (2009). Although the underlying NLP in NMPC is also parametric to the initial state, unlike linear MPC, NMPC does not have a general explicit state feedback law. Grancharova and Johansen (2012) locally approximates the parametric NLP with a parametric QP, the solution of which can be obtained and stored offline. However, the stored piecewise linear functions are only approximate solutions to the NMPC problem.

On the contrary, implicit methods or iterative methods, as its name indicates, solve the optimization problem online in an iterative manner. Due to the advantages in its scalability and flexibilities in optimality control and online parameter modifying, iterative methods have received tremendous attention in both industry and academia. The reviews of iterative methods for linear MPC and NMPC are given as follows.

Iterative methods for linear MPC

Depending how each iteration is performed, iterative methods can be further categorized into first- and second-order methods. First-order methods such as Nesterov's fast gradient projection method (Nesterov, 1983) and the operator splitting methods (Douglas & Rachford, 1956) perform cheap iterations that mainly consist of matrix-vector multiplications or require only first-order information. In comparison, second-order methods such as the interior-point method and the active-set method require solving a linear equation at each iteration. We first describe the first-order methods for linear MPC. The fast gradient projection method applied to linear MPC problems can be found in Richter, Jones, and Morari (2009) and Kögel and Findeisen (2011a), and cold start, warm start, preconditioning in the context of MPC are discussed. The fast gradient projection method directly applied to the primal problem is limited only to MPC problems with bounded input constraints. To solve linear MPC problems with both input and state constraints, methods such as the combination of the fast gradient projection method and the augmented Lagrangian method (Kögel & Findeisen, 2011b), the fast gradient projection method with Lagrange relaxation (Richter, Morari, & Jones, 2011), and the dual fast gradient projection method (Patrinos & Bemporad, 2013), are proposed. An alternative first-order method to solve the input and state constrained linear MPC problems is the operator splitting method, e.g., the alternating direction method of multipliers (ADMM). Linear MPC based on ADMM can be found in O'Donoghue, Stathopoulos, and Boyd (2013) and Jerez et al. (2014). When Nesterov's acceleration is applied to ADMM, the resulting method is called the fast ADMM method (Goldstein, O'Donoghue, Setzer, & Baraniuk, 2014), which is applied to solve the linear MPC problem in Pu, Zeilinger, and Jones (2016) and Ghadimi, Teixeira, Shames, and Johansson (2014).

Second-order methods are also referred to as Newton-type methods, in which a Newton step is performed by solving a linear equation at every iteration. Newton-type methods tailored to MPC, such as the active-set and interior-point methods, have been developed. By utilizing warm start and the property of the parametric nature of MPC, the so-called "online active-set method" by Ferreau, Bock, and Diehl (2008) shows better performance compared with a general-purpose QP solver. Wright (1996), Wang and Boyd (2010), and Domahidi, Zgraggen, Zeilinger, Morari, and Jones (2012) propose methods that exploit the banded structure of the coefficient matrix in the Newton step, and a linear computational complexity in the prediction horizon can be obtained. Generally speaking, second-order methods have faster rates of convergence and can deal with more general class of problems than first-order methods, however, with higher memory requirements and computational costs per iteration.

Iterative methods for NMPC

Iterative methods for NMPC can also be categorized into first- and second-order methods. There have been several attempts on solving directly the NMPC problem by using first-order methods. For NMPC problems with only input constraints, the gradient projection method based on Pontryagin's minimum principle (Pontryagin, Boltyanskii, Gamkrelidze, & Mishchenko, 1961) can be applied (Käpernick & Graichen, 2014). In order to handle state or general constraints, the gradient projection method can be extended to combine with the augmented Lagrangian method (Englert, Völz, Mesmer, Rhein, & Graichen, 2019) or project the gradient step onto a linearization of the constraints (Torrisi, Grammatico, Smith, & Morari, 2018).

Second-order methods for NMPC, such as the sequential quadratic programming (SQP) method and the interior-point method, solve QP subproblems with linearized dynamics and constraints at each iteration. Apparently, as proposed in Kouzoupis, Ferreau, Peyrl, and Diehl (2015) and Kalmari, Backman, and Visala (2015), the idea of solving these QP subproblems by using first-order methods can be adopted. However, it should be noted that for convexity-dependent first-order methods, such as the dual gradient method and ADMM, special care should be taken in the Hessian matrices to ensure the convexities of the QP subproblems. Hessian approximation techniques, such as the generalized Gauss-Newton method for nonlinear least-squares NLPs (Bock, 1983; Houska, Ferreau, & Diehl, 2011b) and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) (see, e.g., Nocedal and Wright (2006)) method can be applied to ensure the convexity. Apart from the first-order methods, the particular banded structures of the subproblems are exploited in Glad and Jonson (1984), Rao, Wright, and Rawlings (1998), Jørgensen, Rawlings, and Jørgensen (2004), and Frasch, Sager, and Diehl (2015) such that the search direction can be calculated efficiently. An excellent review of Newton-type methods can be found in Diehl, Ferreau, and Haverbeke (2009). For seeking a suboptimal solution of the NMPC problem, Ohtsuka (2004) proposes a Jacobian-free method that traces the solution by continuation. Another approximate method is the real-time iteration scheme (Diehl, Bock, & Schlöder, 2005), which can be seen as the SQP method with only one QP iteration performed each sampling instant. A comprehensive review of suboptimal solution methods can be found in Wolf and Marquardt (2016).

1.3 Outline and contributions

This thesis presents fast numerical optimization algorithms for both linear MPC and NMPC. Chapter 2 introduces some preliminaries of numerical optimization and formulates the MPC problem. Chapter 3 presents a fast optimization method for linear

MPC. Chapters 4, 5, and 6 present fast optimization methods and an efficient implementation for NMPC for different contexts. We conclude the thesis and present an outlook in Chapter 7. We shortly discuss the main contributions of each chapter as follows.

Chapter 2 – Model predictive control. This chapter formulates the continuous-time MPC problem and gives the discretized form of the MPC problem, which is used throughout the thesis. The discretized MPC problem is obtained by using the so-called reverse-time discretization method, which leads to a well-structured discretized MPC problem so that succinct descriptions of the proposed methods in the remaining chapters can be achieved. Since the inequality constraints in this thesis are transferred into barrier functions under the framework of the interior-point method, the basics of numerical optimization and the interior-point method are introduced in this chapter.

Chapter 3 – Combined first- and second-order method for linear MPC. This chapter presents a simple iterative method that combines first- and second-order approaches for linear MPC. Approximate value functions requiring only first-order derivatives and incorporating fixed second-order information are employed, which leads to a method that splits the MPC problem into subproblems along the prediction horizon, and only the states and costates (Lagrange multipliers corresponding to the state equations) are exchanged between consecutive subproblems during iteration. The convergence is guaranteed under the framework of the majorization-minimization principle. For efficient implementation, practical details are discussed, and the performance is assessed against both first- and second-order methods with two numerical examples. The results indicate that the proposed method can obtain a moderately accurate solution with a small number of cheap iterations.

Chapter 4 – Highly parallelizable Newton-type method for NMPC – Algorithm. This chapter presents a highly parallelizable Newton-type method for NMPC by exploiting the particular structure of the Karush-Kuhn-Tucker (KKT) conditions. These equations are approximately decoupled into single-step subproblems along the prediction horizon for parallelization. The coupling variable of each subproblem is approximated to its optimal value using a simple, efficient, and effective method at each iteration. The rate of convergence of the proposed method is proved to be superlinear under mild conditions. Numerical simulation of using the proposed method to control a quadrotor shows that the proposed method is highly parallelizable and converges in only a few iterations, even to a high accuracy. Comparison of the proposed method's performance with that of several state-of-the-art methods shows that it is faster.

Chapter 5 – Highly parallelizable Newton-type method for NMPC – Implementation. The highly parallelizable method for NMPC introduced in Chapter 4 does not specify a particular method to handle the inequality constraints. This chapter presents an efficient implementation, which is called ParNMPC, of the highly parallelizable method under the framework of the primal-dual interior-point method. The implementation details of ParNMPC are introduced, including a framework that unifies search direction calculation done using Newton's method and the parallel method, line search methods for guaranteeing convergence, and a warm start strategy for the interior-point method. To assess the performance of ParNMPC under different configurations, three experiments including a closed-loop simulation of a quadrotor, a real-world control example of a laboratory helicopter, and a closed-loop simulation of a robot manipulator are shown. These experiments show the effectiveness and efficiency of ParNMPC both in serial and parallel.

Chapter 6 – Sparsity-exploiting Jacobi method for NMPC. This chapter presents an efficient Jacobi optimization method for NMPC by exploiting the temporal sparsity of the KKT matrix and lower-layer problem-dependent sparsity. The NMPC problem is solved by the Jacobi method, in which the temporal couplings of either the state or costate (Lagrange multiplier corresponding to the state equation) equations are ignored so that the lower-layer sparsity is preserved. Convergence analysis indicates that the convergence of the proposed method is related to the prediction horizon and regularization. To demonstrate its efficiency of the proposed method, we concentrate on the NMPC control of partial differential equation (PDE) systems. The NMPC problem to be solved is formulated by discretizing the PDE system in space and time by using the finite difference method, which results in a large-scale and sparse problem. A lower-layer Jacobi method is proposed to exploit the lower-layer sparsity (spatial sparsity of the PDE-constrained NMPC problem). Numerical experiment of controlling a heat transfer process shows that the proposed method is two orders of magnitude faster than the conventional structure-exploiting Newton's method.

Appendix A – Explicitly discretized MPC. The MPC problem used throughout the thesis is formulated on the basis of the so-called reverse-time discretization method, which is an implicit discretization method. In this appendix, we discuss the extensions of the proposed methods in the previous chapters to the MPC problem based on the explicit discretization method.

Chapter 2

Model Predictive Control

Numerical optimization plays a central role on finding the optimal control input in the context of model predictive control (MPC). In this chapter, we provide a basic introduction to optimization and formulate the discrete-time MPC problem used throughout the thesis.

This chapter is organized as follows. Section 2.1 introduces the basics of numerical optimization and the interior-point method. In Section 2.2, the continuous-time MPC problem is formulated, a particular discretization method called the reverse-time discretization method is introduced, and the discretized MPC problem is given.

2.1 Preliminaries

2.1.1 Numerical optimization

Consider the following optimization problem:

$$\begin{aligned} \min_{x} \;& l(x) \\ \text{s.t.} \;& c(x) = 0, \\ & g(x) \geq 0, \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the optimization variable, $l : \mathbb{R}^n \to \mathbb{R}$ is the cost function to be minimized, $c : \mathbb{R}^n \to \mathbb{R}^m$ is the equality constraint function, and $g : \mathbb{R}^n \to \mathbb{R}^w$ is the inequality constraint function.

To characterize the optimization problem (2.1), we have the following definitions:

Definition 2.1. *(Feasible set). The feasible set Ω is defined as the set of points x that satisfy the constraints; that is,*

$$\Omega := \{x | c(x) = 0,\ g(x) \geq 0\}. \tag{2.2}$$

Definition 2.2. *(Feasibility). The optimization problem* (2.1) *is said to be feasible if the feasible set* Ω *is not empty.*

Definition 2.3. *(Strict feasibility). The optimization problem* (2.1) *is said to be strictly feasible if the set* $\{x|c(x) = 0,\ g(x) > 0\}$ *is not empty.*

Definition 2.4. *(Local minimum). The point* x^* *is said to be a local minimum of* (2.1) *if* $x^* \in \Omega$ *and there exists a neighborhood* O *of* x^* *such that* $l(x^*) \leq l(x)$ *holds for any* $x \in O \cap \Omega$.

Definition 2.5. *(Global minimum). The point* x^* *is said to be a global minimum of* (2.1) *if* $x^* \in \Omega$ *and* $l(x^*) \leq l(x)$ *holds for any* $x \in \Omega$.

Definition 2.6. *(Active set). The active set* $\mathcal{A}(x)$ *at any feasible* $x \in \Omega$ *is defined as*

$$\mathcal{A}(x) := \left\{i|g_{(i)}(x) = 0\right\}.$$

Definition 2.7. *(Linear independence constraint qualification). Given a point* $x \in \Omega$, *the linear independence constraint qualification (LICQ) holds at* x *if the set of the constraint gradients*

$$\left\{\nabla c_{(i)}(x),\ i \in \{1, \cdots, m\}\right\} \cup \left\{\nabla g_{(i)}(x),\ i \in \mathcal{A}(x)\right\}$$

is linearly independent.

Definition 2.8. *(Convex function). A scalar-valued function, e.g.,* $l(x)$, *is said to be convex if*

$$l(\theta x_1 + (1-\theta)x_2) \leq \theta l(x_1) + (1-\theta)l(x_2)$$

holds for any $\theta \in [0, 1]$ *and points* x_1 *and* x_2 *in the domain of definition of* l.

Definition 2.9. *(Convex set). The set* Ω *is said to be convex if*

$$\theta x_1 + (1-\theta)x_2 \in \Omega$$

holds for any $\theta \in [0, 1]$ *and points* $x_1 \in \Omega$ *and* $x_2 \in \Omega$.

Definition 2.10. *(Convex optimization problem). The optimization problem* (2.1) *is said to be convex if the cost function* $l(x)$ *is a convex function and the feasible set* Ω *in* (2.2) *is a convex set.*

The optimization problem (2.1) encodes a large class of optimization problems, which can be categorized from the types of cost and constraints as follows.

- Linear program (LP). LPs have a linear cost function and linear equality and inequality constraints as follows.

$$\begin{aligned} \min_{x} \ & v^T x \\ \text{s.t.} \quad & Ax = a, \\ & Bx \geq b. \end{aligned}$$

  LPs are convex optimization problems.

- Quadratic programs (QP). QPs have a quadratic cost function and linear equality and inequality constraints as follows.

$$\begin{aligned} \min_{x} \ & \|x\|_P^2 + v^T x \\ \text{s.t.} \quad & Ax = a, \\ & Bx \geq b, \end{aligned}$$

  where $P = P^T$. QPs are convex optimization problems if $P \geq 0$. The discretized linear MPC problems are essentially QPs, and their convexities can be guaranteed when the weighting matrices are chosen to be positive-semidefinite.

- Nonlinear program (NLP). NLPs are general optimization problems, e.g., with nonlinear constraints. Nonlinear MPC (NMPC) problems under the constraints of nonlinear dynamics are examples of NLPs.

Theorem 2.11. *(Karush-Kuhn-Tucker conditions). Consider the optimization problem defined in* (2.1)*. Assume that the optimization problem* (2.1) *is feasible, that x^* is a local minimum of* (2.1)*, that the functions l, c, and g are continuously differentiable, and that the LICQ holds at x^*. Then, there exist unique vectors $\lambda^* \in \mathbb{R}^m$ and $z^* \in \mathbb{R}^w$ such that the following conditions hold:*

$$c(x^*) = 0, \tag{2.3a}$$
$$\nabla l(x^*)^T + \nabla c(x^*)^T \lambda^* - \nabla g(x^*)^T z^* = 0, \tag{2.3b}$$
$$z^*_{(i)} g_{(i)}(x^*) = 0, \ i = \{1, \cdots, w\}, \tag{2.3c}$$
$$g(x^*) \geq 0, \tag{2.3d}$$
$$z^* \geq 0. \tag{2.3e}$$

The conditions (2.3) are known as the Karush-Kuhn-Tucker (KKT) conditions, which are the first-order necessary conditions for optimality. The vectors λ and z are known as the Lagrange multipliers corresponding to the constraints $c(x) = 0$ and $g(x) \geq 0$, respectively, and the LICQ guarantees the uniqueness of the optimal Lagrange multipliers at x^*. Note that the KKT conditions are the equivalent conditions (Boyd & Vandenberghe, 2004) for the global minimum if (2.1) is convex.

2.1.2 Interior-point method

In this section, we introduce the interior-point method to solve the optimization problem (2.1) and show the updates of the optimization variable and the Lagrange multipliers. Since the interior-point method depends on second-order derivatives and introduces a barrier function for the inequality constraint, we make the following assumptions:

Assumption 2.1. *The functions l, c, and g are twice continuously differentiable.*

Assumption 2.2. *The optimization problem* (2.1) *is strictly feasible.*

Moreover, we make the following assumption so that the LICQ holds in the interior-point method:

Assumption 2.3. *The set of the constraint gradients $\left\{\nabla c_{(i)}(x),\ i \in \{1, \cdots, m\}\right\}$ is linearly independent.*

In the interior-point method, the inequality constraint $g(x) \geq 0$ is transferred into a logarithmic barrier function added to the cost. That is, we obtain the following relaxed optimization problem:

$$\begin{aligned} \min_{x,s}\quad & l(x) - \rho \sum_{i=1}^{w} \ln s_{(j)} \\ \text{s.t.}\quad & c(x) = 0, \\ & g(x) = s, \end{aligned} \tag{2.4}$$

where $\rho > 0$ is the barrier parameter.

We denote $\lambda \in \mathbb{R}^m$ and $z \in \mathbb{R}^w$ as the Lagrange multipliers corresponding to the constraints $c(x) = 0$ and $g(x) = s$, respectively. Let $\mathcal{L}(\lambda, z, x)$ be the Lagrangian defined by

$$\mathcal{L}(\lambda, z, x) := l(x) + \lambda^T c(x) - z^T g(x).$$

The KKT conditions for (2.4) are

$$c(x) = 0, \tag{2.5a}$$

$$s - g(x) = 0, \tag{2.5b}$$

$$\nabla_x \mathcal{L}(\lambda, z, x)^T = 0, \tag{2.5c}$$

$$-\rho S^{-1} e + z = 0, \tag{2.5d}$$

where $S := \operatorname{diag}(s_{(1)}, \cdots, s_{(w)})$ and $e := [1, \cdots, 1]$. Note that (2.5) differs from (2.3) only in (2.3c) and (2.5d). That is, the condition (2.3c) is relaxed by a positive value

ρ in (2.5d). It can be easily checked from Assumption 2.3 that the LICQ holds at (x^*, s^*), which indicates the (existence and) uniqueness of the Lagrange multipliers λ^* and z^*.

Note that the strategy of decreasing the barrier parameter ρ during iteration in order to obtain an accurate solution is out the scope of this introduction. Next, we only show how an iteration is performed in the primal and primal-dual interior-point method with a fixed barrier parameter ρ.

Primal interior-point method

The primal interior-point method is to directly apply Newton's method to solve the KKT conditions (2.5d). The search direction $(\Delta\lambda, \Delta z, \Delta x, \Delta s)$ is calculated by solving

$$\begin{bmatrix} 0 & 0 & \nabla c(x) & 0 \\ 0 & 0 & -\nabla g(x) & I \\ \nabla c(x)^T & -\nabla g(x)^T & \nabla^2_{xx}\mathcal{L}(\lambda, z, x) & 0 \\ 0 & I & 0 & \Sigma \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta z \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} c(x) \\ s - g(x) \\ \nabla_x \mathcal{L}(\lambda, z, x)^T \\ -\rho S^{-1} e + z \end{bmatrix}, \tag{2.6}$$

where

$$\Sigma = \rho S^{-2}. \tag{2.7}$$

Primal-dual interior-point method

The primal-dual interior-point method also applies Newton's method to solve the KKT conditions (2.5), however, with the condition (2.5d) replaced by

$$-\rho e + Sz = 0.$$

The search direction $(\Delta\lambda, \Delta z, \Delta x, \Delta s)$ is calculated by solving

$$\begin{bmatrix} 0 & 0 & \nabla c(x) & 0 \\ 0 & 0 & -\nabla g(x) & I \\ \nabla c(x)^T & -\nabla g(x)^T & \nabla^2_{xx}\mathcal{L}(\lambda, z, x) & 0 \\ 0 & S & 0 & Z \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta z \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} c(x) \\ s - g(x) \\ \nabla_x \mathcal{L}(\lambda, z, x)^T \\ -\rho e + Sz \end{bmatrix}, \tag{2.8}$$

where and $Z := \operatorname{diag}(z_{(1)}, \cdots, z_{(w)})$. By rewriting (2.8) into the symmetric form, we obtain

$$\begin{bmatrix} 0 & 0 & \nabla c(x) & 0 \\ 0 & 0 & -\nabla g(x) & I \\ \nabla c(x)^T & -\nabla g(x)^T & \nabla^2_{xx}\mathcal{L}(\lambda, z, x) & 0 \\ 0 & I & 0 & \Sigma \end{bmatrix} \begin{bmatrix} \Delta\lambda \\ \Delta z \\ \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} c(x) \\ s - g(x) \\ \nabla_x \mathcal{L}(\lambda, z, x)^T \\ -\rho S^{-1} e + z \end{bmatrix}, \tag{2.9}$$

where

$$\Sigma = S^{-1}Z. \tag{2.10}$$

As can be seen from (2.6) and (2.9), the search direction calculation in the primal interior-point method differs from that of the primal-dual method only in Σ.

After the search direction $(\Delta\lambda, \Delta z, \Delta x, \Delta s)$ is calculated by using either the primal method or the primal-dual method, the new iterate $(\lambda^+, z^+, x^+, s^+)$ is computed by

$$(\lambda^+, x^+, s^+) = (\lambda, x, s) - \alpha_s(\Delta\lambda, \Delta x, \Delta s),$$

and

$$z^+ = z - \alpha_z \Delta z,$$

where $\alpha_s \in (0, \alpha_s^{\max}]$ and $\alpha_z \in (0, \alpha_z^{\max}]$ are the step sizes, which are determined by line search methods, such as the merit-function-based line search method and the filter line search method. Here, $\alpha_s^{\max}$ and $\alpha_z^{\max}$ are the maximum admissible step sizes obtained by using the fraction-to-the-boundary rule (Nocedal & Wright, 2006):

$$\alpha_s^{\max} = \max\left\{\alpha \in (0,1] : s - \alpha\Delta s \geq (1-\tau)s\right\}$$

and

$$\alpha_z^{\max} = \max\left\{\alpha \in (0,1] : z - \alpha\Delta z \geq (1-\tau)z\right\},$$

where the fraction-to-the-boundary parameter τ is chosen to be $\tau = \min\{\tau^{\min}, \rho\}$ with $\tau^{\min} = 0.005$ (typical value).

2.2 MPC problem

2.2.1 Continuous-time MPC

Consider a continuous-time (nonlinear) system governed by the following differential equation:

$$\dot{x}(t) = f\left(u(t), x(t)\right), \tag{2.11}$$

where $u \in \mathbb{R}^{n_u}$ is the control input and $x \in \mathbb{R}^{n_x}$ is the system state. The MPC problem for (2.11) is formulated on the basis of the following finite-horizon optimal control problem:

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T l\left(u(\tau), x(\tau)\right) d\tau + \varphi\left(x(T)\right) \\ \text{s.t.} \quad & x(0) = \bar{x}_0, \\ & \dot{x}(\tau) = f\left(u(\tau), x(\tau)\right),\ \tau \in [0,T], \\ & C(u(\tau), x(\tau)) = 0,\ \tau \in [0,T], \\ & G(u(\tau), x(\tau)) \geq 0,\ \tau \in [0,T], \end{aligned} \tag{2.12}$$

where $\bar{x}_0$ is the current state or initial state, $l : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \to \mathbb{R}$ is the stage cost function, $\varphi : \mathbb{R}^{n_x} \to \mathbb{R}$ is the terminal cost function, $C : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_\mu}$ is the equality path constraint function, and $G : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ is the inequality path constraint function.

2.2.2 Discretized MPC

MPC based on the direct approach (Stryk & Bulirsch, 1992) requires the continuous-time system (2.11) to be discretized. Generally, different discretization methods lead to different discretization accuracies, computational costs, and problem structures. In this thesis, we consider a special discretization method that discretizes (2.11) into

$$x_- + \mathcal{F}(u, x) = 0, \tag{2.13}$$

where x_- is the predecessor state. The discretization method is herein called the "reverse-time discretization method," which is a class of implicit discretization methods. The reverse-time discretization method comes from the fact that the predecessor state x_- can be directly obtained from x. For a given explicit discretization method, its reverse-time variation involves simply applying the discretization backward in time, i.e., with a negative step size. For example, the backward Euler method with $x_i = x_{i-1} + f(u_i, x_i)\Delta\tau$ is a reverse-time discretization method with $\mathcal{F}(u_i, x_i) = f(u_i, x_i)\Delta\tau - x_i$, where $\Delta\tau$ is the discretization step size. It corresponds to the forward Euler method performed backward in time. Any explicit integration method, e.g., the Runge-Kutta method, can be used backward in time to result in the reverse-time discretization. It should be noted that in solving the MPC problem, the state integration is still propagated forward in time; thus, stable dynamics stay stable when the discretization accuracy is high.

Regarding the discretization accuracy, we show in the following proposition that the discretization accuracy of an explicit method is preserved in its reverse-time variation.

Proposition 2.12. *Let $y(t) \in \mathbb{R}^m$ and consider the ordinary differential equation*

$$\dot{y}(t) = f(y(t), t).$$

Let h be the discretization step size and the differential equation be discretized by using an explicit discretization method with a local truncation error of $\mathcal{O}(|h|^n)$:

$$Y(t + h) = F(y(t), h, t),$$

that is, $Y(t+h)$ is the approximation to $y(t+h)$ satisfying

$$\|Y(t+h) - y(t+h)\| = \mathcal{O}(|h|^n).$$

Let $\tilde{Y}(t)$ be obtained by using the corresponding reverse-time discretization method, i.e.,

$$y(t-h) = F(\tilde{Y}(t), -h, t).$$

Then, the local truncation error of the reverse-time discretization method is also $\mathcal{O}(|h|^n)$, that is,

$$\|\tilde{Y}(t) - y(t)\| = \mathcal{O}(|h|^n). \tag{2.14}$$

Proof. Without loss of generality, we assume $h > 0$. It can also be obtained that

$$\|y(t-h) - Y(t-h)\| = \mathcal{O}(h^n), \tag{2.15}$$

where

$$Y(t-h) = F(y(t), -h, t).$$

According to the mean value theorem for vector-valued functions (McLeod, 1965), the following equation holds:

$$F(\tilde{Y}(t), -h, t) - F(y(t), -h, t) = \sum_{i=1}^{n} \lambda_i \nabla_y F(\xi_i, -h, t) \left(\tilde{Y}(t) - y(t)\right), \tag{2.16}$$

where $\xi_i \in [y(t), Y(t)]$, $\lambda_i \geq 0$, and $\sum_{i=1}^{n} \lambda_i = 1$. Since $y(t-h) = F(\tilde{Y}(t), -h, t)$ and $Y(t-h) = F(y(t), -h, t)$, by taking norms of both sides, we obtain

$$\|\tilde{Y}(t) - y(t)\| \leq \left\| \left(\sum_{i=1}^{n} \lambda_i \nabla_y F(\xi_i, -h, t) \right)^{-1} \right\| \|(y(t-h) - Y(t-h))\|.$$

The result (2.14) then follows from (2.15) and the fact that $\nabla_y F(\xi_i, -h, t) \to I$ when $h \to 0$. $\square$

By using the reverse-time discretization method, we discretize the continuous-time MPC problem (2.12) into the following N-stage MPC problem:

$$\begin{aligned}
\min_{x_i, u_i} \; & \sum_{i=1}^{N} \frac{T}{N} l(u_i, x_i) + \varphi(x_N) \\
\text{s.t.} \quad & x_0 = \bar{x}_0, \\
& x_{i-1} + \mathcal{F}(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\
& C(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\
& G(u_i, x_i) \geq 0, \quad i \in \{1, \cdots, N\},
\end{aligned} \tag{2.17}$$

where N is the number of the discretization grid points in the prediction horizon. By introducing slack variables s_i, we can transfer (2.17) into

$$\begin{aligned}
&\min_{x_i,u_i,s_i} \sum_{i=1}^{N} \frac{T}{N} l(u_i, x_i) + \varphi(x_N) && \text{(2.18a)}\\
&\text{s.t.} \quad x_0 = \bar{x}_0, && \text{(2.18b)}\\
&\qquad x_{i-1} + \mathcal{F}(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, && \text{(2.18c)}\\
&\qquad C(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, && \text{(2.18d)}\\
&\qquad G(u_i, x_i) - s_i = 0, \quad i \in \{1, \cdots, N\}, && \text{(2.18e)}\\
&\qquad s_i \geq 0, \quad i \in \{1, \cdots, N\}. && \text{(2.18f)}
\end{aligned}$$

A more general and structured form of (2.18) can be expressed as

$$\begin{aligned}
&\min_{x_i,u_i} \sum_{i=1}^{N} L_i(u_i, x_i)\\
&\text{s.t.} \quad x_0 = \bar{x}_0,\\
&\qquad x_{i-1} + \mathcal{F}_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\},\\
&\qquad C_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\},\\
&\qquad G_i(u_i, x_i) \geq 0, \quad i \in \{1, \cdots, N\},
\end{aligned} \tag{2.19}$$

where the slack variables are combined into the input vector u, the equality constraints (2.18d) and (2.18e) form the new equality constraints $C_i(u_i, x_i) = 0$, $i \in \{1, \cdots, N\}$, and the problem is made time-dependent to deal with, e.g., time-dependent dynamics and reference. Moreover, we assume that $G_i(u, x) \geq 0$ is a general polytopic constraint on u and x, that is, G can be expressed as $G_i(u, x) = D_i u + E_i x + b_i$, where D_i and E_i are matrices and b_i is a vector. For the following reasons, we formulate the MPC problem with a polytopic inequality constraint. First, the polytopic constraint can be satisfied with a backtrack line search, i.e., the fraction-to-the-boundary rule. Second, polytopic constraints, such as box or softened box constraints, are common in the context of MPC, and therefore there is no need to introduce slack variables. MPC problems with nonlinear inequality constraints can be reformulated into the form of (2.19) by introducing slack input variables.

In order to use the interior-point method to solve the discretized MPC problem (2.19), which is essentially a NLP, we make the following assumptions:

Assumption 2.4. *The functions L_i, $\mathcal{F}_i$, C_i, and G_i, $i \in \{1, \cdots, N\}$, are twice continuously differentiable with respect to u and x.*

Assumption 2.5. *The discretized MPC problem* (2.19) *is strictly feasible.*

Assumption 2.6. *The LICQ holds at the optimal solution for the discretized MPC problem* (2.19)*.*

Remark 2.1. *When the interior-point method is referred in the remaining chapters, it is not to directly apply the interior-point method introduced in Section 2.1.2 to solve the discretized MPC problem* (2.19)*. Specifically, the inequalities are transferred into barrier functions added to the cost, the primal or primal-dual interior-point method in the remaining chapters means that the matrix Σ is calculated by* (2.7) *or* (2.10)*, but the search direction is not calculated by Newton's method.*

Chapter 3

Combined First- and Second-Order Method for Linear MPC

3.1 Introduction

Iterative methods for linear model predictive control (MPC) can be categorized into first- and second-order methods. Second-order methods such as the interior-point method and the active-set method require solving a linear equation at each iteration. First-order methods such as Nesterov's fast gradient method (Nesterov, 1983) and the operator splitting methods (Douglas & Rachford, 1956) perform cheap iterations that mainly consist of matrix-vector multiplications or require only first-order information. Compared with second-order methods, first-order methods have cheaper computational costs per iteration, however, with slower rates of convergence; hence, they require more iterations. Therefore, iterative methods suffer from the trade-off between the computational cost per iteration and the rate of convergence or, consequently, number of iterations (NOIs).

To tackle this, we propose an iterative method that requires only the first-order derivatives of value functions and incorporates fixed second-order information to speed up convergence. The linear MPC problem is first relaxed by using the interior-point method. Iterations for solving the relaxed MPC problem are performed on the basis of its approximate value function, where the value function is partially linearized and regularized with fixed second-order derivatives. Convergence is guaranteed by choosing an appropriate regularization term under the majorization-minimization principle (Ortega & Rheinboldt, 1970). This method is herein called the "iterative horizon-splitting method." Moreover, the accuracy of the solution obtained by using the iterative horizon-splitting method can be further improved with solution polishing. For efficient implementation, we discuss practical details such as the selection of the regularization parameter, limiting the NOIs, and the barrier strategy. The performance of the practical implementation, without showing its convergence theoretically, is assessed against both first- and second-order methods with two numerical examples.

This chapter is organized as follows. The MPC problem and its relaxed form under the interior-point method are presented in Section 3.2. The iterative horizon-splitting method and its convergence are shown in Section 3.3. The polishing procedure is introduced in Section 3.4. Some practical implementation details are discussed in Section 3.5. The numerical experiments are described and discussed in Section 3.6. The key points are summarized, and future work is mentioned in Section 3.7.

3.2 Problem statement

In this chapter, we consider the linear form of (2.18), i.e., the following linear MPC problem:

$$\begin{aligned} \min_{X,U} \ & \sum_{i=1}^{N} L_i(x_i, u_i) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_{i-1} + Ax_i + Bu_i = 0, \quad \forall i \in \{1, \cdots, N\}, \\ & Dx_i + Eu_i + b \geq 0, \quad \forall i \in \{1, \cdots, N\}. \end{aligned} \tag{3.1}$$

Here, $x \in \mathbb{R}^{n_x}$ is the state, $u \in \mathbb{R}^{n_u}$ is the control input, $\bar{x}_0$ is the initial state, $U = (u_1, \cdots, u_N)$ and $X = (x_0, \cdots, x_N)$ are the sequences of the control inputs and states along the prediction horizon N, respectively, and $L_i(x, u) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the stage cost function defined by

$$L_i(x, u) := \frac{1}{2}\left(\|x - x_{r_i}\|_{Q_i}^2 + \|u - u_{r_i}\|_{R_i}^2\right)$$

with weighting matrices $Q_i > 0$ and $R_i > 0$, where x_{r_i} and u_{r_i} are the state and input references, respectively. Each inequality constraint has a dimensionality of n_c.

We adopt the interior-point method to relax the MPC problem (3.1) by transferring the inequality constraint into a logarithmic barrier function added to the cost. We obtain the following relaxed optimal control problem (OCP) with a barrier parameter $\rho > 0$:

$$\begin{aligned} \min_{X,U} \ & \sum_{i=1}^{N} \{L_i(x_i, u_i) + \Phi(x_i, u_i, \rho)\} \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_{i-1} + Ax_i + Bu_i = 0, \quad \forall i \in \{1, \cdots, N\}, \end{aligned} \tag{3.2}$$

where $\Phi(x, u, \rho) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ is the barrier cost function defined as

$$\Phi(x, u, \rho) := -\rho \sum_{j=1}^{n_c} \log \left(C_{(j)} x + D_{(j)} u + c_{(j)} \right). \tag{3.3}$$

3.3 Iterative horizon-splitting method

In this section, we derive a method that depends on the approximate value functions of OCP (3.2) and analyze its convergence. We first give some definitions including the value functions in the following subsection.

3.3.1 Preparations

To guarantee the existence of a feasible solution, we define the set of admissible states as follows.

Definition 3.1. *For $i \in \{1, \cdots, N\}$, let $\mathbb{A}_{i-1} \subset \mathbb{R}^{n_x}$ be the set of admissible x_{i-1} defined as*

$$\begin{aligned} \mathbb{A}_{i-1} := \{x_{i-1} :\ \exists x_j, u_j \ \text{s.t.}\ & Dx_j + Eu_j + b > 0, \\ & x_{j-1} + Ax_j + Bu_j = 0,\ \forall j \in \{i, \cdots, N\}\}. \end{aligned}$$

Definition 3.2. *For $i \in \{1, \cdots, N\}$, we define the i-th value function $V_{N-(i-1)}(x_{i-1}, \rho) : \mathbb{A}_{i-1} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ of the OCP (3.2) as*

$$\begin{aligned} V_{N-(i-1)}(x_{i-1}, \rho) := & \min_{x_i, u_i} \mathcal{L}_i(x_i, u_i, \rho) \\ & \text{s.t.} \qquad x_{i-1} + Ax_i + Bu_i = 0, \end{aligned} \tag{3.4}$$

where

$$\mathcal{L}_i(x, u, \rho) := L_i(x, u) + \Phi(x, u, \rho) + V_{N-i}(x, \rho) \tag{3.5}$$

is the cost function with $V_0(x, \rho) = 0$. Let $\lambda_i \in \mathbb{R}^{n_x}$ be the Lagrange multiplier (costate) corresponding to the equality constraint in (3.4). We define $\mathcal{S}_{N-(i-1)}(x_{i-1}, \rho) : \mathbb{A}_{i-1} \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_x}$ as the mapping that recovers the optimal solutions denoted by x_i^*, u_i^*, and λ_i^* for given values of x_{i-1} and ρ.

Remark 3.1. *The value function $V_{N-(i-1)}(x_{i-1}, \rho)$ is strongly convex in $x_{i-1} \in \mathbb{A}_{i-1}$ for $\rho \in \mathbb{R}_{\geq 0}$, $\forall i \in \{1, \cdots, N\}$.*

Definition 3.3. *We denote the regularized quadratic expansion of the barrier function (3.3) at the k-th iteration by $\Phi^k(x,u,\rho,\beta):\mathbb{R}^{n_x}\times\mathbb{R}^{n_u}\times\mathbb{R}_{\geq 0}\times(0,1]\to\mathbb{R}$, that is,*

$$\Phi^k(x,u,\rho,\beta):=\Phi(x^k,u^k,\rho)+\begin{bmatrix} x-x^k \\ u-u^k \end{bmatrix}^T\begin{bmatrix} \nabla_x^T\Phi(x^k,u^k,\rho) \\ \nabla_u^T\Phi(x^k,u^k,\rho) \end{bmatrix}$$
$$+\frac{1}{2\beta}\left\|\begin{bmatrix} x-x^k \\ u-u^k \end{bmatrix}\right\|^2_{H_\Phi(x^k,u^k,\rho)},$$

where $H_\Phi(x^k,u^k,\rho)$ denotes the Hessian matrix of $\Phi(x,u,\rho)$ with respect to (x,u) evaluated at (x^k,u^k,ρ).

Definition 3.4. *For $i\in\{1,\cdots,N\}$, we denote the i-th primal regularization of the linearized value function at the k-th iteration by $\hat{V}^k_{N-(i-1)}(x_{i-1},\rho,\Lambda_i):\mathbb{A}_{i-1}\times\mathbb{R}_{\geq 0}\times\mathbb{R}^{n_x\times n_x}\to\mathbb{R}$, that is,*

$$\begin{aligned}\hat{V}^k_{N-(i-1)}(x_{i-1},\rho,\Lambda_i):=V_{N-(i-1)}(x^k_{i-1},\rho)+&\nabla_{x_{i-1}}V_{N-(i-1)}(x^k_{i-1},\rho)(x_{i-1}-x^k_{i-1})\\ &+\frac{1}{2}\|x_{i-1}-x^k_{i-1}\|^2_{\Lambda_i}.\end{aligned}\tag{3.6}$$

Here, the regularization matrix satisfies $\Lambda_i>0$.

3.3.2 Iterative horizon-splitting method

Under Definition 3.2, the relaxed problem (3.2) translates into calculating $\mathcal{S}_N(\bar{x}_0,\rho)$, i.e., solving the following problem:

$$\begin{aligned}&\min_{x_1,u_1}\{L_1(x_1,u_1)+\Phi(x_1,u_1,\rho)+V_{N-1}(x_1,\rho)\}\\ &\text{s.t.}\qquad \bar{x}_0+Ax_1+Bu_1=0.\end{aligned}\tag{3.7}$$

Solving (3.7) with Newton's method first approximates the cost function with its second-order Taylor series. Then, the resulting quadratic program (QP) is solved at each iteration. Calculating the second-order derivative of the value function $V_{N-1}(x_1,\rho)$ leads to a Riccati recursion, which has a computational complexity of $\mathcal{O}(N(n_x+n_u)^3)$. Alternatively, when only the first-order derivative is involved during iteration, the proximal regularization of the linearized value function, i.e.,

$$V_{N-1}(x_1^k,\rho)+\nabla_{x_1}V_{N-1}(x_1^k,\rho)(x_1-x_1^k)+\frac{1}{2\gamma}\|x_1-x_1^k\|_2^2,\tag{3.8}$$

is used. Here, the regularization matrix is simply an identity matrix scaled by a factor of $1/\gamma$, where $\gamma>0$ is chosen to be sufficiently small. This method, which is based on only the first-order derivative, is less computationally expensive than Newton's method when performing one iteration. However, the regularization term does not take the dynamics and cost function into consideration and therefore has a slower rate of convergence; thus it needs more iterations.

We propose solving the following approximate problem iteratively:

$$\begin{aligned}\min_{x_1,u_1} & \left\{L_1(x_1,u_1)+\Phi^k(x_1,u_1,\rho,\beta)+\hat{V}^k_{N-1}(x_1,\rho,\Lambda_2)\right\} \\ \text{s.t.} \quad & \bar{x}_0+Ax_1+Bu_1=0,\end{aligned} \tag{3.9}$$

where Φ^k and $\hat{V}^k_{N-1}$ are defined in Definitions 3.3 and 3.4, respectively. As can be seen from the expression of $\hat{V}^k_{N-1}(x_1,\rho,\Lambda_2)$ in (3.6), we choose the regularization matrix as a general constant matrix Λ_2 instead of $1/\gamma I$. The choice of Λ_2 is discussed later in Section 3.5.1 to account for the dynamics and cost function, and Λ_2 therefore serves as second-order information.

Note that the costate λ can be interpreted as the "sensitivity" of the value function with respect to the current state (see, e.g., Nocedal and Wright (2006)), that is,

$$\nabla^T_{x_{i-1}}V_{N-(i-1)}(x^k_{i-1},\rho)=\lambda^*_i(x^k_{i-1},\rho)$$

holds. Solving problem (3.9) requires the optimal costate $\lambda^*_2(x^k_1,\rho)$, which is obtained from $\mathcal{S}_{N-1}(x^k_1,\rho)$. In calculating $\mathcal{S}_{N-1}(x^k_1,\rho)$, the same iterative method is applied by solving a series of approximate problems as in (3.9), which requires $\lambda^*_3(x_2,\rho)$ for some given x_2. Recursively, the approximation procedure is conducted until the last problem, i.e., calculating $\mathcal{S}_1(x_{N-1},\rho)$. This leads to a method with which only approximate problems are needed to be solved. To simplify notation, we define the approximate problem as follows.

Definition 3.5. *For given values of $\beta\in(0,1]$ and $\Lambda_{i+1}>0$, an optimization problem $\mathcal{P}^k_i$, $i\in\{1,\cdots,N\}$, which is parametric to x_{i-1}, λ_{i+1}, and ρ, is defined as*

$$\begin{aligned}\mathcal{P}^k_i(x_{i-1},\lambda_{i+1},\rho):\min_{x_i,u_i} & \hat{\mathcal{L}}^k_i(x_i,u_i,\lambda_{i+1},\rho) \\ \text{s.t.} \quad & x_{i-1}+Ax_i+Bu_i=0,\end{aligned} \tag{3.10}$$

where

$$\begin{aligned}\hat{\mathcal{L}}^k_i(x,u,\lambda,\rho):=&L_i(x,u)+\Phi^k(x,u,\rho,\beta)\\ &+V_{N-i}(x^k,\rho)+(x-x^k)^T\lambda+\frac{1}{2}\|x-x^k\|^2_{\Lambda_{i+1}}\end{aligned} \tag{3.11}$$

is the cost function.

Remark 3.2. *The approximate problem* (3.10) *can be seen as a single-step OCP that approximates the optimization problem in* (3.4) *and is essentially a QP, which can be solved efficiently with a computational complexity of* $\mathcal{O}(n_u^3)$ *by condensing to obtain* u_i *and an extra complexity of* $\mathcal{O}(n_x n_u + n_x^2)$ *to recover* x_i *and* λ_i*, i.e.,* $\mathcal{O}(n_u^3 + n_x^2)$ *in total.*

The resulting recursion for calculating $\mathcal{S}_{N-(i-1)}(x_{i-1}, \rho)$ is summarized in Algorithm 3.1. Here, we denote the operation $(x_i^{k+1}, u_i^{k+1}, \lambda_i^{k+1}) \leftarrow \mathcal{P}_i^k(x_{i-1}, \lambda_{i+1}^{k+1}, \rho)$ as solving $\mathcal{P}_i^k$. To simply illustrate the concept, we assume that the iterates always stay in the interior of the space defined by the inequality constraints. This method splits the OCP (3.2) into simple subproblems (3.10) along the prediction horizon, and only the states and costates are shared between consecutive subproblems during iteration. We herein call this the "iterative horizon-splitting (IHS) method." IHS can be seen as a combination of first- and second-order methods either from the fact that the first-order iteration is incorporated with fixed second-order information Λ or from the fact that each subproblem is solved with Newton's method while the whole problem is solved by exchanging only first-order information (states and costates). Notice that the inner iteration of IHS involves calling itself for the next time step until computing $\mathcal{S}_1(x_{N-1}, \rho)$, which indicates that IHS is a recursive method. To speed up each layer of the recursion, the initial guesses x_i^0 and u_i^0 are warm started from their previous optimal solutions.

Algorithm 3.1 IHS for (3.4) $(x_i, u_i, \lambda_i) \leftarrow \mathcal{S}_{N-(i-1)}(x_{i-1}, \rho)$

```
Input: $(x_{i-1}, \rho)$
Output: $(x_i, u_i, \lambda_i)$
1: Initialize: $k \leftarrow 0$; initial guesses $x_i^0$ and $u_i^0$ satisfying $Dx_i^0 + Eu_i^0 + b > 0$
2: repeat:
3:     $(-, -, \lambda_{i+1}^{k+1}) \leftarrow \mathcal{S}_{N-i}(x_i^k, \rho)$
4:     $(x_i^{k+1}, u_i^{k+1}, \lambda_i^{k+1}) \leftarrow \mathcal{P}_i^k(x_{i-1}, \lambda_{i+1}^{k+1}, \rho)$
5:     $k \leftarrow k + 1$
6: until convergence criterion is met
7: $(x_i, u_i, \lambda_i) \leftarrow (x_i^k, u_i^k, \lambda_i^k)$
```

We then show that the steady iterates of IHS correspond to the solutions of $\mathcal{S}_{N-(i-1)}(x_{i-1}, \rho)$ in the following lemma.

Lemma 3.1. *For given* $x_{i-1} \in \mathbb{A}_{i-1} \neq \emptyset$ *and* $\rho > 0$*, let* $\{x_i^k\}$ *and* $\{u_i^k\}$ *be the iterates generated by IHS in Algorithm 3.1. If* $x_i^k = x_i^{k+1}$ *and* $u_i^k = u_i^{k+1}$ *hold, then* $(x_i^{k+1}, u_i^{k+1}, \lambda_i^{k+1}) = \mathcal{S}_{N-(i-1)}(x_{i-1}, \rho)$.

Proof. Since x_i^{k+1} and u_i^{k+1} are the optimal solutions to the optimization problem $\mathcal{P}_i^k$, the Karush-Kuhn-Tucker (KKT) conditions for $\mathcal{P}_i^k$ are stated as follows.

$$\nabla_{x_i}^T \hat{\mathcal{L}}_i^k(x_i^{k+1}, u_i^{k+1}, \lambda_{i+1}^{k+1}, \rho) + A^T \lambda_i^{k+1} = 0 \tag{3.12}$$

$$\nabla_{u_i}^T \hat{\mathcal{L}}_i^k(x_i^{k+1}, u_i^{k+1}, \lambda_{i+1}^{k+1}, \rho) + B^T \lambda_i^{k+1} = 0 \tag{3.13}$$

$$x_{i-1} + Ax_i^{k+1} + Bu_i^{k+1} = 0 \tag{3.14}$$

Since λ_{i+1}^{k+1} is recovered from $V_{N-i}(x_i^k, \rho)$, or equivalently, $V_{N-i}(x_i^{k+1}, \rho)$, we have

$$\lambda_{i+1}^{k+1} = \nabla_{x_i}^T V_{N-i}(x_i^{k+1}, \rho). \tag{3.15}$$

Substituting (3.15) into (3.12) and (3.13), together with $x_i^k = x_i^{k+1}$ and $u_i^k = u_i^{k+1}$, we obtain

$$\nabla_{x_i}^T L_i(x_i^{k+1}, u_i^{k+1}) + \nabla_{x_i}^T \Phi(x_i^{k+1}, u_i^{k+1}, \rho) + \nabla_{x_i}^T V_{N-i}(x_i^{k+1}, \rho) + A^T \lambda_i^{k+1} = 0, \tag{3.16}$$

and

$$\nabla_{u_i}^T L_i(x_i^{k+1}, u_i^{k+1}) + \nabla_{u_i}^T \Phi(x_i^{k+1}, u_i^{k+1}, \rho) + B^T \lambda_i^{k+1} = 0. \tag{3.17}$$

Note that equations (3.14), (3.16), and (3.17) correspond to the KKT conditions for the optimization problem in (3.4), which is strongly convex. The result then follows. □

We show in the next subsection that the convergence of IHS can be guaranteed by the selections of β and Λ_{i+1}.

3.3.3 Convergence

In this subsection, the convergence of IHS is analyzed under the framework of the majorization-minimization principle, which was originally proposed by Ortega and Rheinboldt (Ortega & Rheinboldt, 1970). The majorization-minimization principle gives a guide to designing algorithms for convex optimization problems with convergence guaranteed. The key idea is to find an approximate cost function that is easy to be optimized and acts as the upper bound of the original cost function. In IHS, approximate cost function (3.11) is used rather than the original one (3.5). Therefore, the tuning parameters β and Λ_{i+1} play an important role in the convergence of IHS. In the following two lemmas, we show the existence of β and Λ_{i+1}, which result in upper bound functions. The convergence of IHS is then given in Theorem 3.6.

Lemma 3.2. *For any $x > 0$ and $y > 0$, there exists $\beta \in (0, 1]$ such that $-\log(y) - \frac{1}{y}(x - y) + \frac{1}{2\beta y^2}(x - y)^2 \geq -\log(x)$.*

Proof. The result follows from the second-order expansion of $-\log(x)$ at y. □

Lemma 3.3. *For a given $\rho > 0$, there exists $\Lambda_{i+1} > 0$ such that $\hat{V}^k_{N-i}(x_i, \rho, \Lambda_{i+1}) \geq V_{N-i}(x_i, \rho)$ for any $x_i \in \mathbb{A}_i$.*

Proof. For each $i \in \{0, \cdots, N-1\}$, from the fact that $V_{N-i}(x_i, \rho)$ is smooth and strongly convex in $x_i \in \mathbb{A}_i$ for $\rho > 0$, we have the second-order expansion of $V_{N-i}(x_i, \rho)$ at $x^k_i \in \mathbb{A}_i$:

$$V_{N-i}(x_i, \rho) = V_{N-i}(x^k_i, \rho) + \nabla_{x_i} V_{N-i}(x^k_i, \rho)(x_i - x^k_i) + \frac{1}{2}\|x_i - x^k_i\|^2_{H_{V_{N-i}}(t,\rho)}, \tag{3.18}$$

where $H_{V_{N-i}}(t, \rho) > 0$ is the Hessian matrix of $V_{N-i}(x_i, \rho)$ with respect to x_i evaluated at $(tx_i + (1-t)x^k_i, \rho)$ for some $t \in [0, 1]$. Moreover, it can be known that $H_{V_{N-i}}(t, \rho)$ has finite entries for a given $\rho > 0$. Comparing the expression of $\hat{V}^k_{N-i}(x_i, \rho, \Lambda_{i+1})$ with (3.18) and the finite property of $H_{V_{N-i}}(t, \rho)$, the result follows if Λ_{i+1} is selected such that $\Lambda_{i+1} - H_{V_{N-i}}(t, \rho) \geq 0$. □

Theorem 3.6. *For given $x_{i-1} \in \mathbb{A}_{i-1} \neq \emptyset$ and $\rho > 0$, let $\{x^k_i\}$ and $\{u^k_i\}$ be the iterates generated by IHS in Algorithm 3.1. Then, there exist $\beta \in (0, 1]$ and $\Lambda_{i+1} > 0$ such that $\{x^k_i\}$ and $\{u^k_i\}$ converge with the descent cost*

$$\mathcal{L}_i(x^{k+1}_i, u^{k+1}_i, \rho) \leq \mathcal{L}_i(x^k_i, u^k_i, \rho) \tag{3.19}$$

while satisfying $x_{i-1} + Ax^k_i + Bu^k_i = 0$ and $x_{i-1} + Ax^{k+1}_i + Bu^{k+1}_i = 0$ for $k = 1, 2, \cdots$, and the equality holds only if $x^k_i = x^{k+1}_i = x^*_i$ and $u^k_i = u^{k+1}_i = u^*_i$.

Proof. Since x^k_i and u^k_i are the optimal solutions to $\mathcal{P}^k_i$ and $\mathbb{A}_{i-1} \neq \emptyset$, the constraint $x_{i-1} + Ax^k_i + Bu^k_i = 0$ is satisfied inherently for $k = 1, 2, \cdots$.

According to Lemma 3.2, we choose $\beta \in (0, 1]$ such that

$$\Phi^k(x_i, u_i, \rho, \beta) \geq \Phi(x_i, u_i, \rho). \tag{3.20}$$

In Algorithm 3.1, since λ^{k+1}_{i+1} is recovered from $V_{N-i}(x^k_i, \rho)$, $\lambda^{k+1}_{i+1} = \lambda^*_{i+1}(x^k_i, \rho) = \nabla^T_{x_i} V_{N-i}(x^k_i, \rho)$ holds. Then, the summation of the last three terms of $\hat{\mathcal{L}}^k_i(x_i, u_i, \lambda^{k+1}_{i+1}, \rho)$ in (3.11) satisfies

$$V_{N-i}(x^k_i, \rho) + (x_i - x^k_i)^T \lambda^{k+1}_{i+1} + \frac{1}{2}\|x_i - x^k_i\|^2_{\Lambda_{i+1}} = \hat{V}^k_{N-i}(x_i, \rho, \Lambda_{i+1}).$$

From Lemma 3.3, we choose $\Lambda_{i+1} > 0$ such that

$$\hat{V}^k_{N-i}(x_i, \rho, \Lambda_{i+1}) \geq V_{N-i}(x_i, \rho). \tag{3.21}$$

Together with (3.20) and (3.21), we have

$$\hat{\mathcal{L}}^k_i(x_i, u_i, \lambda^{k+1}_{i+1}, \rho) \geq \mathcal{L}_i(x_i, u_i, \rho). \tag{3.22}$$

Substituting x^k_i and u^k_i into both sides of (3.22) yields

$$\hat{\mathcal{L}}^k_i(x^k_i, u^k_i, \lambda^{k+1}_{i+1}, \rho) = \mathcal{L}_i(x^k_i, u^k_i, \rho). \tag{3.23}$$

It should be noted that x^{k+1}_i and u^{k+1}_i are the optimal solutions to $\mathcal{P}^k_i(x_{i-1}, \lambda^{k+1}_{i+1}, \rho)$. Therefore, we have a descent cost:

$$\hat{\mathcal{L}}^k_i(x^{k+1}_i, u^{k+1}_i, \lambda^{k+1}_{i+1}, \rho) \leq \hat{\mathcal{L}}^k_i(x^k_i, u^k_i, \lambda^{k+1}_{i+1}, \rho), \tag{3.24}$$

and the equality holds only if $x^k_i = x^{k+1}_i$ and $u^k_i = u^{k+1}_i$, which indicates optimal solutions x^*_i and u^*_i from Lemma 3.1. The conclusion then follows from (3.22), (3.23), and (3.24). $\square$

3.4 Polishing

When β and Λ_{i+1} are chosen so that the convergence of IHS is guaranteed for a sufficiently small ρ according to Theorem 3.6, IHS behaves like a first-order method and therefore requires lots of iterations. When $\rho > 0$ is set to be not too small, a solution with a low or medium accuracy can be obtained with a limited NOIs. For this case, we discuss solution polishing in this section.

After obtaining an approximate solution using Algorithm 3.1, we can guess which constraints are active and solve an optimization problem with only equality constraints. Alternatively, we can iteratively polish the solution by applying the IHS algorithm to the inequality-constrained OCP (3.1).

Similarly to Definitions 3.2, 3.4, and 3.5, we have the following definitions of the value function for the OCP (3.1), linearized primal regularization, and the subproblem, respectively.

Definition 3.7. *For $i \in \{1, \cdots, N\}$, we define the i-th value function $V^{(AS)}_{N-(i-1)}(x_{i-1}) : \mathbb{A}_{i-1} \to \mathbb{R}$ of the OCP (3.1) as*

$$\begin{aligned} V^{(AS)}_{N-(i-1)}(x_{i-1}) := \min_{x_i, u_i} \; & L_i(x_i, u_i) + V^{(AS)}_{N-i}(x_i) \\ \text{s.t.} \quad & x_{i-1} + Ax_i + Bu_i = 0, \\ & Dx_i + Eu_i + b \geq 0, \end{aligned} \tag{3.25}$$

where $V_0^{(AS)}(x_N) = 0$. Here, the superscript "AS" stands for "active set," which indicates that the inequality constraints are handled explicitly in (3.25)*, rather than being transferred into a barrier function in* (3.4)*. We define $\mathcal{S}_{N-(i-1)}^{(AS)}(x_{i-1}) : \mathbb{A}_{i-1} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_x}$ as the mapping that recovers the optimal solutions denoted by $x_i^{*(AS)}$, $u_i^{*(AS)}$, and $\lambda_i^{*(AS)}$ for a given value of x_{i-1}. Here, $\lambda_i^{(AS)}$ is the Lagrange multiplier corresponding to the dynamic constraint in* (3.25)*. It should be noted that $\lambda_i^{(AS)}$ and λ_i in the interior-point problem* (3.4) *generally have different values even when the same optimal solution has been reached.*

Definition 3.8. *For $i \in \{1, \cdots, N\}$, we denote the i-th primal regularization of the linearized value function at the k-th iteration by $\hat{V}_{N-(i-1)}^{k(AS)}(x_{i-1}, \Lambda_i) : \mathbb{A}_{i-1} \times \mathbb{R}^{n_x \times n_x} \to \mathbb{R}$, that is,*

$$\begin{aligned}&\hat{V}_{N-(i-1)}^{k(AS)}(x_{i-1}, \Lambda_i)\\ &:= V_{N-(i-1)}^{(AS)}(x_{i-1}^k) + (x_{i-1} - x_{i-1}^k)^T \lambda_i^{*(AS)}(x_{i-1}^k) + \frac{1}{2}\|x_{i-1} - x_{i-1}^k\|_{\Lambda_i}^2,\end{aligned} \tag{3.26}$$

where $\Lambda_i > 0$ holds, and $\lambda_i^{*(AS)}(x_{i-1}^k)$ is obtained from $\mathcal{S}_{N-(i-1)}^{(AS)}(x_{i-1}^k)$.

Definition 3.9. *For a given value of Λ_{i+1}, an optimization problem $\mathcal{P}_i^{k(AS)}$, $i \in \{1, \cdots, N\}$, which is parametric to x_{i-1} and λ_{i+1}, is defined as follows.*

$$\begin{aligned}\mathcal{P}_i^{k(AS)}(x_{i-1}, \lambda_{i+1}) : \min_{x_i, u_i} \; & L_i(x_i, u_i) + V_{N-i}^{(AS)}(x_i^k) + (x_i - x_i^k)^T \lambda_{i+1} + \frac{1}{2}\|x_i - x_i^k\|_{\Lambda_{i+1}}^2\\ \text{s.t.} \quad & x_{i-1} + Ax_i + Bu_i = 0,\\ & Dx_i + Eu_i + b \geq 0\end{aligned} \tag{3.27}$$

Likewise, we perform IHS directly on the inequality-constrained problem in Algorithm 3.2, which is also a recursive algorithm.

To show the convergence of Algorithm 3.2, we make an assumption on the quality of the approximate solution as follows.

Assumption 3.1. *The approximate solution obtained by Algorithm 3.1, i.e., x_i^0 in Algorithm 3.2, is closed to x_i^* such that there is no active set change after the first iteration when solving $\mathcal{P}_j^{k(AS)}$, $j = i+1, \cdots, N$. Meanwhile, we assume that the constraints in the active set are strongly active.*

From the point of view of guessing which constraints are active, Assumption 3.1 guarantees that the active set guess is exactly the optimal active set. It can be shown similarly to Lemma 3.1 that the optimal solution to the inequality-constrained

Algorithm 3.2 IHS for (3.25) $(x_i^{(\mathrm{AS})}, u_i^{(\mathrm{AS})}, \lambda_i^{(\mathrm{AS})}) \leftarrow \mathcal{S}_{N-(i-1)}^{(\mathrm{AS})}(x_{i-1})$

```
Input: $x_{i-1}$
Output: $(x_i^{(\mathrm{AS})}, u_i^{(\mathrm{AS})}, \lambda_i^{(\mathrm{AS})})$
1: Initialize: $k \leftarrow 0$; $x_i^0$ and $u_i^0$ coming from the results returned by
               Algorithm 3.1
2: repeat:
3:     $(-, -, \lambda_{i+1}^{k+1(\mathrm{AS})}) \leftarrow \mathcal{S}_{N-i}^{(\mathrm{AS})}(x_i^k)$
4:     $(x_i^{k+1}, u_i^{k+1}, \lambda_i^{k+1(\mathrm{AS})}) \leftarrow \mathcal{P}_i^{k(\mathrm{AS})}(x_{i-1}, \lambda_{i+1}^{k+1(\mathrm{AS})})$
5:     $k \leftarrow k + 1$
6: until convergence criterion is met
7: $(x_i^{(\mathrm{AS})}, u_i^{(\mathrm{AS})}, \lambda_i^{(\mathrm{AS})}) \leftarrow (x_i^k, u_i^k, \lambda_i^{k(\mathrm{AS})})$
```

problem is approached if Algorithm 3.2 converges and the strongly active assumption holds, which indicates that the approximate solution is polished. The convergence of Algorithm 3.2 can be guaranteed under Assumption 3.1 as shown in the following theorem.

Theorem 3.10. *Let x_i^0 and u_i^0 be generated by Algorithm 3.1 and Assumption 3.1 hold. Then, there exists $\Lambda_{i+1} > 0$ such that the iterates $\{x_i^k\}$ and $\{u_i^k\}$ converge to $x_i^{*(AS)}$ and $u_i^{*(AS)}$, respectively.*

Proof. It can be known from Bemporad et al. (2002) that $V_{N-i}^{(\mathrm{AS})}(x_i)$ is a piecewise quadratic function of x_i. Since Assumption 3.1 holds, $V_{N-i}^{(\mathrm{AS})}(x_i)$ is differentiable (because of the strongly active property) at x_i^k satisfying

$$\nabla_{x_i}^T V_{N-i}^{(\mathrm{AS})}(x_i^k) = \lambda_{i+1}^{*(\mathrm{AS})}(x_i^k) \tag{3.28}$$

and can be expanded as

$$V_{N-i}^{(\mathrm{AS})}(x_i^{k+1}) = V_{N-i}^{(\mathrm{AS})}(x_i^k) + (x_i^{k+1} - x_i^k)^T \lambda_{i+1}^{*(\mathrm{AS})}(x_i^k) + \frac{1}{2}\|x_i^{k+1} - x_i^k\|_{\Lambda_{i+1}^*}^2, \tag{3.29}$$

where $\Lambda_{i+1}^* > 0$ holds. Similarly to the proof of Theorem 3.6, if $\Lambda_{i+1} - \Lambda_{i+1}^* \geq 0$, the result then follows. □

3.5 Practical implementation

In this section, we discuss the details of the practical implementation of IHS. It should be noted that these discussions apply to the polishing procedure as well.

3.5.1 Choosing the regularization matrix

It is no doubt that the regularization matrix Λ plays an important role in the rate of convergence of IHS. Motivated by the fact that the second-order derivatives of the value functions for an unconstrained OCP are constants (see Lemma 3.4) and can be computed offline, these constant second-order derivatives can be used when choosing the regularization matrices to improve the rate of convergence while maintaining a cheap iteration.

Lemma 3.4. *Let*

$$\nabla^2_{x_{i-1}} V_{N-(i-1)}(x_{i-1}, \rho)$$

be the second-order derivative of $V_{N-(i-1)}(x_{i-1}, \rho)$ with respect to x_{i-1} evaluated at (x_{i-1}, ρ). Then, $\nabla^2_{x_{i-1}} V_{N-(i-1)}(x_{i-1}, 0)$ is a constant positive definite matrix denoted by W_i for all $x_{i-1} \in \mathbb{A}_{i-1}$ and $i \in \{1, \cdots, N\}$.

Proof. From that facts that $V_{N-(i-1)}(x_{i-1}, \rho)$ is strongly convex in $x_{i-1} \in \mathbb{A}_{i-1}$ for $\rho \in \mathbb{R}_{\geq 0}$ and $V_{N-(i-1)}(x_{i-1}, 0)$ corresponds to the value function of the unconstrained OCP, i.e., linear quadratic regulator, we can know that the statement holds and these constant matrices can be computed by the recursion

$$W_i = (BR_i^{-1}B^T + A(Q_i + W_{i+1})^{-1}A^T)^{-1} \tag{3.30}$$

with $W_{N+1} = 0$. Note that (3.30) is essentially the Riccati recursion performed for the reverse-time system. □

For example, we can choose the regularization matrices to be

$$\Lambda_{i+1} = \frac{1}{\gamma} W_{i+1}, \ \gamma \in (0, 1], \ i \in \{1, \cdots, N\}, \tag{3.31}$$

where $1/\gamma$ is a scaling factor that guarantees convergence from Theorem 3.6.

Remark 3.3. *The regularization matrices can be chosen considering the distances to constraints' boundaries. For example, Λ_i can be obtained from the negative of the n_x-th leading principal submatrix of the KKT matrix*

$$\begin{bmatrix} 0 & 0 & 0 & B & A \\ 0 & 0 & -I & D & C \\ 0 & -I & \Delta & 0 & 0 \\ B^T & D^T & 0 & R_i & 0 \\ A^T & C^T & 0 & 0 & Q_i + \Lambda_{i+1} \end{bmatrix}^{-1},$$

where $\Delta \in \mathbb{R}^{n_c \times n_c}$ is a positive semi-definite diagonal matrix with its i-th diagonal entry indicating how close the i-th constraint approaches its boundary. When $\Delta = 0$, i.e., the unconstrained case, Λ_i decays into W_i.

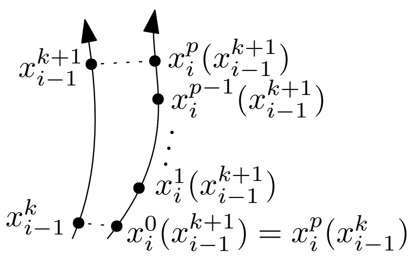


Figure 3.1: Iterates of $x_i = x_i(x_{i-1}^{k+1})$ when calculating $\mathcal{S}_{N-(i-1)}(x_{i-1}^{k+1}, \rho)$ warm started from its last optimal solution $x_i^p(x_{i-1}^k)$.

3.5.2 Limiting NOIs

Note that IHS is a recursive method and requires solving a large number of the subproblems (3.10). Let us assume that p iterations are needed in Algorithm 3.1 to converge for each $i \in \{1, \cdots, N-1\}$. Then, the total number of the subproblems solved in IHS has a complexity of $\mathcal{O}(p^N)$, which is unacceptable when either p or N is large.

To decrease the computation time (CT), the NOIs p can be limited to a small number, which may result in a divergent algorithm. To tackle this problem, we notice that, in IHS, the i-th subproblem $\mathcal{P}_i^k$ is visited p times more than the $(i-1)$-th subproblem $\mathcal{P}_{i-1}^k$ when $i \neq N$. This can be interpreted as the i-th subproblem taking a shorter or more conservative step at every iteration than the $(i-1)$-th subproblem does, which is illustrated in Fig. 3.1. To mimic IHS while setting $p = 1$, we add an artificial damping parameter $\alpha \in (0, 1]$ to limit the propagation of the state so that the initial state of the later subproblem varies slowly and thus results in a small update step. The approximate IHS (AIHS) method is summarized in Algorithm 3.3, in which a basic barrier mechanism is integrated. Here, the fraction-to-the-boundary (Nocedal & Wright, 2006) procedure is used to prevent the variables from approaching their bounds too quickly. The barrier parameter is decreased at each iteration.

Compared with $\mathcal{O}(N(n_x + n_u)^3)$ for the method of Riccati recursion, the computational complexity is reduced to $\mathcal{O}(N(n_u^3 + n_x^2))$ for AIHS as shown in Remark 3.2. The reduction in the computation is mainly caused by avoiding performing Riccati recursion online. Moreover, the KKT matrices for $\{\mathcal{P}_i^k\}_{i=1}^N$ can be factorized in parallel, reducing the computational complexity to $\mathcal{O}(n_u^3 + N(n_x n_u + n_x^2))$.

Remark 3.4. *Similarly to AIHS in Algorithm 3.3, when performing only one iteration for Algorithm 3.2 in the polishing procedure, a damping parameter α is needed consequently, and the barrier strategy is no longer required.*

Algorithm 3.3 AIHS for (3.4) with $i = 1$ and $x_0 = \bar{x}_0$

```
Input: $\bar{x}_0$
Output: $X^*$ and $U^*$
 1: Initialize: $k \leftarrow 0$; $\alpha \in (0,1]$; barrier parameters $\rho^0 > 0$, $\delta \geq 1$, $\rho_{min} > 0$,
                and $\tau \leftarrow 0.1$; initial guesses $X^0$ and $U^0$ satisfying
                $Dx_i^0 + Eu_i^0 + b > 0, \forall i \in \{1, \cdots, N\}$
 2: repeat:
 3:     $x_0^k \leftarrow \bar{x}_0$, $x_0^{k+1} \leftarrow \bar{x}_0$, and $\lambda_{N+1}^{k+1} \leftarrow 0$
 4:     for $i = N$ to 1 do
 5:         $(-, -, \lambda_i^{k+1}) \leftarrow \mathcal{P}_i^k(x_{i-1}^k, \lambda_{i+1}^{k+1}, \rho^k)$
 6:     end for
 7:     for $i = 1$ to $N$ do
 8:         $(x_i^{k+1}, u_i^{k+1}, -) \leftarrow \mathcal{P}_i^k(x_{i-1}^{k+1}, \lambda_{i+1}^{k+1}, \rho^k)$
 9:         $(x_i^{k+1}, u_i^{k+1}) \leftarrow (1-\alpha)(x_i^k, u_i^k) + \alpha(x_i^{k+1}, u_i^{k+1})$
10:     end for
11:     procedure FRACTION-TO-THE-BOUNDARY
12:         for $i = 1$ to $N$ do
13:             $G_i^k \leftarrow Dx_i^k + Eu_i^k + b$
14:             $G_i^{k+1} \leftarrow Dx_i^{k+1} + Eu_i^{k+1} + b$
15:         end for
16:         $\eta = \max\{\eta \in (0,1] : (1-\eta)G_i^k + \eta G_i^{k+1} \geq \tau G_i^k,\ \forall i \in \{1, \cdots, N\}\}$
17:         $(X^{k+1}, U^{k+1}) \leftarrow (1-\eta)(X^k, U^k) + \eta(X^{k+1}, U^{k+1})$
18:     end procedure
19:     $\rho^{k+1} \leftarrow \max\{\rho_{min}, \rho^k/\delta\}$
20:     $k \leftarrow k + 1$
21: until convergence criterion is met
22: $(X^*, U^*) \leftarrow (X^k, U^k)$
```

3.6 Numerical experiments

Two numerical examples were considered: an aircraft example with simple bound constraints and a quadrotor example with both bound and linear inequality constraints.

AIHS was compared with the method of Riccati recursion, the active-set (AS) method implemented with qpOASES (Ferreau, Kirches, Potschka, Bock, & Diehl, 2014), and the alternating direction method of multipliers (ADMM) implemented with OSQP (Banjac et al., 2017). The method of Riccati recursion was implemented in the same way as AIHS in Algorithm 3.3, however, with $\alpha = \beta = 1$ and the second-order derivatives of the value functions computed at every iteration, which is equivalent to performing Newton's method on the relaxed problem (3.2). The convergence criterion for AIHS and the method of Riccati recursion was chosen as

$$\|U^k - U^{k-1}\|_2 \leq \epsilon, \tag{3.32}$$

where $\epsilon > 0$ is the tolerance. We applied the algorithm discussed in Remark 3.4 for solution polishing and used the same parameters as those in AIHS.

All state constraints were softened with an extra slack variable s introduced. For example, a state constraint $x \leq 0$ is softened into two constraints $x - s \leq 0$ and $s \geq 0$ with a quadratic penalty on s. For AIHS and the method of Riccati recursion, one slack variable was introduced for each condensed subproblem, which leads to an OCP with $N(n_u + 1)$ variables. For AS and ADMM, a condensed OCP with only $Nn_u + 1$ variables was obtained.

For AS, all constraints were initialized to be inactive, and a pre-computed Cholesky factor of the Hessian matrix was fed to the solver. For ADMM, a fixed NOIs, which was chosen to be 100, was performed since up to thousands of iterations are needed to reach a comparable accuracy to other methods, and only the CT per iteration was compared. The initial guesses were chosen to be $X^0 = 0$, $U^0 = 0$, and $s^0 = 1$ for all methods.

All methods were implemented as MEX functions in MATLAB 2018a on a 3.40 GHz (Turbo Boost was disabled) Intel Core i7-6700 computer running Ubuntu 18.04.1. To reduce the effect of the operating system on the CT, every MPC problem was solved 100 times, and the minimal CT was chosen.

3.6.1 Aircraft

The aircraft considered here can be found in Kapasouris, Athans, and Stein (1988). The state vector of the aircraft is $x = [v, q, \dot{\theta}, \theta]^T \in \mathbb{R}^4$, which represents the forward velocity, angle of attack, pitch rate, and pitch angle. The input vector $u = [\delta_e, \delta_f]^T \in \mathbb{R}^2$ represents the elevator angle and flaperon angle. The goal is to track given references of q and θ under input and state constraints: $[-25, -25]^T \leq u \leq [25, 25]^T$ and $-15 \leq \dot{\theta} \leq 15$. The weighting matrices were chosen as $Q_i = \text{diag}(0.01, 10, 0.01, 1)$ and $R_i = 0.01I$, $\forall i \in \{1, \cdots, N\}$. The quadratic penalty on the slack variable was 1000. The sampling interval was 0.05 s.

For AIHS, the following parameters were chosen: $\alpha = 0.9$, $\beta = 0.7$, $\rho_0 = 10$, $\delta = 2$, and $\epsilon = \rho_{min} = 10^{-3}$. The regularization matrices $\{\Lambda_{i+1}\}_{i=1}^{N}$ were computed following (3.31) with $\gamma = 0.7$. For the method of Riccati recursion, we chose $\rho_0 = 1000$, $\delta = 2$, and $\epsilon = \rho_{min} = 10^{-3}$. We collected 1000 samples for the initial state $\bar{x}_0$ and reference x_{r_i} that were picked randomly from $\{x : [-10, -2, -15, -11]^T \leq x \leq [10, 2, 15, 11]^T\}$ and $\{x : [0, -2, 0, -11]^T \leq x \leq [0, 2, 0, 11]^T\}$, respectively. All of the sampled MPC problems had feasible solutions, that is, $\bar{x}_0 \in \mathbb{A}_0$ for the corresponding x_{r_i}. Fig. 3.2

shows the average and maximum CT per iteration with different prediction horizons, and the corresponding NOIs are shown in Fig. 3.3.

We denote $U^{*(\mathrm{IP})}$ and $U^{*(\mathrm{AS})}$ as the solution returned by Algorithm 3.3 and the solution to the original MPC problem (3.1), respectively. The polished solution $U^{k(\mathrm{Pol})}$ is obtained following Remark 3.4 with k iterations. The average errors defined by

$$e^{*(\mathrm{IP})} := \|U^{*(\mathrm{IP})} - U^{*(\mathrm{AS})}\|_2$$

and

$$e^{k(\mathrm{Pol})} := \|U^{k(\mathrm{Pol})} - U^{*(\mathrm{AS})}\|_2$$

are shown in Fig. 3.4.

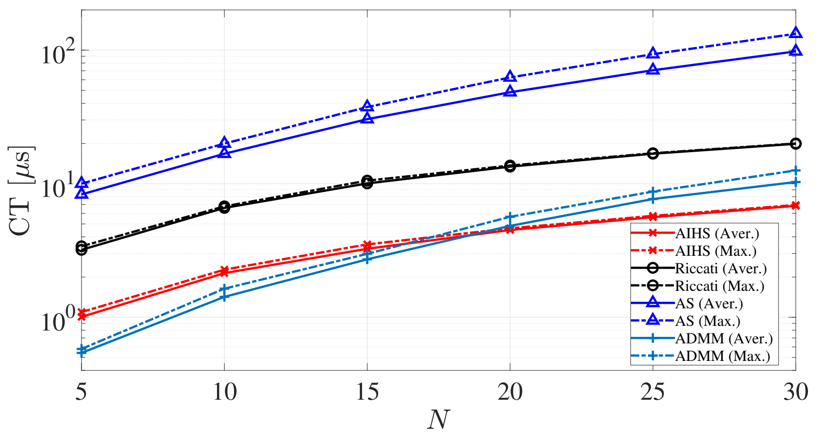


Figure 3.2: Average and maximum CT per iteration (aircraft).

3.6.2 Quadrotor

The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \theta, \varphi, \psi]^T \in \mathbb{R}^9$, where (X, Y, Z) and (θ, φ, ψ) are the position and angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. The model of the quadrotor is obtained by linearizing the following nonlinear model (Hehn & D'Andrea, 2011) at the equilibrium $x = 0$ and

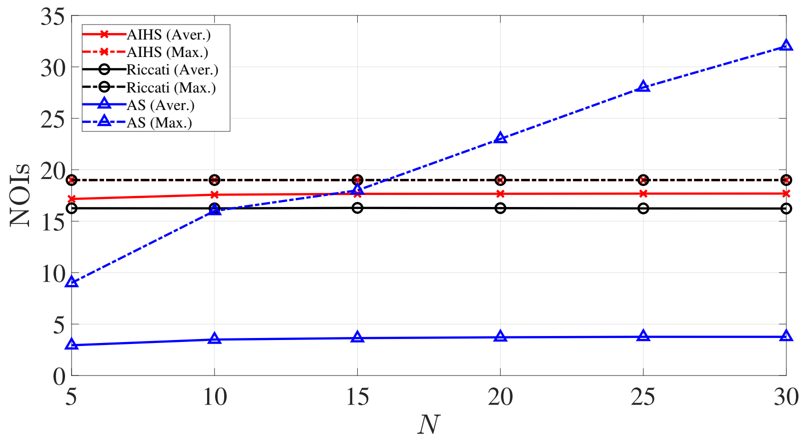


Figure 3.3: Average and maximum NOIs (aircraft).

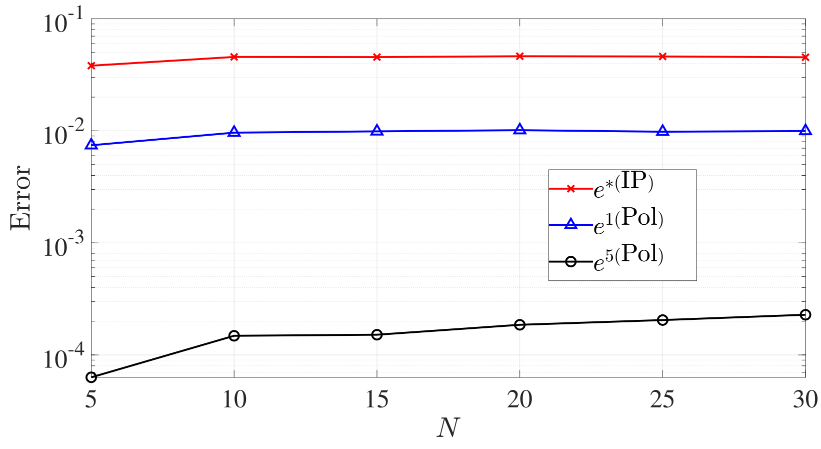


Figure 3.4: Errors after polishing with one and five iterations (aircraft).

$u = [g, 0, 0, 0]^T$ with $g = 9.81$.

$$\begin{aligned}
\ddot{X} &= a(\cos\theta \sin\varphi \cos\psi + \sin\theta \sin\psi) \\
\ddot{Y} &= a(\cos\theta \sin\varphi \sin\psi - \sin\theta \cos\psi) \\
\ddot{Z} &= a \cos\theta \cos\varphi - g \\
\dot{\theta} &= (\omega_X \cos\theta + \omega_Y \sin\theta)/\cos\varphi \\
\dot{\varphi} &= -\omega_X \sin\theta + \omega_Y \cos\theta \\
\dot{\psi} &= \omega_X \cos\theta \tan\varphi + \omega_Y \sin\theta \tan\varphi + \omega_Z
\end{aligned} \tag{3.33}$$

The goal is to track a given position reference under the following constraints: $-0.2 \leq \theta, \varphi, \psi \leq 0.2$ to guarantee linearity, $[0, -1, -1, -1]^T \leq u \leq [11, 1, 1, 1]^T$, and a maximum power output constraint $a + |\omega_X| + |\omega_Y| + |\omega_Z| \leq g + 2$, which is transferred equivalently into eight linear inequality constraints. The weighting matrices were chosen as $Q_i = \text{diag}(10, 1, 2, 1, 10, 1, 1, 1, 1)$ and $R_i = I$, $\forall i \in \{1, \cdots, N\}$. The quadratic penalty on the slack variable was 1000. The sampling interval was 0.05 s.

For AIHS, the following parameters were chosen: $\alpha = 0.7$, $\beta = 1$, $\rho_0 = 10$, $\delta = 2$, and $\epsilon = \rho_{min} = 10^{-3}$. The same barrier parameters and ϵ were chosen for the method of Riccati recursion. The regularization matrices $\{\Lambda_{i+1}\}_{i=1}^{N}$ were given by (3.31) with $\gamma = 1$. We collected 1000 samples for the initial state $\bar{x}_0$ and reference x_{r_i} that were chosen randomly from $\{x : -x_{\max} \leq x \leq x_{\max}\}$ and $\{x : -x_{r\max} \leq x \leq x_{r\max}\}$, respectively, where $x_{\max} = [1, 1, 1, 1, 1, 1, 0.2, 0.2, 0.2]^T$ and $x_{r\max} = [1, 0, 1, 0, 1, 0, 0, 0, 0]^T$. All of the sampled MPC problems had feasible solutions, that is, $\bar{x}_0 \in \mathbb{A}_0$ for the corresponding x_{r_i}. Fig. 3.5 shows the average and maximum CT per iteration with different prediction horizons, and the corresponding NOIs are shown in Fig. 3.6. The errors before and after polishing are shown in Fig. 3.7.

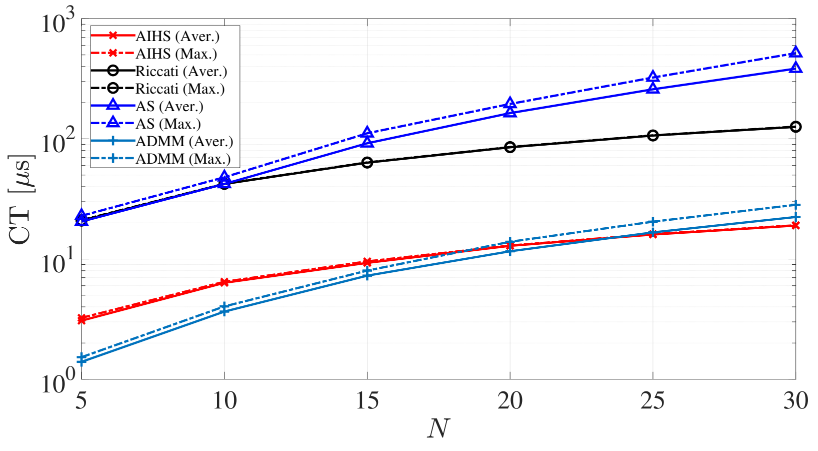


Figure 3.5: Average and maximum CT per iteration (quadrotor).

The quadrotor example shows the performance of AIHS when approaching constrained solutions, which can be seen from the NOIs of AS in Fig. 3.6 (a similar result can be observed for the aircraft example in Fig. 3.3). In addition, we demonstrate the performance of AIHS under an aggressive problem setting for the quadrotor example. Here, we chose the weighting matrices to be $Q_i = \text{diag}(10, 1, 2, 1, 10, 1, 10^{-3}, 10^{-3}, 10^{-3})$ and $R_i = 10^{-3}I$ so that the constraints could be easily active. The regularization matrices $\{\Lambda_{i+1}\}_{i=1}^{N}$ were chosen following Remark 3.3 with different values of Δ for comparison. We measured the suboptimality defined in (3.34) of AIHS during itera-

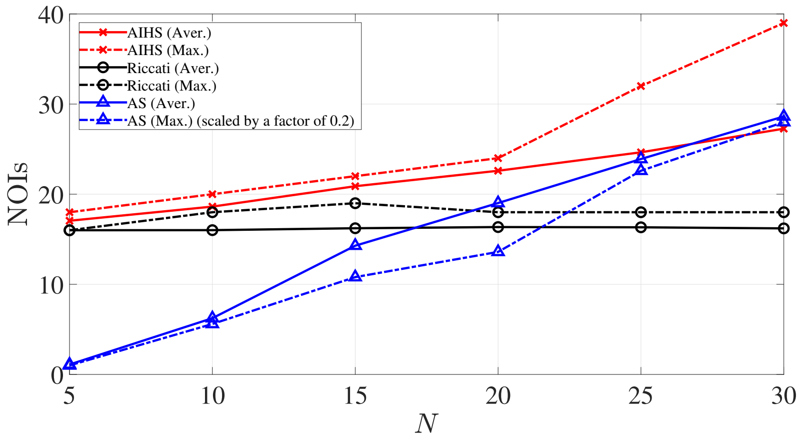


Figure 3.6: Average and maximum NOIs for the quadrotor example (the maximum NOIs of AS is scaled by a factor of 0.2).

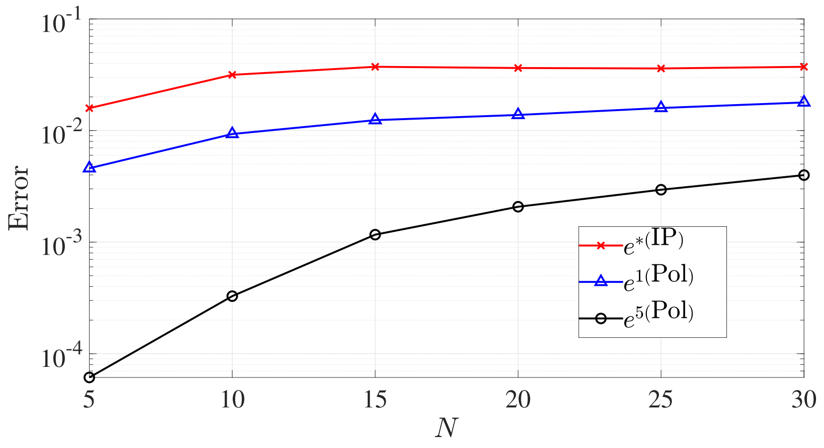


Figure 3.7: Errors after polishing with one and five iterations (quadrotor).

tion.

$$\sigma^k := \frac{\sum_{i=1}^{N} \left\{ L_i(x_i^k, u_i^k) - L_i(x_i^{*(\mathrm{AS})}, u_i^{*(\mathrm{AS})}) \right\}}{\sum_{i=1}^{N} L_i(x_i^{*(\mathrm{AS})}, u_i^{*(\mathrm{AS})})} \times 100\% \tag{3.34}$$

The percentages of samples that satisfied $\sigma^k < 1\%$ with different values of Δ are shown in Fig. 3.8, and the maximum and average suboptimalities are shown in Table 3.1.

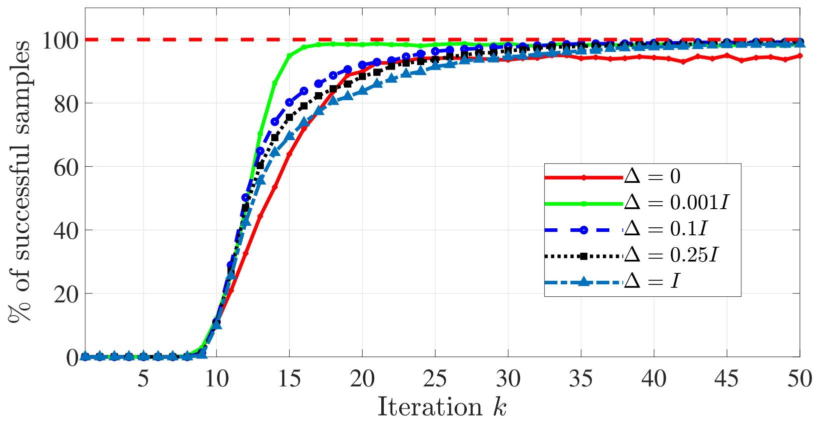


Figure 3.8: Percentages of successful samples with different values of Δ for the quadrotor example ($N = 15$).

Table 3.1: Maximum and average suboptimalities [%] with different values of Δ at different iterations (quadrotor).

		Δ				
		0	$0.001I$	$0.1I$	$0.25I$	I
σ^{20}	Max.	15.1	3.0	15.3	21.3	30.2
	Aver.	0.44	0.16	0.38	0.50	0.65
σ^{40}	Max.	101.0	3.8	3.2	4.4	6.1
	Aver.	0.40	0.16	0.15	0.16	0.19
σ^{60}	Max.	13.1	3.2	2.0	2.3	2.8
	Aver.	0.32	0.15	0.13	0.13	0.14
σ^{80}	Max.	10.1	6.4	1.8	1.9	2.0
	Aver.	0.28	0.16	0.13	0.13	0.13
σ^{100}	Max.	9.6	2.2	1.7	1.8	1.8
	Aver.	0.31	0.15	0.13	0.13	0.13

3.6.3 Analysis

It should be noted that AS converges to the exact solution while the others can only obtain suboptimal solutions to certain accuracies. AS and ADMM can take advantage of warm start to have fewer NOIs when performing closed-loop simulations. However, we demonstrated only the performance of these methods when iterating from scratch, which can be seen in the assessments of their worst-case performance, e.g., when facing a rapidly changing initial state or reference.

We can see from Fig. 3.2 and Fig. 3.5 that AIHS was much faster than the method of Riccati recursion and AS in terms of the average and maximum CT per iteration, even comparable to ADMM. Meanwhile, AIHS took just slightly more iterations than the method of Riccati recursion as can be seen in Fig. 3.3 and Fig. 3.6, and both maximum values of these two methods are lower than that of AS when the prediction horizon N is large. Note again that ADMM took hundreds of iterations in average. Although only suboptimal solutions were obtained for AIHS, their accuracies could be improved a lot with only several polishing iterations, which can be seen from Figs. 3.4 and 3.7. When the weighting matrices were chosen with a large condition number, most of the samples could converge to the given tolerance with a small NOIs from Fig. 3.8. Even for those samples that could not converge, a small suboptimality could be reached with a limited amount of iterations, as can be seen in Table 3.1, when Δ was chosen properly. It can be observed from the maximum suboptimalities in Table 3.1 that if Δ is chosen with larger diagonal entries, AIHS can have slower but convergent iterations.

3.7 Summary

This chapter presented an iterative method that depends on updating only the first-order derivatives of value functions. Unlike the method of Riccati recursion, where a backward recursion of updating second-order derivatives is required, the proposed method decomposes the MPC problem into easy-to-solve single-step OCPs, and only states and costates are exchanged between consecutive subproblems. Numerical experiments showed that AIHS breaks the trade-off of implicit methods when converging to a reasonable accuracy, that is, AIHS not only has a short CT per iteration but also keeps a small NOIs. The convergence analysis was conducted only for IHS and remains as future work for AIHS. Another direction is to construct subproblems that consist of multiple steps to improve the rate of convergence.

Chapter 4

Highly Parallelizable Newton-Type Method for NMPC – Algorithm

4.1 Introduction

Due to the rapid development of parallel devices such as multi-core processors, graphics processing units, and field-programmable gate arrays (FPGAs), there is growing demand for tailored parallel nonlinear model predictive control (NMPC) methods. However, most real-time methods can only be parallelized to a certain extent, such as parallel system simulation in the context of multiple shooting (Bock & Plitt, 1984) or parallel matrix operations. Although first-order methods such as the alternating direction method of multipliers (Jerez et al., 2014; O'Donoghue et al., 2013) and the fast gradient method (Jerez et al., 2014) can be easily parallelized and efficiently implemented for linear model predictive control (MPC), they suffer from slow rates of convergence compared with the Newton-type methods, dealing with complicated constraints, and time-varying dynamics in the underlying linearized problems when using sequential quadratic programming (SQP) for NMPC. Although the Newton-type methods are basically able to deal with general constraints and have fast rates of convergence, parallelization is still challenging. Methods based on parallel Riccati recursion (Frasch et al., 2015; Nielsen & Axehill, 2015) have been proposed. A computational complexity of $\mathcal{O}(\log N)$ for N threads can be achieved, where N is the number of prediction steps. Several tailored parallel methods for NMPC have been reported. The advanced multi-step NMPC (Yang & Biegler, 2013), which is parallelized along the time axis, concurrently solves several predicted optimal control programs (OCPs) beforehand, and then the input is updated on the basis of the state measurement. Particle swarm optimization (Xu, Chen, Gong, & Mei, 2016), with its inherent parallelism, has been implemented on FPGA for NMPC. An augmented

Lagrangian-based method tailored to nonlinear OCPs has been reported (Kouzoupis, Quirynen, Houska, & Diehl, 2016) that concurrently solves subproblems along the prediction steps and then solves a centralized consensus quadratic program (QP) to update the dual variables at each iteration. This method has a linear rate of convergence and its speed-up depends on the computation time of the consensus step in accordance with Amdahl's law (Amdahl, 1967).

In this chapter, a highly parallelizable Newton-type method is proposed. The discretized NMPC problem based on the reverse-time discretization method is first given. Then, a sequential method, which we call the backward correction method, is formulated. The backward correction method is proved to be exactly identical to Newton's method but with a clearer structure for possibility of parallelization. In the backward correction method, the coupling variable in each subproblem is calculated recursively in a backward manner, which is time-intensive. Reasonable approximations of the coupling variables are used to break down the recursion process so that the calculation can be done in parallel. The resulting algorithm is then proved to converge superlinearly. Numerical simulation of controlling a quadrotor demonstrated that the proposed method is highly parallelizable and can converge to a specified tolerance in only a few iterations.

This chapter is organized as follows. First, the discretized NMPC problem is given in Section 4.2. The backward correction method is formulated and proved to be exactly identical to Newton's method in Section 4.3. The proposed method and its rate of convergence are shown in Section 4.4. The numerical experiment is described and the results are discussed in Section 4.5. The key points are summarized and future work is mentioned in Section 4.6.

4.2 Problem statement

In this chapter, we consider the NMPC problem (2.19), however, without considering the inequality constraints $G_i(u_i, x_i) \geq 0$, $i \in \{1, \cdots, N\}$. The inequality constraints can be transferred into barrier costs under the framework of the interior-point method as introduced in Section 2.1.2 or into nonlinear algebraic equality constraints with additional slack variables introduced (Ohtsuka, 2004). We thus consider solving the following discretized NMPC problem with only equality constraints:

$$\begin{aligned}
&\min_{X,U} \sum_{i=1}^{N} L_i(u_i, x_i) \\
&\text{s.t.} \quad x_0 = \bar{x}_0, \\
&\qquad x_{i-1} + \mathcal{F}_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\
&\qquad C_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\},
\end{aligned} \tag{4.1}$$

where $X = (x_0, x_1, \cdots, x_N)$ and $U = (u_1, u_2, \cdots, u_N)$ are, respectively, the sequences of states and inputs along the horizon.

Note that we do not assume any particular discretization method in this chapter. Any explicit integration method, e.g., the Runge-Kutta method, can be used backward in time to result in the reverse-time discretization, and the obtained results, including the parallelization and convergence, apply as well. Discretization using the conventional explicit method is discussed in Remark 4.1.

4.2.1 KKT conditions

Let λ_i (costate) $\in \mathbb{R}^{n_x}$ and $\mu_i \in \mathbb{R}^{n_\mu}$ be the Lagrange multipliers corresponding to the i-th state equation and the equality constraint $C_i(u_i, x_i) = 0$, respectively. For the sake of brevity, we define

$$s := (\lambda, \mu, u, x).$$

Let $\mathcal{H}_i(s)$ be the Hamiltonian defined by

$$\mathcal{H}_i(s) := L_i(u, x) + \lambda^T \mathcal{F}_i(u, x) + \mu^T C_i(u, x).$$

Let $\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1})$ be defined by

$$\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1}) := \begin{bmatrix} x_{i-1} + \mathcal{F}_i(u_i, x_i) \\ C_i(u_i, x_i) \\ \nabla_u \mathcal{H}_i(s_i)^T \\ \lambda_{i+1} + \nabla_x \mathcal{H}_i(s_i)^T \end{bmatrix}.$$

The Karush-Kuhn-Tucker (KKT) conditions for the discretized NMPC problem (4.1) are

$$\mathcal{K}_i(x^*_{i-1}, s^*_i, \lambda^*_{i+1}) = 0, \ \forall i \in \{1, \cdots, N\}, \tag{4.2}$$

with $x^*_0 = \bar{x}_0$ and $\lambda^*_{N+1} = 0$.

Note that equations (4.2) are also called the discrete-time Euler-Lagrange equations, which are the first-order conditions for optimality of the optimization problem

(4.1). The nonlinear model predictive controller is implemented at each sampling instant by solving the KKT conditions (4.2) given current state $\bar{x}_0$, then using u_1^* as the actual input.

Remark 4.1. *Discretization using the reverse-time discretization method minimizes the complexity of coupling between neighboring stages of the KKT conditions. Namely,* x_{i-1} *and* λ_{i+1} *enter the KKT conditions with index* i *linearly. In the case of explicit discretization, e.g., using the forward Euler method, a similar structure with linear couplings can also be obtained for the corresponding reordered KKT conditions (Biegler & Thierry, 2018), and the results of parallelization and convergence can in principle be extended. However, it can be seen from the similar linear coupling structure that explicit discretization will not lead to better computational performance; thus, we herein focus on the reverse-time discretization.*

4.2.2 Notation

We define the following extra notations used in this chapter:

- $s_{i:j} := (s_i, s_{i+1}, \cdots, s_j)$, $\mathcal{K}_{i:j} := (\mathcal{K}_i, \mathcal{K}_{i+1}, \cdots, \mathcal{K}_j)$.
- $\mathcal{K}_{i:j}^k := \mathcal{K}_{i:j}(x_{i-1}^k, s_{i:j}^k, \lambda_{j+1}^k)$, $J_{i:j}^k := \nabla_{s_{i:j}} \mathcal{K}_{i:j}(x_{i-1}^k, s_{i:j}^k, \lambda_{j+1}^k)$, $\mathcal{K}_i^k := \mathcal{K}_{i:i}^k$, $J_i^k := J_{i:i}^k$.
- $S := s_{1:N}$, $\mathcal{K} := \mathcal{K}_{1:N}$, $J(S) := \nabla \mathcal{K}(S)$.
- For a matrix $A \in \mathbb{R}^{m \times n}$, we denote by $A[\alpha, \beta]$ the submatrix of A consisting of rows α and columns β. We define $[A]_k^U := A[1:k, 1:k]$ as the k-th leading principal submatrix of A and define $[A]_k^L := A[m-k+1:m, n-k+1:n]$.

4.3 Backward correction method

Newton's method for solving equations (4.2) is difficult to parallelize due to the coupling between stages. In this section, we will take a look at the iteration of each stage and investigate the coupling between stages. We formulate the backward correction method and prove that it is exactly identical to Newton's method.

4.3.1 Motivation

Newton's method is practical and powerful for solving the nonlinear algebraic equation $\mathcal{K}(S) = 0$ as it is simple to implement and converges quadratically in general. The full-step Newton's method performs the following iteration starting from an initial guess S^0 that is sufficiently close to S^*:

$$S^{k+1} = S^k - J(S^k)^{-1}\mathcal{K}(S^k), \ k = 0, 1, \cdots .$$

However, performing Newton's iteration is computationally expensive because one has to first evaluate $\mathcal{K}(S^k)$ and its Jacobian $J(S^k)$ and then solve a linear equation. The computational complexity of computing $J(S^k)^{-1}\mathcal{K}(S^k)$, in terms of N, can be made either $\mathcal{O}(N^2)$ by condensing (Frison & Jorgensen, 2013), or $\mathcal{O}(N)$ by exploiting the banded structure of $J(S)$, as proposed in Rao et al. (1998). Due to the coupling between stages ($J(S^k)$ is not block-diagonal), parallelization is challenging. Although methods using parallel Riccati recursion (Frasch et al., 2015; Nielsen & Axehill, 2015) can have complexities of $\mathcal{O}(\log(N))$, they are still computationally expensive. Note that, thanks to the reverse-time discretization of the state equation, the KKT conditions (4.2) at stage i are coupled with the neighboring stages linearly, that is, x_{i-1} and λ_{i+1} enter $\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1}) = 0$ linearly. Consequently, equation $\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1}) = 0$ can be formulated as the necessary condition for a single-period OCP with initial state x_{i-1} and terminal penalty function $\varphi\left(x_i\right) = \lambda_{i+1}^T x_i$. Therefore, solving $\mathcal{K}(S) = 0$ can be parallelized by solving a series of equations $\mathcal{K}_i(x^*_{i-1}, s_i, \lambda^*_{i+1}) = 0$ with respect to s_i for $i \in \{1, \cdots, N\}$ if the coupling variables x^*_{i-1} and λ^*_{i+1} are given for each stage i in advance. It should be noted that this resembles the principle of optimality, which states that any part of the optimal trajectory itself must be optimal. Unfortunately, x^*_{i-1} and λ^*_{i+1} cannot be known in advance, and only suboptimal values are given in general. When iterative methods are used for convergence, the iterations at the stage level should be investigated in order to parallelize them.

Consider the following Newton's iteration for solving $\mathcal{K}_i(\tilde{x}_{i-1}, s_i, \tilde{\lambda}_{i+1}) = 0$ approximately:

$$s_i^{k+1} = s_i^k - \nabla_{s_i}\mathcal{K}_i(\tilde{x}_{i-1}, s_i^k, \tilde{\lambda}_{i+1})^{-1}\mathcal{K}_i(\tilde{x}_{i-1}, s_i^k, \tilde{\lambda}_{i+1}), \tag{4.3}$$

where $\tilde{x}_{i-1}$ and $\tilde{\lambda}_{i+1}$ are the estimation of x^*_{i-1} and λ^*_{i+1}, respectively. One of the simplest methods takes $\tilde{x}_{i-1} = x^k_{i-1}$ and $\tilde{\lambda}_{i+1} = \lambda^k_{i+1}$. This method has complexity $\mathcal{O}(n^3)$ in parallel but it might diverge. The Gauss-Seidel scheme performs (4.3) recursively from $i = 1$ to N, with $\tilde{x}_{i-1} = x^{k+1}_{i-1}$ and $\tilde{\lambda}_{i+1}$, for example, estimated with a coarse-grained high-level controller in advance, as proposed in Zavala (2016). The convergence of the Gauss-Seidel scheme cannot be guaranteed either, unless λ_{i+1}^* is well estimated.

Next, we present a new method that estimates λ_{i+1}^* on the basis of the current k-th iteration's information. In the Gauss-Seidel method, stages $2, \cdots, N$ are updated after the update of stage 1, that is, $s_{2:N}^{k+1}$ is a function of x_1:

$$s_{2:N}^{k+1}(x_1) = s_{2:N}^k - \nabla_{s_{2:N}} \mathcal{K}_{2:N}(x_1, s_{2:N}^k, \lambda_{N+1})^{-1} \mathcal{K}_{2:N}(x_1, s_{2:N}^k, \lambda_{N+1}). \tag{4.4}$$

As x_1 enters $\mathcal{K}_{2:N}$ linearly, the following equation holds:

$$\mathcal{K}_{2:N}(x_1, s_{2:N}^k, \lambda_{N+1}) = \mathcal{K}_{2:N}(x_1^k, s_{2:N}^k, \lambda_{N+1}) + \begin{bmatrix} (x_1 - x_1^k)^T & 0^T \end{bmatrix}^T.$$

Therefore, the Jacobian matrix of $\mathcal{K}_{2:N}$ with respect to $s_{2:N}$ does not depend on x_1, and the update of λ_2 can be extracted from the first n_x elements of (4.4) and expressed as

$$\lambda_2^{k+1}(x_1) = \lambda_2^k + W_2^k(x_1 - x_1^k) - d_{\lambda_2}^k, \tag{4.5}$$

where $W_2^k = -\left[\left(J_{2:N}^k\right)^{-1}\right]_{n_x}^U \in \mathbb{R}^{n_x \times n_x}$ and $d_{\lambda_2}^k = \left(\left(J_{2:N}^k\right)^{-1}\right)[1:n_x, :] \cdot \mathcal{K}_{2:N}^k \in \mathbb{R}^{n_x}$. Although λ_2^* cannot be known in advance, it can be regarded as the correction of λ_2^k by (4.5) so that the iteration of solving $\mathcal{K}_1(x_0, s_1, \lambda_2^{k+1}(x_1)) = 0$ with respect to s_1 is given by

$$\begin{aligned} s_1^{k+1} &= s_1^k - \nabla_{s_1} \mathcal{K}_1(\bar{x}_0, s_1^k, \lambda_2^{k+1}(x_1^k))^{-1} \mathcal{K}_1(\bar{x}_0, s_1^k, \lambda_2^{k+1}(x_1^k)) \\ &= s_1^k - \left(J_1^k + \begin{bmatrix} 0 & 0 \\ 0 & W_2^k \end{bmatrix}\right)^{-1} \left(\mathcal{K}_1^k - \begin{bmatrix} 0 \\ d_{\lambda_2}^k \end{bmatrix}\right). \end{aligned}$$

After x_1^{k+1} is obtained, stages $2, \cdots, N$ can be further split and iterated recursively, which results in the algorithm described in the next subsection.

4.3.2 Algorithm

As stage N is the last stage and cannot be further split, the update of λ_N as a function of x_{N-1} can be directly formulated. After $\lambda_N^{k+1}(x_{N-1})$ is obtained, $\lambda_{N-1}^{k+1}(x_{N-2})$ can be formulated. Recursively, the update of λ_i, i.e., $\lambda_i^{k+1}(x_{i-1})$, is formulated on the basis of W_i^k and $d_{\lambda_i}^k$ from $i = N$ to 1. Then, s_i is updated from $i = 1$ to N following the Gauss-Seidel scheme. The resulting method is called the backward correction method (Algorithm 4.1), which has a computational complexity of $\mathcal{O}(Nn^3)$.

Next, we show that the backward correction method results in exactly the same update as Newton's method.

Algorithm 4.1 k-th iteration of backward correction method

```
Input: $\bar{x}_0$, $S^k$
Output: $S^{k+1}$
1: Initialize: $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^k = 0$, $W_{N+1}^k = 0$, $d_{\lambda_{N+1}}^k = 0$
2: for $i = N$ to 1 do
3:
       $$H_i^k \leftarrow \nabla_{s_i}\mathcal{K}_i(x_{i-1}^k, s_i^k, \lambda_{i+1}^{k+1}(x_i^k))^{-1}, \quad (4.6)$$
       $$\text{where } \mathcal{K}_i = \mathcal{K}_i\left(x_{i-1}, s_i, \lambda_{i+1}^{k+1}(x_i)\right), \quad (4.7)$$
       $$\text{and } \lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k + W_{i+1}^k(x_i - x_i^k) - d_{\lambda_{i+1}}^k. \quad (4.8)$$
       $$W_i^k \leftarrow -\left[H_i^k\right]_{n_x}^U \quad (4.9)$$
       $$d_{\lambda_i}^k \leftarrow H_i^k[1:n_x,:] \cdot \mathcal{K}_i\left(x_{i-1}^k, s_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right) \quad (4.10)$$
4: end for
5: for $i = 1$ to $N$ do
6:
       $$s_i^{k+1} \leftarrow s_i^k - H_i^k \cdot \mathcal{K}_i\left(x_{i-1}^{k+1}, s_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right) \quad (4.11)$$
7: end for
```

Theorem 4.1. *The backward correction method in Algorithm 4.1 is identical to Newton's method, that is, starting from the same S^k at the k-th iteration, the updated value obtained using Newton's method denoted by $S_{(nt)}^{k+1}$ is equal to the value obtained using the backward correction method denoted by $S_{(bc)}^{k+1}$.*

Proof. Define $\Delta S_{(nt)}^k := S_{(nt)}^{k+1} - S^k$ and $\Delta S_{(bc)}^k := S_{(bc)}^{k+1} - S^k$. First, the calculation of $\Delta S_{(nt)}^k$ is given. Note that stage i is coupled with only its neighboring stages linearly, so we know that

$$\nabla_{s_2}\mathcal{K}_1 = \nabla_{s_3}\mathcal{K}_2 = \cdots = \nabla_{s_N}\mathcal{K}_{N-1} = \begin{bmatrix} 0 & 0 \\ I_{n_x} & 0 \end{bmatrix} =: M_U \in \mathbb{R}^{n\times n} \tag{4.12}$$

and

$$\nabla_{s_1}\mathcal{K}_2 = \nabla_{s_2}\mathcal{K}_3 = \cdots = \nabla_{s_{N-1}}\mathcal{K}_N = \begin{bmatrix} 0 & I_{n_x} \\ 0 & 0 \end{bmatrix} =: M_L \in \mathbb{R}^{n\times n}. \tag{4.13}$$

Thus, Jacobian $J(S)$ is a block tridiagonal matrix with the form

$$J(S) = \begin{bmatrix} J_1 & M_U & & \\ M_L & J_2 & \ddots & \\ & \ddots & \ddots & M_U \\ & & M_L & J_N \end{bmatrix}. \tag{4.14}$$

The linear equation $J(S^k)\Delta S^k_{(nt)} + \mathcal{K}(S^k) = 0$ can be solved using block Gaussian elimination, that is, by solving

$$\begin{bmatrix} X_1^k & & & \\ M_L & X_2^k & & \\ & \ddots & \ddots & \\ & & M_L & X_N^k \end{bmatrix} \begin{bmatrix} \Delta s^k_{1(nt)} \\ \Delta s^k_{2(nt)} \\ \vdots \\ \Delta s^k_{N(nt)} \end{bmatrix} = - \begin{bmatrix} Y_1^k \\ Y_2^k \\ \vdots \\ Y_N^k \end{bmatrix}, \tag{4.15}$$

where the recursions are given by

$$\begin{aligned} &X_N^k = J_N^k,\ Y_N^k = \mathcal{K}_N^k, \\ &X_i^k = J_i^k - M_U (X_{i+1}^k)^{-1} M_L, \\ &Y_i^k = \mathcal{K}_i^k - M_U (X_{i+1}^k)^{-1} Y_{i+1}^k. \end{aligned}$$

Then (4.15) is solved recursively from $i = 1$ to N using

$$\Delta s^k_{i(nt)} = -(X_i^k)^{-1} \left(Y_i^k + M_L \Delta s^k_{i-1(nt)} \right), \tag{4.16}$$

where $\Delta s^k_{0(nt)} = 0$.

Next, we show that the recursion in Algorithm 4.1 matches the one in the block Gaussian elimination to prove the identity between $\Delta S^k_{(bc)}$ and $\Delta S^k_{(nt)}$. Combining (4.6) and (4.9) results in the recursion for calculating H_i^k being given by

$$H_i^k = \left(J_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1}^k \end{bmatrix} \right)^{-1} = \left(J_i^k - M_U H_{i+1}^k M_L \right)^{-1}. \tag{4.17}$$

Note that $H_N^k = (J_N^k)^{-1}$, which implies $H_N^k = (X_N^k)^{-1}$. It then follows from the definition of X_i^k and (4.17) that $H_i^k = (X_i^k)^{-1}$ for any $i \in \{1, \cdots, N\}$. From the definition of $\lambda_{i+1}^{k+1}(x_i)$ in (4.8), the following leftmost equation holds, and from (4.10), the rightmost equation holds.

$$\mathcal{K}_i \left(x_{i-1}^k, s_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = \mathcal{K}_i^k - \begin{bmatrix} 0 \\ d^k_{\lambda_{i+1}} \end{bmatrix} = \mathcal{K}_i^k - M_U H_{i+1}^k \cdot \mathcal{K}_{i+1} \left(x_i^k, s_{i+1}^k, \lambda_{i+2}^{k+1}(x_{i+1}^k) \right)$$

Together with $\mathcal{K}_N \left(x_{N-1}^k, s_N^k, \lambda_{N+1}^{k+1}(x_N^k) \right) = \mathcal{K}_N^k = Y_N^k$, we have $\mathcal{K}_i \left(x_{i-1}^k, s_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) = Y_i^k,\ i \in \{1, \cdots, N\}$. Then from (4.11), the recursion for calculating $\Delta s^k_{i(bc)}$ is given by

$$\begin{aligned} \Delta s^k_{i(bc)} &= -H_i^k \cdot \mathcal{K}_i \left(x_{i-1}^{k+1}, s_i^k, \lambda_{i+1}^{k+1}(x_i^k) \right) \\ &= -(X_i^k)^{-1} (Y_i^k + M_L \Delta s^k_{i-1(bc)}),\ i \in \{1, \cdots, N\}. \end{aligned} \tag{4.18}$$

A comparison of (4.16) with (4.18), together with the boundary condition $\Delta s^k_{0(bc)} = \Delta s^k_{0(nt)} = 0$, shows that $\Delta s^k_{i(bc)} = \Delta s^k_{i(nt)}$ for $i \in \{1, \cdots, N\}$. Thus, $S^{k+1}_{(bc)} = S^{k+1}_{(nt)}$ holds. □

The backward correction method is one of the methods exploiting the banded structure of $J(S)$ using block Gauss elimination. However, discretization of the system dynamics using the reverse-time method leads to linear couplings between neighboring stages in the KKT conditions (4.2), which makes the recursion in the backward correction method extremely simple compared with that in other structure-exploiting methods (see Glad and Jonson (1984), Rao, Wright, and Rawlings (1998), and Jørgensen, Rawlings, and Jørgensen (2004)).

4.4 Parallel method

The backward correction method is inherently non-parallelizable due to the recursion for calculating H_i^k in (4.6) from $i = N$ to 1 in order to formulate the expression of $\lambda_{i+1}^{k+1}(x_i)$ in (4.8) for each stage, which is the most computationally expensive part. Although the iterations are sequential, the iteration of each stage is clearly shown in (4.11). We show next that the backward correction method can be parallelized by using a reasonable approximation.

Recall that, in the iteration (4.3) for solving the KKT conditions (4.2), its convergence depends on the estimation of the coupling variables x_{i-1}^* and λ_{i+1}^* for each stage i. The estimation of λ_{i+1}^* in the backward correction method is performed using (4.8) and can be seen as an accurate estimation of λ_{i+1}^*. Consequently, a quadratic rate of convergence can be obtained. A slower rate of convergence can be achieved by using a relatively coarse estimation of λ_{i+1}^*.

To visualize the small variations of $W_{1,\cdots,N}^k$ as k increases, let us consider the situation in which the cost function $L(u,x,p)$ is quadratic, $\mathcal{F}(u,x,p)$ and the constraint $C(u,x,p)$ are linear, and the time-varying parameter p is fixed. Under these conditions, the Jacobian matrix of $\mathcal{K}$ with respect to S is constant, which means that $W_{1,\cdots,N}^k$ in (4.9) are always constant during iteration. As for general problems, we can expect small variations of $W_{1,\cdots,N}^k$ as k increases when the problem is less nonlinear and p varies slowly. Therefore, we propose estimating λ_{i+1}^* on the basis of W_{i+1}^{k-1} at the k-th iteration. That is, (4.8) in the backward correction method is replaced by

$$\lambda_{i+1}^{k+1}(x_i) = \lambda_{i+1}^k + W_{i+1}^{k-1}(x_i - x_i^k) - d_{\lambda_{i+1}}^k,\ i \in \{1,\cdots,N\}. \tag{4.19}$$

This removes the recursion for calculating H_i^k from $i = N$ to 1, so $H_{1,\cdots,N}^k$ can be calculated in parallel using

$$H_i^k = \left(J_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1}^{k-1} \end{bmatrix} \right)^{-1}. \tag{4.20}$$

4.4.1 Algorithm

Calculating $H^k_{1,\cdots,N}$ in parallel leads to an algorithm with complexity $\mathcal{O}(n^3 + Nn^2)$ for N threads, where n^3 is the complexity of the matrix inverse in (4.20), and Nn^2 indicates the complexities of carrying out (4.10) and (4.11) backward and forward, respectively.

Moreover, further parallelization is achieved by arranging the order of computations as shown in Algorithm 4.2. After $H^k_{1,\cdots,N}$ is obtained, an initial coarse update is done using $s^{k+1}_i = s^k_i - H^k_i \cdot \mathcal{K}^k_i$ (Step 1 of Algorithm 4.2). This coarse update introduces approximations for coupling variables (x_{i-1} and λ_{i+1}) compared with (4.11). The correction due to the approximation of λ_{i+1} is conducted in a backward manner (Step 2 of Algorithm 4.2). In the sequential part of Step 2, the approximation error $d_{\lambda_{i+1}}$ of λ_{i+1} is calculated for all stages, and then the other variables are corrected on the basis of the approximation error in the parallel part. Likewise, the correction due to the approximation of x_{i-1} is conducted in a forward manner (Step 3 of Algorithm 4.2). As a result of these two correction steps, the further parallelized implementation produces exactly the same update as the proposed method and reduces the complexity to $\mathcal{O}(n^3 + Nn_x^2)$ for N threads.

4.4.2 Convergence

In this subsection, the rate of convergence is shown for the proposed method by measuring the distance to the backward correction method.

At the k-th iteration, the proposed method differs from the backward correction method only in how $\lambda^{k+1}_{2,\cdots,N+1}$ are formulated: $\lambda^{k+1}_{i+1}(x_i) = \lambda^k_{i+1} + W^k_{i+1}(x_i - x^k_i) - d^k_{\lambda_{i+1}}$ for the backward correction method, $\lambda^{k+1}_{i+1}(x_i) = \lambda^k_{i+1} + W^{k-1}_{i+1}(x_i - x^k_i) - d^k_{\lambda_{i+1}}$ for the proposed method. To distinguish these two methods, let $v_{(bc)}$ be the variable in the backward correction method for a variable v. It can be seen that the proposed method approximates the backward correction method, where the approximation is introduced by

$$h^k_i := \Theta^{k-1}_i - \Theta^k_{i(bc)},\ i \in \{1, \cdots, N\}, \tag{4.21}$$

where

$$\Theta^k_{i(bc)} := \begin{bmatrix} 0 & 0 \\ 0 & -W^k_{i(bc)} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

and

$$\Theta^{k-1}_i := \begin{bmatrix} 0 & 0 \\ 0 & -W^{k-1}_i \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

Algorithm 4.2 Proposed parallelized implementation of the backward correction method (k-th iteration)

```
Input: $\bar{x}_0$, $S^k$, $W^{k-1}_{1,\cdots,N}$
Output: $S^{k+1}$, $W^{k}_{1,\cdots,N}$
 1: Initialize: $x_0^k = \bar{x}_0$, $\lambda^k_{N+1} = 0$, $W^{k-1}_{N+1} = 0$
 2: Step 1. Coarse update:
 3: for $i = 1$ to $N$ do in parallel
 4:     $\mathcal{K}_i^k \leftarrow \mathcal{K}_i(x^k_{i-1}, s^k_i, \lambda^k_{i+1})$
 5:     $J_i^k \leftarrow \nabla_{s_i}\mathcal{K}_i(x^k_{i-1}, s^k_i, \lambda^{k+1}_{i+1})$
 6:     $H_i^k \leftarrow \left(J_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W^{k-1}_{i+1} \end{bmatrix}\right)^{-1}$
 7:     $W_i^k \leftarrow -\left[H_i^k\right]^U_{n_x}$
 8:     $s_i^{k+1} \leftarrow s_i^k - H_i^k \cdot \mathcal{K}_i^k$
 9: end for
10: Step 2. Backward correction due to approximation of $\lambda$:
11: for $i = N-1$ to 1 do
12:     $d^{k+1}_{\lambda_{i+1}} \leftarrow \lambda^{k+1}_{i+1} - \lambda^k_{i+1}$
13:     $\lambda_i^{k+1} \leftarrow \lambda_i^{k+1} - H_i^k[1:n_x, n-n_x+1:n] \cdot d^{k+1}_{\lambda_{i+1}}$
14: end for
15: for $i = 1$ to $N-1$ do in parallel
16:     $s_i^{k+1}[n_x+1:n,:] \leftarrow s_i^{k+1}[n_x+1:n,:] - H_i^k[n_x+1:n, n-n_x+1:n] \cdot d^{k+1}_{\lambda_{i+1}}$
17: end for
18: Step 3. Forward correction due to approximation of $x$:
19: for $i = 2$ to $N$ do
20:     $d^{k+1}_{x_{i-1}} \leftarrow x^{k+1}_{i-1} - x^k_{i-1}$
21:     $x_i^{k+1} \leftarrow x_i^{k+1} - H_i^k[n-n_x+1:n, 1:n_x] \cdot d^{k+1}_{x_{i-1}}$
22: end for
23: for $i = 2$ to $N$ do in parallel
24:     $s_i^{k+1}[1:n-n_x,:] \leftarrow s_i^{k+1}[1:n-n_x,:] - H_i^k[1:n-n_x, 1:n_x] \cdot d^{k+1}_{x_{i-1}}$
25: end for
```

Theorem 4.1 has shown that the backward correction method is equivalent to Newton's method, so the iteration can be written as

$$S^{k+1}_{(bc)} = S^k - H(S^k) \cdot \mathcal{K}(S^k),$$

where $H(S^k) := J(S^k)^{-1}$. Let $D_{h_{1,\cdots,N}}$ and D_S be the open subsets of $\mathbb{R}^{n\times n}$ and $\mathbb{R}^{nN}$, respectively. Denote $h = (h_1, h_2, \cdots, h_N) \in D_{h_1} \times D_{h_2} \times \cdots \times D_{h_N} =: D_h$. Assume that $0 \in D_h$ and $S^* \in D_S$. The k-th iteration of the proposed method is expressed as

$$S^{k+1} = S^k - G(S^k, h^k),$$

where $G : D_S \times D_h \to \mathbb{R}^{nN}$ is a mapping that defines the increment with the proposed method. We have the following assumptions:

Assumption 4.1. *The solution S^* is an isolated solution to $\mathcal{K}(S) = 0$.*

Note that Assumption 4.1 guarantees the nonsingularities for not only the Jacobian matrix $J_{1:N}$ but the Jacobian matrices $J_{2,\cdots,N:N}$ in a neighborhood of S^*.

Assumption 4.2. *The function $\mathcal{K} : \mathbb{R}^{nN} \to \mathbb{R}^{nN}$ is Lipschitz continuous on D_S with Lipschitz constant L. The norm $\|J(S)^{-1}\|$ is bounded on D_S .*

Recall that the proposed method has the same algorithm as the backward correction method, except that λ^*_{i+1} is estimated by (4.19) instead of (4.8). Therefore, in the following proof, we refer to the equations in Algorithm 4.1 to show the convergence of Algorithm 4.2 for convenience.

The proof is structured as follows. Lemma 4.3 shows that $\lim_{k\to\infty} h_i^k = 0$ for all $i \in \{1, \cdots, N\}$ when the iteration converges. This implies that the proposed method approaches the backward correction method when it converges. To further show this, the explicit expression of $G(S, h)$ is given in Lemma 4.4 by $G(S, h) = P(S, h) \cdot \mathcal{K}(S)$, and the distance between $P(S, h)^{-1}$ and $J(S)$ is shown to be arbitrarily boundable in Lemma 4.4. Finally, it is shown in Theorem 4.2 that if S^0 is chosen close to S^* and h^0 close to 0, the convergence of the proposed method can be guaranteed. Moreover, since the proposed method approaches the backward correction method, which converges quadratically, the rate of convergence is shown to be superlinear in Theorem 4.2.

Lemma 4.1. *(Ortega & Rheinboldt, 1970) Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, $E \in \mathbb{R}^{n\times n}$, and $\|A^{-1}E\| < 1$. Then*

$$\|(A + E)^{-1}\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}E\|}.$$

Lemma 4.2. *(Ortega & Rheinboldt, 1970) Let $A \in \mathbb{R}^{n\times n}$ be nonsingular, $E \in \mathbb{R}^{n\times n}$, and $\|A^{-1}E\| < 1$. Then*

$$\|(A + E)^{-1} - A^{-1}\| \leq \|A^{-1}\| \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|}.$$

Lemma 4.3. *If the iterates $\{S^k\}$ given by the proposed method converge to S^*, then $h^k \to 0$ $(k \to \infty)$.*

Proof. This lemma can be proved by showing that $h_i^k \to 0$ $(k \to \infty)$ for all $i \in \{1, \cdots, N\}$. For each $i \in \{1, \cdots, N\}$, denote $M := N-i+1$ and $\beta := \left\| \left([J(S^*)]_{nM}^L \right)^{-1} \right\|$. Since $S^k \to S^*$ as $k \to \infty$, and $J(S)$ is continuous, then, for any $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$, there exists $k_m > 0$ such that, for $k > k_m$,

$$\left\| \left[J(S^j)\right]_{nM}^L - [J(S^*)]_{nM}^L \right\| < \frac{1}{2}\epsilon_J, \text{ for all } j \in \{k-M, \cdots, k\} \tag{4.22}$$

holds. Hence, the following must hold:

$$\left\| \left[J(S^j)\right]_{nM}^L - \left[J(S^k)\right]_{nM}^L \right\| < \epsilon_J, \text{ for all } j \in \{k-M, \cdots, k\}. \tag{4.23}$$

As the inequality (4.22) also holds for $j = k$, we can obtain, with the aid of Lemma 4.1 ($A = [J(S^*)]_{nM}^L$, $E = \left[J(S^k)\right]_{nM}^L - [J(S^*)]_{nM}^L$, $\|A^{-1}E\| \le \|A^{-1}\|\|E\| \le \frac{1}{2}\beta\epsilon_J < 1$),

$$\left\| \left(\left[J(S^k)\right]_{nM}^L \right)^{-1} \right\| \le \frac{2\beta}{2-\beta\epsilon_J} =: \eta.$$

From the backward correction method, $\Theta_{i(bc)}^k$ is obtained from the last M stages (stages $i : N$) by using

$$\begin{aligned}
\Theta_{i(bc)}^k &= \left[\left(\left[J(S^k)\right]_{nM}^L \right)^{-1} \right]_{n_x}^U \\
&= \begin{bmatrix} M_U & 0 \end{bmatrix} \left(\left[J(S^k)\right]_{nM}^L \right)^{-1} \begin{bmatrix} M_L \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} M_U & 0 \end{bmatrix} \begin{bmatrix} J_i^k & M_U & & \\ M_L & J_{i+1}^k & \ddots & \\ & \ddots & \ddots & M_U \\ & & M_L & J_N^k \end{bmatrix}^{-1} \begin{bmatrix} M_L \\ 0 \end{bmatrix}.
\end{aligned} \tag{4.24}$$

Alternatively, as $\Theta_{i(bc)}^k = M_U H_{i(bc)}^k M_L$, together with (4.17), we can recursively calculate $\Theta_{i(bc)}^k$ by using $\Theta_{i(bc)}^k = M_U (J_i^k - \Theta_{i+1(bc)}^k)^{-1} M_L$. Similar to $\Theta_{i(bc)}^k$, Θ_i^{k-1} has a recursive formulation: $\Theta_i^{k-1} = M_U (J_i^{k-1} - \Theta_{i+1}^{k-2})^{-1} M_L$. Therefore, Θ_i^{k-1} can be written as

$$\Theta_i^{k-1} = \begin{bmatrix} M_U & 0 \end{bmatrix} \begin{bmatrix} J_i^{k-1} & M_U & & \\ M_L & J_{i+1}^{k-2} & \ddots & \\ & \ddots & \ddots & M_U \\ & & M_L & J_N^{k-M} \end{bmatrix}^{-1} \begin{bmatrix} M_L \\ 0 \end{bmatrix}. \tag{4.25}$$

A comparison of (4.24) with (4.25) shows that Θ_i^{k-1} can be expressed as

$$\Theta_i^{k-1} = \begin{bmatrix} M_U & 0 \end{bmatrix} \left(\left[J(S^k)\right]_{nM}^L + E_{\epsilon_J} \right)^{-1} \begin{bmatrix} M_L \\ 0 \end{bmatrix},$$

where $E_{\epsilon_J} = \mathrm{diag}(J_i^{k-1} - J_i^k, J_{i+1}^{k-2} - J_{i+1}^k, \cdots, J_N^{k-M} - J_N^k)$. As E_{ϵ_J} is a block-diagonal matrix, together with (4.23), the following inequalities hold:

$$\begin{aligned}\|E_{\epsilon_J}\| &\le \sum_{j=k-M}^{k-1} \left\| J_{k+i-1-j}^{j} - J_{k+i-1-j}^{k} \right\| \\ &\le \sum_{j=k-M}^{k-1} \left\| \left[J(S^j)\right]_{nM}^{L} - \left[J(S^k)\right]_{nM}^{L} \right\| \\ &< M\epsilon_J \le N\epsilon_J.\end{aligned}$$

On the other hand,

$$\begin{aligned}\|\Theta_i^{k-1} - \Theta_{i(bc)}^k\| &= \left\| \begin{bmatrix} M_U & 0 \end{bmatrix} \left(\left(\left[J(S^k)\right]_{nM}^{L} + E_{\epsilon_J} \right)^{-1} - \left(\left[J(S^k)\right]_{nM}^{L} \right)^{-1} \right) \begin{bmatrix} M_L \\ 0 \end{bmatrix} \right\| \\ &\le \left\| \left(\left[J(S^k)\right]_{nM}^{L} + E_{\epsilon_J} \right)^{-1} - \left(\left[J(S^k)\right]_{nM}^{L} \right)^{-1} \right\|.\end{aligned}$$

By Lemma 4.2 ($A = \left[J(S^k)\right]_{nM}^{L}$, $E = E_{\epsilon_J}$, $\|A^{-1}E\| \le \|A^{-1}\|\|E\| \le \eta N \epsilon_J < 1$),

$$\|h_i^k\| = \|\Theta_i^{k-1} - \Theta_{i(bc)}^k\| \le \eta \frac{N\eta\epsilon_J}{1 - N\eta\epsilon_J} =: \epsilon_h.$$

Since ϵ_h monotonically increases with respect to $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$, and $\epsilon_h \to 0$ as $\epsilon_J \to 0$ and $\epsilon_h \to \infty$ as $\epsilon_J \to \frac{2}{2N+1}\beta^{-1}$, then $\epsilon_h \in (0, \infty)$ for $\epsilon_J \in (0, \frac{2}{2N+1}\beta^{-1})$. This implies that, for any $\epsilon_h > 0$, there exists $k_m > 0$ such that, for any $k > k_m$, $\|h_i^k\| \le \epsilon_h$ holds for each $i \in \{1, \cdots, N\}$ when the iterates $\{S^k\}$ converge to S^*. Therefore, $h^k \to 0$ $(k \to \infty)$ when $\{S^k\}$ converge. □

Lemma 4.4. *$G(S,h)$ can be expressed as $G(S,h) = P(S,h) \cdot \mathcal{K}(S)$, where $P(S,h) : D_S \times D_h \to \mathbb{R}^{nN \times nN}$. Moreover, $\lim_{h \to 0} P(S,h)^{-1} = J(S)$ holds for all $S \in D_S$.*

Proof. First, we find the expression of $P(S,h)$. As the proposed method has the same algorithm structure as the backward correction method, the iteration of s_i can be found from (4.11):

$$\begin{aligned} s_i^{k+1} &= s_i^k + \Delta s_i^k \\ &= s_i^k - H_i^k \cdot \mathcal{K}_i\left(x_{i-1}^{k+1}, s_i^k, \lambda_{i+1}^{k+1}(x_i^k)\right) \\ &= s_i^k - H_i^k \cdot \left(\mathcal{K}_i^k + M_L \Delta s_{i-1}^k - \begin{bmatrix} 0^T & (d_{\lambda_{i+1}}^k)^T \end{bmatrix}^T \right), \end{aligned} \tag{4.26}$$

and the recursion for calculating $d_{\lambda_i}^k$ is given in (4.10) by

$$\begin{bmatrix} 0^T & (d_{\lambda_i}^k)^T \end{bmatrix}^T = M_U H_i^k \cdot \left(\mathcal{K}_i^k - \begin{bmatrix} 0^T & (d_{\lambda_{i+1}}^k)^T \end{bmatrix}^T \right). \tag{4.27}$$

From (4.26) and (4.27), recursively, we can have

$$\begin{aligned}\Delta s_1^k &= -H_1^k \cdot \mathcal{K}_1^k + H_1^k M_U H_2^k \cdot \mathcal{K}_2^k - \cdots \\ &= -H_1^k \sum_{j=1}^{N} (-1)^{j-1} \left(\prod_{l=2}^{j} M_U H_l^k \right) \cdot \mathcal{K}_j^k,\end{aligned} \tag{4.28}$$

and

$$s_i^{k+1} = s_i^k - H_i^k \sum_{j=i}^{N} (-1)^{j-i} \left(\prod_{l=i+1}^{j} M_U H_l^k \right) \cdot \mathcal{K}_j^k - H_i^k M_L \Delta s_{i-1}^k. \tag{4.29}$$

Note that, from (4.28) and (4.29), for any $i \in \{1, \cdots, N\}$, Δs_i^k is a linear function of $\mathcal{K}_{1,\cdots,N}^k$. That is, $G(S,h)$ can be expressed as $G(S,h) = P(S,h) \cdot \mathcal{K}(S)$, where $P(S,h)$ is the coefficient matrix. Let us denote $P_{i,j}(S^k, h^k)$ and $H_{i,j}(S^k) \in \mathbb{R}^{n \times n}$ as the (i,j)-th block entries of the partitioned matrices $P(S^k, h^k)$ and $H(S^k)$, $i, j \in \{1, \cdots, N\}$, respectively. Again, from (4.28) and (4.29), we obtain the expression of $P_{i,j}^k(S^k, h^k)$ as the summation of the terms in (4.30), with a maximum number of terms N,

$$\alpha H_{r_0}^k \prod_{l=1}^{m} M_l H_{r_l}^k \tag{4.30}$$

where $\alpha \in \{1, -1\}$, $r_l \in \{1, \cdots, N\}$, $l \in \{0, 1, \cdots, 2N\}$, $m \in \{0, 1, \cdots, 2N\}$, and $M_l \in \{M_U, M_L\}$. Since $H_i^k = (J_i^k - \Theta_{i+1}^{k-1})^{-1}$, $H_{i(bc)}^k = (J_i^k - \Theta_{i+1(bc)}^k)^{-1}$, and the proposed method differs from the backward correction method only in Θ_i, $i \in \{1, \cdots, N\}$, so $H_{i,j}(S^k)$ has the same form as $P_{i,j}(S^k, h^k)$. For example, $P_{1,1}(S^k, h^k) = H_1^k$, $H_{1,1}(S^k) = H_{1(bc)}^k$, $P_{2,2}(S^k, h^k) = H_2^k + H_2^k M_L H_1^k M_U H_2^k$, and $H_{2,2}(S^k, h^k) = H_{2(bc)}^k + H_{2(bc)}^k M_L H_{1(bc)}^k M_U H_{2(bc)}^k$.

We then prove

$$\lim_{h \to 0} P(S,h)^{-1} = J(S), \tag{4.31}$$

for all $S \in D_S$. It can be proved by induction on l, together with the boundedness of $\|J(S)^{-1}\|$ in Assumption 4.2, that, for any $\epsilon > 0$, there exists $\delta_h > 0$ such that, if $h^k \in \{h | \|h_i\| < \delta_h,\ i \in \{1, \cdots, N\}\} \subset D_h$,

$$\left\| \alpha H_{r_0}^k \prod_{l=1}^{m} M_l H_{r_l}^k - \alpha H_{r_0(bc)}^k \prod_{l=1}^{m} M_l H_{r_l(bc)}^k \right\| < \epsilon \tag{4.32}$$

holds for any $\alpha \in \{1, -1\}$, $r_l \in \{1, \cdots, N\}$, $l \in \{0, 1, \cdots, 2N\}$, $m \in \{0, 1, \cdots, 2N\}$, and $M_l \in \{M_U, M_L\}$. Since $P_{i,j}(S^k, h^k)$ is the summation of terms of (4.30) with maximum number N and satisfying (4.32), we obtain

$$\left\| P_{i,j}(S^k, h^k) - H_{i,j}(S^k) \right\| < N\epsilon,\ i, j \in \{1, \cdots, N\}.$$

Hence,

$$\|P(S^k,h^k) - H(S^k)\| \leq \sum_{i=1}^{N}\sum_{j=1}^{N} \|P_{i,j}^k(S^k,h^k) - H_{i,j}^k(S^k)\| < N^3\epsilon.$$

Since $\epsilon > 0$ is arbitrary, $\|P(S^k,h^k) - H(S^k)\|$ can be bounded arbitrarily. Thus, $\|P(S^k,h^k)^{-1} - J(S^k)\|$ can also be bounded arbitrarily due to the continuity of matrix inverse on D_S, that is, (4.31) holds. This completes the proof. Moreover, (4.31) can also be restated to show that $P(S,h)^{-1}$ is a consistent approximation (Ortega & Rheinboldt, 1970) to $J(S)$ on D_S. □

Theorem 4.2. *There are balls $O_S := \{S \| \|S - S^*\| < \delta_S\} \subset D_S$ and $O_h := \{h \| \|h_i\| < \delta_h,\ i \in \{1,\cdots,N\}\} \subset D_h$ such that, for any $S^0 \in O_S$ and $h^0 \in O_h$, the iterates $\{S^k\}$ given by the proposed method remain in O_S and converge to S^* superlinearly.*

Proof. Set $\beta = \|H(S^*)\|$. For any $\epsilon \in (0, \frac{1}{2}\beta^{-1})$, there exists $\delta_1 > 0$ such that, if $S^k \in O_1 := \{S \| \|S - S^*\| < \delta_1\} \subset D_S$, then

$$\|J(S^k) - J(S^*)\| < \frac{1}{2}\epsilon \tag{4.33}$$

and

$$\|\mathcal{K}(S^k) - \mathcal{K}(S^*) - J(S^*)(S^k - S^*)\| < L\|S^k - S^*\|^2 \tag{4.34}$$

hold by the continuity of J at S^* and the Lipschitz continuity in Assumption 4.2, respectively. For such ϵ, from Lemma 4.4, we can choose $\delta_2 > 0$ such that $h^k \in O_2 := \{h \| \|h_i\| < \delta_2,\ i \in \{1,\cdots,N\}\} \subset D_h$ and

$$\|P(S^k,h^k)^{-1} - J(S^k)\| < \frac{1}{2}\epsilon. \tag{4.35}$$

From (4.33) and (4.35), we have

$$\|P(S^k,h^k)^{-1} - J(S^*)\| < \epsilon.$$

Since $\beta = \|H(S^*)\| = \|J(S^*)^{-1}\|$, then $\|P(S^k,h^k)\|$ can be bounded using Lemma 4.1 ($A = J(S^*)$, $E = P(S^k,h^k)^{-1} - J(S^*)$, $\|A^{-1}E\| \leq \|A^{-1}\|\|E\| < \beta\epsilon < 1/2$) by

$$\|P(S^k,h^k)\| \leq \frac{\beta}{1-\beta\epsilon} < 2\beta. \tag{4.36}$$

From (4.34) and (4.36),

$$\begin{aligned}
\|S^{k+1} - S^*\| &= \|S^k - P(S^k,h^k)\mathcal{K}(S^k) - S^*\| \\
&= \|P(S^k,h^k)[P(S^k,h^k)^{-1}(S^k - S^*) - \mathcal{K}(S^k)]\| \\
&\leq 2\beta\left(\|P(S^k,h^k)^{-1} - J(S^k)\| + \|J(S^k) - J(S^*)\|\right)\|S^k - S^*\| \\
&\quad + 2\beta\|\mathcal{K}(S^k) - \mathcal{K}(S^*) - J(S^*)(S^k - S^*)\| \\
&< \omega(S^k,h^k)\|S^k - S^*\|,
\end{aligned}$$

where

$$\omega(S^k, h^k) = 2\beta \left(\|P(S^k, h^k)^{-1} - J(S^k)\| + \|J(S^k) - J(S^*)\| + L\|S^k - S^*\| \right)$$

is the upper bound on the rate of convergence. The result of Lemma 4.4 ensures that $\|P(S^k, h^k)^{-1} - J(S^k)\| \to 0$ as $h^k \to 0$, and, due to the continuity of J at S^*, $\|J(S^k) - J(S^*)\| \to 0$ as $S^k \to S^*$. Therefore, there exist balls $O_S := \{S | \|S - S^*\| < \delta_S\} \subset O_1 \cap O_3$ and $O_h := \{h | \|h_i\| < \delta_h, \ i \in \{1, \cdots, N\}\} \subset O_2$ such that, for any $S^k \in O_S$ and $h^k \in O_h$, $\omega(S^k, h^k) < 1$ holds, where O_3 is a subset of D_S such that, if S^k is in O_3, then $h^k \in O_h$ always holds from Lemma 4.3. Therefore, if $S^0 \in O_S$ and $h^0 \in O_h$, then the iterates $\{S^k\}$ converge and remain in O_S. Moreover, since $S^k \to S^*$ as $k \to \infty$, we have $h^k \to 0$ as $k \to \infty$ from Lemma 4.3. Therefore, $\omega(S^k, h^k) \to 0$ as $k \to \infty$, which indicates that the proposed method has a superlinear rate of convergence. □

Remark 4.2. *Theorem 4.2 gives only conditions of convergence for the update of only one sampling instant. However, the convergence of the proposed method should be guaranteed for every sampling instant. When the iteration at one sampling instant converges, a sufficiently large number of iterations k can guarantee sufficiently small $\|h_i^k\|$ for all $i \in \{1, \cdots, N\}$ from Lemma 4.3. Moreover, if the time-dependent parameter p varies smoothly with respect to time, a small sampling interval can ensure that the new state $\bar{x}_0$ and the time-dependent parameter p are close to their previous values even if there are model mismatches or disturbances in the actual system, which implies that the initial guess S^0 is close to the optimal value S^* when warm-start is used. Therefore, the conditions in Theorem 4.2 can always be satisfied in the next sampling instant.*

4.5 Numerical experiment

4.5.1 NMPC for a quadrotor

In order to demonstrate the computation time and rate of convergence of the proposed method, reference tracking of a quadrotor is considered. The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T \in \mathbb{R}^9$, where (X, Y, Z) and (γ, β, α) are the position and angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. We use a previously defined nonlinear model in (3.33). The control input is bounded with $[0, -1, -1, -1]^T \le u \le [11, 1, 1, 1]^T$. The system starts from the initial state $x_0 =$

$[1, 0, 1, 0, 1, 0, 0, 0, 0]^T$. The state reference is set using the time-varying parameter p with $x_{ref} = 0$ from 0 to 10 s and $x_{ref} = [1.5, 0, 1.5, 0, 1.5, 0, 0, 0, 0]^T$ from 10 to 20 s, and $u_{ref} = [g, 0, 0, 0]^T$. The stage cost function is $L(u, x, p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2)$ with $Q = \mathrm{diag}(10, 1, 2, 1, 10, 1, 1, 1, 1)$ and $R = I_4$, and the terminal cost function $\varphi(x, p)$ is not imposed. The prediction horizon is $T = 1$ s and is divided into $N = 8, 16, 24, 48, 96$ steps for comparison. Simulation is performed for 20 s with a sampling interval of 0.01 s.

4.5.2 Computation time

The proposed method was implemented using the open-source toolkit ParNMPC (version Beta1.0) (https://github.com/deng-haoyang/ParNMPC), which can automatically generate parallel C/C++ code with OpenMP (Dagum & Menon, 1998). For comparison, the code generation tool (Houska et al., 2011b) in the ACADO Toolkit (Houska, Ferreau, & Diehl, 2011a) and the AutoGenU (Ohtsuka, 2015) code generation toolkit are used to perform closed-loop NMPC simulation. ACADO Code Generation is based on the SQP method with Gauss-Newton Hessian approximation (GNHA); the qpOASES (Ferreau et al., 2014) dense QP solver and the qpDUNES (Frasch et al., 2015) sparse QP solver are used respectively to solve the underlying QP problems. AutoGenU is based on the C/GMRES method. The simulation was performed on a desktop computer with dual 2.2 GHz Intel Xeon E5-2650 V4 processors with 24 cores in total with the Hyper-Threading and Turbo Boost features are disabled.

In principle, ACADO Code Generation is based on the real-time iteration (RTI) scheme, in which only one SQP iteration is performed. The KKT tolerance is controlled by performing several SQP iterations, and the computation time is the sum of the CPU times returned by the solver. The KKT tolerances of the proposed method and the SQP methods are set to $5 \cdot 10^{-3}$, while in the C/GMRES method, only one iteration is performed per update. It should be noted that these methods do not converge to the same solution even when the same KKT tolerances are achieved because of the differences in handling inequality constraints. Namely, the SQP methods converge to the optimal solution of the inequality constrained optimization problem while the proposed method and C/GMRES only approximately take the inequality constraints into account by introducing dummy inputs (Ohtsuka, 2004). The resulting suboptimality is defined as

$$e_o := \frac{\|U^*_{dummy} - U^*_{ieq}\|}{\|U^*_{ieq}\|},$$

where U^*_{dummy} and U^*_{ieq} denote the solutions of the control inputs sequences obtained by introducing dummy inputs and by dealing directly with inequality constraints, respectively.

The discretization within the SQP methods is based on the explicit Euler method with multiple shooting. The QP problems solved by qpOASES is condensed, and warm-start is enabled. Although SQP methods can, in principle, be parallelized to a certain degree, e.g., for multiple shooting and the qpDUNES QP solver, they are run in a fully sequential fashion. The explicit Euler method is used for the C/GMRES method, and the number of iterations in GMRES is set to 7. The proposed method is based on the implicit Euler method.

The approximation to the backward correction method is defined in (4.21). Let us denote the relative approximation error by e_h, which measures the distance to Newton's method:

$$e_h := \max_{i\in\{1,\cdots,N\}} \frac{\|h_i\|}{\|\Theta_{i(bc)}\|}.$$

The simulation results for the inputs, position, suboptimality e_o, and relative approximation error e_h after each update with the proposed method for $N = 24$ are shown in Figs. 4.1, 4.2, 4.3, and 4.4, respectively. The results obtained by introducing dummy inputs are almost the same as those with the SQP methods, i.e., active input bound constraints can be achieved when tracking the position reference. Since the degree

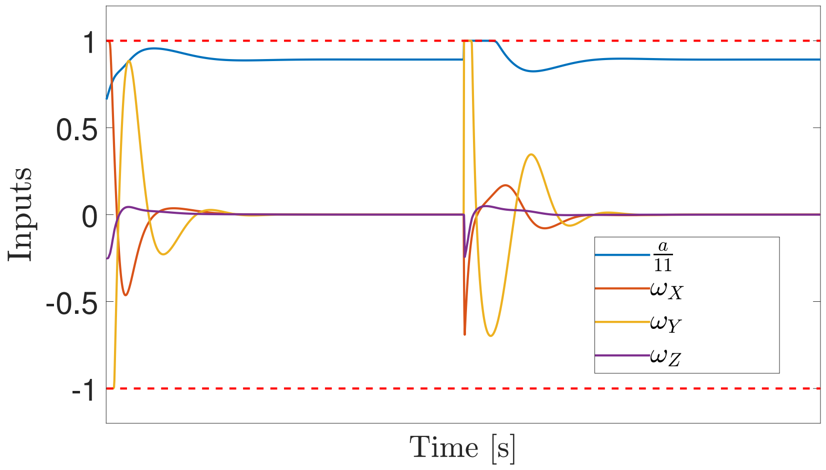


Figure 4.1: Time histories of quadrotor's inputs.

of parallelism of the proposed method is N, the program can be run on $\min\{24, N\}$ cores concurrently. Table 4.1 lists the average computation time and number of iterations per update. Although the computation time of the proposed method is long

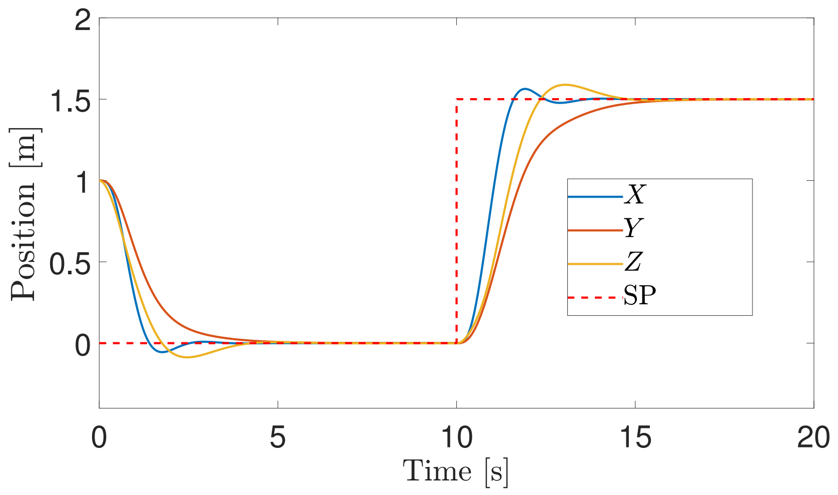


Figure 4.2: Time histories of quadrotor's position (SP is position reference).

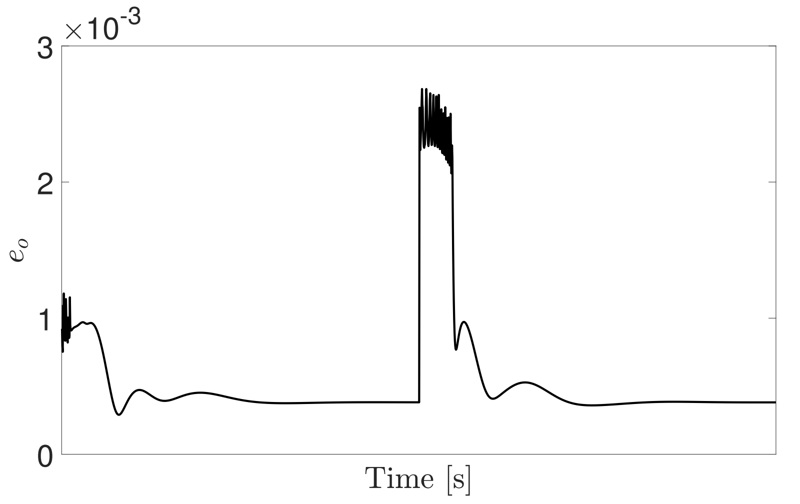


Figure 4.3: Time histories of suboptimality e_o.

when running on one core in a fully sequential fashion, it becomes faster than other methods in the medium and high range of N when running on multiple cores.

As the computation time of the proposed method is in the μs range, the overhead time is non-negligible. Let us denote t_E as the average time per update running on $\min\{24, N\}$ cores in practice and t_p and t_s as the times of the parallelizable and sequential parts running on one core, respectively. The theoretical minimum execution time is given by $t_A := t_s + t_p / \min\{24, N\}$ according to Amdahl's law. From Table 4.2, we can see that t_s is less than 1% of the total elapsed time, indicating that the proposed method is highly parallelizable. In this example, the overhead time accounts for a substantial part of the total elapsed time, even greater than 60%.

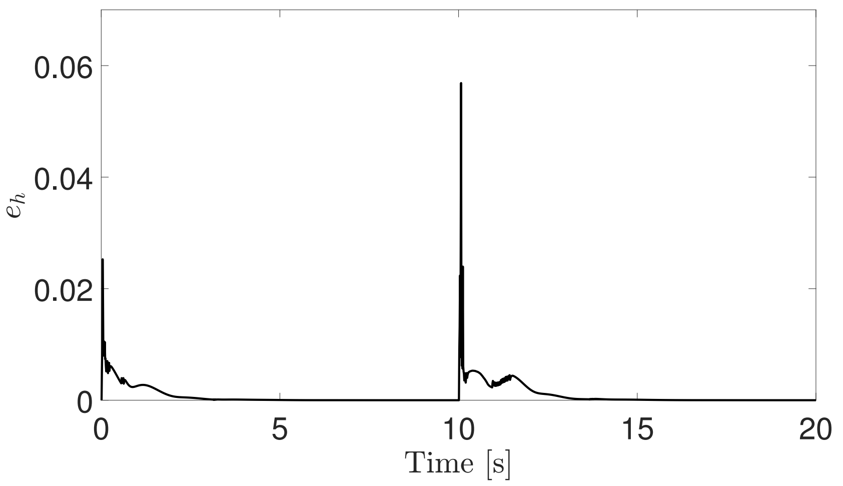


Figure 4.4: Time histories of relative approximation error e_h after each update.

The overhead time can also be observed from the parallel scaling results in Table 4.3. Although a speed-up of more than 10× can be achieved, the efficiency (speed-up per core) degenerates with the growth in the number of cores. One reason for these overheads is the communication between the dual processors when carrying out the backward and forward correction steps. This overhead can be reduced by running on a single-processor-based computer. Also, interruptions by other threads on Windows unbalance the parallel tasks, which also explains why the overhead time is proportionately the highest for $N = 24$ in Table 4.2 and explains the lowest efficiency when using all 24 cores in Table 4.3. These overhead time results show the potential of the proposed method if it is run on platforms with less overhead to maximize its efficiency.

Table 4.1: Average computation time [μs] (upper) and number of iterations (lower) per update.

N	8	16	24	48	96
Proposed method on min$\{24, N\}$ cores	112	139	173	285	454
	1.01	1.01	1.02	1.03	1.04
Proposed method on one core	455	795	1171	2453	4666
SQP with qpOASES	95	349	866	5920	65745
	1.09	1.13	1.14	1.17	1.24
SQP with qpDUNES	189	361	563	1142	2487
	1.39	1.42	1.44	1.48	1.54
C/GMRES	201	335	470	868	1648

Table 4.2: Practical and theoretical average computation times per update for proposed method.

N	8	16	24	48	96
t_p [μs]	452	790	1163	2433	4623
t_s [μs]	2.8	5.4	8.1	20.2	43.3
$t_s/(t_s+t_p)\cdot 100$ [%]	0.61	0.67	0.69	0.82	0.93
t_E [μs]	112	139	173	285	454
t_A [μs]	59	55	57	122	236
$(t_E-t_A)/t_E\cdot 100$ [%]	47	61	67	57	48

Table 4.3: Speed-ups with different numbers of cores for $N = 96$.

Number of cores	1	2	4	6
speed-up	1.00	1.95	3.50	4.84
Number of cores	8	12	16	24
speed-up	6.26	8.35	9.72	10.29

4.5.3 Number of iterations for different tolerances

Although the proposed method is proven to converge superlinearly, the rate of convergence is only characterized for $k \to \infty$. The proposed method was compared with Newton's method (the backward correction method) and the SQP method (qpOASES was used) with GNHA. The proposed method and Newton's method converge to the same solution, and the suboptimalities e_o for both methods under the set tolerances are nearly the same as the suboptimality shown in Fig. 4.3. The numbers of iterations shown in Fig. 4.5 are filtered using a 10-th order moving average filter to remove the jitter that arose on the critical point of satisfying the KKT tolerance, so the numbers of iterations become distinguishable for different methods. Fig. 4.5 shows that the proposed method converge to the specified tolerance within several iterations, even to a high accuracy. Moreover, it converges faster than the SQP method with GNHA. The relative approximation error e_h in Fig. 4.4 shows that the proposed method is extremely close to Newton's method. Therefore, the proposed method is slightly slower than Newton's method.

Examples handling inequalities using the interior-point method and with complicated constraints, such as state and terminal constraints, can be found on the website of the toolkit (https://github.com/deng-haoyang/ParNMPC).

4.6 Summary

This chapter presents a highly parallelizable Newton-type method for NMPC. First, the original NMPC problem is discretized using the reverse-time integration method

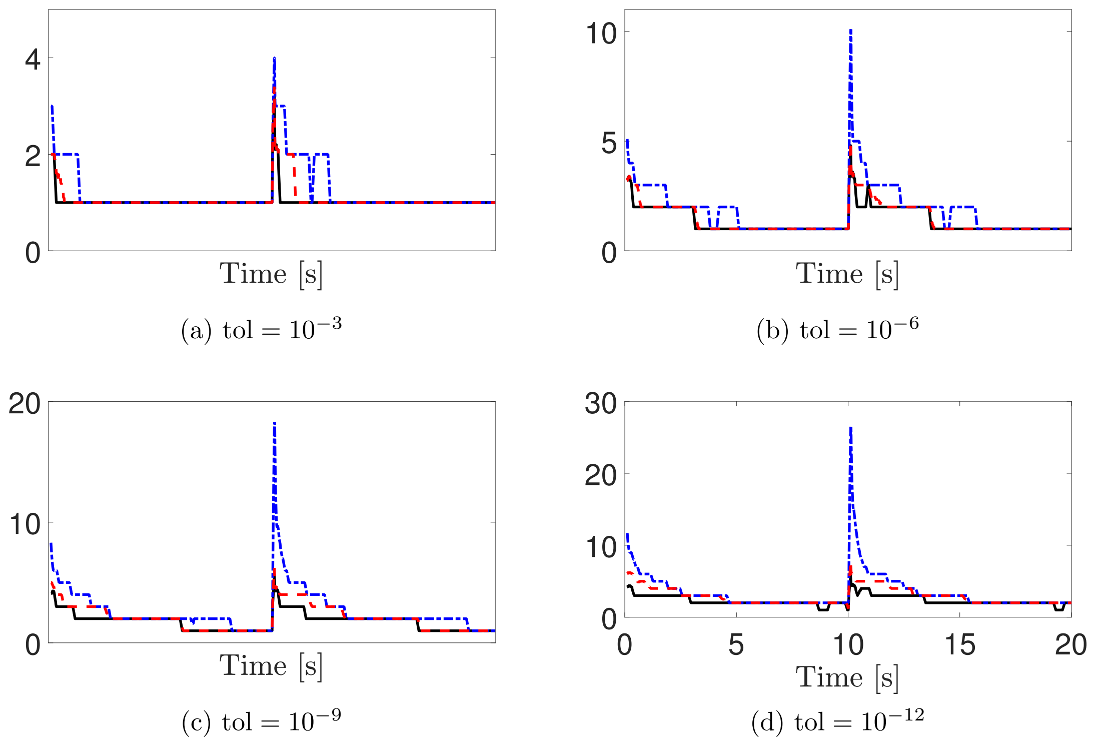


Figure 4.5: Filtered number of iterations for different tolerances (black solid line: Newton's method; blue dash-dot line: SQP with GNHA; red dashed line: proposed method).

so that the Euler-Lagrange equations of the resulting subproblems are linearly coupled between neighboring stages. Next, the backward correction method, by predicting the coupling variables recursively from the last stage to the first, is formulated and proved to be identical to Newton's method. Since this method is computationally expensive, it is approximated using the previous iteration's information so that the recursion is broken down. The proposed method (parallelized implementation) has a complexity of $\mathcal{O}(n^3 + Nn_x^2)$ running on N threads. The evaluations of functions $\{\mathcal{K}_i^k\}_{i=1}^N$ and their Jacobian matrices $\{J_i^k\}_{i=1}^N$ are done in parallel. Numerical simulation of controlling a quadrotor shows that the proposed method is highly parallelizable (sequential code running time $\leq 1\%$) and faster than conventional methods. Since the experimental environment has inherently high overhead and a limitation on the number of cores, implementation of the proposed method on other platforms, such as ones with FPGAs and many-core processors, would improve its efficiency. This remains for future work.

Chapter 5

Highly Parallelizable Newton-Type Method for NMPC – Implementation

5.1 Introduction

In the last chapter, we present a parallel method for solving the discretized nonlinear model predictive control (NMPC) problem (4.1) without specifying the method that deals with the inequality constraint $G(u, x, p) \geq 0$. Obviously, different inequality handling methods lead to different performance and algorithm structures. In this chapter, we introduce an efficient implementation of the parallel method that explicitly uses the primal-dual interior-point method to handle the inequality constraint.

Although considerable effort has been devoted in recent years to the real-time optimization software of NMPC, unfortunately, there is still a lack of efficient implementations for the parallel optimization of NMPC to the authors' knowledge. Current existing parallel algorithms either require a large problem size, e.g., long prediction horizon, to surpass serial algorithms or depend on careful designs for parallelization, which are not user-friendly for implementation and have difficulty incorporating existing nonlinear optimization techniques to achieve robust numerical performance. In this chapter, ParNMPC, which is an effective and efficient parallel implementation for the optimization of NMPC, is presented. ParNMPC is a MATLAB open-source (https://github.com/deng-haoyang/ParNMPC) software, and the parallelization part is implemented by using OpenMP (Dagum & Menon, 1998), which is designed for shared-memory multi-core processors and supported by most of the operating systems. ParNMPC comes with the following features.

- The NMPC problem can be formulated symbolically as easily as other toolkits.

- The degree of parallelism (DOP) is made configurable so that it can be adapted to the rate of convergence and the number of cores, enabling deployment to both single- and multi-core processors.

- Parallel code can be automatically generated.

Compared with Chapter 4, the main contributions of this chapter are:

- The parallel code generation toolkit ParNMPC is introduced. The primal-dual interior-point method is inherently integrated in ParNMPC to deal with the inequality constraints, and its warm start strategy in the context of NMPC is introduced.

- The search direction calculation procedure is generalized from algebraic operations in Chapter 4 to nonlinear programs (NLPs) so that the existing nonlinear optimization techniques, such as Hessian approximation, regularization, and condensing, can be applied. As a consequence, the method can be applied to a wider class of problems, and a speed up of more than $2\times$ was observed in Section 5.6.

The performance of ParNMPC is assessed with several challenging applications.

This chapter is organized as follows. The NMPC problem is introduced in Section 5.2. Search direction calculation done using Newton's method and the parallel method are given in a unified framework in Section 5.3. Section 5.4 introduces two commonly used line search methods for guaranteeing convergence. Warm start and the barrier strategy are discussed in Section 5.5. The performance evaluation of ParNMPC is introduced in Section 5.6. Finally, this chapter is summarized in Section 5.7.

5.2 Problem statement

In this chapter, we consider the NMPC problem in the form of (2.19), i.e., the following discretized NMPC problem with polytopic inequality constraints:

$$
\begin{aligned}
\min_{X,U} \quad & \sum_{i=1}^{N} L_i(u_i, x_i) \\
\text{s.t.} \quad & x_0 = \bar{x}_0, \\
& x_{i-1} + \mathcal{F}_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\
& C_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\
& G_i(u_i, x_i) \geq 0, \quad i \in \{1, \cdots, N\},
\end{aligned}
\tag{5.1}
$$

where $\bar{x}_0$ is the initial state, $X := (x_0, x_1, x_2, \cdots, x_N)$ and $U := (u_1, u_2, \cdots, u_N)$ are the state and input sequences along the horizon, respectively. We note again that $G_i(u, x) \geq 0$ is a polytopic constraint of u and x.

We adopt the interior-point method to relax the NMPC problem (5.1) by transferring the inequality constraint $G_i(u, x) \geq 0$ into a logarithmic barrier function added to the cost. Notice that the inequality constraint $G_i(u, x) \geq 0$ is a single-side constraint, which may lead to an unbounded solution since the corresponding barrier function tends to be $-\infty$ as its argument goes to ∞. To prevent this, a linear damping term (Wächter & Biegler, 2006b) is added to the barrier function, and we obtain the following relaxed NMPC problem:

$$\begin{aligned} \min_{X,U} \; & \sum_{i=1}^{N} L_i(u_i, x_i) + \rho \Phi_i(u_i, x_i) \\ \text{s.t.} \; & x_0 = \bar{x}_0, \\ & x_{i-1} + \mathcal{F}_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \\ & C_i(u_i, x_i) = 0, \quad i \in \{1, \cdots, N\}, \end{aligned} \tag{5.2}$$

where $\rho > 0$ is the barrier parameter and

$$\Phi_i(u, x) := \sum_{j=1}^{n_z} \big(-\ln(G_{i(j)}(u, x)) + \sigma G_{i(j)}(u, x) \big)$$

with a fixed small damping constant $\sigma > 0$ (e.g., 10^{-4}).

5.2.1 KKT conditions

Let λ_i (costate) $\in \mathbb{R}^{n_x}$, $\mu_i \in \mathbb{R}^{n_\mu}$, and $z_i \in \mathbb{R}^{n_z}$ be the Lagrange multipliers corresponding to the i-th state equation, the equality constraint $C_i(u_i, x_i) = 0$, and the inequality constraint $G_i(u_i, x_i) \geq 0$, respectively. For the sake of brevity, we define

$$s := (\lambda, \mu, u, x).$$

Let $\mathcal{H}_i(s)$ be the Hamiltonian defined by

$$\mathcal{H}_i(s) := L_i(u, x) + \lambda^T \mathcal{F}_i(u, x) + \mu^T C_i(u, x).$$

Let $\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1})$ be defined by

$$\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1}) := \begin{bmatrix} x_{i-1} + \mathcal{F}_i(u_i, x_i) \\ C_i(u_i, x_i) \\ \nabla_u \mathcal{H}_i(s_i)^T + \rho \nabla_u \Phi_i(u_i, x_i)^T \\ \lambda_{i+1} + \nabla_x \mathcal{H}_i(s_i)^T + \rho \nabla_x \Phi_i(u_i, x_i)^T \end{bmatrix}. \tag{5.3}$$

The Karush-Kuhn-Tucker (KKT) conditions for relaxed NMPC problem (5.2) are

$$\mathcal{K}_i(x_{i-1}^*, s_i^*, \lambda_{i+1}^*) = 0,\ \forall i \in \{1, \cdots, N\},$$

with $x_0^* = \bar{x}_0$ and $\lambda_{N+1}^* = 0$.

5.2.2 Notation

We define the following extra notations used in this chapter:

- $U := (u_1, \cdots, u_N)$, $X := (x_1, \cdots, x_N)$, $S := (s_1, \cdots, s_N)$, $Z := (z_1, \cdots, z_N)$.
- $\mathcal{K}_i^k := \mathcal{K}_i(x_{i-1}^k, s_i^k, \lambda_{i+1}^k)$, $\nabla_u \mathcal{F}_i^k := \nabla_u \mathcal{F}_i(u_i^k, x_i^k)$, etc.

5.3 Search direction calculation

Search direction calculation is the most computationally expensive part of real-time optimization. In some particular algorithms dedicated for NMPC, such as the C/GMRES (Ohtsuka, 2004) method and the RTI scheme (Diehl et al., 2005), a full-step iteration is performed each sampling time, requiring only the computation of the search direction. In this section, search direction calculation performed using Newton's method and the parallel method is given in a unified framework.

5.3.1 Newton's method

For Newton's method, the search direction $(\Delta s_1, \cdots, \Delta s_N)$ is obtained by solving the following regularized primal-dual (Nocedal & Wright, 2006) system:

$$\left[\begin{array}{c|cc|c} \ddots & I & & \\ \hline I & \begin{matrix} 0 & 0 \\ 0 & -\delta_C I \end{matrix} & J_i^k & \\ & (J_i^k)^T & M_i^k & I \\ \hline & & I & \ddots \end{array}\right] \begin{bmatrix} \vdots \\ \Delta s_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathcal{K}_i^k \\ \vdots \end{bmatrix}, \tag{5.4}$$

where

$$J_i^k := \begin{bmatrix} \nabla_u \mathcal{F}_i^k & \nabla_x \mathcal{F}_i^k \\ \nabla_u C_i^k & \nabla_x C_i^k \end{bmatrix}$$

and

$$
\begin{aligned}
M_i^k := & \underbrace{\begin{bmatrix} \nabla_{uu}^2 \mathcal{H}_i^k & \nabla_{ux}^2 \mathcal{H}_i^k \\ \nabla_{xu}^2 \mathcal{H}_i^k & \nabla_{xx}^2 \mathcal{H}_i^k \end{bmatrix}}_{①} + \delta_H I \\
& + \underbrace{\begin{bmatrix} \nabla_u G_i^k & \nabla_x G_i^k \end{bmatrix}^T \operatorname{diag}(z_i^k \oslash G_i^k) \begin{bmatrix} \nabla_u G_i^k & \nabla_x G_i^k \end{bmatrix}}_{②},
\end{aligned} \tag{5.5}
$$

with $\delta_C, \delta_H \geq 0$. Apart from the regularization terms $-\delta_C I$ and $\delta_H I$, solving system (5.4) is equivalent to applying Newton's method to KKT conditions (5.3) for relaxed problem (5.2), however, with the Hessian of the barrier function $\rho\Phi_i$ estimated by ② in (5.5), which is in contrast with the primal system where the exact Hessian is used. The primal-dual system is preferred against the primal system since the primal system tends to perform poorly when a small barrier parameter ρ is chosen (Nocedal & Wright, 2006). After obtaining the search direction $(\Delta s_1, \cdots, \Delta s_N)$, the search direction of z is obtained from

$$
\Delta z_i = (z_i^k \odot G_i^{k+1} - \rho e) \oslash G_i^k, \tag{5.6}
$$

where $G_i^{k+1} := G_i(u_i^k - \Delta u_i, x_i^k - \Delta x_i)$ and $e := [1, \cdots, 1]^T$.

It is generally easy to guarantee the full rank property of the Jacobian of $\mathcal{F}_i$, e.g., when the discretization step size $\Delta\tau$ is small. We add a regularization term $\delta_C I \geq 0$ to avoid singularity (Nocedal & Wright, 2006) of the primal-dual matrix in (5.4) caused by the rank deficiency of $[\nabla_u C_i^k \ \nabla_x C_i^k]$. In ParNMPC, δ_C is chosen to be sufficiently small (10^{-9}).

To guarantee that the obtained search direction is a descent direction for certain merit functions, a regularization term $\delta_H I \geq 0$ is added to the Hessian so that the primal-dual matrix has a desired inertia. The descent property is required to guarantee convergence when the current iterate is distant from the optimal solution. However, choosing a proper δ_H on-the-fly requires successive factorizations of the primal-dual matrix, which is computationally heavy for real-time optimization. In the context of NMPC, δ_H can be decided offline with prior knowledge of the NMPC problem. We show in Section 5.3.3 that the descent property is ensured inherently if the Hessian matrix is approximated properly, thus requiring no regularization procedure, i.e., $\delta_H = 0$.

We show next that solving (5.4) can be interpreted as solving a series of subproblems. We first define the following single-stage subproblem:

$$\begin{aligned} \min_{x_i,u_i} \ & L_i(u_i,x_i) + \rho\Phi_i(u_i,x_i) + \underbrace{(x_i - x_i^k)^T\lambda_{i+1} + \frac{1}{2}\|x_i - x_i^k\|^2_{W_{i+1}}}_{①} \\ \text{s.t.} \ & x_{i-1} + \mathcal{F}_i(u_i,x_i) = 0, \\ & C_i(u_i,x_i) = 0, \end{aligned} \tag{5.7}$$

where its regularized primal-dual matrix at the k-th iteration is given by

$$\left[\begin{array}{c|c} \begin{matrix} 0 & 0 \\ 0 & -\delta_C I \end{matrix} & J_i^k \\ \hline (J_i^k)^T & M_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1} \end{bmatrix} \end{array}\right]. \tag{5.8}$$

Given a barrier parameter ρ, we denote

$$(s_i, W_i) \leftarrow \mathcal{P}_i^k(x_{i-1}, \lambda_{i+1}, W_{i+1}) \tag{5.9}$$

as the operation of performing a full-step primal-dual iteration of (5.7) at the k-th iteration for given parameters of x_{i-1}, λ_{i+1}, and W_{i+1}. Here, as in (4.8), $W_i \in \mathbb{R}^{n_x \times n_x}$ denotes the sensitivity matrix of λ_i with respect to the initial state x_{i-1} of subproblem (5.7). Specifically, as in (4.9), W_i is the negation of the n_x-th leading principal submatrix of the inversion of (5.8), which is a by-product of iteration (5.9).

Notice that the primal-dual matrix in (5.4) is a block-tridiagonal matrix. It has been shown in the last chapter that the backward correction method in Section 4.3 is equivalent to solving (5.4) by using the block Gaussian elimination method, which is also equivalent to performing the Riccati recursion for the linear quadratic problem having regularized KKT conditions (5.4). Moreover, it can be shown by checking the KKT conditions for (5.7) that the backward correction method, or equivalently, the block Gaussian elimination method is equivalent to solving single-stage subproblems (5.7) recursively as shown in Algorithm 5.1, which can be seen as a compact form of Algorithm 4.1.

5.3.2 Parallel method

The search direction calculation done using Newton's method in Algorithm 5.1 involves a recursion of computing the sensitivity matrix W_i from $i = N$ to 1. The parallel method in Section 4.4 iterates on the basis of the information stored at the previous iteration so that the recursion is broken, and the sensitivity matrices are then

Algorithm 5.1 Search direction calculation using Newton's method

```
Input: $\bar{x}_0$, $S^k$, $Z^k$, $\rho$
Output: $\Delta S$, $\Delta Z$
 1: Initialization: $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^{k+1} = 0$, $W_{N+1}^k = 0$
 2: for $i = N$ to 1 do
 3:     $(s_i^{k+1}, W_i^k) \leftarrow \mathcal{P}_i^k(x_{i-1}^k, \lambda_{i+1}^{k+1}, W_{i+1}^k)$
 4: end for
 5: for $i = 1$ to $N$ do
 6:     $(s_i^{k+1}, -) \leftarrow \mathcal{P}_i^k(x_{i-1}^{k+1}, \lambda_{i+1}^{k+1}, W_{i+1}^k)$
 7:     $\Delta s_i = s_i^k - s_i^{k+1}$
 8:     $\Delta z_i \leftarrow (5.6)$
 9: end for
```

updated, which can be referred to as "first iterate, then update," while in Newton's method, it is "first update, then iterate." In the parallel method, the regularized primal-dual matrices are independent of each other and can be factorized in parallel. The parallel method is summarized in Algorithm 5.2, in which the parallelizable part is explicitly given. As shown in Algorithm 5.2, not only are the function evaluations performed in parallel, matrix factorizations, which are the most computationally expensive part of (5.9), are also done in parallel. The parallelization is not fully shown in this chapter, and details can be found in Section 4.4, in which the non-parallelizable part consists only of $2N$ matrix-vector multiplications with a complexity of $\mathcal{O}(Nn_x^2)$.

Algorithm 5.2 Search direction calculation using parallel method

```
Input: $\bar{x}_0$, $S^k$, $Z^k$, $W_{1,\cdots,N}^{k-1}$, $\rho$
Output: $\Delta S$, $\Delta Z$, $W_{1,\cdots,N}^k$
 1: Initialize: $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^{k+1} = 0$, $W_{N+1}^{k-1} = 0$
 2: for $i = 1$ to $N$  do in parallel
 3:     Evaluate $\mathcal{K}_i^k$ (KKT function)
 4:     Evaluate $J_i^k$ (Jacobian)
 5:     Evaluate $M_i^k$ (Hessian)
 6:     Factorize (5.8) (KKT matrix)
 7: end for
 8: for $i = N$ to 1 do
 9:     $(s_i^{k+1}, W_i^k) \leftarrow \mathcal{P}_i^k(x_{i-1}^k, \lambda_{i+1}^{k+1}, W_{i+1}^{k-1})$
10: end for
11: for $i = 1$ to $N$ do
12:     $(s_i^{k+1}, -) \leftarrow \mathcal{P}_i^k(x_{i-1}^{k+1}, \lambda_{i+1}^{k+1}, W_{i+1}^{k-1})$
13:     $\Delta s_i = s_i^k - s_i^{k+1}$
14:     $\Delta z_i \leftarrow (5.6)$
15: end for
```

Remark 5.1. *Newton's method and the parallel method have DOPs of one and N, respectively. It should be noted that, for each iteration, the parallel method has the same total computational cost as Newton's method, and parallelization is achieved with no extra computational cost. Compared with the quadratic rate of convergence for Newton's method, a superlinear rate of convergence is shown in Section 4.4.2 for the parallel method. It is therefore expected to have a faster rate of convergence when fewer sensitivity matrices W_i are approximated by shifting between or combining these two methods. In practice, depending on the number of cores, the DOP is made configurable to speed convergence up without sacrificing the computational performance. When the DOP is set to one, ParNMPC decays to Newton's method in Algorithm 5.1 and behaves exactly the same as the structure-exploiting primal-dual interior-point method. When the DOP is set to N, a fully parallelized method that can use N cores is obtained as shown in Algorithm 5.2.*

The convergence of the parallel method is expected to behave like Newton's method when the system is fast-sampled, less parallelized (i.e., DOP is small), and the dynamics and parameters are slowly varying. That is, the sensitivity matrices W_i can be updated in real time. In practice, the parallel method was observed to also converge fast even when it was fully parallelized (DOP $= N$) and for highly nonlinear systems, such as the robot manipulator in Section 5.6 and the double inverted pendulum on a cart, which can be found on ParNMPC's homepage (https://github.com/denghaoyang/ParNMPC).

Remark 5.2. *Compared with Algorithm 4.2 in Section 4.4, where the parallelization is given by algebraic operations , the parallel method is generalized in this chapter from algebraic operations to NLPs* (5.7) *so that existing nonlinear optimization techniques, such as regularization, condensing, Hessian approximation, and line search, can be applied. These techniques broaden the range of application of ParNMPC and make the algorithm numerically more efficient and robust. For example, the Hessian matrices in robot applications have to be approximated to avoid time-consuming evaluations of their exact values, and a large speed up was observed in Section 5.6 due to condensing.*

In Newton's method, the nonsingular and descent regularization procedures are performed on the overall primal-dual matrix in (5.4), while, in the parallel method, it is generally difficult to have an explicit form of the overall primal-dual matrix. An interpretation of ① in (5.7) is that ① approximates the optimal cost-to-go in a quadratic form (the constant term is ignored). Consequently, each subproblem (5.7) approximates an NMPC problem consisting of stages i to N with an initial state of x_{i-1}. When $i = 1$, (5.7) approximates the original relaxed problem (5.2) locally. We require that, after regularizing each subproblem, its primal-dual matrix (5.8) is nonsingular and has a desired inertia, that is, the lower-right submatrix is positive-definite on the null space of J_i^k.

Instead of inverting (5.8) directly in Section 4.4 by using LU factorization, a more efficient way of solving each subproblem is to condense the primal-dual system by eliminating Δx_i and $\Delta\lambda_i$. Compared with $\mathcal{O}((2n_x + n_\mu + n_u)^3)$ for inverting (5.8) directly, condensing can lead to significant computational improvement and is used by default in ParNMPC. It should be noted that the inherent implicit property of the state and costate equations due to the use of the reverse-time discretization method incurs an extra cost on equation solving, i.e., requiring $\nabla_x\mathcal{F}_i^k$ to be inverted, which is usually not needed for the sequential methods (Diehl et al., 2009) where a forward and backward simulation is performed so that the state and costate equations are eliminated. We show that this extra cost for inverting $\nabla_x\mathcal{F}_i^k$ can be avoided by using approximation. Consider, for example, the reverse-time Euler method with $\mathcal{F}_i(u,x) = f(u,x)\Delta\tau - x$. We can have the following approximation based on truncated Neumann series when the condition $\|\nabla_x f_i^k \Delta\tau\| \le 1$ is satisfied:

$$(\nabla_x\mathcal{F}_i^k)^{-1} \approx -I - \nabla_x f_i^k \Delta\tau,$$

where the truncation error is $\mathcal{O}(\Delta\tau^2)$. It is not difficult to show that, for the family of reverse-time Runge-Kutta methods, the approximation is obtained by

$$(\nabla_x\mathcal{F}_i^k)^{-1} \approx -2I - \nabla_x\mathcal{F}_i^k. \tag{5.10}$$

A good approximation can be obtained if (2.11) is not stiff and $\Delta\tau$ is small. Approximation (5.10) stays an option in ParNMPC.

5.3.3 Hessian approximation

Although ① in (5.5) is evaluated in parallel as shown in Algorithm 5.2, it sometimes still dominates the entire computation, e.g., when complicated dynamics and high-order discretization schemes are involved. Moreover, even when the exact Hessian is obtained, the descent property of the search direction is not guaranteed; thus, a regularization procedure is potentially required. Hessian approximation plays an important role in both numerical efficiency and robustness. For example, the Hessian of a high-order discretized problem can be approximated by that of a low-order discretized problem with a cheaper cost. We discuss in this subsection a commonly used Hessian approximation method in NMPC.

When the cost function is in the least-squares form:

$$L_i(u,x) := \frac{1}{2}\|l_i(u,x)\|_2^2,$$

where l_i is a vector-valued function, the generalized Gauss-Newton method (Bock, 1983) approximates the Hessian matrix ① in (5.5) by

$$\begin{bmatrix} \nabla_u l_i^k & \nabla_x l_i^k \end{bmatrix}^T \begin{bmatrix} \nabla_u l_i^k & \nabla_x l_i^k \end{bmatrix}. \tag{5.11}$$

The Gauss-Newton method performs well if the least-square residual is small and functions $\mathcal{F}_i(u,x)$ and $C_i(u,x)$ are less nonlinear such that their second-order derivatives are negligible. The Gauss-Newton Hessian approximation method is favoured in the context of NMPC, where a quadratic cost function is commonly chosen, e.g., in regulation and tracking control problems. Meanwhile, the positive definiteness of the lower-right block of (5.8) can be guaranteed for any DOP settings, which is shown in the following proposition.

Proposition 5.1. *Assume $\delta_H = 0$, $\delta_C > 0$ is chosen such that* (5.8) *is nonsingular, and the Jacobian of l is of full column rank. If W_i is initialized to satisfy $W_i \geq 0$ for $i \in \{1, \cdots, N\}$, then the lower-right block of* (5.8) *generated by the Gauss-Newton method is always positive-definite.*

Proof. From the full rank property of the Jacobian of l, we know that (5.11), i.e., ① in (5.5), is positive-definite. Since $z_i^k > 0$ and $G_i^k > 0$, ② in (5.5) always satisfies ② ≥ 0. Consequently, we know that $M_i^k > 0$ always holds. In addition, due to the fact that the initial value of W_{i+1} satisfies $W_{i+1} \geq 0$, the lower-right block of (5.8) is positive-definite for the initial value of W_{i+1}. We show next the definiteness of the updated W_i for $i \in \{1, \cdots, N\}$.

For each $i \in \{1, \cdots, N\}$, the Schur complement of the lower-right block of matrix (5.8) is nonsingular since we have shown that the lower-right block is positive-definite and (5.8) is nonsingular from the choice of δ_C. Specifically, the upper-left block of the inversion of (5.8) can be expressed as

$$\left(\begin{bmatrix} 0 & 0 \\ 0 & -\delta_C I \end{bmatrix} - J_i^k \left(M_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1} \end{bmatrix} \right)^{-1} (J_i^k)^T \right)^{-1}. \tag{5.12}$$

For an arbitrary vector $v \in \mathbb{R}^{n_x+n_\mu}$, the inequality

$$v^T J_i^k \left(M_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1} \end{bmatrix} \right)^{-1} (J_i^k)^T v \geq 0 \tag{5.13}$$

holds. Considering that $\delta_C > 0$, (5.13) holds, and (5.12) is nonsingular, we know that (5.12) < 0. Since the updated W_i is the negation of the n_x-th leading principal submatrix of (5.12), $W_i > 0$ always holds for $i \in \{1, \cdots, N\}$. In turn, the result then follows. □

Another type of Hessian approximation method is the quasi-Newton method, e.g., the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method (see, e.g., Nocedal and Wright (2006)), which requires only the first-order derivatives of the Hamiltonian to approximate the Hessian matrix and can estimate the curvature in the Hamiltonian. However, it is not easy to guarantee the positive definiteness of the Hessian update, especially for constrained problems. Although there are methods, such as the damped BFGS method (Nocedal & Wright, 2006), for guaranteeing the positive definiteness, quasi-Newton Hessian approximation methods are less favoured in the context of real-time optimization due to their fluctuating performance as reported by Quirynen (2017).

5.4 Line search

After obtaining the search directions ΔS and ΔZ, we first determine the maximum step size of an iteration so that the primal and dual feasibility conditions $G_i(u_i, x_i) \geq 0$ and $z_i > 0$ are satisfied for any $i \in \{1, \cdots, N\}$. It is recommended by Nocedal and Wright (2006) to have different step sizes for s and z (say, $\alpha_s^{\max}$ and $\alpha_z^{\max}$) to improve performance. $\alpha_s^{\max}$ and $\alpha_z^{\max}$ are obtained by using the fraction-to-the-boundary rule (Nocedal & Wright, 2006):

$$\alpha_z^{\max} = \max\{\alpha \in (0, 1] : z_i^k - \alpha \Delta z_i \geq (1 - \tau) z_i^k, \ \forall i \in \{1, \cdots, N\}\}$$

$$\alpha_s^{\max} = \max\{\alpha \in (0, 1] : G_i^k - \alpha \Delta G_i \geq (1 - \tau) G_i^k, \ \forall i \in \{1, \cdots, N\}\}$$

where

$$\Delta G_i = G_i^k - G_i(u_i^k - \Delta u_i, x_i^k - \Delta x_i)$$

and the fraction-to-the-boundary parameter τ is chosen to be $\tau = \min\{\tau^{\min}, \rho\}$ with $\tau^{\min} = 0.005$ (typical value). The update of z is performed by

$$Z^{k+1} = Z^k - \alpha_z^{\max} \Delta Z.$$

Although ΔS is a descent direction, the convergence of a full feasible step iteration cannot be guaranteed, especially when the current iterate is far from the optimal solution, e.g., a far reference. To ensure convergence, we perform the iterate

$$S^{k+1} = S^k - \alpha_s \Delta S, \;\; \alpha_s \in (0, \alpha_s^{\max}],$$

only when a specified merit function $Q(U, X)$ has been decreased by trying a sequence of values of α_s with a constant decay rate or α_s has reached its specified minimum value, e.g., 10^{-3}. The step size α_s is accepted if

$$Q(U^k - \alpha_s \Delta U, X^k - \alpha_s \Delta X) \leq Q(U^k, X^k) + \nu \alpha_s D_Q(U^k, X^k; \Delta U, \Delta X),$$

where $\nu \in (0, 1)$ and $D_Q(U^k, X^k; \Delta U, \Delta X)$ denotes the directional derivative of Q in the direction $(\Delta U, \Delta X)$. At the k-th iteration, we choose the merit function to be

$$\begin{aligned} Q(U, X) = \sum_{i=1}^{N} \{ & L_i(u_i, x_i) + \rho \Phi_i(u_i, x_i) \\ & + (\lambda^{\max})^T |x_{i-1} + \mathcal{F}_i(u_i, x_i)| \\ & + (\mu^{\max})^T |C_i(u_i, x_i)| \}, \end{aligned} \tag{5.14}$$

where $x_0 = \bar{x}_0$, $\lambda^{\max} \in \mathbb{R}^{n_x}$, and $\mu^{\max} \in \mathbb{R}^{n_z}$. We here choose the following heuristic penalty coefficients for better practical performance:

$$\lambda_{(j)}^{\max} = \max_{i \in \{1, \cdots, N\}} |\lambda_{i(j)}^k - \alpha_s^{\max} \Delta \lambda_{i(j)}|$$

and

$$\mu_{(j)}^{\max} = \max_{i \in \{1, \cdots, N\}} |\mu_{i(j)}^k - \alpha_s^{\max} \Delta \mu_{i(j)}|,$$

where $\lambda_{(j)}^{\max}$ and $\mu_{(j)}^{\max}$ are the j-th component of $\lambda^{\max}$ and $\mu^{\max}$, respectively.

An alternative to ensure convergence (Wächter & Biegler, 2006a) is the filter line search method (Fletcher & Leyffer, 2002), in which a trial step is accepted if it decreases the cost function or improves the constraint violation instead of a combination of those two in (5.14). This method is favoured against the merit-function-based line search method because it has a smaller computational cost per step and does not need to maintain a merit function, which depends on the choices of the penalty terms $\lambda^{\max}$ and $\mu^{\max}$. Both methods can be easily parallelized and are available in ParNMPC.

It should be noted that line search introduces an extra computational cost, which is not favoured in the context of real-time optimization. Moreover, special care should be taken with expensive computational costs, such as a feasibility restoration phase (Nocedal & Wright, 2006) and second-order corrections (Nocedal & Wright, 2006), when the step size becomes too small or the trial step has been rejected. Under what circumstances should we enable line search? Of course, line search is recommended offline in the case where the very first NMPC problem is deterministic and can be solved offline to provide an accurate initial guess. We discuss the online case in Remark 5.3.

Remark 5.3. *For NMPC, where successive optimization problems are solved, line search can be avoided, i.e., by setting $\alpha_s = \alpha_s^{max}$, if the initial state $\bar{x}_0$ varies smoothly with respect to time. This strategy has been applied in many of the real-time optimization methods for NMPC. For example, under certain conditions, the C/GMRES method can trace a KKT solution without line search. For the case of a distant solution caused by, e.g., far reference, line search should be enabled so that the convergence to a local optimum can be guaranteed (Yamashita, 1998) under certain assumptions.*

5.5 Warm start and barrier strategy

In many optimization methods for MPC, such as the active-set method, warm start can significantly reduce the number of iterations (NOI) since the optimal solution stays close to that of the last sampling time. However, the warm start of the interior-point method is less straightforward and remains challenging. In the interior-point method, relaxed problem (5.2) is solved successively with a decreasing barrier parameter $\rho > 0$, and an accurate solution is obtained when the barrier parameter ρ is decreased to be sufficiently small. The difficulty of warm start for the interior-point method is that the barrier parameter has to return back to its initial value at the next sampling time, which makes the optimization problem no longer close to its previous one.

One solution is to choose a fixed value (Wang & Boyd, 2010) of ρ, i.e., to obtain an approximate solution. This method is pretty acceptable in the context of MPC, where we care more about the closed-loop performance or cost. However, when a small value of ρ is chosen, it always leads to poor performance since the optimization problem is highly nonlinear at the boundary of the constraints. The iterates can hardly escape from the boundary due to the large entries of ② in (5.5).

To resolve the problem of warm start when pursuing a highly accurate solution, we adopt the two-phase strategy proposed by Zanelli, Quirynen, Jerez, and Diehl (2017). In the first phase, the relaxed problem is solved with a fixed barrier parameter ρ^0. Then, the barrier parameter is decreased with a constant rate at every iteration in the second phase. The strategy is illustrated in Fig. 5.1 showing the variation of ρ. At sampling time t, the relaxed problem with an initial barrier parameter ρ^0 is solved

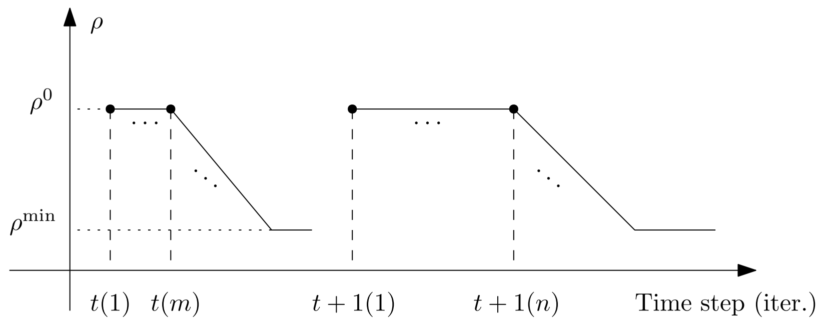


Figure 5.1: Barrier parameter ρ under two-phase strategy.

and the solution is stored to warm start the first iteration of the next sampling time $t+1$, that is, the solution at $t(m)$ is used to warm start $t+1(1)$.

The overall parallel optimization algorithm, together with the two-phase strategy, is summarized in Algorithm 5.3. The termination criterion is met when the optimality error for the relaxed problem is smaller than a predefined tolerance or the maximum NOI has been reached. We denote e_i^x, e_i^μ, e_i^u, e_i^λ as the ℓ_∞ norms of the four expressions in (5.3), respectively. The optimality error e_o is defined as

$$e_o := \max_{i\in\{1,\cdots,N\}} \left\{ e_i^x,\ e_i^\mu,\ \frac{e_i^u}{\sigma_u},\ \frac{e_i^\lambda}{\sigma_\lambda} \right\},$$

where $\sigma_u, \sigma_\lambda \geq 1$ are the scaling parameters. We adopt Wächter and Biegler's scaling strategy (Wächter & Biegler, 2006b) to choose σ_u and σ_λ as

$$\sigma_u = \max\left\{ \frac{\sum_{i=1}^N (\|\lambda_i\|_1 + \|\mu_i\|_1)}{N(n_x+n_\mu)},\ \sigma^{\max} \right\} / \sigma^{\max}$$

and

$$\sigma_\lambda = \max\left\{ \frac{\sum_{i=1}^N \|\lambda_i\|_1}{N n_x},\ \sigma^{\max} \right\} / \sigma^{\max}.$$

Here, $\sigma^{\max} \geq 1$ is a fixed number, e.g., $\sigma^{\max} = 100$.

Regarding the initialization of the very first optimization problem, for those which can be solved offline, an accurate initial guess can be provided with the offline solution. For those which cannot be solved offline, e.g., with an unknown initial state beforehand, the basic requirements for S^{init}, Z^{init}, and $W_{1,\cdots,N}^{\text{init}}$ are that the inequality $G_i(u_i, x_i) > \epsilon$ ($\epsilon > 0$ is a small number) should be satisfied for all $i \in \{1,\cdots,N\}$, its corresponding Lagrange multiplier z should be positive, and the sensitivity matrices should be positive-semidefinite. Conventional initialization techniques used in other methods can be applied to S^{init} and Z^{init}. For example, a state trajectory can be

Algorithm 5.3 Parallel optimization with two-phase strategy

```
Input: $\bar{x}_0$, $S^{\text{init}}$, $Z^{\text{init}}$, $W^{\text{init}}_{1,\cdots,N}$
Output: $S^{\text{init}}$, $Z^{\text{init}}$, $W^{\text{init}}_{1,\cdots,N}$, $S^*$, $Z^*$
 1: Initialize: $k = 0$; initial guesses: $S^0 = S^{\text{init}}$, $Z^0 = Z^{\text{init}}$, $W^{-1}_{1,\cdots,N} = W^{\text{init}}_{1,\cdots,N}$;
                barrier parameters: $\rho^0 > 0$, $\eta \in (0,1]$, $\rho^{\min} \in (0,\rho^0]$, $\rho = \rho^0$
 2: repeat
 3:     $(S^{k+1}, Z^{k+1}, W^{k}_{1,\cdots,N}) \leftarrow \mathit{Iter}(\bar{x}_0, S^k, Z^k, W^{k-1}_{1,\cdots,N}, \rho)$
 4:     $k \leftarrow k+1$
 5: until termination criterion is met
 6: $(S^{\text{init}}, Z^{\text{init}}, W^{\text{init}}_{1,\cdots,N}) \leftarrow (S^k, Z^k, W^{k-1}_{1,\cdots,N})$
 7: repeat
 8:     $\rho = \max\{\rho^{\min}, \eta\rho\}$
 9:     $(S^{k+1}, Z^{k+1}, W^{k}_{1,\cdots,N}) \leftarrow \mathit{Iter}(\bar{x}_0, S^k, Z^k, W^{k-1}_{1,\cdots,N}, \rho)$
10:     $k \leftarrow k+1$
11: until termination criterion is met
12: $(S^*, Z^*) \leftarrow (S^k, Z^k)$
  Procedure Iter
    Input: $\bar{x}_0$, $S^k$, $Z^k$, $W^{k-1}_{1,\cdots,N}$, $\rho$
    Output: $S^{k+1}$, $Z^{k+1}$, $W^{k}_{1,\cdots,N}$
    Calculate search directions (Algorithm 5.2)
    Perform line search (Section 5.4)
  End
```

obtained by interpolating from the initial state to the set-point. The initialization of the sensitivity matrices $W^{\text{init}}_{1,\cdots,N}$ is less straightforward. We show a possible initialization for the input-constrained problem with a quadratic cost function, that is, the stage cost function of relaxed problem (5.2) is in the form of

$$L_i(u_i, x_i) + \rho\Phi_i(u_i) = \frac{1}{2}\|x_i - x_{ref}\|^2_{Q_i} + \phi_i(u_i, \rho),$$

where x_{ref} is the given state reference, $Q_i > 0$ is the weighting matrix, and $\phi_i(u_i, \rho)$ is a function encoding the input-related cost. The following equality can be shown:

$$\lim_{\Delta\tau \to 0} W_i = \sum_{j=i}^{N} Q_i, \ i \in \{1, \cdots, N\},$$

and the right-hand side can be used as W^{init}_i.

5.6 Performance evaluation

In this section, the performance of ParNMPC is evaluated by using three examples. In the first, an example of quadrotor control, we demonstrate the problem formulation and parallel code generation procedures of ParNMPC. The computation time (CT) of ParNMPC is compared in detail against several state-of-the-art NMPC toolkits, and the performance of the two-phase and inversion approximation strategies is assessed as well. We show in the second example, in which a real-world laboratory helicopter is controlled, that ParNMPC converges fast when tracking a dramatically changed reference signal. The third experiment is a closed-loop simulation of a robot manipulator that demonstrates the Gauss-Newton Hessian approximation method and the parallel performance for systems with complicated dynamics.

All experiments were performed on a hexa-core 2.9-GHz (Turbo Boost and Hyper-Threading were disabled) Intel Core i9-8950HK laptop. The helicopter experiment was run in Simulink on Windows 10, and the others were run on Ubuntu 18.04. All code was automatically generated by using ParNMPC. To reduce the effect of the computing environment, the CT at each time step was measured by taking the minimum one of ten runs of the closed-loop simulation.

It should be noted that all three experiments were started from deterministic states, which made it possible to solve the very first optimization problem offline to provide an accurate initial guess, and line search in ParNMPC was only enabled then.

5.6.1 Quadrotor

Problem formulation

Consider a quadrotor with four inputs and nine states. The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T \in \mathbb{R}^9$, where (X, Y, Z) and (γ, β, α) are the position and angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. The dynamics of the quadrotor are given in (3.33). The goal was to control the quadrotor to position $(1, 1, 1)$ under the constraints of the inputs: $[0, -1, -1, -1]^T \leq u \leq [11, 1, 1, 1]^T$. We chose the cost function to be quadratic as

$$L_i(u, x) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2),$$

where x_{ref} encodes the position reference, $u_{ref} = [g, 0, 0, 0]^T$, and the weighting matrices were $Q = \mathrm{diag}(10, 1, 10, 1, 10, 1, 1, 1, 1)$ and $R = 0.01 \times I$. The prediction horizon was $T = 0.5$ s, which was discretized into $N = 24$ grids by using the reverse-time Heun's method.

The above NMPC problem is formulated in ParNMPC as shown in Listing 5.1.

```
% Create an OCP(n_u,n_x,n_p,N)
OCP = OptimalControlProblem(4,9,0,24);
X = OCP.x(1); dX = OCP.x(2); ...
a = OCP.u(1); omegaX = OCP.u(2); ...
OCP.setT(0.5);
OCP.setDiscretizationMethod('RK2');
f = [dX;a*(cos(gamma)*sin(beta)*cos(alpha)+sin(gamma)*sin(alpha));...];
Q = diag([10, 1, 10, 1, 10, 1, 1, 1, 1]); R = 0.01*eye(4);
xRef = [1;0;1;0;1;0;0;0;0]; uRef = [g;0;0;0];
L = 0.5*(OCP.x-xRef).'*Q*(OCP.x-xRef)+0.5*(OCP.u-uRef).'*R*(OCP.u-uRef);
G =[[11;1;1;1]-OCP.u;[0;1;1;1]+OCP.u];
OCP.setf(f); OCP.setL(L); OCP.setG(G);
OCP.codeGen();
```

Listing 5.1: Problem formulation in ParNMPC (some repeated code was omitted)

After formulating the NMPC problem, the NMPC solver is configured as shown in Listing 5.2. The exact Hessian is calculated, and the regularization parameters are made default in this example, i.e., $\delta_C = 10^{-9}$ and $\delta_H = 0$. The exact value of $(\nabla_x \mathcal{F})^{-1}$ is used by default.

```
% Configurate the NMPC solver
sol = NMPCSolver(OCP);
sol.setHessianApproximation('Newton');
% sol.setNonsingularRegularization(1e-9); % \delta_C
% sol.setDescentRegularization(0); % \delta_H
% sol.setInvFxMethod('exact');
sol.codeGen();
```

Listing 5.2: NMPC solver configuration in ParNMPC

Code generation

The parallel code for DOP = 6 of the NMPC controller can be automatically generated as shown in Listing 5.3. We fix the barrier parameter to be $\rho^0 = \rho^{\min} = 10^{-3}$. The two-phase strategy in Section 5.5 can be adopted by adjusting parameters ρ^0, $\rho^{\min}$, and η. Regarding the termination criterion, the optimality tolerance is 10^{-3}, and the maximum NOI is 10. The generated code is self-contained, easy-to-use, and can be easily deployed, e.g., using Visual Studio with *OpenMP Support* enabled or GCC with an extra *-fopenmp* flag. Given an initial state, the corresponding optimal control input can be obtained by simply calling function *NMPC_Solve* from the generated code.

```
options = createOptions();
options.DoP = 6;
options.isLineSearch = false;
options.rhoInit = 1e-3; % \rho^0
options.rhoEnd = 1e-3; % \rho^min
% options.rhoDecayRate = 0.1; % \eta
options.tolEnd = options.rhoEnd;
options.maxIterTotal = 10;
% Generate parallel C code
NMPC_Solve_CodeGen('lib','C',options);
```

Listing 5.3: Parallel code generation in ParNMPC

Evaluation of computation time

The CT of ParNMPC was evaluated by performing a three-second closed-loop simulation starting from $\bar{x}_0 = 0$ with a sampling period of 10 ms. We compared the following toolkits (the last three toolkits are based on the first-order methods):

- ParNMPC (version 1903-1): the toolkit introduced in this chapter.
- ParNMPC-primitive: the primitive version of ParNMPC, of which the algorithm is introduced in Section 4.4. The mechanism for handling inequalities was modified in this chapter to be the primal interior-point method.
- ACADO Code Generation Tool with qpOASES and qpDUNES: the SQP method based on a dense QP solver, qpOASES, and a sparse QP solver, qpDUNES.
- GRAMPC: a gradient-based augmented Lagrangian method.
- VIATOC: a gradient projection method for NMPC problems with input and state box constraints.
- FalcOpt: a projected gradient descent method.

Apart from the default parameters, some key tuning parameters for the different toolkits are shown in Table 5.1.

The closed-loop trajectories of the inputs and position when using ParNMPC are shown in Figs. 5.2 and 5.3, respectively. The CTs during the closed-loop simulation for the different toolkits are shown in Fig. 5.4. In addition, the optimality for each toolkit is defined as the normalized distance to its corresponding optimal trajectory, i.e.,

$$\frac{\sum_{t/0.01=0}^{300}\{L(\tilde{u}(t),\tilde{x}(t),p(t))-L(u^*(t),x^*(t),p(t))\}}{\sum_{t/0.01=0}^{300}L(u^*(t),x^*(t),p(t))}\times 100\%, \tag{5.15}$$

where $\tilde{u}(t)$ is the control input obtained by each toolkit, $u^*(t)$ is the optimal control input obtained by solving the corresponding OCP exactly, and $\tilde{x}(t)$ and $x^*(t)$ are the

Table 5.1: Tuning parameters for different toolkits in quadrotor example (e.g., RT-Heun stands for reverse-time Heun's method).

Toolkit	Discretization	Tuning parameters
ParNMPC	RT-Heun	As shown in Listings 5.1, 5.2, and 5.3
ParNMPC-primitive	RT-Euler	Barrier parameter: 0.001
ACADO (qpOASES)	Heun	GAUSS_NEWTON, MULTIPLE_SHOOTING, FULL_CONDENSING_N2, HOTSTART_QP, tolerence: 0.001
ACADO (qpDUNES)	Heun	GAUSS_NEWTON, MULTIPLE_SHOOTING, tolerence: 0.001
ACADO-RTI (qpDUNES)	Heun	Same as above but performing RTI scheme
GRAMPC	Heun	MaxMultIter: 1, MaxGradIter: 10
VIATOC	Heun	Number of iterations: 10
FalcOpt	Euler	Tolerance: 0.01

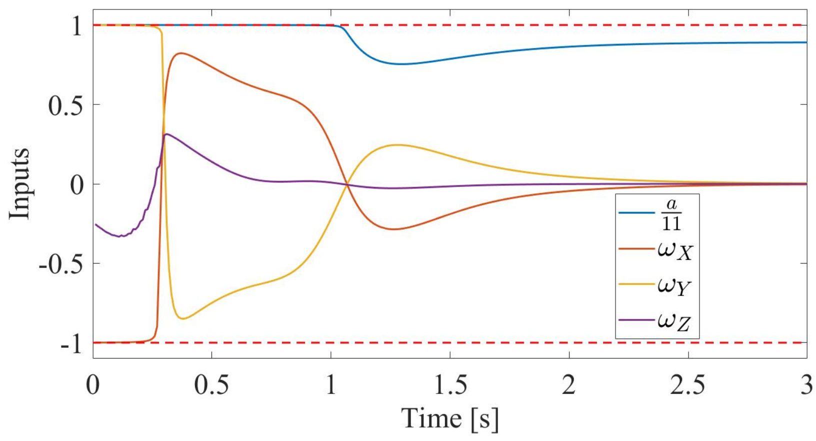


Figure 5.2: Time histories of inputs for quadrotor example.

corresponding closed-loop responses, respectively. The closed-loop optimalities of the different toolkits are shown in Table 5.2.

Table 5.2: Closed-loop optimalities [%] of different toolkits for quadrotor example

ParNMPC	0.1177	GRAMPC	1.5233
ParNMPC-primitive	0.0819	VIATOC	1.2743
ACADO (qpDUNES)	0.0002	FalcOpt	0.9719
ACADO-RTI (qpDUNES)	0.0108	-	-

Note that even though the first-order methods (GRAMPC, VIATOC, and FalcOpt) have relatively low optimalities, they can still drive the quadrotor to the reference position. In some circumstances, a sub-optimal solution obtained by performing only several iterations is acceptable. That is, we are also interested in the CT per iteration, which is compared for different N for these toolkits in Table 5.3.

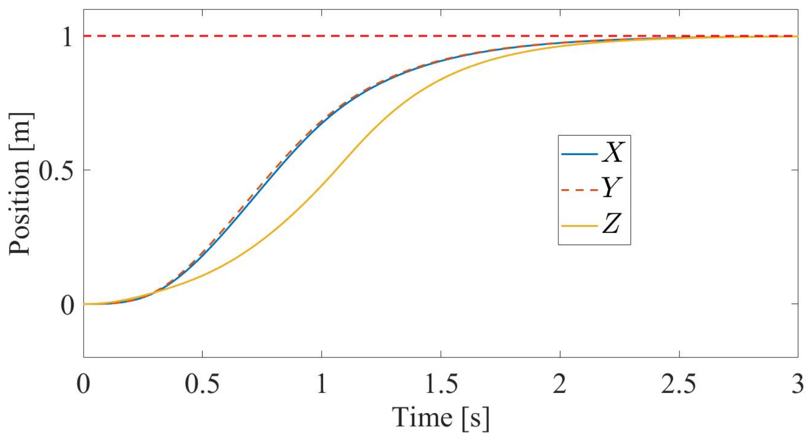


Figure 5.3: Time histories of position for quadrotor example.

We conclude from the comparison results as shown in Fig. 5.4 and Tables 5.2 and 5.3 that

- For ParNMPC, after parallelization, a speed-up of more than $4\times$ was achieved for $N = 96$ and even $2.5\times$ for a small number of discretion grid points of $N = 6$.
- Compared with ParNMPC-primitive, ParNMPC was a factor of 2-3 faster in terms of CT per iteration as shown in Table 5.3 both in serial and parallel, which was caused by condensing.
- Compared with ACADO, where a QP has to be solved at each iteration, the CT per iteration of ParNMPC in Table 5.3 was shorter than that of ACADO (qpDUNES) even in serial and did not vary too much during the closed-loop simulation due to the usage of the interior-point method. That is, the CT per time step can be roughly estimated from the NOI. However, ACADO was more effective than ParNMPC in terms of the closed-loop optimality as shown in Table 5.2.
- The first-order methods had short CTs per iteration, and ParNMPC with parallelization was only several times slower than first-order methods as shown in Table 5.3. However, the first-order methods converged slowly as indicated by their closed-loop optimalities in Table 5.2.

In summary, ParNMPC has a good trade-off between optimality and CT, resulting in an overall advantageous CT for the closed-loop control in Fig. 5.4, while maintaining a good closed-loop optimality as shown in Table 5.2.

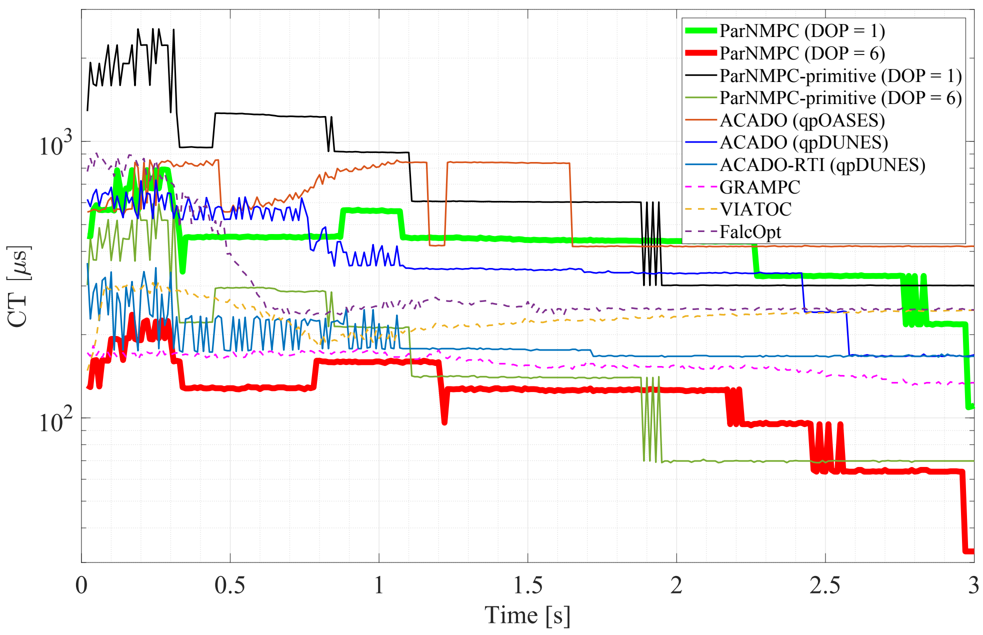


Figure 5.4: Computation times per time step of closed-loop simulation.

Evaluation of the two-phase and inversion approximation strategies

In the above quadrotor example, the performance of the two-phase and inversion approximation (5.10) strategies is not evaluated. In this part, we extended the simulation time to six seconds and set the position reference x_{ref} to zero for the last three seconds. We ran the experiment with different DOPs, barrier strategies (with different ρ^0 and $\rho^{\min}$), and $(\nabla_x \mathcal{F})^{-1}$ calculation methods as shown in Table 5.4. It should be noted that all of these patterns converged to the same optimality tolerance with the same terminal barrier parameter, and therefore, the same level of optimality.

Due to the choice of the small terminal barrier parameter $\rho^{\min}$, input signals approached the constraints very closely, and the closed-loop optimality (5.15) was decreased to 0.002. The NOIs and CTs for different patterns are shown in Table 5.5. By comparing patterns (b)(d)(e)(f) with (a)(c), it can be seen that the two-phase strategy illustrated in Fig. 5.1 significantly reduced the maximum NOIs for both DOPs. The approximation of $(\nabla_x \mathcal{F})^{-1}$ using (5.10) for patterns (e)(f) reduced the CT by 14% and 6% per iteration compared with patterns (b)(d), respectively, however, with a slight increase in the NOIs.

Table 5.3: Computation times [μs] per iteration for different N for quadrotor example.

N		6	12	24	48	96
	min	27	54	108	218	437
ParNMPC (DOP = 1)	median	28	55	110	222	444
	max	29	57	114	241	456
	min	11	18	31	55	101
ParNMPC (DOP = 6)	median	11	19	32	56	103
	max	12	20	33	63	111
	min	78	152	301	627	1251
ParNMPC-primitive (DOP = 1)	median	79	154	302	631	1261
	max	82	166	322	640	1287
	min	20	37	69	135	263
ParNMPC-primitive (DOP = 6)	median	21	38	70	137	267
	max	24	42	76	141	284
	min	30	81	269	1015	4571
ACADO (qpOASES)	median	32	99	417	2463	20377
	max	34	102	427	2496	20573
	min	41	83	120	248	520
ACADO (qpDUNES)	median	43	86	171	355	751
	max	60	112	245	492	1226
	min	42	83	166	345	719
ACADO-RTI (qpDUNES)	median	44	87	176	368	774
	max	79	161	363	819	1855
	min	3	6	13	26	53
GRAMPC	median	4	8	16	31	62
	max	5	9	18	35	71
	min	6	8	15	29	57
VIATOC	median	7	12	23	46	94
	max	8	13	31	100	469
	min	3	6	12	24	49
FalcOpt	median	3	6	12	25	51
	max	4	8	17	35	73

Table 5.4: Configurations.

Pattern	DOP	Two-phase strategy ($\eta = 0.1$)		$(\nabla_x \mathcal{F})^{-1}$
		ρ^0	$\rho^{\min}$	
(a)	1	10^{-5}	10^{-5}	Exact
(b)	1	1	10^{-5}	Exact
(c)	6	10^{-5}	10^{-5}	Exact
(d)	6	1	10^{-5}	Exact
(e)	1	1	10^{-5}	Approx.
(f)	6	1	10^{-5}	Approx.

5.6.2 Helicopter

In this experiment, we controlled Quanser's 3 degree-of-freedom (DOF) helicopter as shown in Fig. 5.5. The data acquisition and communication were done by using Quanser Q8-USB, which is able to provide a maximum closed-loop control rate of 2

Table 5.5: Number of iterations and computation times under different configurations.

Pattern	Max. NOI	Avg. NOI	Median CT/Iter. [μs]
(a)	37	6.0	111
(b)	19	13.0	111
(c)	38	6.5	32
(d)	21	13.4	32
(e)	20	13.2	95
(f)	21	13.4	30

kHz. The hardware blocks in Simulink were provided by Quanser's real-time control software, QUARC.

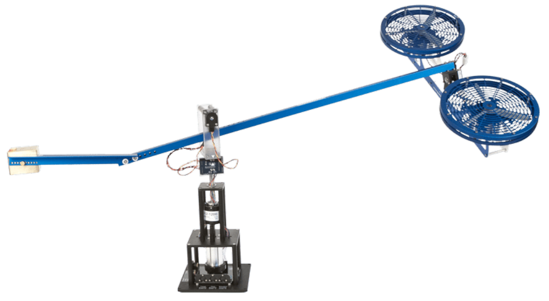


Figure 5.5: Quanser's 3 DOF helicopter (https://www.quanser.com/products/3-dof-helicopter/).

The helicopter has two inputs $u = [V_f, V_b]^T$: the voltage on the front motor V_f and the voltage on the back motor V_b. It has six states including three angles $q = [q_\epsilon, q_\rho, q_\lambda]^T$ (the elevation angle q_ϵ, pitch angle q_ρ, and yaw angle q_λ) and their time derivatives. We use the benchmark model in Brentari, Bosetti, Queinnec, and Zaccarian (2018) in the following form:

$$\ddot{q} = -\begin{bmatrix} \sin q_\epsilon (a_{\epsilon_1} + a_{\epsilon_2} \cos q_\rho) + C_\epsilon \dot{q}_\epsilon \\ -a_\rho \cos q_\epsilon \sin q_\rho + C_\rho \dot{q}_\epsilon \\ C_\lambda \dot{q}_\epsilon \end{bmatrix} + K_f \begin{bmatrix} b_\epsilon \cos q_\rho & 0 \\ 0 & b_\rho \\ b_\lambda \cos q_\epsilon \sin q_\rho & 0 \end{bmatrix} \begin{bmatrix} V_f + V_b \\ V_f - V_b \end{bmatrix},$$

where the parameters are given as: $a_{\epsilon_1} = 2.356$, $a_{\epsilon_2} = 0.799$, $a_\rho = 0.858$, $C_\epsilon = 0.053$, $C_\rho = 0.048$, $C_\lambda = 0.274$, $b_\epsilon = 0.719$, $b_\rho = 9.336$, $b_\lambda = 0.327$, and $K_f = 0.1188$. The goal was to control the helicopter to track a given reference q_{ref} under the constraints of the input voltages: $V_f, V_b \in [5, 10]$. We choose the cost function to be quadratic as follows.

$$L_i(u, x) = \frac{1}{2}(\|q - q_{ref}\|^2_{Q_q} + \|\dot{q}\|^2_{Q_{\dot{q}}} + \|u - \bar{u}\|^2_R)$$

Here, $\bar{u} = [7.5, 7.5]^T$ denotes the approximate voltage needed to eliminate the effect of gravity, $p = q_{ref}$, and the weighting matrices are $Q_q = 10 \times I$, $Q_{\dot{q}} = 0.1 \times I$, and $R = 0.1 \times I$. The prediction horizon was $T = 4$ s, which was discretized into $N = 24$ grids by using the reverse-time Runge-Kutta method method. The barrier parameter was fixed to 0.001, and DOP $= 1$. The exact Hessian and $(\nabla_x \mathcal{F})^{-1}$ were used.

The experiment was performed for 90 s with a sampling period of 5 ms. We first controlled the helicopter to track several step yaw angle references and then a sine signal with a skew rate of $\pi/2$ rad/s. The closed-loop responses of the angles are shown in Figs. 5.6 and 5.7, and the corresponding input signals are shown in Fig. 5.8. Despite the offset when tracking the piecewise constant reference, the NMPC controller could track the given reference well while satisfying the input constraints. As shown in Fig. 5.9, the proposed method converged to the specified tolerance with only five iterations at most, even though the reference signal was changed dramatically, resulting in sudden changes in the input signals flipping from one side to another. The time histories of the CT are shown in Fig. 5.10.

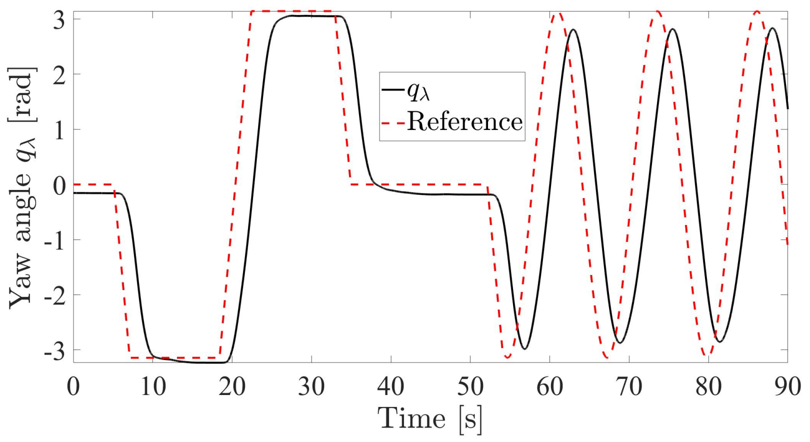


Figure 5.6: Time histories of helicopter's yaw angle.

5.6.3 Robot manipulator

The manipulator is a 7-DOF lightweight robot manipulator KUKA LBR iiwa 14, which has seven joint torque inputs $u = \tau \in \mathbb{R}^7$ and 14 states including seven joint angles $q \in \mathbb{R}^7$ and their corresponding angular velocities $\dot{q} \in \mathbb{R}^7$. The goal was to control the manipulator to track given joint angles' references q_{ref} under the constraints of the torques and angular velocities; each joint had a maximum torque output of 10 Nm, and the angular velocity for each joint was limited to have a maximum value

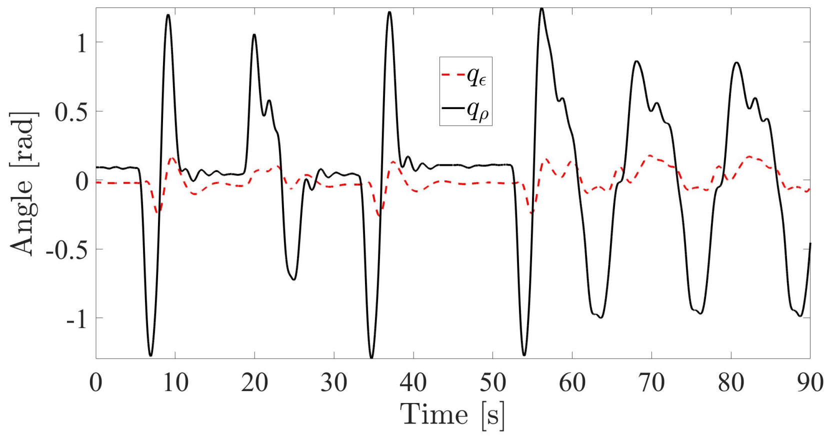


Figure 5.7: Time histories of helicopter's elevation and pitch angles.

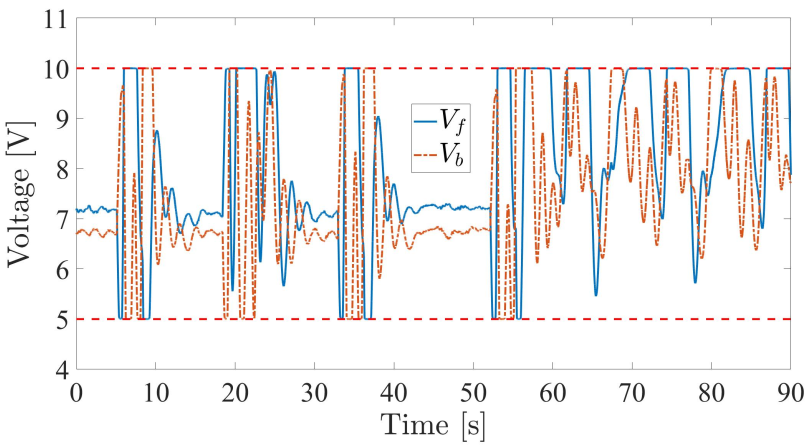


Figure 5.8: Time histories of helicopter's input signals.

of $\pi/2$ rad/s in order to achieve smooth movement. For the NMPC problem formulation, the angular velocity constraints were softened by introducing a slack variable $v \geq 0$, which denotes the violation of the constraints. The inequality constraint is

$$G_i(u,x) = \begin{bmatrix} \tau + 10e \\ -\tau + 10e \\ v \\ \dot{q} + \frac{\pi}{2}e + ve \\ -\dot{q} + \frac{\pi}{2}e + ve \end{bmatrix} \geq 0,$$

where $e = [1, \cdots, 1]^T$. We choose the cost function to be quadratic as follows.

$$L_i(u,x) = \frac{1}{2}(\|q - q_{ref}\|^2_{Q_q} + \|\dot{q}\|^2_{Q_{\dot{q}}} + \|\tau\|^2_R + 1000v^2)$$

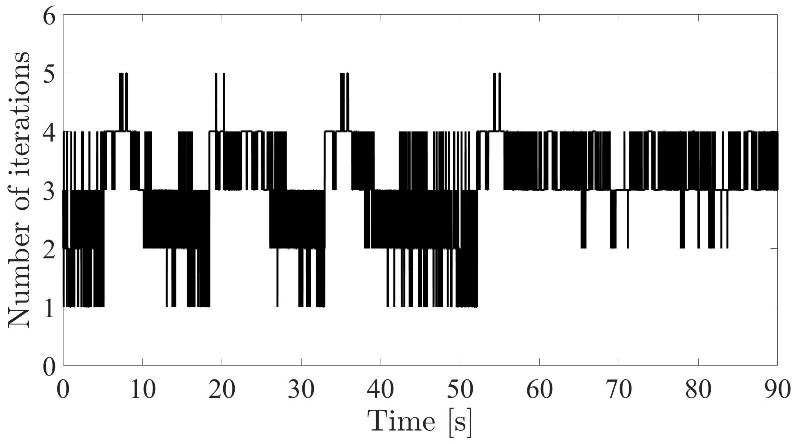


Figure 5.9: Time histories of number of iterations.

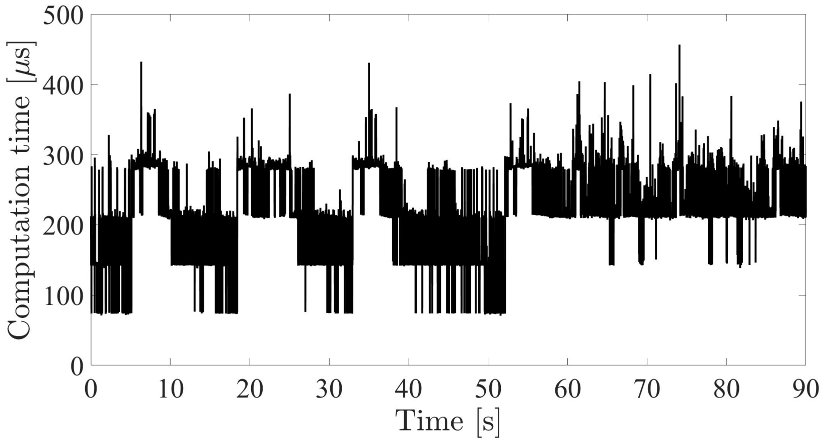


Figure 5.10: Time histories of computation time.

Here, $p = q_{ref}$ and the weighting matrices were $Q_q = I$, $Q_{\dot{q}} = 0.1 \times I$, and $R = 0.001 \times I$. The weighting imposed on v was 1000, which was at least three orders of magnitude larger than the other weightings. The prediction horizon was $T = 1$ s, which was discretized into $N = 18$ grids by using the reverse-time Euler's method method. The Gauss-Newton Hessian approximation method was chosen. The barrier parameter was fixed to 5×10^{-4}, and the exact $(\nabla_x \mathcal{F})^{-1}$ was used. The gravity was set to zero. The system function $f(u, x)$ and its derivatives were implemented using Pinocchio (Carpentier et al., 2019), which is a C++ library for efficient rigid multi-body dynamics computations.

For the closed-loop simulation, the system was started from an initial state of $\bar{x}_0 = 0$ and $q_{ref} = 0$. The simulation was performed for 8 s with a sampling period of 1 ms. The joints' reference was set to $q_{ref} = [0, \pi/2, 0, \pi/2, 0, \pi/2, 0]^T$ for the first four seconds and $q_{ref} = [\pi/2, 0, \pi/2, 0, \pi/2, 0, \pi/2]^T$ for the last four seconds. The reference was fed to the controller with a rate limitation of 4π rad/s.

In this experiment, we compared the CTs under different DOP settings (in serial and parallel). We note again that when DOP is set to one, ParNMPC behaves exactly the same as the structure-exploiting primal-dual interior-point method. For the measurement of the CT, we ran the simulation 10 times and chose the minimal one to eliminate the effect of performance fluctuation. Five sampled plots for the simulated manipulator are shown in Fig. 5.11, and the time histories of the joint torque inputs, joint angles, and the angular velocities are shown in Figs. 5.12, 5.13, and 5.14, respectively. Here, the last three torques are omitted in Fig. 5.12 because their magnitudes are in the range of $[-1, 1]$. We can see from the simulation results that the NMPC controller could control the robot manipulator to its desired reference position smoothly while satisfying the specified constraints. The NOIs for DOP = 1 and 6 are compared in Fig. 5.15, which shows that both converged to the specified tolerance with nearly the same rates, even though approximate information was used in the parallel mode. However, the parallel one was much faster than the serial one in terms of the CT shown in Fig. 5.16. A speed-up of more than $4\times$ could be achieved on the hexa-core processor. We noticed that, for both DOPs, the NMPC controller could still drive the manipulator to its reference position by using even only one iteration each sampling time. Considering that the CT per iteration for the parallel method was less than 170 μs, a higher sampling rate can be achieved with either more cores or fewer iterations.

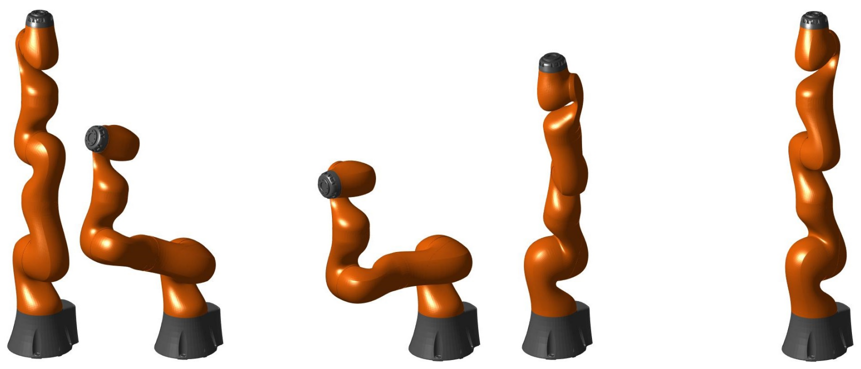


Figure 5.11: Plots of manipulator at 0, 1, 4, 5, and 8 s.

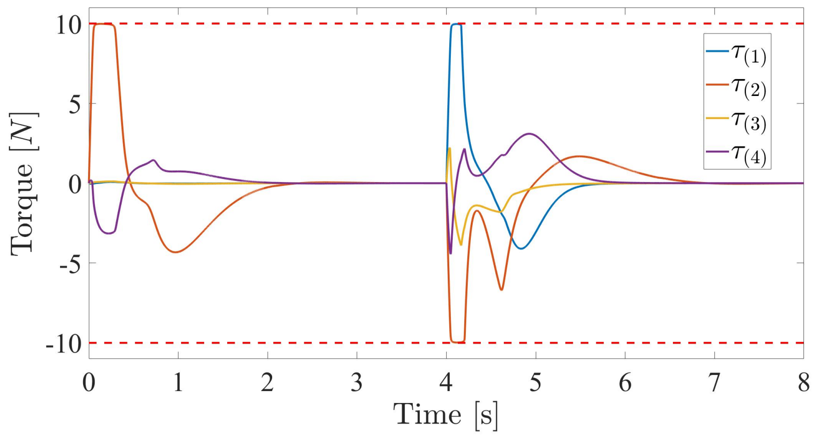


Figure 5.12: Time histories of manipulator's first four joint torque inputs.

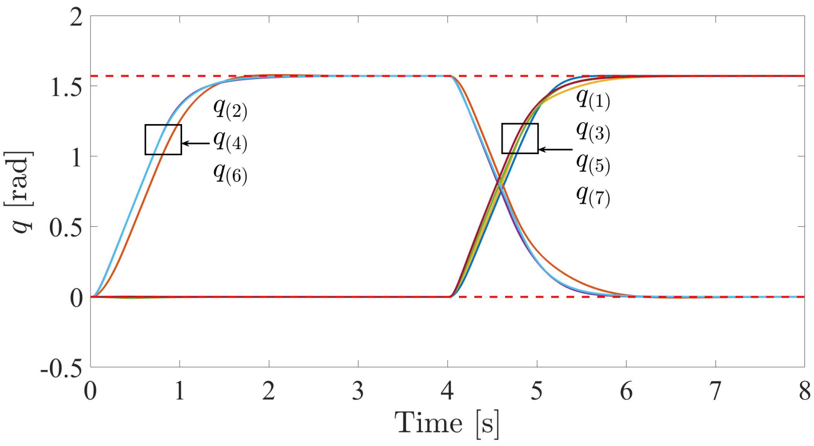


Figure 5.13: Time histories of manipulator's joint angles q.

5.6.4 Discussion

Since ParNMPC is exactly the primal-dual interior-point method applied to NMPC when DOP = 1, it inherits the good convergence and robustness of the interior-point method. Moreover, the increase in speed for ParNMPC with parallelization (DOP > 1) can be large on a hexa-core processor, so ParNMPC with parallelization is expected to behave better with more cores. Parallel performance can be achieved even for highly nonlinear systems (e.g., a robot manipulator) and a small N, with almost the same number of iterations. However, parallelization with OpenMP introduces an overhead time, which is usually proportional to DOP and is not negligible when either the overall CT is small or the DOP is large.

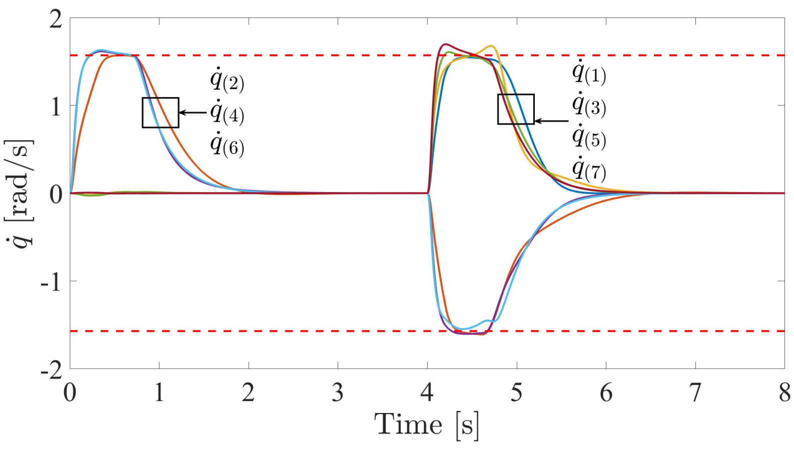


Figure 5.14: Time histories of manipulator's joint angular velocities $\dot{q}$.

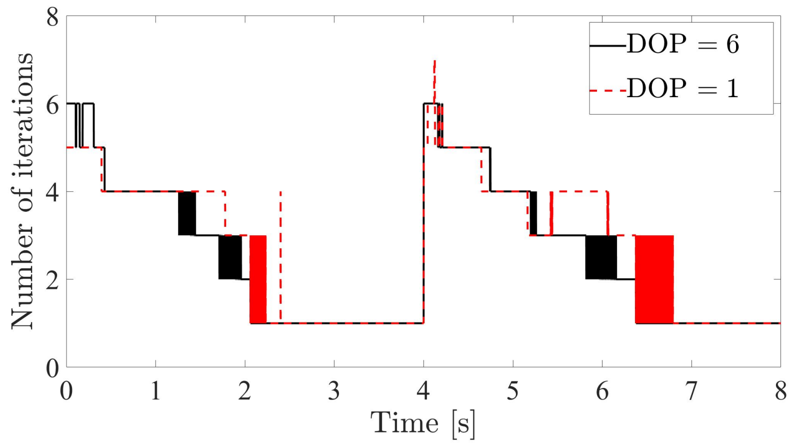


Figure 5.15: Time histories of numbers of iterations for parallel method (DOP = 6) and Newton's method (DOP = 1).

5.7 Summary

This chapter presented an efficient implementation for the parallel optimization of NMPC. The reverse-time discretization method dedicated for parallelization is introduced and its accuracy was shown. Since the parallel method is in the same framework as the conventional primal-dual interior-point method, nonlinear optimization techniques, such as regularization, Hessian approximation, and line search, are applied and optimized for the parallel optimization of NMPC, making the implementation both fast and numerically robust. Three experiments showed that ParNMPC was effective and efficient both in serial and parallel.

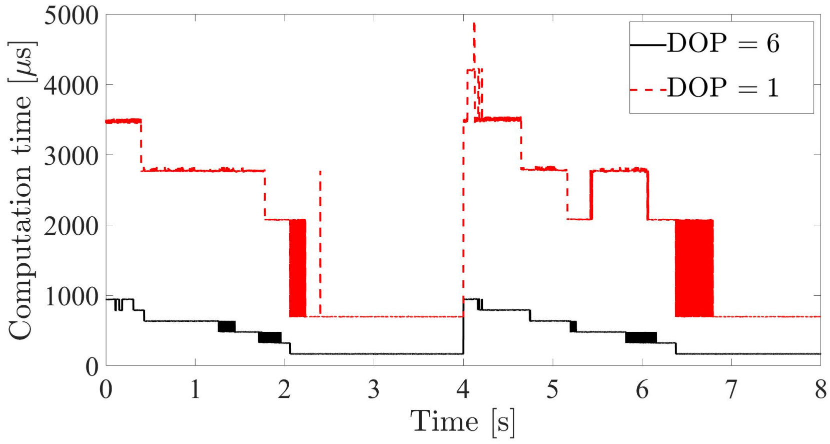


Figure 5.16: Time histories of computation times for parallel method (DOP = 6) and Newton's method (DOP = 1).

Future directions include integrating algorithmic differentiation to calculate the required derivatives for large-scale systems and replacing OpenMP with lower level parallel computing interfaces to reduce the overhead time.

Chapter 6

Sparsity-Exploiting Jacobi Method for NMPC

6.1 Introduction

In nonlinear model predictive control (NMPC), especially in the NMPC of large-scale systems, there are two kinds of sparsities in the Karush-Kuhn-Tucker (KKT) matrix, the upper-layer temporal sparsity along the prediction horizon and the lower-layer problem-dependent sparsity. Structure-exploiting methods for NMPC, such as the backward correction method (or, equivalently, Newton's method) and the parallel method introduced in Chapter 4, can be seen as methods that exploit the temporal sparsity along the prediction horizon in the KKT matrix. However, when these methods are applied, the lower-layer sparsity is not preserved, which results in computationally expensive dense matrix operations.

In this chapter, we present a sparsity-exploiting Jacobi method that exploits both the upper- and lower-layer sparsities. The upper-layer Jacobi (or Jacobi-type) method is derived by ignoring the temporal couplings of either the state or costate equations such that the lower-layer sparsity can be preserved. Convergence analysis shows that the upper-layer Jacobi method can be guaranteed to converge by introducing regularization and choosing a short prediction horizon. For the lower-layer sparsity exploitation, we concentrate on the NMPC control of systems governed by partial differential equations (PDEs) and present a tailored lower-layer Jacobi solver for the underlying linear systems of the PDE-constrained NMPC problem. The performance of the proposed method is assessed by controlling the temperature distribution of a two-dimensional heat transfer process on a thin plate. The proposed method is matrix-free and has a complexity of $\mathcal{O}(N(n_u + n_x))$ for the heat transfer example.

The numerical example shows that the proposed method is two orders of magnitude faster than the conventional structure-exploiting method.

This chapter is organized as follows. The problem is formulated in Section 6.2. The proposed Jacobi method for NMPC is introduced in Section 6.3. Section 6.4 introduces the application of the proposed method to PDE systems. Section 6.5 demonstrates the performance of the proposed method. Finally, this chapter is summarized in Section 6.6.

6.2 Problem statement

We consider the following N-stage NMPC problem with a temporal discretization step size of $h := T/N$:

$$\begin{aligned} \min_{\substack{u_1,\cdots,u_N,\\ x_0,\cdots,x_N}} \ & \sum_{i=1}^{N} h l_i(u_i, x_i) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_i = x_{i-1} + h f(u_i, x_i), \quad i \in \{1, \cdots, N\}, \\ & G_i(u_i, x_i) \geq 0, \quad i \in \{1, \cdots, N\}. \end{aligned} \tag{6.1}$$

For the sake of brevity, we omit the equality constraint $C_i(u, x) = 0$ and limit the discretization method to be the backward Euler's method with $\mathcal{F}_i(u_i, x_i) = f(u_i, x_i)h - x_i$ so that a maximum sparsity can be achieved. It should be noted that the main results obtained in Section 6.3 apply to the general discretized NMPC problem (2.19).

6.2.1 Regularized NMPC problem

Instead of solving the original NMPC problem (6.1), we add the following regularization term to the cost:

$$\frac{\gamma}{2}\|u_i - \tilde{u}_i^*\|^2, \tag{6.2}$$

where $\gamma \geq 0$ is the regularization parameter and $\tilde{u}_i^*$ is the given regularization reference and regarded as the estimation of the optimal control input of (6.1). We discuss later in Remark 6.1 the role of regularization and the selection of $\tilde{u}_i^*$.

We adopt the interior-point method to relax the regularized NMPC problem by transferring the inequality constraints into a logarithmic barrier function added to the cost. We obtain the following relaxed regularized NMPC (RR-NMPC) problem:

$$\begin{aligned} \min_{\substack{u_1,\cdots,u_N,\\ x_0,\cdots,x_N}} \ & \sum_{i=1}^{N} \Big(h l_i(u_i,x_i) + h\Phi_i(u_i,x_i) + \frac{\gamma}{2}\|u_i - \tilde{u}_i^*\|^2 \Big) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_i = x_{i-1} + hf(u_i,x_i), \quad i \in \{1,\cdots,N\}, \end{aligned} \tag{6.3}$$

where $\Phi_i(u,x) := -\tau \sum_j \ln G_{i(j)}(u,x)$ ($\tau > 0$ is the barrier parameter). The RR-NMPC problem (6.3) approaches the original NMPC problem (6.1) when $\tau \to 0$ and either $\gamma = 0$ or $\tilde{u}_i^*$ is given to be the optimal control input of (6.1).

6.2.2 KKT conditions

Let $\lambda_i \in \mathbb{R}^{n_x}$ be the Lagrange multiplier (costate) corresponding to the i-th state equation. For the sake of brevity, we define

$$s := (x,u,\lambda) \text{ and } S := (s_1,\cdots,s_N).$$

Let $\mathcal{H}_i(s)$ be the Hamiltonian defined by

$$\mathcal{H}_i(s) := l_i(u,x) + \Phi_i(u,x) + \lambda^T f(u,x).$$

Let $\mathcal{K}_i(x_{i-1},s_i,\lambda_{i+1})$ be defined by

$$\mathcal{K}_i(x_{i-1},s_i,\lambda_{i+1}) := \begin{bmatrix} x_{i-1} - x_i + hf(u_i,x_i) \\ h\nabla_u \mathcal{H}_i(s_i)^T + \gamma(u_i - \tilde{u}_i^*) \\ \lambda_{i+1} - \lambda_i + h\nabla_x \mathcal{H}_i(s_i)^T \end{bmatrix}$$

with $x_0 = \bar{x}_0$ and $\lambda_{N+1} = 0$. The KKT conditions for the RR-NMPC problem (6.3) are

$$\mathcal{K}_i(x_{i-1}^*, s_i^*, \lambda_{i+1}^*) = 0, \ i \in \{1,\cdots,N\}. \tag{6.4}$$

Although the KKT conditions (6.4) are only the necessary conditions for optimality, we solve the RR-NMPC problem (6.3) by solving the nonlinear algebraic equations (6.4).

We introduce the following shorthand at the k-th iteration:

$$\begin{aligned} &\mathcal{K}_i^k := \mathcal{K}_i(x_{i-1}^k, s_i^k, \lambda_{i+1}^k), \ \mathcal{K}^k := (\mathcal{K}_1^k,\cdots,\mathcal{K}_N^k), \\ &\nabla_{s_i}\mathcal{K}_i^k := \nabla_{s_i}\mathcal{K}_i(x_{i-1}^k, s_i^k, \lambda_{i+1}^k), \quad \text{etc.} \end{aligned}$$

6.2.3 Newton's method

In solving the KKT conditions (6.4) by using the Newton's method, the search direction $\Delta S^k := (\Delta s_1^k, \cdots, \Delta s_N^k)$ is obtained by solving the following KKT system:

$$\begin{bmatrix} \ddots & & & & \\ \cdots & D_{i-1}^k & M_U & & \\ & M_L & D_i^k & M_U & \\ & & M_L & D_{i+1}^k & \cdots \\ & & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ \Delta s_{i-1}^k \\ \Delta s_i^k \\ \Delta s_{i+1}^k \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathcal{K}_{i-1}^k \\ \mathcal{K}_i^k \\ \mathcal{K}_{i+1}^k \\ \vdots \end{bmatrix}. \tag{6.5}$$

Here, $D_i^k := \nabla_{s_i} \mathcal{K}_i^k$ (the expression can be found in (6.15)) and the constant matrices M_L and M_U given as follows show the couplings of the state and costate equations, respectively.

$$M_L := \begin{bmatrix} I_{n_x} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad M_U := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_{n_x} \end{bmatrix} \tag{6.6}$$

Note that M_U and M_L differ from (4.12) and (4.13) in Chapter 4, respectively, since the order of the unknown variables in s has been shifted. After the search direction is calculated, a line search is performed to guarantee the primal feasibility $G(u, x) \geq 0$, i.e.,

$$S^{k+1} = S^k - \alpha^{\max} \Delta S^k, \ i \in \{1, \cdots, N\}, \tag{6.7}$$

where $\alpha^{\max}$ is obtained from the fraction-to-the-boundary rule (Nocedal & Wright, 2006):

$$\alpha^{\max} = \max \left\{ \alpha^{\max} \in (0, 1] : \ G_i^{k+1} \geq 0.005 G_i^k, \ i \in \{1, \cdots, N\} \right\}. \tag{6.8}$$

Since the KKT matrix in (6.5) is a block-tridiagonal matrix, solving (6.5) by using the block Gaussian elimination method has a computational complexity of $\mathcal{O}(N(n_x + n_u)^3)$. Many of the existing structure-exploiting methods tailored to NMPC are of this complexity. Note that sparsity exists in both the upper-level KKT matrix in (6.5) and the lower-level Jacobian matrices D_i^k, $i \in \{1, \cdots, N\}$. However, since the recursion $\hat{D}_i^k := D_i^k - M_U (\hat{D}_{i+1}^k)^{-1} M_L$ with $\hat{D}_{N+1}^k := 0$ needs be performed from $i = N$ to 1 in the block Gaussian elimination method and $\hat{D}_i^k$ is not necessarily sparse (dense matrix inversion needs to be performed), the lower-level sparsity is not preserved, which makes the structure-exploiting methods computationally expensive for systems with large numbers of states.

6.3 Jacobi method for NMPC

6.3.1 Preliminaries

We first review the Jacobi method (see, e.g., Saad (2003)) for solving linear equations as follows.

Lemma 6.1. *Let*

$$Av = b, \tag{6.9}$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. Let A be decomposed into $A = D + L + U$, where D, L, and U are the diagonal, strict lower triangular, and strict upper triangular elements (blocks) of A, respectively. Assume that D is invertible. The Jacobi method for solving (6.9) *is given by*

$$v^{k+1} = D^{-1}(b - (L+U)v^k).$$

The Jacobi method converges if

$$\rho(D^{-1}(L+U)) < 1. \tag{6.10}$$

Likewise, for the element-wise decomposition, the Jacobi method converges if the matrix A is strictly diagonally dominant.

We show in the following some results on the convergence of general iterations.

Definition 6.1. *(Point of attraction (Ortega & Rheinboldt, 1970)). Consider the iteration*

$$v^{k+1} = H(v^k), \tag{6.11}$$

where $v \in \mathbb{R}^n$ and $H : P \to \mathbb{R}^n$ for a subset $P \subset \mathbb{R}^n$. Let v^* be an interior point of P and a fixed point of the iteration (6.11), *i.e., $v^* = H(v^*)$. Then, v^* is said to be a point of attraction of the iteration* (6.11) *if there is an open neighborhood $O \subset P$ of v^* such that the iterates defined by* (6.11) *all lie in O and converge to v^* for any $v^0 \in O$.*

Lemma 6.2. *(Ortega & Rheinboldt, 1970) Consider the iteration* (6.11). *Then, v^* is a point of attraction of the iteration* (6.11) *if the following condition holds:*

$$\rho(\nabla_v H(v^*)) < 1. \tag{6.12}$$

Lemma 6.3. *(Convergence factor and rate (Ortega & Rheinboldt, 1970)). If v^* is a point of attraction of the iteration* (6.11)*, the following holds:*

$$\rho(\nabla_v H(v^*)) = \lim_{k\to\infty} \sup \|v^k - v^*\|^{1/k},$$

and $\rho(\nabla_v H(v^*))$ is called the convergence factor. The convergence rate is defined by $-\ln \rho(\nabla_v H(v^*))$.

6.3.2 Algorithm

Let D^k, L, and U be the diagonal, strict lower triangular, and strict upper triangular blocks of the KKT matrix in (6.5) as follows.

$$\begin{aligned}
D^k &:= \text{block-diag}(D_1^k, \cdots, D_N^k)\\
L &:= \text{lower-block-diag}(M_L, \cdots, M_L)\\
U &:= \text{upper-block-diag}(M_U, \cdots, M_U)
\end{aligned}$$

The block-diagonal matrix D^k can be guaranteed to be invertible if h is sufficiently small and $\gamma > 0$. The Jacobi method for solving the KKT conditions (6.4) is given by

$$S^{k+1} = S^k - \alpha^{\max}(D^k)^{-1}\mathcal{K}^k, \tag{6.13}$$

where $\alpha^{\max} \in (0, 1]$ is a scalar obtained from the fraction-to-the-boundary rule (6.8) and S^0 is chosen such that the primal feasibility condition $G(u_i, x_i) > 0$ is satisfied for all $i \in \{1, \cdots, N\}$. Herein, we call the Jacobi method (6.13) for solving the KKT conditions (6.4) the upper-layer Jacobi method.

The upper-layer Jacobi method exploits the banded structure (temporal sparsity) of the KKT matrix by ignoring its off-diagonal blocks. That is, the couplings introduced by the state and costate equations are ignored during iteration so that (6.13) can be performed block-wisely (D^k is a block-diagonal matrix). Although the Jacobi method in (6.13) can be regarded as Newton's method ignoring off-diagonal blocks and Newton's method is known to be locally quadratically convergent under mild assumptions, the convergent property might not be preserved for the Jacobi method. The convergence of the upper-layer Jacobi method is analyzed in the following subsection.

6.3.3 Convergence

We first give a general convergence condition for the upper-layer Jacobi method.

Theorem 6.2. *S^* is a point of attraction of the iteration (6.13) if the following condition holds:*

$$\rho((D^*)^{-1}(L+U)) < 1. \tag{6.14}$$

Proof. Since $G_i(u_i^*, x_i^*) > 0$ is satisfied for all $i \in \{1, \cdots, N\}$, there exists an open neighborhood O of S^* such that for any $S^0 \in O$, the fraction-to-the-boundary rule (6.8) will never be triggered when the iteration converges, i.e., $\alpha^{\max} = 1$ in the neighborhood of S^*. The result then follows by applying Lemma 6.2 with $\alpha^{\max} = 1$, $\mathcal{K}^* = 0$, and $\nabla_S \mathcal{K}^* = D^* + L + U$. □

Theorem 6.2 gives a general sufficient condition for convergence. However, the condition (6.14) can only be verified afterward and does not provide significant insights related to the NMPC problem. We show in the following lemma and theorem that the upper-layer Jacobi method can be guaranteed to converge by tuning the NMPC parameters, such as the prediction horizon T and the regularization parameter γ.

Lemma 6.4. *Let D_i^* be decomposed into*

$$\begin{aligned} D_i^* &= \begin{bmatrix} h\nabla_x f_i^* - I & 0 & 0 \\ h\nabla_{ux}^2 \mathcal{H}_i^* & h\nabla_{uu}^2 \mathcal{H}_i^* + \gamma I & 0 \\ h\nabla_{xx}^2 \mathcal{H}_i^* & h\nabla_{xu}^2 \mathcal{H}_i^* & h(\nabla_x f_i^*)^T - I \end{bmatrix} \\ &\quad + h \begin{bmatrix} 0 & \nabla_u f_i^* & 0 \\ 0 & 0 & (\nabla_u f_i^*)^T \\ 0 & 0 & 0 \end{bmatrix} \\ &=: \bar{D}_i^* + h\tilde{D}_i^*. \end{aligned} \tag{6.15}$$

Let $\bar{D}^*$ and $\tilde{D}^*$ be defined as follows.

$$\begin{aligned} \bar{D}^* &:= \text{block-diag}(\bar{D}_1^*, \cdots, \bar{D}_N^*) \\ \tilde{D}^* &:= \text{block-diag}(\tilde{D}_1^*, \cdots, \tilde{D}_N^*) \end{aligned}$$

Then, for any $h > 0$ and $\gamma \geq 0$ such that $\bar{D}^*$ is invertible, e.g., when h is sufficiently small and γ is nonzero, the following holds:

$$\rho((\bar{D}^*)^{-1}(L+U)) = 0.$$

Proof. If $N = 1$ ($L = U = 0$), the result can be easily obtained. We discuss the case of $N \geq 2$. The proof is done by showing that the eigenvalues of $(\bar{D}^*)^{-1}(L+U)$ are all zero. In fact, the expression $\det(\sigma I - (\bar{D}^*)^{-1}(L+U)) = \sigma^{N(2n_x+n_u)}$ is obtained by using Schur complement recursively as shown below.

Define a set of matrices

$$\mathbb{A} := \left\{ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ P & 0 & 0 \end{bmatrix} \in \mathbb{C}^{(2n_x+n_u)\times(2n_x+n_u)},\ P \in \mathbb{C}^{n_x \times n_x} \right\}.$$

Define the following shorthand:

$$\bar{D}_i^L := -(\bar{D}_i^*)^{-1} M_L \text{ and } \bar{D}_i^U := -(\bar{D}_i^*)^{-1} M_U$$

so that

$$(\bar{D}^*)^{-1}(L+U) = -\begin{bmatrix} 0 & \bar{D}_1^U & & & \\ \bar{D}_2^L & 0 & \bar{D}_2^U & & \\ & \bar{D}_3^L & \ddots & \ddots & \\ & & \ddots & 0 & \bar{D}_{N-1}^U \\ & & & \bar{D}_N^L & 0 \end{bmatrix}.$$

For any $A \in \mathbb{A}$, $\sigma \in \mathbb{C}$, and $i \in \{2, \cdots, N\}$, it can be examined that

$$\bar{D}_{i-1}^U (\sigma I - A)^{-1} \bar{D}_i^L \in \mathbb{A}. \tag{6.16}$$

Let $K \in \{2, \cdots, N\}$ and $A_K \in \mathbb{A}$. We define a K-size (K blocks of rows and columns) block-tridiagonal matrix W_K by

$$\left[\begin{array}{cccc|c} \sigma I & \bar{D}_1^U & & & \\ \bar{D}_2^L & \sigma I & \bar{D}_2^U & & \\ & \bar{D}_3^L & \ddots & \ddots & \\ & & \ddots & \sigma I & \bar{D}_{K-1}^U \\ \hline & & & \bar{D}_K^L & \sigma I - A_K \end{array}\right] =: \begin{bmatrix} M_A & M_B \\ M_C & M_D \end{bmatrix}. \tag{6.17}$$

The determinant of W_K is given by

$$\det W_K = \det(\sigma I - A_K) \det(\operatorname{schur}(W_K, \sigma I - A_K)), \tag{6.18}$$

where $\operatorname{schur}(W_K, \sigma I - A_K)$ denotes the Schur complement of the block $\sigma I - A_K$ of W_K, i.e.,

$$\operatorname{schur}(W_K, \sigma I - A_K) = M_A - M_B M_D^{-1} M_C.$$

Let us then calculate the right hand side of (6.18). It can be shown that

$$\det(\sigma I - A_K) = \sigma^{2n_x+n_u}. \tag{6.19}$$

Since $A_K \in \mathbb{A}$, we can know from (6.16) that the only nonzero block (lower right corner) of $M_B M_D^{-1} M_C$ belongs to $\mathbb{A}$, i.e,

$$\bar{D}_{K-1}^U (\sigma I - A_K)^{-1} \bar{D}_K^L \in \mathbb{A}. \tag{6.20}$$

By choosing A_{K-1} to be the left hand side of (6.20), the Schur complement $\mathrm{schur}(W_K, \sigma I - A_K)$ can be seen as a $(K-1)$-size block-tridiagonal matrix in the form of (6.17), i.e.,

$$\mathrm{schur}(W_K, \sigma I - A_K) =: W_{K-1}. \tag{6.21}$$

By substituting (6.19) and (6.21) into (6.18), we obtain the following recursion:

$$\det W_K = \sigma^{2n_x+n_u} \det W_{K-1}.$$

Following the procedures above and together with $W_1 = \sigma I_{2n_x+n_u}$, we obtain

$$\det W_K = \sigma^{K(2n_x+n_u)}. \tag{6.22}$$

Now we choose $K = N$ and $A_K = 0 \in \mathbb{A}$, i.e.,

$$\det W_N = \det(\sigma I - (\bar{D}^*)^{-1}(L+U)) = \sigma^{N(2n_x+n_u)}. \tag{6.23}$$

From (6.23), we can know that $(\bar{D}^*)^{-1}(L+U)$ has only zero eigenvalues. The conclusion $\rho((\bar{D}^*)^{-1}(L+U)) = 0$ then follows. □

Theorem 6.3. *The convergence of the upper-layer Jacobi method* (6.13) *is described as follows.*

(i) Let γ be chosen to be nonzero. There exists $T_0 > 0$ such that $\rho((D^*)^{-1}(L+U)) < 1$ holds for any $T < T_0$.

(ii) There exists $\epsilon > 0$ such that $\rho((D^*)^{-1}(L+U)) < 1$ holds for any $\tilde{D}_i^*$ satisfying $\|\tilde{D}_i^*\| < \epsilon \|\bar{D}_i^*\|$, $i \in \{1, \cdots, N\}$.

Proof. Proof of (i). From the definitions in Lemma 6.4, we know that

$$\rho((D^*)^{-1}(L+U)) = \rho((\bar{D}^* + h\tilde{D}^*)^{-1}(L+U)). \tag{6.24}$$

We first choose $T_0 > 0$ to be sufficiently small. Since γ is a nonzero constant, $T < T_0$, and $h = T/N$ is sufficiently small, the right-hand side of (6.24) can be seen as a small perturbation of $\rho((\bar{D}^*)^{-1}(L+U))$ in terms of h. That is, together with Lemma 6.4, we obtain that

$$\lim_{h \to 0} \rho((D^*)^{-1}(L+U)) = 0.$$

From the continuities of matrix inverse and spectral radius, there always exists $T_0 > 0$ such that $\rho((D^*)^{-1}(L+U)) < 1$ holds for any $T < T_0$.

Proof of (ii). Let $\epsilon > 0$ be chosen to be sufficiently small. Since ϵ is a small number and $\|\tilde{D}_i^*\| < \epsilon \|\bar{D}_i^*\|$, $i \in \{1, \cdots, N\}$, $\rho((D^*)^{-1}(L+U)) = \rho((\bar{D}^* + h\tilde{D}^*)^{-1}(L+U))$ can also be seen as a small perturbation of $\rho((\bar{D}^*)^{-1}(L+U)) = 0$ in terms of $\tilde{D}^*$. Similarly to the proof of (i), the result then follows. □

Theorem 6.3 can be interpreted as follows. Theorem 6.3 (i) indicates that a nonzero regularization parameter γ and a short prediction horizon T guarantee the convergence of the upper-layer Jacobi method. However, note that neither a nonzero γ nor a small $T > 0$ is a necessary condition for satisfying $\rho((D^*)^{-1}(L+U)) < 1$. Since $\bar{D}_i^*$ consists of the sensitivity $\nabla_u f_i^*$, Theorem 6.3 (ii) can be interpreted as that $\rho((D^*)^{-1}(L+U)) < 1$ holds if the dynamical system (6.34) is not sensitive to the control input u.

Remark 6.1. *(Role of regularization). As can be seen from the proof of Theorem 6.3 (i), the nonzero regularization parameter γ makes $(\bar{D}^*)^{-1}$ less ill-conditioned (less sensitive to h) and therefore guarantees the convergence of the method when selecting a short prediction horizon T. However, a large γ makes the optimal solution of the RR-NMPC problem* (6.3) *far away from the original NMPC problem* (6.1) *unless the regularization reference $\tilde{u}_i^*$ in the regularization term* (6.2) *is chosen to be a good estimation of the optimal control input of* (6.1)*. Since the RR-NMPC problem has to be solved successively at every time step, $\tilde{u}_i^*$ can be fixed to be the optimal control input of the last time step or updated at each iteration, i.e., $\tilde{u}_i^* = u_i^k$. If $\tilde{u}_i^* = u_i^k$, the optimal solution to the RR-NMPC problem is equivalent to the optimal solution to the NMPC problem with only relaxation.*

6.3.4 Jacobi-type variants

The reason the iteration (6.13) is called the Jacobi method is that its convergence condition (6.14) mimics the condition (6.10) for solving linear equations. This can be seen as applying the Jacobi method to solve the KKT conditions. Likewise, Jacobi-type methods (see, e.g., Saad (2003)) for solving linear equations, such as the Gauss-Seidel and successive over-relaxation (SOR) methods, can be applied to solve the KKT conditions as well. We show several Jacobi-type methods and their conditions for convergence as follows.

- Forward Gauss-Seidel method (FGS):

$$S^{k+1} = S^k - \alpha^{\max}(D^k + L)^{-1}\mathcal{K}^k \tag{6.25}$$

Condition for convergence:

$$\rho((D^* + L)^{-1}U) < 1. \tag{6.26}$$

- Backward Gauss-Seidel method (BGS):

  $$S^{k+1} = S^k - \alpha^{\max}(D^k + U)^{-1}\mathcal{K}^k$$

  Condition for convergence:

  $$\rho((D^* + U)^{-1}L) < 1. \tag{6.27}$$

- SOR method:

  $$S^{k+1} = S^k - \alpha^{\max}\omega(D^k + \omega L)^{-1}\mathcal{K}^k,\ \omega > 0.$$

  Condition for convergence:

  $$\rho((D^* + \omega L)^{-1}(\omega U + (\omega - 1)D^*)) < 1.$$

- Symmetric Gauss-Seidel (SGS) method:

  $$S^{k+1} = S^k - \alpha^{\max}(D^k + L)^{-1}(\mathcal{K}^k - U(D^k + U)^{-1}\mathcal{K}^k). \tag{6.28}$$

  Condition for convergence:

  $$\rho((D^* + L)^{-1}U(D^* + U)^{-1}L) < 1. \tag{6.29}$$

Note that FGS and BGS have the same amount of computation as the Jacobi method. The difference is the rate of convergence as shown in the following theorem.

Theorem 6.4. *If the convergence condition* (6.14) *holds for the Jacobi method, then the convergence conditions* (6.26) *and* (6.27) *also hold for FGS and BGS, respectively. That is, S^* is a point of attraction of the FGS and BGS iterations. Moreover, both the FGS and BGS methods converge twice as fast as the Jacobi method.*

Proof. Since the KKT matrix in (6.5) is a block-tridiagonal matrix, the KKT matrix is consistently ordered (Hageman & Young, 1981). It is known from Saad (2003) that for a consistently ordered matrix, the spectral radius of FGS is the square of that of the Jacobi method, i.e.,

$$\rho((D^* + L)^{-1}U) = \rho((D^*)^{-1}(L + U))^2.$$

The conclusion above can be shown similarly for BGS that

$$\rho((D^* + U)^{-1}L) = \rho((D^*)^{-1}(L + U))^2.$$

Recall the definition of the convergence rate in Lemma 6.3. The result then follows. □

As can be seen from Theorems 6.3 and 6.4, a short prediction horizon T and a nonzero γ can also guarantee the convergence of FGS and BGS. The discussion on the role of the regularization procedure in Remark 6.1 applies to FGS and BGS as well. As for the SGS iteration (6.28), it can be seen as a BGS iteration followed by a FGS iteration, i.e., a backward sweep followed by a forward sweep. The iteration in the previous work Zavala (2016) is similar to the FGS iteration (6.25). However, the inequality constraints are kept and the regularization procedure is not introduced, so the convergence is difficult to guarantee.

The upper-layer Jacobi method and its variants exploit the upper-level sparsity of the KKT matrix in (6.5) and preserve the lower-level sparsity of the Jacobian matrices D_i^k, $i \in \{1, \cdots, N\}$. We show in the next section that the lower-level sparsity can be further exploited for the NMPC control of PDE systems.

6.4 Application to PDE systems

6.4.1 Motivation

Besides the NMPC control of systems governed by ordinary differential equations (ODEs), NMPC control of PDE systems, such as Navier-Stokes equations for fluid flow and heat transfer equations for chemical processes, has gained increasing attention due to the optimal and constraint-handling properties of NMPC. However, PDE-constrained NMPC presents a great challenge for real-time optimization due to the infinite-dimensional state space of PDE systems. The solution methods for PDE-constrained NMPC can be generally categorized as indirect or direct. Indirect methods analytically derive the optimality conditions for PDE-constrained NMPC and then solve these conditions numerically. For example, in Hashimoto, Yoshioka, and Ohtsuka (2013), the analytic optimality conditions for the NMPC control of a class of parabolic PDEs are derived. These optimality conditions are discretized to a set of nonlinear algebraic equations, the solution of which is then traced by using the C/GMRES method (Ohtsuka, 2004). Moreover, the so-called contraction mapping method can be applied efficiently if the nonlinear algebraic equations satisfy certain structure conditions. In contrast to indirect methods, which "first optimize, then discretize," direct methods need PDE systems to be first discretized both in space and time. The NMPC problem is formulated on the basis of the discretized PDE system, which leads to a nonlinear program. Since a fine-grained spatial discretization results in a large number of states or optimization variables, model reduction techniques are frequently applied. A common way, e.g., in Ou and Schuster (2009), is to combine the proper orthogonal decomposition method (Sirovich, 1987) with Galerkin projection to obtain a low-dimensional dynamical system. An alternative approach is to use Koopman operator-based reduced order models (Peitz & Klus, 2019). The key idea is to transform the dynamical system into switched autonomous systems by restricting the control input to a finite number of constant values. The switched autonomous systems are approximated by low-dimensional linear systems using the Koopman operator.

Unlike the model reduction methods, we deal directly with the discretized PDE system. PDE systems can be discretized in space into ODE systems by using, e.g., the finite difference method. Although conventional NMPC methods for ODE systems can in principle be applied, they are computationally expensive due to the large number of states. For example, structure-exploiting methods for NMPC, e.g., in Steinbach (1994) and Zanelli, Domahidi, Jerez, and Morari (2020), perform Riccati recursions and have computational complexities of $\mathcal{O}(N(n_u + n_x)^3)$. Even the state variables can be first eliminated in condensing-based methods, the elimination procedure is roughly of $\mathcal{O}(N^2 n_x^2)$ (Andersson, Frasch, Vukov, & Diehl, 2017). Note again that sparsity exists both in the spatial and temporal directions of PDE-constrained NMPC problems. Structure-exploiting methods can only make use of the temporal sparsity along the prediction horizon, and the lower-layer spatial sparsity is destroyed due to Riccati recursion. In this section, we apply the upper-layer Jacobi method to PDE-constrained NMPC problems and present a lower-layer Jacobi method that solves efficiently the underlying linear systems by exploiting the lower-layer spatial sparsity.

6.4.2 PDE systems

We consider the NMPC control of a general class of PDE systems defined on the spatial domain $\Omega \subset \mathbb{R}^n$ and temporal domain $\Gamma \subset \mathbb{R}$ (p here stands for the spatial position):

$$\begin{aligned} a(u(t), w(p,t))\frac{\partial^2 w(p,t)}{\partial t^2} + b(u(t), w(p,t))\frac{\partial w(p,t)}{\partial t} \\ = c(u(t), w(p,t))\triangle w(p,t) + d(u(t), w(p,t)), \end{aligned} \tag{6.30}$$

where w is the scalar PDE state, $\triangle w(p,t) := \sum_{i=1}^{n} \partial^2 w(p,t)/\partial p_{(i)}^2$ denotes the Laplacian of w, and a, b, c and d are twice-differentiable nonlinear functions of u and w. The boundary conditions, such as the Dirichlet and Neumann boundary conditions, can be given to be input- and state-dependent, i.e., as functions of u and w. We

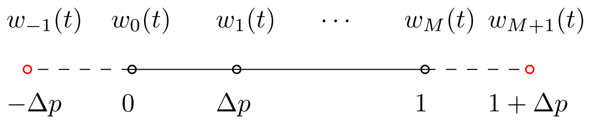


Figure 6.1: Spatial discretization points and fictitious points

assume that a is not zero for every admissible (u, w) when the second-order time derivative of w is involved.

Note that (6.30) is a very general description of PDE systems. Many of the PDE systems, such as the heat transfer equation and wave equation, fall into the form of (6.30). For PDE systems that are not in this form, e.g., the Navier-Stokes equations including both gradients and algebraic variables, we will discuss later in Remark 6.2 that the results of this chapter can in principle be extended.

6.4.3 Spatial discretization

We first introduce the spatial discretization of (6.30) by using the finite difference method. Without loss of generality, we demonstrate the discretization by using a one-dimensional system on an unit space interval $\Omega := [0, 1]$ satisfying the following Neumann boundary condition:

$$\frac{\partial w(p,t)}{\partial p} = e(u(t), w(p,t)), \ p = 0 \text{ and } 1. \tag{6.31}$$

Let $M + 1$ be the number of the spatial discretization grid points and Δp be the corresponding step size. The finite difference method is to approximate derivatives by using finite differences, i.e., for $j \in \{0, \cdots, M\}$,

$$\triangle w(j\Delta p, t) \approx \frac{w_{j+1}(t) - 2w_j(t) + w_{j-1}(t)}{\Delta p^2},$$

where $w_j(t) := w(j\Delta p, t)$. At $j = 0$ and M, two fictitious points $w_{-1}(t)$ and $w_{M+1}(t)$, as illustrated in Fig. 6.1, are introduced to deal with the boundary condition. By using the finite difference method, the Neumann boundary condition (6.31) translates into the difference equations as follows.

$$\frac{w_1(t) - w_{-1}(t)}{2\Delta p} = e(u(t), w_0(t)) \tag{6.32a}$$

$$\frac{w_{M+1}(t) - w_{M-1}(t)}{2\Delta p} = e(u(t), w_M(t)) \tag{6.32b}$$

The PDE system (6.30) is then discretized into

$$\begin{aligned}&a(u(t), w_j(t))\ddot{w}_j(t) + b(u(t), w_j(t))\dot{w}_j(t)\\&= c(u(t), w_j(t))\frac{w_{j+1}(t) - 2w_j(t) + w_{j-1}(t)}{\Delta p^2} + d(u(t), w_j(t)),\ j \in \{0, \cdots, M\},\end{aligned} \tag{6.33}$$

where the fictitious points $w_{-1}(t)$ and $w_{M+1}(t)$ can be eliminated by using the discretized boundary conditions (6.32). Note that the discretized PDE system (6.33) is described by a finite number of states:

$$\begin{aligned}x(t) &:= (W(t), \dot{W}(t))\\&:= (w_0(t), \cdots, w_M(t), \dot{w}_0(t), \cdots, \dot{w}_M(t)) \in \mathbb{R}^{n_x},\end{aligned}$$

where $n_x = 2(M + 1)$. The dynamics of the discretized PDE system are given by

$$\dot{x}(t) = \begin{bmatrix} \dot{W}(t) \\ g(u(t), x(t)) \end{bmatrix} =: f(u(t), x(t)), \tag{6.34}$$

where $g(u(t), x(t))$ denotes the expression of $\ddot{W}(t)$ obtained from (6.33).

6.4.4 Lower-layer Jacobi method

The upper-layer Jacobi method and its variants essentially consist of solving linear equations with the coefficient matrices D_i^k of the structure in (6.15) for $i \in \{1, \cdots, N\}$. Since D_i^k is sparse and its structure is fixed, efficient exact or iterative solution methods usually exist. For example, the Jacobi method for solving linear equations can be applied directly when h is sufficiently small. In this subsection, we introduce another iterative method by exploiting the particular structure of D_i^k.

The linear systems are reordered to have the following coefficient matrix:

$$\begin{bmatrix} 0 & h\nabla_x f_i^k - I & h\nabla_u f_i^k \\ h(\nabla_x f_i^k)^T - I & h\nabla_{xx}^2 \mathcal{H}_i^k & h\nabla_{xu}^2 \mathcal{H}_i^k \\ h(\nabla_u f_i^k)^T & h\nabla_{ux}^2 \mathcal{H}_i^k & h\nabla_{uu}^2 \mathcal{H}_i^k + \gamma I \end{bmatrix}. \tag{6.35}$$

For the sake of brevity, the linear system with the coefficient matrix (6.35) is expressed by using the following shorthand:

$$\begin{bmatrix} 0 & F_x & F_u \\ F_x^T & A_{xx} & A_{xu} \\ F_u^T & A_{ux} & A_{uu} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{6.36}$$

Equation (6.36) can be solved by first eliminating (v_1, v_2) and then solving for v_3, i.e., by performing the following two steps.

$$\begin{aligned}\left(A_{uu} - \begin{bmatrix} F_u^T & A_{ux} \end{bmatrix} \begin{bmatrix} 0 & F_x \\ F_x^T & A_{xx} \end{bmatrix}^{-1} \begin{bmatrix} F_u \\ A_{xu} \end{bmatrix}\right) v_3 \\ = b_3 - \begin{bmatrix} F_u^T & A_{ux} \end{bmatrix} \begin{bmatrix} 0 & F_x \\ F_x^T & A_{xx} \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\end{aligned} \tag{6.37a}$$

$$\begin{bmatrix} 0 & F_x \\ F_x^T & A_{xx} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} F_u \\ A_{xu} \end{bmatrix} v_3 \tag{6.37b}$$

Solving (6.37) consists of solving several linear equations of the following form:

$$\begin{bmatrix} 0 & F_x \\ F_x^T & A_{xx} \end{bmatrix} \begin{bmatrix} v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} b_4 \\ b_5 \end{bmatrix}. \tag{6.38}$$

The linear equation (6.38) can be solved by first solving $F_x v_5 = b_4$ and then solving $F_x^T v_4 = b_5 - A_{xx} v_5$. That is, linear equations with the coefficient matrices F_x and F_x^T are solved essentially. We show next that these linear equations can be solved efficiently by using the Jacobi method.

Recall that for the discretized PDE system (6.34), we have

$$F_x = h\nabla_x f_i^k - I = \begin{bmatrix} -I & hI \\ h\nabla_W g_i^k & h\nabla_{\dot{W}} g_i^k - I \end{bmatrix}.$$

Therefore, a linear equation with the coefficient matrix F_x, i.e., the following equation,

$$\begin{bmatrix} -I & hI \\ h\nabla_W g_i^k & h\nabla_{\dot{W}} g_i^k - I \end{bmatrix} \begin{bmatrix} v_6 \\ v_7 \end{bmatrix} = \begin{bmatrix} b_6 \\ b_7 \end{bmatrix}$$

can be solved by first eliminating v_6 with $v_6 = hv_7 - b_6$ and then solving a linear system with the following coefficient matrix:

$$h\nabla_{\dot{W}} g_i^k - I + h^2 \nabla_W g_i^k. \tag{6.39}$$

Since that $\nabla_{\dot{W}} g_i^k$ is diagonal, the off-diagonal entries of (6.39) have orders of $\mathcal{O}(h^2)$. It is easy to show that (6.39) is diagonally dominant if h is sufficiently small. Moreover, the off-diagonal entries of $\nabla_W g_i^k$ are sufficiently small for many of the PDE equations, such as the heat transfer equation and the Navier-Stokes equation with a large Reynolds number. That is, according to Lemma 6.1, convergence of the Jacobi method can be achieved for solving linear equations with the coefficient matrices (6.39). The same conclusion can be made for the F_x^T system.

We herein call the introduced Jacobi method the lower-layer Jacobi method. The lower-layer Jacobi method is concluded as follows. Since $n_x \gg n_u$ for PDE-constrained NMPC problems, the major computational cost for solving (6.36) comes from solving linear equations with the coefficient matrices (6.39), which can be solved efficiently by using the Jacobi method if, e.g., the NMPC problem is finely discretized in time, i.e., with a sufficiently small h.

Remark 6.2. *The key point of the lower-layer method is to find a method that solves the equation* (6.36) *efficiently, e.g., the introduced lower-layer Jacobi method for the NMPC control of the PDE system* (6.30)*. For PDE systems that are not in the form of* (6.30) *or even general systems, the proposed upper layer's iteration can be performed efficiently if a structure-exploiting linear solver can be designed.*

6.5 Numerical experiment

The upper-layer Jacobi method for solving the NMPC problem (6.1) and the lower-layer Jacobi method for solving the linear equation (6.36) form a double-layer Jacobi method for PDE-constrained NMPC problems. In this section, we demonstrate the performance of the double-layer Jacobi method in terms of the computation time, number of iterations, and convergence factor by using a heat transfer closed-loop control example. The experiment was implemented in C and performed on a 3.9-GHz (turbo boost frequency) Intel Core i5-8265U laptop computer. To reduce the effect of the computing environment, the computation time at each time step was measured by taking the minimum one of ten runs of the closed-loop simulation.

6.5.1 System description

We consider a nonlinear heat transfer process in a thin copper plate (Mathworks, 2020). Because the plate is relatively thin compared with the planar dimensions, temperature can be assumed constant in the thickness direction. The system is described by the following two-dimensional PDE:

$$\rho C_p t_z \frac{\partial w(p,t)}{\partial t} - k t_z \triangle w(p,t) + 2Q_c + 2Q_r = 0,$$

where w is the plate temperature, $p \in \Omega := \{(x,y)|x,y \in [0,1]\}$ (x here stands for the horizontal axis) and Q_c and Q_r are, respectively, the convection and radiation heat transfers defined as follows.

$$Q_c := h_c(w(p,t) - T_a)$$
$$Q_r := \epsilon\delta(w(p,t)^4 - T_a^4)$$

The boundary conditions are the zero Neumann boundary conditions. The parameters of the heat transfer process are given in Table 6.1. There are 16 actuators distributed

Table 6.1: Parameters in the heat transfer process.

ρ	8960	Density of copper [kgm^{-3}]
C_p	386	Specific heat of copper [Jkg^{-1}K^{-1}]
t_z	0.01	Plate thickness [m]
k	400	Thermal conductivity of copper [Wm^{-1}K^{-1}]
h_c	1	Convection coefficient [Wm^{-2}K^{-1}]
T_a	300	Ambient temperature [K]
ϵ	0.5	Emissivity of the plate surface
δ	$5.67 \cdot 10^{-8}$	Stefan-Boltzmann constant [Wm^{-2}K^{-4}]

uniformly under the plate to heat or cool the plate above the ambient temperature in the range of $[T_a, T_a + 400]$ K. The temperatures of the plate at the positions of the actuators can be controlled directly. We assume that the actuators negligibly impact the convection and radiation heat transfer processes. We are interested in controlling the temperature distribution across the plate under constraints, which is a typical control problem that arises in semiconductor manufacturing. For example, a temperature gradient needs to be maintained within a wafer to ensure catalytic activation (Bleris, Garcia, Kothare, & Arnold, 2006).

6.5.2 NMPC problem statement

The plate was uniformly discretized into 13 × 13 spatial grid points as shown in Fig. 6.2. Since the temperatures at the positions of the actuators can be controlled directly, the temperatures of the red squared points are regard as control inputs. We obtain a system with 16 inputs and 153 states. The inputs are constrained by

$$G(u, x, p) = \begin{bmatrix} u - T_a e \\ -u + (T_a + 400)e \end{bmatrix} \geq 0,$$

where $e = [1, \cdots, 1]^T$. We chose the cost function to be quadratic as

$$l(u, x, p) := \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2),\ i \in \{1, \cdots, N\},$$

where x_{ref} and u_{ref} encoded the temperature distribution reference and the weighting matrices were $Q = I$ and $R = 0.1 \times I$.

Note that the lower-layer Jacobi method converged fast and needed only two iterations due to the small thermal diffusivity $k\rho^{-1}C_p^{-1}$. Moreover, we noticed that the coefficient matrix in (6.37a) was dominated by the diagonal matrix A_{uu}. The

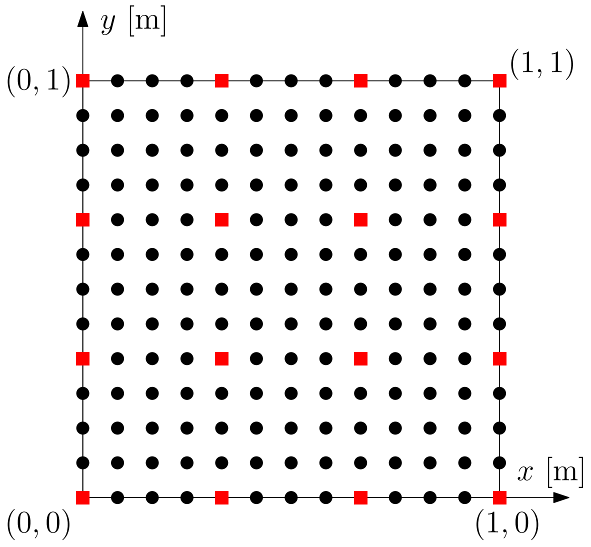


Figure 6.2: Spatial discretization grid points on plate (red squared points: actuators' positions).

linear equation (6.37a) in the lower-layer Jacobi method was solved iteratively by performing two of the following iterations:

$$A_{uu}v_3^{k+1} = b_3 + \begin{bmatrix} F_u^T & A_{ux} \end{bmatrix} \begin{bmatrix} 0 & F_x \\ F_x^T & A_{xx} \end{bmatrix}^{-1} \left(\begin{bmatrix} F_u \\ A_{xu} \end{bmatrix} v_3^k - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right), \tag{6.40}$$

which can be solved efficiently due to the diagonal property of A_{uu}. The parameters of the NMPC controller are given in Table 6.2. Since all of the matrices during iteration

Table 6.2: NMPC parameters.

Name	Value
Prediction horizon T	100 [s]
# of temporal discretization points N	20
Barrier parameter τ	100
Regularization reference $\tilde{u}_i^*$	u_i^k
Regularization parameter γ	0.5
Stopping criterion	$\|\mathcal{K}^k\|_\infty < 1$
Upper-layer method	SGS (6.28)
Lower-layer method	Two Jacobi iterations

were sparse, the expressions of the matrix-vector multiplications were pre-computed offline, which made the proposed method matrix-free. The computational complexity of the proposed method for the heat transfer example is $\mathcal{O}(N(n_x + n_u))$.

6.5.3 Closed-loop simulation

The system was started from an initial state of $\bar{x}_0 = [T_a, \cdots, T_a]^T$. The simulation was performed for 1000 s with a sampling period of 5 s. The temperature distribution reference, as shown in Fig. 6.3, was set to a slope shape for the first 500 seconds and a V-like shape for the last 500 seconds. The second reference was fed to the controller

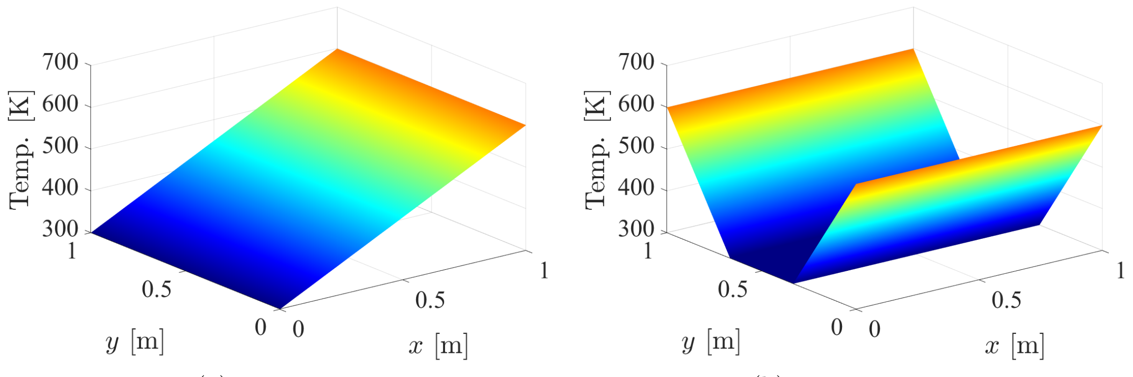
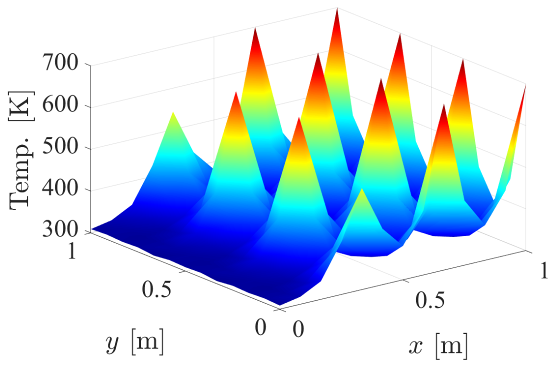


Figure 6.3: Temperature distribution references at different time periods.

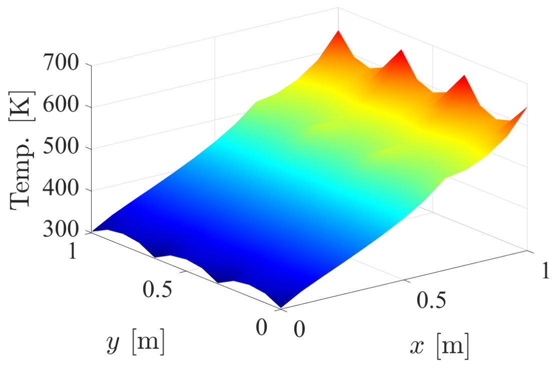
by changing continuously from the first reference within 50 seconds.

For tracking the first reference of the closed-loop simulation, two sampled plots at $t = 50$ s and $t = 500$ s are shown in Fig. 6.4 (a) and (b). For the second reference, two sampled plots at $t = 550$ s and $t = 1000$ s are shown in Fig. 6.4 (c) and (d). As shown by these plots, the references were tracked well by using the NMPC controller. The time histories of the control inputs are shown in Fig. 6.5. Although the barrier parameter τ was fixed to 100, a high accuracy was still achieved such that the inputs approached the boundaries very closely.

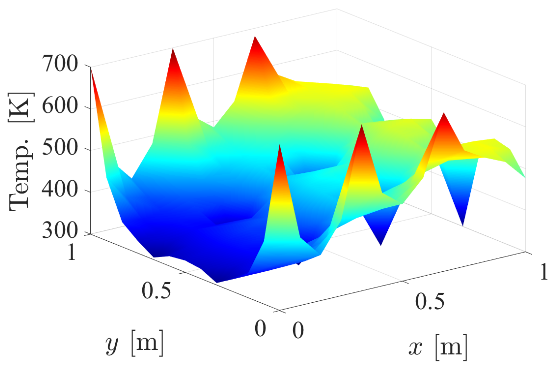
To demonstrate the performance of the proposed method, we compared the conventional Newton's method introduced in Section 6.2.3. The search direction (6.5) in Newton's method was calculated by using the block Gaussian elimination method, which was implemented by using NMPC real-time optimization software ParNMPC (version 1903-1) (https://github.com/deng-haoyang/ParNMPC), i.e., the toolkit introduced in Chapter 5 with a degree of parallelism of one. Note that since both methods were based on the interior-point method, their computation times per iteration were consistent throughout the closed-loop simulation. The mean computation time per iteration for Newton's method was 0.180 s, which was about 433 times of that of the proposed method (0.416 ms). Considering that their numbers of iterations



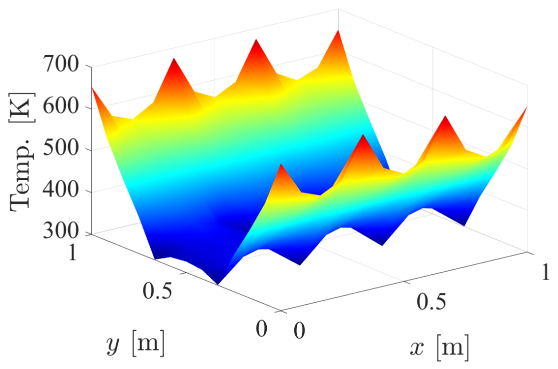
(a) $t = 50$ s



(b) $t = 500$ s



(c) $t = 550$ s



(d) $t = 1000$ s

Figure 6.4: Temperature distributions at different time t.

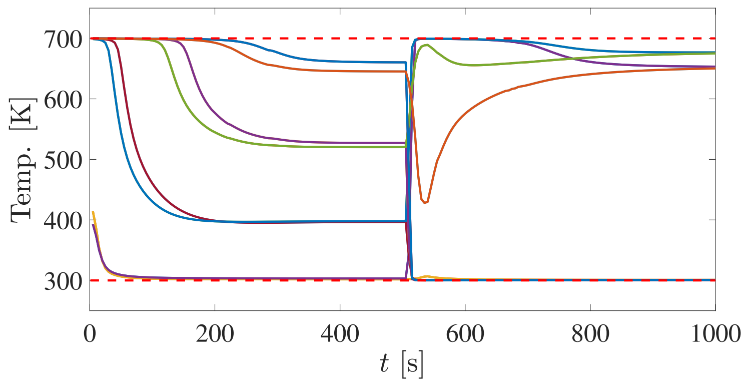


Figure 6.5: Time histories of inputs (some inputs coincide with each other).

shown in Fig. 6.6 were in the same range, the proposed method was much faster than Newton's method in terms of the computation time per time step shown in Fig. 6.7. Furthermore, since the proposed method for the heat transfer example was matrix-free, its compiled executable file size (440 KB) was only about one tenth of that of Newton's method, which enables embedded applications for the proposed method.

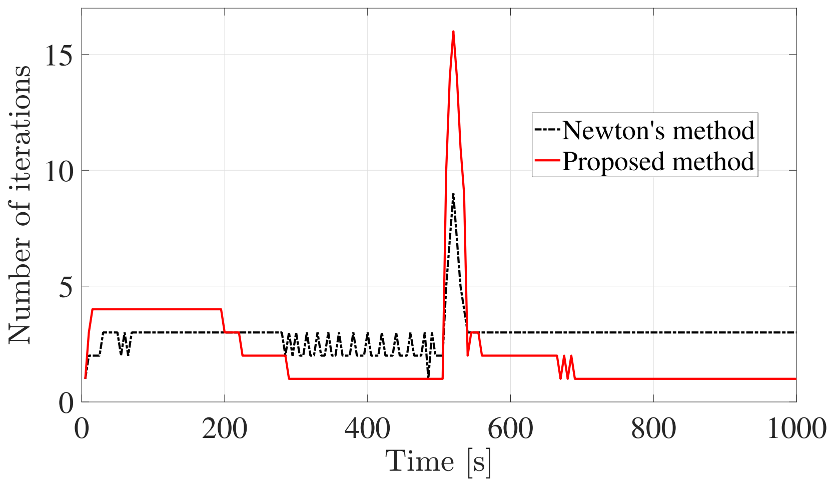


Figure 6.6: Time histories of numbers of iterations.

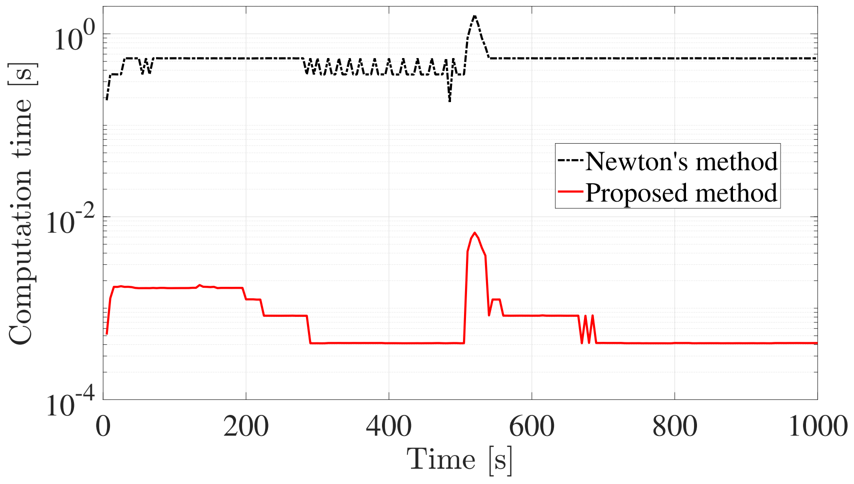


Figure 6.7: Time histories of computation times per time step.

Lastly, we discuss the effects of the prediction horizon and regularization. In the numerical experiment, the proposed method could not converge without regularization ($\gamma = 0$). According to Theorem 6.3 and Remark 6.1, the convergence can be guaranteed by shortening the prediction horizon and introducing a positive regularization parameter γ. We compared the convergence factor $\rho((D^*+L)^{-1}U(D^*+U)^{-1}L)$ for the upper layer's SGS iteration along the closed-loop simulation under different prediction horizons ($T = 20$ and 100) and regularization parameters ($\gamma = 0$ and 0.5) in Fig. 6.8. It can be seen that the convergence condition $\rho((D^* + L)^{-1}U(D^* + U)^{-1}L) < 1$ was satisfied with either regularization or a short prediction horizon.

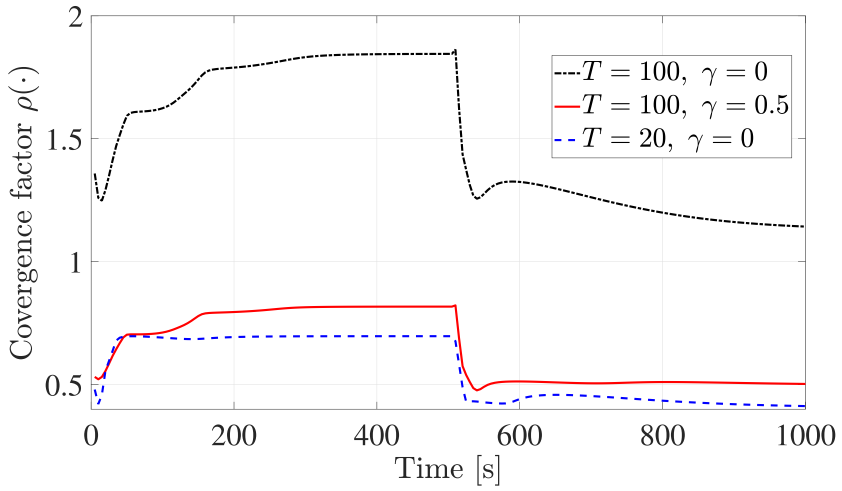


Figure 6.8: Time histories of convergence factors of SGS under different settings.

6.6 Summary

This chapter presents the Jacobi method for NMPC and its application to the NMPC control of PDE systems. The proposed upper-layer Jacobi method performs simple Jacobi-type iterations to solve the KKT conditions to make full use of the sparsities exist in the upper and lower layers. Furthermore, the convergence of the proposed method can be guaranteed by adjusting the prediction horizon and regularization parameter. The proposed method is applied to PDE-constrained NMPC problems, and a lower-layer Jacobi method is proposed to solve the underlying linear equations. The results of the numerical experiment show that the proposed method can significantly reduce the computation time and program size.

Future research directions include extending the proposed method to the NMPC control of other large-scale systems, such as power grids and multi-agent systems.

Chapter 7

Conclusions and Outlook

In this thesis, we have proposed a variety of fast numerical optimization methods for both linear model predictive control (MPC) and nonlinear MPC (NMPC). The arising quadratic programs and nonlinear programs in linear MPC and NMPC, respectively, are optimization problems with particular structures and properties, which are captured in designing the efficient proposed methods. The combined first- and second-order method in Chapter 3 makes use of the convexity and multi-stage property of the linear MPC problem. In Chapters 4 and 5, the parallel method and its implementation depend on the smoothly varying property of the sensitivity matrix. The Jacobi method in Chapter 6 takes advantage of the double-layer sparsities arising in large-scale NMPC problems. The experiments of aircraft, quadrotor, helicopter, robot manipulator, and heat-transfer process control have demonstrated the efficiency and potential of the proposed methods. Therefore, it is expected that the real-time MPC control of systems with complicated dynamics, many inputs and states, and high sampling rates can be made possible by using the proposed methods.

As we have mentioned in Chapter 1, the success of linear MPC lies in its simplicity and easy understanding. Due to the convexity, optimization methods for linear MPC can be extremely reliable for practical industrial and safety-critical applications. On the other hand, the generality of NMPC makes it powerful to handle more types of costs, dynamics, and constraints, however, brings complexities to its underlying optimization methods. Although it is generally hard to find the global optimal solution for a nonconvex NMPC problem within a limited amount of time, real-time optimization methods for NMPC should be at least fast and reliable in finding local solutions. In addition, optimization methods for NMPC should be able to report and recover from numerical failures.

Appendix A

Explicitly Discretized MPC

A.1 Problem statement

In the previous chapters, the model predictive control (MPC) problem is formulated on the basis of the following discrete-time system obtained by using the reverse-time discretization method:

$$x_{i-1} + \mathcal{F}_i(u_i, x_i) = 0.$$

We here discuss how to extend the proposed methods to solve the MPC problems on the basis of the explicit discretization methods, i.e., the following discrete-time system:

$$x_{i+1} + \mathcal{F}_i(u_i, x_i) = 0. \tag{A.1}$$

The discrete-time system (A.1) can be obtained by using, e.g., the forward Euler method with $\mathcal{F}_i(u_i, x_i) = -x_i - hf(u_i, x_i)$, where $h > 0$ is the step size.

We first show that the combined first- and second-order method in Chapter 3 can be extended easily to the explicitly discretized linear MPC problem in the following section.

A.2 Combined first- and second-order method for linear MPC

Consider a conventional MPC problem on the basis of the following discrete-time system:

$$x_{i+1} + Ax_i + Bu_i = 0.$$

By multiplying A^{-1} and redefining the index of u, we can obtain the following "reverse-time" system:

$$x_i + A^{-1}x_{i+1} + A^{-1}Bu_{i+1} = 0.$$

That is, the conventional linear MPC problem and the linear MPC problem (3.1) on the basis of the "reverse-time" system can be transferred mutually if A is nonsingular. In conclusion, the combined first- and second-order method for linear MPC in Chapter 3 can be extended by transforming the conventional MPC problem to the form of (3.1).

A.3 Highly parallelizable Newton-type method for NMPC

Since the equality constraint $C(u, x) = 0$ and the inequality constraint $G(u, x) \geq 0$ are the constraints on the current stage's variables and thus introduce no couplings between the neighboring stages, we consider the following nonlinear MPC (NMPC) problem on the basis of the explicitly discretized system (A.1):

$$\begin{aligned} &\min_{X,U} \sum_{i=0}^{N-1} L_i(u_i, x_i) + \varphi(x_N) \\ &\text{s.t.} \quad x_0 = \bar{x}_0, \\ &\qquad\;\; x_{i+1} + \mathcal{F}_i(u_i, x_i) = 0, \quad i \in \{0, \cdots, N-1\}, \end{aligned} \tag{A.2}$$

where $X = (x_0, x_1, \cdots, x_N)$ and $U = (u_0, u_1, \cdots, u_{N-1})$ are, respectively, the sequences of states and inputs along the horizon.

A.3.1 KKT conditions

Let $\lambda_{i+1} \in \mathbb{R}^{n_x}$ be the Lagrange multiplier (costate) corresponding to the i-th state equation. For the sake of brevity, we define

$$s_i := (\lambda_i, u_{i-1}, x_i) \text{ and } S := (s_1, \cdots, s_N).$$

Let $\mathcal{H}_i(\lambda_{i+1}, u_i, x_i)$ be the Hamiltonian defined by

$$\mathcal{H}_i(\lambda_{i+1}, u_i, x_i) := L_i(u_i, x_i) + \lambda_{i+1}^T \mathcal{F}_i(u_i, x_i).$$

For $i \in \{1, \cdots, N-1\}$, let $\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1})$ be defined by

$$\mathcal{K}_i(x_{i-1}, s_i, \lambda_{i+1}) := \begin{bmatrix} x_i + \mathcal{F}_{i-1}(u_{i-1}, x_{i-1}) \\ \nabla_u \mathcal{H}_{i-1}(\lambda_i, u_{i-1}, x_{i-1})^T \\ \lambda_i + \nabla_x \mathcal{H}_i(\lambda_{i+1}, u_i, x_i)^T \end{bmatrix} \tag{A.3}$$

with $x_0 = \bar{x}_0$. Let $\mathcal{K}_N(x_{N-1}, s_N, \lambda_{N+1})$ be defined by

$$\mathcal{K}_N(x_{N-1}, s_N, \lambda_{N+1}) := \begin{bmatrix} x_N + \mathcal{F}_{N-1}(u_{N-1}, x_{N-1}) \\ \nabla_u \mathcal{H}_{N-1}(\lambda_N, u_{N-1}, x_{N-1})^T \\ \lambda_N + \nabla_x \varphi(x_N) \end{bmatrix} \tag{A.4}$$

with $\lambda_{N+1} = 0$. The Karush-Kuhn-Tucker (KKT) conditions for the explicitly discretized NMPC problem (A.2) are

$$\mathcal{K}_i(x^*_{i-1}, s^*_i, \lambda^*_{i+1}) = 0, \ \forall i \in \{1, \cdots, N\}, \tag{A.5}$$

with $x^*_0 = \bar{x}_0$ and $\lambda^*_{N+1} = 0$.

A.3.2 Parallel method

It can be observed from (A.3) and (A.4) that the state and costate also enter the KKT conditions linearly, which can also be shown from the constant off-diagonal block entries of the KKT matrix. By applying the Newton's method to the KKT conditions (A.5), we obtain the search direction ΔS associated with the following partitioned KKT matrix (or Jacobian $J(S^k)$):

$$J(S^k) = \left[\begin{array}{c|c|c|c|c}
\begin{array}{cc} 0 & \nabla_u\mathcal{F}_0^k \\ (\nabla_u\mathcal{F}_0^k)^T & \nabla^2_{uu}\mathcal{H}_0^k \end{array} & \begin{array}{l} I \\ 0 \end{array} & & & \\ \hline
\begin{array}{cc} I & 0 \end{array} & J_1^k & \begin{array}{c} 0 \\ I \\ 0 \end{array} & & \\ \hline
 & \begin{array}{ccc} 0 & I & 0 \end{array} & \ddots & \ddots & \\ \hline
 & & \ddots & J_{N-1}^k & \begin{array}{c} 0 \\ I \\ 0 \end{array} \\ \hline
 & & & \begin{array}{ccc} 0 & I & 0 \end{array} & \nabla^2_{xx}\varphi^k
\end{array}\right], \tag{A.6}$$

where the diagonal block J_i^k is given by

$$J_i^k := \begin{bmatrix} \nabla^2_{xx}\mathcal{H}_i^k & (\nabla_x\mathcal{F}_i^k)^T & \nabla^2_{xu}\mathcal{H}_i^k \\ \nabla_x\mathcal{F}_i^k & 0 & \nabla_u\mathcal{F}_i^k \\ \nabla^2_{ux}\mathcal{H}_i^k & (\nabla_u\mathcal{F}_i^k)^T & \nabla^2_{uu}\mathcal{H}_i^k \end{bmatrix}.$$

As can be seen from the KKT matrix (A.6), the constant off-diagonal blocks indicate linear couplings between stages. By comparing the Jacobian (4.14) in Chapter 4 and (A.6) and following Section 4.3, we can derive the backward correction method to calculate the search direction ΔS for the explicitly discretized NMPC problem. Similarly, the parallel method can be derived by approximating the backward correction method.

As indicated by Remark 4.1 in Chapter 4, we note again that explicit discretization will not lead to better computational performance compared with the reverse-time discretization since the diagonal blocks J_i^k are of the same components as that in Chapter 4.

A.4 Sparsity-exploiting Jacobi method for NMPC

We consider discretizing the continuous-time dynamical system by using the forward Euler method. Similarly to (6.3), we consider the following relaxed regularized nonlinear MPC (RR-NMPC) problem:

$$\begin{aligned} \min_{\substack{u_0,\cdots,u_{N-1},\\ x_0,\cdots,x_N}} \quad & \sum_{i=0}^{N-1} \Big(hl_i(u_i,x_i) + h\Phi_i(u_i,x_i) + \frac{\gamma}{2}\|u_i - \tilde{u}_i^*\|^2 \Big) + h\varphi(x_N) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_{i+1} = x_i + hf(u_i,x_i), \quad i \in \{0,\cdots,N-1\}, \end{aligned} \tag{A.7}$$

where Φ_i is the barrier cost function and φ is the terminal cost function.

A.4.1 KKT conditions

Let λ_{i+1} (costate) $\in \mathbb{R}^{n_x}$ be the Lagrange multiplier corresponding to the i-th state equation. For the sake of brevity, we define

$$s_i := (x_i, u_{i-1}, \lambda_i) \text{ and } S := (s_1,\cdots,s_N).$$

Let $\mathcal{H}_i(x_i,u_i,\lambda_{i+1})$ be the Hamiltonian defined by

$$\mathcal{H}_i(x_i,u_i,\lambda_{i+1}) := l_i(u_i,x_i) + \Phi_i(u_i,x_i) - \lambda_{i+1}^T f(u_i,x_i).$$

For $i \in \{1,\cdots,N-1\}$, let $\mathcal{K}_i(x_{i-1},s_i,\lambda_{i+1})$ be defined by

$$\mathcal{K}_i(x_{i-1},s_i,\lambda_{i+1}) := \begin{bmatrix} x_i - x_{i-1} - hf(u_{i-1},x_{i-1}) \\ h\nabla_u\mathcal{H}_{i-1}(x_{i-1},u_{i-1},\lambda_i)^T + \gamma(u_{i-1} - \tilde{u}_{i-1}^*) \\ \lambda_i - \lambda_{i+1} + h\nabla_x\mathcal{H}_i(x_i,u_i,\lambda_{i+1})^T \end{bmatrix}$$

with $x_0 = \bar{x}_0$. Let $\mathcal{K}_N(x_{N-1},s_N,\lambda_{N+1})$ be defined by

$$\mathcal{K}_N(x_{N-1},s_N,\lambda_{N+1}) := \begin{bmatrix} x_N - x_{N-1} - hf(u_{N-1},x_{N-1}) \\ h\nabla_u\mathcal{H}_{N-1}(x_{N-1},u_{N-1},\lambda_N)^T + \gamma(u_{N-1} - \tilde{u}_{N-1}^*) \\ \lambda_N + h\nabla_x\varphi(x_N) = 0 \end{bmatrix}$$

with $\lambda_{N+1} = 0$. The KKT conditions for the RR-NMPC problem (A.7) are

$$\mathcal{K}_i(x^*_{i-1}, s^*_i, \lambda^*_{i+1}) = 0, \; i \in \{1, \cdots, N\}. \tag{A.8}$$

A.4.2 Newton's method

In solving the KKT conditions (A.8) by using the Newton's method, the search direction $\Delta S^k := (\Delta s^k_1, \cdots, \Delta s^k_N)$ is obtained by solving the following KKT system:

$$\begin{bmatrix} \ddots & & & & \\ \cdots & D^k_{i-1} & M^k_{U_{i-1}} & & \\ & M^k_{L_i} & D^k_i & M^k_{U_i} & \\ & & M^k_{L_{i+1}} & D^k_{i+1} & \cdots \\ & & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ \Delta s^k_{i-1} \\ \Delta s^k_i \\ \Delta s^k_{i+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathcal{K}^k_{i-1} \\ \mathcal{K}^k_i \\ \mathcal{K}^k_{i+1} \\ \vdots \end{bmatrix}. \tag{A.9}$$

Here, $D^k_i := \nabla_{s_i}\mathcal{K}^k_i$, $M^k_{U_i} := \nabla_{s_{i+1}}\mathcal{K}^k_i$, and $M^k_{L_i} := \nabla_{s_{i-1}}\mathcal{K}^k_i$. The expressions of D^k_i are

$$D^k_i = \begin{bmatrix} I & -h\nabla_u f^k_i & 0 \\ 0 & h\nabla^2_{uu}\mathcal{H}^k_{i-1} + \gamma I & -h(\nabla_u f^k_i)^T \\ h\nabla^2_{xx}\mathcal{H}^k_i & 0 & I \end{bmatrix}, \; i \in \{1, \cdots, N-1\}, \tag{A.10}$$

and

$$D^k_N = \begin{bmatrix} I & -h\nabla_u f^k_N & 0 \\ 0 & h\nabla^2_{uu}\mathcal{H}^k_{N-1} + \gamma I & -h(\nabla_u f^k_N)^T \\ h\nabla^2_{xx}\varphi^k & 0 & I \end{bmatrix}. \tag{A.11}$$

Note that unlike (6.6), the off-diagonal block entries of the KKT matrix in (A.9), i.e., $M^k_{U_i}$ and $M^k_{L_i}$, are not constant. The expressions of $M^k_{U_i}$ and $M^k_{L_i}$ are, respectively,

$$M^k_{U_i} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & h\nabla^2_{xu}\mathcal{H}^k_i & -h(\nabla_x f^k_i)^T - I \end{bmatrix}$$

and

$$M^k_{L_i} = \begin{bmatrix} -h\nabla_x f^k_{i-1} - I & 0 & 0 \\ h\nabla^2_{ux}\mathcal{H}^k_{i-1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

A.4.3 Jacobi method for NMPC

Let D^k, L^k, and U^k be the diagonal, strict lower triangular, and strict upper triangular blocks of the KKT matrix in (A.9) as follows.

$$\begin{aligned} D^k &:= \text{block-diag}(D^k_1, \cdots, D^k_N) \\ L^k &:= \text{lower-block-diag}(M^k_{L_2}, \cdots, M^k_{L_N}) \\ U^k &:= \text{upper-block-diag}(M^k_{U_1}, \cdots, M^k_{U_{N-1}}) \end{aligned}$$

Similarly to (6.13) in Chapter 6, the upper-layer Jacobi method for solving the KKT conditions (A.8) is given by

$$S^{k+1} = S^k - \alpha^{\max}(D^k)^{-1}\mathcal{K}^k, \tag{A.12}$$

where $\alpha^{\max} \in (0, 1]$ is a scalar obtained from the fraction-to-the-boundary rule.

Remark A.1. *Since the Jacobi method requires the invertibility of the diagonal blocks and the first diagonal block in* (A.6) *might be singular, the partition of KKT matrix* (A.6) *in the parallel method can not be applied to the Jacobi method.*

We are interested in whether the convergence of the Jacobi method (6.13) in Chapter 6 still holds for the iteration (A.12). We show in the following lemma and theorem that the similar convergence results can be obtained for (A.12).

Lemma A.1. *Let D_i^* be decomposed into*

$$\begin{aligned} D_i^* &= \bar{D}_i^* + h \begin{bmatrix} 0 & -\nabla_u f_i^* & 0 \\ 0 & 0 & -(\nabla_u f_i^*)^T \\ 0 & 0 & 0 \end{bmatrix} \\ &=: \bar{D}_i^* + h\tilde{D}_i^*. \end{aligned} \tag{A.13}$$

Let $\bar{D}^*$ and $\tilde{D}^*$ be defined as follows.

$$\begin{aligned} \bar{D}^* &:= \text{block-diag}(\bar{D}_1^*, \cdots, \bar{D}_N^*) \\ \tilde{D}^* &:= \text{block-diag}(\tilde{D}_1^*, \cdots, \tilde{D}_N^*) \end{aligned}$$

Then, for any $h > 0$ and $\gamma \geq 0$ such that $\bar{D}^*$ is invertible, e.g., when h is sufficiently small and γ is nonzero, the following holds:

$$\rho((\bar{D}^*)^{-1}(L^* + U^*)) = 0.$$

Proof. This lemma can be proved following the proof of Lemma 6.4 in Chapter 6. □

Theorem A.1. *The convergence of the upper-layer Jacobi method* (A.12) *is described as follows.*

(i) Let γ be chosen to be nonzero. There exists $T_0 > 0$ such that $\rho((D^*)^{-1}(L^* + U^*)) < 1$ holds for any $T < T_0$.

(ii) There exists $\epsilon > 0$ such that $\rho((D^*)^{-1}(L^* + U^*)) < 1$ holds for any $\tilde{D}_i^*$ satisfying $\|\tilde{D}_i^*\| < \epsilon\|\bar{D}_i^*\|$, $i \in \{1, \cdots, N\}$.

Proof. This lemma can be proved following the proof of Theorem 6.3 in Chapter 6. □

As can be seen from Theorem A.1, the convergence results of the upper-layer Jacobi method also hold for (A.12). That is, a nonzero regularization parameter γ and a short prediction horizon T guarantee the convergence of the upper-layer Jacobi method (A.12). Likewise, the Jacobi-type iterations, such as the Gauss-Seidel and successive over-relaxation methods, can be applied, and their convergence can be shown as well.

In conclusion, the upper-layer Jacobi method for solving the NMPC problem on the basis of the reverse-time discretization method in Chapter 6 can also be extended to solve the explicitly discretized NMPC problem. Same convergence results and extensions can be obtained. Moreover, since the diagonal blocks D_i^k in (A.10) and (A.11) is simpler than these in (6.15), it takes less time to perform one iteration for (A.12) than (6.13).

References

Alessio, A., & Bemporad, A. (2009). A survey on explicit model predictive control. In L. Magni, D. M. Raimondo, & F. Allgöwer (Eds.), *Nonlinear Model Predictive Control* (pp. 345–369). Springer, Berlin, Heidelberg.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the 1967 Sprint Joint Computer Conference* (pp. 483–485). Atlantic City, USA.

Andersson, J. A., Frasch, J. V., Vukov, M., & Diehl, M. (2017). A condensing algorithm for nonlinear MPC with a quadratic runtime in horizon length. *Optimization Online*, No. 5837.

Banjac, G., Stellato, B., Moehle, N., Goulart, P., Bemporad, A., & Boyd, S. (2017). Embedded code generation using the OSQP solver. In *Proceedings of the 56th IEEE Conference on Decision and Control* (pp. 1906–1911). Melbourne, Australia.

Bemporad, A., Bernardini, D., Long, R., & Verdejo, J. (2018). *Model predictive control of turbocharged gasoline engines for mass production* (Tech. Rep. No. 2018-01-0875). SAE Technical Paper.

Bemporad, A., & Filippi, C. (2003). Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming. *Journal of Optimization Theory and Applications*, *117*(1), pp. 9–38.

Bemporad, A., Morari, M., Dua, V., & Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, *38*(1), pp. 3–20.

Biegler, L. T., & Thierry, D. M. (2018). Large-scale optimization formulations and strategies for nonlinear model predictive control. In *Proceedings of the 6th IFAC Conference on Nonlinear Model Predictive Control* (pp. 1–15). Madison, USA.

Bleris, L. G., Garcia, J., Kothare, M. V., & Arnold, M. G. (2006). Towards embedded model predictive control for system-on-a-chip applications. *Journal of Process Control*, *16*(3), pp. 255–264.

Bock, H. G. (1983). Recent advances in parameter identification techniques for ODE. In P. Deuflhard & E. Hairer (Eds.), *Numerical Treatment of Inverse Problems in Differential and Integral Equations* (pp. 95–121). Birkhäuser Boston.

Bock, H. G., & Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress* (pp. 1603–1608). Budapest, Hungary.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press.

Brentari, M., Bosetti, P., Queinnec, I., & Zaccarian, L. (2018). Benchmark model of Quanser's 3 DOF helicopter. *Rapport LAAS 18040*, hal-01711135.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., & Mansard, N. (2019). The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *Proceedings of the 11th IEEE/SICE International Symposium on System Integration* (pp. 614–619). Paris, France.

Clarke, D. W., Mohtadi, C., & Tuffs, P. (1987). Generalized predictive control – Part I. The basic algorithm. *Automatica*, *23*(2), pp. 137–148.

Cutler, C. R., & Ramaker, B. L. (1980). Dynamic matrix control – A computer control algorithm. In *Proceedings of the 1980 Joint Automatic Control Conference* (p. 72). San Francisco, USA.

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, *5*(1), pp. 46–55.

Del Favero, S., Bruttomesso, D., Di Palma, F., Lanzola, G., Visentin, R., Filippi, A., ... (2014). First use of model predictive control in outpatient wearable artificial pancreas. *Diabetes Care*, *37*(5), pp. 1212–1215.

Diehl, M., Bock, H. G., & Schlöder, J. P. (2005). A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, *43*(5), pp. 1714–1736.

Diehl, M., Ferreau, H. J., & Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In L. Magni, D. M. Raimondo, & F. Allgöwer (Eds.), *Nonlinear Model Predictive Control* (pp. 391–417). Springer, Berlin, Heidelberg.

Domahidi, A., Zgraggen, A. U., Zeilinger, M. N., Morari, M., & Jones, C. N. (2012). Efficient interior point methods for multistage problems arising in receding horizon control. In *Proceedings of the 51th IEEE Conference on Decision and Control* (pp. 668–674). Maui, USA.

Douglas, J., & Rachford, H. H. (1956). On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, *82*(2), pp. 421–439.

Englert, T., Völz, A., Mesmer, F., Rhein, S., & Graichen, K. (2019). A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (GRAMPC). *Optimization and Engineering*, *20*(3), pp. 769–809.

Ferreau, H. J., Bock, H. G., & Diehl, M. (2008). An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, *18*(8), pp. 816–830.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., & Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, *6*(4), pp. 327–363.

Fletcher, R., & Leyffer, S. (2002). Nonlinear programming without a penalty function. *Mathematical Programming*, *91*(2), pp. 239–269.

Frasch, J. V., Sager, S., & Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*, *7*(3), pp. 289–329.

Frison, G., & Jorgensen, J. B. (2013). A fast condensing method for solution of linear-quadratic control problems. In *Proceedings of the 52th IEEE Conference on Decision and Control* (pp. 7715–7720). Florence, Italy.

Garcia, C. E., & Morshedi, A. (1986). Quadratic programming solution of dynamic matrix control (QDMC). *Chemical Engineering Communications*, *46*(1-3), pp. 73–87.

Geyer, T., Torrisi, F. D., & Morari, M. (2008). Optimal complexity reduction of polyhedral piecewise affine systems. *Automatica*, *44*(7), pp. 1728–1740.

Ghadimi, E., Teixeira, A., Shames, I., & Johansson, M. (2014). Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems. *IEEE Transactions on Automatic Control*, *60*(3), pp. 644–658.

Glad, T., & Jonson, H. (1984). A method for state and control constrained linear quadratic control problems. In *Proceedings of the 9th IFAC World Congress* (pp. 1583–1587). Budapest, Hungary.

Goldstein, T., O'Donoghue, B., Setzer, S., & Baraniuk, R. (2014). Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, *7*(3), pp. 1588–1623.

Grancharova, A., & Johansen, T. A. (2012). *Explicit Nonlinear Model Predictive Control: Theory and Applications* (Vol. 429). Springer Science & Business Media.

Hageman, L. A., & Young, D. M. (1981). *Applied Iterative Methods*. Academic Press.

Hashimoto, T., Yoshioka, Y., & Ohtsuka, T. (2013). Receding horizon control with numerical solution for nonlinear parabolic partial differential equations. *IEEE Transactions on Automatic Control*, *58*(3), pp. 725–730.

Hehn, M., & D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation* (pp. 763–770). Shanghai, China.

Houska, B., Ferreau, H. J., & Diehl, M. (2011a). ACADO toolkit – An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, *32*(3), pp. 298–312.

Houska, B., Ferreau, H. J., & Diehl, M. (2011b). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, *47*(10), pp. 2279–2285.

Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., & Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, *59*(12), pp. 3238–3251.

Jørgensen, J. B., Rawlings, J. B., & Jørgensen, S. B. (2004). Numerical methods for large scale moving horizon estimation and control. In *Proceedings of the 7th IFAC Dynamics and Control of Process Systems* (pp. 895–900). Cambridge, USA.

Kalmari, J., Backman, J., & Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering*

Practice, *39*, pp. 56–66.

Kapasouris, P., Athans, M., & Stein, G. (1988). Design of feedback control systems for stable plants with saturating actuators. In *Proceedings of the 27th IEEE Conference on Decision and Control* (pp. 469–479). Austin, USA.

Käpernick, B., & Graichen, K. (2014). The gradient based nonlinear model predictive control software GRAMPC. In *Proceedings of the 13th European Control Conference* (pp. 1170–1175). Strasbourg, France.

Kögel, M., & Findeisen, R. (2011a). A fast gradient method for embedded linear predictive control. *IFAC Proceedings Volumes*, *44*(1), pp. 1362–1367.

Kögel, M., & Findeisen, R. (2011b). Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference* (pp. 501–506). Orlando, USA.

Kouzoupis, D., Ferreau, H. J., Peyrl, H., & Diehl, M. (2015). First-order methods in embedded nonlinear model predictive control. In *Proceedings of the 14th European Control Conference* (pp. 2617–2622). Linz, Austria.

Kouzoupis, D., Quirynen, R., Houska, B., & Diehl, M. (2016). A block based ALADIN scheme for highly parallelizable direct optimal control. In *Proceedings of the 2016 American Control Conference* (pp. 1124–1129). Boston, USA.

Mathworks. (2020). *Nonlinear heat transfer in thin plate.* https://www.mathworks.com/help/pde/ug/nonlinear-heat-transfer-in-a-thin-plate.html.

McLeod, R. M. (1965). Mean value theorems for vector valued functions. *Proceedings of the Edinburgh Mathematical Society*, *14*(3), pp. 197–209.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady* (Vol. 269, pp. 543–547).

Nielsen, I., & Axehill, D. (2015). A parallel structure exploiting factorization algorithm with applications to model predictive control. In *Proceedings of the 54th IEEE Conference on Decision and Control* (pp. 3932–3938). Osaka, Japan.

Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization, Second Edition.* New York: Springer Science and Business Media.

O'Donoghue, B., Stathopoulos, G., & Boyd, S. (2013). A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, *21*(6), pp. 2432–2442.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, *40*(4), pp. 563–574.

Ohtsuka, T. (2015). A tutorial on C/GMRES and automatic code generation for nonlinear model predictive control. In *Proceedings of the 14th European Control Conference* (pp. 73–86). Linz, Austria.

Ordys, A. W., & Clarke, D. W. (1993). A state-space description for GPC controllers. *International Journal of Systems Science*, *24*(9), pp. 1727–1744.

Ortega, J. M., & Rheinboldt, W. C. (1970). *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press.

Ou, Y., & Schuster, E. (2009). Model predictive control of parabolic PDE systems with dirichlet boundary conditions via Galerkin model reduction. In *Proceedings of the 2009 American Control Conference* (pp. 1–7). St. Louis, USA.

Pannocchia, G., Rawlings, J. B., & Wright, S. J. (2007). Fast, large-scale model predictive control by partial enumeration. *Automatica*, *43*(5), pp. 852–860.

Patrinos, P., & Bemporad, A. (2013). An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, *59*(1), pp. 18–33.

Peitz, S., & Klus, S. (2019). Koopman operator-based model reduction for switched-system control of PDEs. *Automatica*, *106*, pp. 184–191.

Pontryagin, L., Boltyanskii, V., Gamkrelidze, R., & Mishchenko, E. (1961). *Mathematical Theory of Optimal Processes.* Fizmatgiz, Moscow.

Pu, Y., Zeilinger, M. N., & Jones, C. N. (2016). Complexity certification of the fast alternating minimization algorithm for linear MPC. *IEEE Transactions on Automatic Control*, *62*(2), pp. 888–893.

Qin, S. J., & Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, *11*(7), pp. 733–764.

Quirynen, R. (2017). *Numerical simulation methods for embedded optimization* (Unpublished doctoral dissertation). Albert-Ludwigs-Universität Freiburg.

Rao, C. V., Wright, S. J., & Rawlings, J. B. (1998). Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, *99*(3), pp. 723–757.

Richter, S., Jones, C. N., & Morari, M. (2009). Real-time input-constrained MPC using fast gradient methods. In *Proceedings of the 48h IEEE Conference on Decision and Control held jointly with the 28th Chinese Control Conference* (pp. 7387–7393).

Richter, S., Morari, M., & Jones, C. N. (2011). Towards computational complexity certification for constrained MPC based on Lagrange relaxation and the fast gradient method. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference* (pp. 5223–5229). Orlando, USA.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems, Second Edition.* SIAM.

Sirovich, L. (1987). Turbulence and the dynamics of coherent structures. Part I: Coherent structures. *Quarterly of Applied Mathematics*, *45*(3), pp. 561–571.

Steinbach, M. C. (1994). A structured interior point SQP method for nonlinear optimal control problems. In *Computational Optimal Control* (pp. 213–222). Birkhäuser Basel.

Stryk, O. V., & Bulirsch, R. (1992). Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, *37*(1), pp. 357–373.

Torrisi, G., Grammatico, S., Smith, R. S., & Morari, M. (2018). A projected gradient and constraint linearization method for nonlinear model predictive control. *SIAM Journal on Control and Optimization*, *56*(3), pp. 1968–1999.

Wächter, A., & Biegler, L. (2006a). Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, *16*(1), pp. 1–31.

Wächter, A., & Biegler, L. T. (2006b). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical*

Programming, *106*(1), pp. 25–57.

Wang, Y., & Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, *18*(2), pp. 267–278.

Wolf, I. J., & Marquardt, W. (2016). Fast NMPC schemes for regulatory and economic NMPC – A review. *Journal of Process Control*, *44*, pp. 162–183.

Wright, S. J. (1996). *Applying new optimization algorithms to more predictive control* (Tech. Rep. No. MCS-P-561-0196; CONF-960178-1 ON: DE96007029). Argonne National Lab., IL (United States).

Xu, F., Chen, H., Gong, X., & Mei, Q. (2016). Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, *63*(1), pp. 310–321.

Yamashita, H. (1998). A globally convergent primal-dual interior point method for constrained optimization. *Optimization Methods and Software*, *10*(2), pp. 443–469.

Yang, X., & Biegler, L. T. (2013). Advanced-multi-step nonlinear model predictive control. *Journal of Process Control*, *23*(8), pp. 1116–1128.

Zanelli, A., Domahidi, A., Jerez, J., & Morari, M. (2020). FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, *93*(1), pp. 13–29.

Zanelli, A., Quirynen, R., Jerez, J., & Diehl, M. (2017). A homotopy-based nonlinear interior-point method for NMPC. In *Proceedings of the 20th IFAC World Congress* (pp. 13188–13193). Toulouse, France.

Zavala, V. M. (2016). New architectures for hierarchical predictive control. *IFAC-PapersOnLine*, *49*(7), pp. 43–48.

List of Publications

Journal articles

1. Deng, H., & Ohtsuka, T. (2019). A parallel Newton-type method for nonlinear model predictive control. *Automatica*, 109, 108560. **Chapter 4**
2. Deng, H., & Ohtsuka, T. ParNMPC – A parallel optimization toolkit for real-time nonlinear model predictive control. *International Journal of Control* (to appear). **Chapter 5**
3. Deng, H., & Ohtsuka, T. A combined first- and second-order approach for model predictive control. *International Journal of Robust and Nonlinear Control* (under review). **Chapter 3**
4. Deng, H., & Ohtsuka, T. A double-layer Jacobi method for PDE-constrained nonlinear model predictive control. *Automatica* (under review). **Chapter 6**

Conference proceedings

1. Deng, H., & Ohtsuka, T. (2018). A highly parallelizable Newton-type method for nonlinear model predictive control. In *Proceedings of the 6th IFAC Conference on Nonlinear Model Predictive Control* (pp. 426–432). Madison, USA. **Chapter 4**
2. Deng, H., & Ohtsuka, T. (2018). A parallel code generation toolkit for nonlinear model predictive control. In *Proceedings of the 57th IEEE Conference on Decision and Control* (pp. 4920–4926). Miami, USA. **Chapter 5**
3. Deng, H., & Ohtsuka, T. (2019). An iterative horizon-splitting method for model predictive control. In *Proceedings of the 58th IEEE Conference on Decision and Control* (pp. 4304–4310). Nice, France. **Chapter 3**