IMT School for Advanced Studies Lucca

Lucca, Italy

Bounded-Variable Least-Squares Methods for Linear and Nonlinear Model Predictive Control

PhD Program in Computer Science and Systems Engineering

XXXI Cycle

By

Nilay Saraf

2019

The dissertation of Nilay Saraf is approved.

Program Coordinator: Prof. Alberto Bemporad,
IMT School for Advanced Studies Lucca, Italy

Supervisor: Prof. Alberto Bemporad,
IMT School for Advanced Studies Lucca, Italy

Co-supervisor: Dr. Daniele Bernardini, ODYS Srl

The dissertation of Nilay Saraf has been reviewed by:

Prof. Eric Kerrigan,
Imperial College London, UK

Prof. Ilya Kolmanovsky,
University of Michigan, Ann Arbor, MI

IMT School for Advanced Studies Lucca

2019

To my parents, my sister, and our pet cockatiel 'chintu'

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to express my deepest gratitude to my advisor Prof. Alberto Bemporad who gave me a dream opportunity to work on my PhD at IMT. His wise mentorship, profound expertise and an inspirational work ethic have all played a cruicial role in achieving the results of this thesis besides making a permanent positive impact on my personal and professional development. I sincerely thank my co-advisor Daniele Bernardini (ODYS) for his time and effort. During this long committment, timely meetings with him and Alberto always granted me the assurance and motivation that I needed to push forward. I also thank Mario Zanon (IMT) for his effort and helpful discussions while we worked together on writing a paper. Next, I would like to sincerely thank Prof. Eric Kerrigan and Prof. Ilya Kolmanovsky for assessing the thesis and providing useful feedback.

Thanks to all current and former members of the DYSCO research unit and study room, I enjoyed a friendly, pleasant and motivating work environment. I thank Vihang Naik, Manas Mejari, Sampath Mulagaleti, Ajay Sampathirao for all the discussions we had, and specially thank Laura Ferrarotti for carefully reviewing all math proofs, besides always being a caring friend. Personally, I would also like to express gratitude to all my friends at IMT or away, oCPS fellows, former teachers, supervisors and colleagues. I thank all staff members at IMT and ODYS for their essential support throughout the last four years. It is a long list of people that I have not mentioned here but they know who they are and they will always be remembered by me. Lastly and most importantly, I would like to thank my parents and sister for always being a

staunch support with unconditional love and care, which has been the cornerstone of all my successes.

I am grateful to the IMT school and the European commission for supporting my PhD studies through the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Marie Skłodowska-Curie grant agreement No. 674875 (oCPS).

Vita

January 21, 1992	Born, Amalner, India
2009-2013	B.E. in Mechanical Engineering Final mark: Distinction University of Mumbai, India
2013-2015	MSc. in Mechanical Engineering Specialization: Control Engineering Delft University of Technology, The Netherlands
10/14-04/15	Research Assistant (*Masterand*) Fraunhofer Institute for Solar Energy Systems, Freiburg, Germany
2015-2019	PhD Program in Computer Science and Systems Engineering Research Unit: Dynamical Systems, Control, and Optimization IMT School for Advanced Studies Lucca, Italy

Publications

1. N. Saraf and A. Bemporad, "Fast model predictive control based on linear input/output models and bounded-variable least squares," in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1919 – 1924.
2. N. Saraf, M. Zanon, and A. Bemporad, "A fast NMPC approach based on bounded-variable nonlinear least squares," in *Proc. 6th IFAC Conference on Nonlinear Model Predictive Control*, Madison, WI, August 2018, pp. 337 – 342.
3. N. Saraf and A. Bemporad, "A bounded-variable least-squares solver based on stable QR updates," *IEEE Transactions on Automatic Control*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8747522
4. N. Saraf and A. Bemporad, "An efficient non-condensed approach for linear and nonlinear model predictive control with bounded variables," in arXiv preprint arXiv:1908.07247, 2019. [Online]. Available: https://arxiv.org/abs/1908.07247

Abstract

This dissertation presents an alternative approach to formulate and solve optimization problems arising in real-time model predictive control (MPC). First it has been shown that by using a quadratic penalty function, the linear MPC optimization problem can be formulated as a least-squares problem subject to bounded variables while directly employing models in their input/output form. A theoretical analysis on stability and optimality is included with a comparison against the conventional *condensed* approach based on linear state-space models. These concepts are straightforwardly extended for fast nonlinear MPC with bounded variables. An active-set algorithm based on a novel application of linear algebra methods is proposed for efficiently solving the resulting box-constrained (nonlinear) least-squares problems with global convergence, numerical robustness, and easy deployability on industrial embedded hardware platforms. Finally, new methods and tools are devised for maximizing efficiency of the solution algorithm considering the numerically sparse structure of the non-condensed MPC problem. Based on these methods, the problem construction phase in MPC design is systematically eliminated by parameterizing the optimization algorithm such that it can adapt to real-time changes in the model and tuning parameters while significantly reducing memory and computational complexity with a potentially self-contained matrix-free implementation. Numerical simulation results included in this thesis testify the potential, applicability, numerical robustness and efficiency of the proposed methods for practical real-time embedded MPC.

Chapter 1

Introduction

1.1 Motivation and research objectives

Model predictive control (MPC) is an advanced control method that is capable of controlling complex systems whose dynamical behaviour may be characterized by means of a mathematical model. Its ability to control multivariable systems while handling constraints has made it one of the most popular methods in advanced control engineering practice with an ever growing range of applications in several industries. MPC evolved over the years from a method developed for controlling slow processes [1, 2] to an advanced multivariable control method that is applicable even to fast-sampling applications, such as in the automotive and aerospace domains [3, 4]. This evolution has been possible because of the significant amount of research on computationally efficient real-time MPC algorithms. For an incomplete list of such efforts and tools the reader is referred to [5–9]. Despite the success of MPC, demand for faster numerical algorithms for a wider scope of applications has been reported for instance in [4]. A common approach to reducing computational load is to solve the MPC problem suboptimally, see for instance [5, 9]. However, even such MPC approaches have limitations that could be prohibitive in some resource-constrained applications, especially in the case of (parameter-varying) nonlinear MPC (NMPC). This denotes that there is

still a large scope of improvement.

This thesis presents an alternative approach to formulate and solve optimization problems arising in real-time model predictive control. The proposed methods aim to stimulate the practical use of MPC in resource constrained applications. They are designed with the motivation to bridge the gap between MPC theory and industrial practice by taking into consideration that: data-based black-box models are often identified as difference equations in input/output (I/O) form; the control variables are often subject to simple bounds; numerically robust algorithms are required for accuracy in limited precision computing which is common in embedded hardware platforms; an easy general deployment needs minimization of calibration requirements; stand-alone code is required for embedded implementation. The main idea studied to meet these needs was to tailor the MPC problem formulation using penalty functions such that fast and simple optimization solvers can be employed. The resulting optimization problems can be solved using box-constrained (nonlinear) least-squares algorithms, which were researched in exhaustive detail.

1.2 Thesis outline and contributions

The thesis is structured into two parts: the first part (Chapters 2-3) focuses on MPC problem formulations whereas the following part (Chapters 4-6) focuses on optimization algorithms and their implementation. We refer to [10] for terminology and basic concepts about MPC. For details about constrained optimization algorithms and relevant terminology used in this thesis we refer to [11]. The content in this thesis is mainly based on the work published in [12–15]. First it has been shown how the linear MPC optimization problem can be formulated as a least-squares problem subject to bounded variables while directly employing models in their I/O form. A theoretical analysis on stability and optimality is included with a comparison against the conventional *condensed* approach based on linear state-space models. These concepts are straightforwardly extended for fast nonlinear MPC. An active-set algorithm based on a novel application of linear algebra methods is pro-

posed for efficiently solving the resulting box-constrained (nonlinear) least squares problems with global convergence, numerical robustness, and easy deployability on industrial embedded hardware platforms. Finally, new methods and tools are devised for maximizing efficiency of the solution algorithm considering the numerically sparse structure of the non-condensed MPC problem. Based on these methods, the problem construction phase in MPC design is systematically eliminated by parameterizing the optimization algorithm such that it can adapt to real-time changes in the model and tuning parameters while significantly reducing memory and computational complexity with a potentially self-contained matrix-free implementation. Numerical simulation results included in this thesis testify the potential, applicability, numerical robustness and efficiency of the proposed methods.

The chapter-wise contribution is described in detail as follows

- Chapter 2, Fast model predictive control based on linear input/output models:

  This chapter introduces a fast and simple model predictive control approach for multivariable discrete-time linear systems described by input/output models subject to bound constraints on inputs and outputs. The proposed method employs a relaxation of the dynamic equality constraints by means of a quadratic penalty function so that the resulting real-time optimization becomes a (sparse), always feasible, bounded-variable least-squares (BVLS) problem. Conditions on the penalty parameter are derived for maintaining closed-loop stability when relaxing the dynamic equality constraints. The approach is not only very simple to formulate, but also leads to a fast way of both *constructing* and *solving* the MPC problem in real time, a feature that is especially attractive when the linear model changes on line, such as when the model is obtained by linearizing a nonlinear model, by evaluating a linear parameter-varying model, or by recursive system identification. A comparison with the conventional state-space based MPC approach is shown in an example, demonstrating the effectiveness of

the proposed method. The content of this chapter is mostly based on [12].

- Chapter 3, Nonlinear model predictive control problem formulations:

  In this chapter, we present an approach for real-time nonlinear model predictive control (NMPC) of constrained multivariable dynamical systems described by nonlinear difference equations. The NMPC problem is formulated by means of a quadratic penalty function as an always feasible, sparse nonlinear least-squares problem subject to box constraints on the decision variables. Linear time-invariant and linear time-varying model predictive control based on BVLS are special cases of the proposed NMPC framework. The proposed formulation and its benefits are demonstrated through a typical numerical example in simulation. An alternative approach based on the augmented-Lagrangian method is also discussed. It is shown that inspite of fundamental differences with the former approach, the nonlinear non-convex optimization problem in this case as well can be formulated to have exactly the same structure, which is favorable to employ fast solution methods. The content of this chapter includes excerpts from [13] and [15].

- Chapter 4, Bounded-variable least-squares solver:

  In this chapter, a numerically robust solver for least-squares problems with bounded variables is presented for applications including, but not limited to, model predictive control. The proposed BVLS algorithm solves the problem efficiently by employing a recursive QR factorization method based on Gram-Schmidt orthogonalization. A reorthogonalization procedure that iteratively refines the QR factors provides numerical robustness for the described primal active-set method, which solves a system of linear equations in each of its iteration via recursive updates. The performance of the proposed BVLS solver, which is implemented in C without external software libraries, is compared in terms of computational efficiency against state-of-the-art quadratic programming

solvers for small to medium-sized random BVLS problems and a typical example of embedded linear MPC application. The numerical tests demonstrate that the solver performs very well even when solving ill-conditioned problems in single precision floating-point arithmetic. Preliminary results in a hardware-in-the-loop setting based on BVLS-based MPC of a mildly nonlinear system with the proposed optimization solver embedded on a programmable logic controller are included, which demonstrate the success of the methods in practically addressing key issues such as numerical robustness. This chapter's content is mainly based on [14].

- Chapter 5, Bounded-variable nonlinear least squares:

  In order to efficiently solve the NMPC problems described in Chapter 3, it is desirable to have a solution method that benefits from warmstarting information, is robust to problem scaling, and exploits structure of the problem. This chapter, which is based on excerpts from [13] and [15], presents an efficient solution algorithm which has the aforementioned advantageous attributes. The proposed solver is a primal feasible line-search method which solves a sequence of BVLS problems until convergence. It can also be seen as an extension of the Gauss-Newton method to handle box constraints. A theoretical analysis of its global convergence property is included. Numerical results based on a typical NMPC example show that the proposed solver is computationally more efficient than the considered benchmarks by a considerable margin.

- Chapter 6, Methods and tools for efficient non-condensed model predictive control:

  This chapter presents a new approach to solving linear and nonlinear model predictive control (MPC) problems that requires minimal memory footprint and throughput and is particularly suitable when the model and/or controller parameters change at runtime. Typically MPC requires two phases: 1) construct an optimization problem based on the given MPC parameters (prediction model, tuning weights, prediction horizon, and constraints), which results

in a quadratic or nonlinear programming problem, and then 2) call an optimization algorithm to solve the resulting problem. In the proposed approach the problem construction step is systematically eliminated, as in the optimization algorithm problem matrices are expressed in terms of abstract functions of the MPC parameters. Furthermore, when using BVLS, an effective use of these operators allows one to exploit sparsity in matrix factors without conventional sparse linear algebra routines while significantly reducing computations. Parameterizing the optimization algorithms in terms of model and tuning parameters not only makes the controller inherently adaptive to any real-time changes in these parameters, but also obviates code-generation requirements. The versatility of the proposed implementation allows one to have a unifying algorithmic framework based on active-set methods with bounded variables that can cope with linear, nonlinear, and adaptive MPC variants based on a broad class of models. The theoretical and numerical results in this chapter are based on [15].

Concluding remarks that highlight the contributions of this thesis and notes on relevant open problems for future research are included in Chapter 7.

Chapter 2

Fast model predictive control based on linear input/output models

2.1 Introduction

The early formulations of Model Predictive Control (MPC), such as Dynamic Matrix Control (DMC) and Generalized Predictive Control (GPC) were based on linear input/ouput models, such as impulse or step response models and transfer functions [16]. On the other hand, most modern MPC algorithms for multivariable systems are formulated based on state-space models. However, black-box models are often identified from input/output (I/O) data, such as via recursive least squares in an adaptive control setting, and therefore require a state-space realization before they can be used by MPC [17]. When the model changes in real time, for example in the case of linear parameter-varying (LPV) systems, converting the black-box model to state-space form and *constructing* the corresponding quadratic programming (QP) matrices might be computationally demanding, sometimes even more time-consuming than *solving* the QP problem. Moreover, dealing directly with I/O models avoids implementing a state estimator, which also adds some numerical burden

and memory occupancy.

In MPC based on I/O models two main approaches are possible for constructing the QP problem. In the "condensed" approach the output variables are eliminated by substitution, exploiting the linear difference equations of the model. As a result, the optimization vector is restricted to the sequence of input moves and the resulting QP problem is *dense* from a numerical linear algebra perspective. In the non-condensed approach, the output variables are also kept as optimization variables, which results in a larger, but *sparse*, QP problem subject to linear equality and inequality constraints.

In our proposed approach we keep the sparse formulation but also eliminate equality constraints by using a quadratic penalty function that relaxes them. The resulting optimization problem, when only subject to lower and upper bounds on variables, is always feasible. Not only does this approach simplify the resulting optimization problem, but it can be interpreted as an alternative way of softening the output constraints, since the error term in satisfying the output equation can be equivalently treated as a relaxation term of the output constraint.

In fact, in practical MPC algorithms feasibility is commonly guaranteed via softening of output constraints by introducing slack variables [10, Section 13.5], [18, 19]. A disadvantage of this approach is that even though the output variables are only subject to box constraints, with the introduction of slack variable(s) the constraints become general (non-box) inequality constraints. This restricts the class of QP solvers that can be used to solve the optimization problem. Instead, the proposed method is similar to the quadratic penalty method (QPM) with single iteration [11, Section 17.1], which guarantees feasibility of the optimization problem without introducing slack variables, and can be solved by Bounded-Variable Least Squares (BVLS), for which simple and efficient algorithms exist [20–22]. Algorithms for BVLS will be discussed later in Chapter 4.

Results for guaranteeing stability when using I/O models in MPC have existed in the literature for a long time, see, e.g., [23,24]. For the unconstrained case, we will show that an existing stabilizing MPC control-

ler based on an I/O model, such as one obtained in [24], is guaranteed to remain stable in the relaxed BVLS formulation if the penalty on violating the equality constraints is chosen to be sufficiently large.

This chapter is organized as follows. We first introduce the BVLS approach based on multivariable discrete-time linear I/O models without stability considerations in Section 2.2. Infeasibility handling is discussed in Section 2.3, where the performance of the proposed formulation is also compared with the soft-constrained MPC approach. In Section 2.4 we analyze the theoretical optimality and closed-loop stability properties of the BVLS approach. Finally, the practical advantages of the approach are demonstrated in Section 2.5, where the proposed method based on I/O models, which we refer to as the "BVLS approach", is compared on a multivariable application example in terms of speed of execution against the standard MPC approach based on state-space models. Final conclusions are drawn in Section 2.6 on the potential benefits and drawbacks of the proposed method.

Notation. $A \in \mathbb{R}^{m\times n}$ denotes a real matrix with m rows and n columns; $\mathrm{rank}(A)$, $A^\top$, A^{-1} (if A is square) and $A^\dagger$ denote its rank, transpose, inverse (if it exists) and pseudo-inverse, respectively. $\mathbb{R}^m$ denotes the set of real vectors of dimension m. For a vector $a \in \mathbb{R}^m$, $\|a\|_2$ denotes its Euclidean norm, $\|a\|_2^2 = a^\top a$. The notation $|\cdot|$ represents the absolute value. Matrix I denotes the identity matrix, and $\mathbf{0}$ denotes a matrix of all zeros.

2.2 Linear input/output models and problem formulation

2.2.1 Linear prediction model

We refer to the time-invariant input/output model typically used in ARX system identification [25], consisting of a noise-free MIMO ARX model with n_y outputs (vector y) and n_u inputs (vector u) described by the dif-

ference equations

$$y_l(k) = \sum_{i=1}^{n_y} \sum_{j=1}^{n_\mathrm{a}} a_{i,j}^{(l)} y_i(k-j) + \sum_{i=1}^{n_u} \sum_{j=1}^{n_\mathrm{b}} b_{i,j}^{(l)} u_i(k-j) \tag{2.1}$$

where y_l is the l^{th} output and u_l is the l^{th} input, $n_\mathrm{a} = \max(n_{i,j}^{(p)})$, $n_\mathrm{b} = 1 + \max(n_{i,j}^{(z)})$, and $n_{i,j}^{(p)}$, $n_{i,j}^{(z)}$ are the number of poles and zeros, respectively, of the transfer function between the i^{th} output and the j^{th} input for all $i \in \{1, 2, \ldots, n_y\}$, $j \in \{1, 2, \ldots, n_u\}$. The coefficients $a_{i,j}^{(l)}$ denote the dependence of the i^{th} output delayed by j samples and the l^{th} output at time instant k, while $b_{i,j}^{(l)}$ denotes the model coefficient between the i^{th} input delayed by j samples and the l^{th} output at time instant k. Note that (2.1) also includes the case of input delays by simply setting the leading coefficients $b_{i,j}^{(l)}$ equal to zero. In matrix notation, (2.1) can be written as

$$y(k) = \sum_{j=1}^{n_\mathrm{a}} A_j y(k-j) + \sum_{j=1}^{n_\mathrm{b}} B_j u(k-j) \tag{2.2}$$

where

$$A_j = \begin{bmatrix} a_{1,j}^{(1)} & a_{2,j}^{(1)} & \cdots & a_{n_y,j}^{(1)} \\ a_{1,j}^{(2)} & a_{2,j}^{(2)} & \cdots & a_{n_y,j}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1,j}^{(n_y)} & a_{2,j}^{(n_y)} & \cdots & a_{n_y,j}^{(n_y)} \end{bmatrix} \in \mathbb{R}^{n_y \times n_y}, \forall j \in \{1, 2, \ldots, n_\mathrm{a}\};$$

$$B_j = \begin{bmatrix} b_{1,j}^{(1)} & b_{2,j}^{(1)} & \cdots & b_{n_u,j}^{(1)} \\ b_{1,j}^{(2)} & b_{2,j}^{(2)} & \cdots & b_{n_u,j}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ b_{1,j}^{(n_y)} & b_{2,j}^{(n_y)} & \cdots & b_{n_u,j}^{(n_y)} \end{bmatrix} \in \mathbb{R}^{n_y \times n_u}, \forall j \in \{1, 2, \ldots, n_\mathrm{b}\};$$

$A_j = \mathbf{0}, \forall j > n_\mathrm{a}$, and $B_j = \mathbf{0}, \forall j > n_\mathrm{b}$.

2.2.2 Performance index

We consider a finite prediction horizon of N_p time steps and take $u(k+j-1)$, $y(k+j)$ as the optimization variables, $\forall j \in \{1, 2, \ldots, N_\mathrm{p}\}$.

To possibly reduce computational effort, we consider a control horizon of N_u steps, $N_\mathrm{u} \leq N_\mathrm{p}$, which replaces variables $u(k+N_\mathrm{u}), u(k+N_\mathrm{u}+1), \ldots, u(k+N_\mathrm{p}-1)$ with $u(k+N_\mathrm{u}-1)$. The following convex quadratic cost function is used

$$\begin{aligned}\min_{u(\cdot),y(\cdot)} J(k) = \min_{u(\cdot),y(\cdot)} & \sum_{j=1}^{N_\mathrm{p}} \frac{1}{2}\|W_y(y(k+j)-y_r)\|_2^2 \\ & + \sum_{j=0}^{N_\mathrm{u}-2} \frac{1}{2}\|W_u(u(k+j)-u_r)\|_2^2 \\ & + \frac{1}{2}(N_\mathrm{p}-N_\mathrm{u}+1)\|W_u(u(k+N_\mathrm{u}-1)-u_r)\|_2^2 \end{aligned} \quad (2.3)$$

where $W_y \in \mathbb{R}^{n_y \times n_y}$ and $W_u \in \mathbb{R}^{n_u \times n_u}$ are positive semidefinite tuning weights, and y_r, u_r are the steady-state references for outputs and inputs, respectively. The latter are usually computed by static optimization of higher-level performance objectives. Practically, these references may be altered for offset-free tracking by estimating steady-state offset. Alternatively, to enforce offset-free tracking or penalize input increments, the same cost function that is described later in its compact form in (2.8a), can also accomodate squared weights on input increments $u(k)-u(k-1), \forall k$, which corresponds to having appropriate entries in additional rows augmented to the matrix of weights on the vector of decision variables, which contains all the input variables.

In case only a vector v_r collecting (a subset of) the output references is provided to MPC for tracking, a reference vector u_r for the inputs and $\tilde{y}_r$ for the outputs for which a set-point has not been specified, which is consistent with model (2.2), can be obtained by solving the linear system

$$v_r = F y_r = \underbrace{F \sum_{j=1}^{n_\mathrm{a}} A_j}_{A_r} y_r + \underbrace{F \sum_{j=1}^{n_\mathrm{b}} B_j}_{B_r} u_r$$

with respect to u_r, $\tilde{y}_r$, where F contains rows of the identity matrix I that extract the known references v_r from the full output reference vector. If A_r^v denotes the matrix obtained by collecting the columns of A_r

corresponding to v_r and $\tilde{A}_r$ the matrix collecting the remaining columns corresponding to $\tilde{y}_r$, solving the linear system

$$\begin{bmatrix}\tilde{A}_r & B_r\end{bmatrix}\begin{bmatrix}\tilde{y}_r \\ u_r\end{bmatrix} = (I - A_r^v)v_r$$

provides the required values for y_r and u_r.

2.2.3 Constraints

The prediction model (2.2) defines the following equality constraints on the output variables

$$y(k+l) = \sum_{j=1}^{n_\mathrm{a}} A_j y(k-j+l) + \sum_{j=1}^{n_\mathrm{b}} B_j u(k-j+l), \forall l \in \{1, 2, \ldots, N_\mathrm{p}\}. \tag{2.4}$$

In order to have a sparse formulation as motivated in [9, 26] and avoid substituting variables via (2.4) in the cost function, we keep the dynamic constraints (2.4) in the following implicit form

$$Gz(k) = H\phi(k) = g(k), \tag{2.5}$$

where $G \in \mathbb{R}^{N_\mathrm{p}\cdot n_y \times (N_\mathrm{u}\cdot n_u + N_\mathrm{p}\cdot n_y)}$, $H \in \mathbb{R}^{N_\mathrm{p}\cdot n_y \times (n_\mathrm{a}\cdot n_y + n_\mathrm{b}\cdot n_u - n_u)}$, $\phi \in \mathbb{R}^{n_\mathrm{a}\cdot n_y + n_\mathrm{b}\cdot n_u - n_u}$ denotes the initial condition vector and

$$z(k) = \left[\begin{array}{c} u(k) \\ y(k+1) \\ u(k+1) \\ y(k+2) \\ \vdots \\ u(k+N_\mathrm{u}-1) \\ y(k+N_\mathrm{u}) \\ \hline y(k+N_\mathrm{u}+1) \\ \vdots \\ y(k+N_\mathrm{p}-1) \\ y(k+N_\mathrm{p}) \end{array}\right] \in \mathbb{R}^{(N_\mathrm{u}\cdot n_u + N_\mathrm{p}\cdot n_y)}$$

denotes the vector of decision variables. Based on the order of decision variables in z, the structure of matrix G depends on the values of $n_{\mathrm{a}}, n_{\mathrm{b}}, N_{\mathrm{u}}$ and N_{p} such that in general

$$G = \left[\begin{array}{ccccccccc|ccccc}
-B_1 & I & 0 & 0 & \cdots & & & & & & \cdots & & 0 \\
-B_2 & -A_1 & -B_1 & I & 0 & \cdots & & & & & & \cdots & 0 \\
 & \vdots & & \ddots & & & & \ddots & & & & & \vdots \\
-B_{N_c} & -A_{N_c-1} & -B_{N_c-1} & -A_{N_c-2} & & \cdots & & -B_1 & I & 0 & \cdots & & 0 \\
\hline
-B_{N_c+1} & -A_{N_c} & -B_{N_c} & -A_{N_c-1} & \cdots & -B_3 & -A_2 & \sum_{i=1}^{2} -B_i & -A_1 & I & 0 & \cdots & 0 \\
-B_{N_c+2} & -A_{N_c+1} & \ddots & & \cdots & -B_4 & -A_3 & \sum_{i=1}^{3} -B_i & -A_2 & -A_1 & I & 0 \cdots & 0 \\
 & \vdots & & \ddots & & & \ddots & & & & & \ddots & 0 \\
-B_{N_p} & -A_{N_p-1} & -B_{N_p-1} & -A_{N_p-2} & \cdots & -B_{N_p-N_c+2} & -A_{N_p-N_c+1} & \sum_{i=1}^{N_p-N_c+1} -B_i & -A_{N_p-N_c} & -A_{N_p-N_c-1} & -A_{N_p-N_c-2} & \cdots -A_1 & I
\end{array}\right] \tag{2.6}$$

and

$$g(k) = \underbrace{\left[\begin{array}{cccc|cccc}
A_1 & A_2 & \cdots & A_{n_{\mathrm{a}}} & B_2 & B_3 & \cdots & B_{n_{\mathrm{b}}} \\
A_2 & \cdots & A_{n_{\mathrm{a}}} & \mathbf{0} & B_3 & \cdots & B_{n_{\mathrm{b}}} & \mathbf{0} \\
\vdots & \iddots & \iddots & \vdots & \vdots & \iddots & \iddots & \vdots \\
 & & & \cdot & B_{n_{\mathrm{b}}} & \mathbf{0} & \cdots & \mathbf{0} \\
 & & & \cdot & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\
A_{n_{\mathrm{a}}} & \mathbf{0} & \cdots & \mathbf{0} & \cdot & \cdot & \cdot & \cdot \\
\mathbf{0} & \cdots & \cdots & \mathbf{0} & \cdot & \cdot & \cdot & \cdot \\
\vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\
\mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0}
\end{array}\right]}_{=H} \underbrace{\left[\begin{array}{c}
y(k) \\
y(k-1) \\
\vdots \\
\cdot \\
\hline
y(k-n_{\mathrm{a}}+1) \\
u(k-1) \\
u(k-2) \\
\vdots \\
u(k-n_{\mathrm{b}}+1)
\end{array}\right]}_{=\phi}.$$

For the typical case of $N_{\mathrm{u}} = N_{\mathrm{p}}$, such that $N_{\mathrm{u}} > \tau$ with $\tau = 1 + \max\{n_{\mathrm{a}}, n_{\mathrm{b}}\}$, we have that

$$
G = \begin{bmatrix}
-B_1 & I & \mathbf{0} & & & \cdots & & & \mathbf{0} \\
-B_2 & -A_1 & -B_1 & I & \mathbf{0} & & & \cdots & \mathbf{0} \\
\vdots & & \ddots & & \ddots & & \ddots & & \vdots \\
-B_\tau & -A_{\tau-1} & \cdots & & -B_1 & I & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{0} & & -B_\tau & -A_{\tau-1} & \cdots & -B_1 & I & \ddots & \vdots \\
\vdots & & \ddots & & & \ddots & & \ddots & \vdots \\
\vdots & & \ddots & & & \ddots & -B_\tau & -A_{\tau-1} \cdots & -B_1 \; I \; \mathbf{0} \\
\mathbf{0} & & & \cdots & & & \mathbf{0} & -B_\tau & -A_{\tau-1} \cdots -B_1 \; I
\end{bmatrix}
$$

is not only sparse but also a block-band matrix. Further details on problem sparsity and its exploitation within the optimization algorithm for computational benefits are discussed in Chapter 6.

In addition to the constraints defined above, we want to impose the following box constraints

$$\underline{u}(k+j) \leq u(k+j) \leq \bar{u}(k+j), \forall j \in \{0, 1, \ldots, N_\mathrm{u} - 1\} \tag{2.7a}$$

$$\underline{y}(k+j) \leq y(k+j) \leq \bar{y}(k+j), \forall j \in \{1, 2, \ldots, N_\mathrm{p}\} \tag{2.7b}$$

where we assume $\underline{u}(k) \leq \bar{u}(k)$, $\underline{y}(k) \leq \bar{y}(k)$, and that $\underline{u}(k)$, $\bar{u}(k)$, $\underline{y}(k)$, $\bar{y}(k)$ may also take infinite values.

Bounds on rate of change of variables would result in general inequalities and are thus not included in order to have a simpler optimization problem, which will be discussed in detail in the following section. However, bounds on the first input increment $\Delta u_{\min} \leq u(k) - u(k-1) \leq \Delta u_{\max}$ can be imposed by replacing $\underline{u}(k)$ with $\max\{\underline{u}(k), u(k-1) + \Delta u_{\min}\}$ and $\bar{u}(k)$ with $\min\{\bar{u}(k), u(k-1) + \Delta u_{\max}\}$. The disadvantage of excluding remaining rate constraints can be partially compensated by penalizing the rate of change of variables in the cost function as discussed earlier, which does not alter the type of the optimization problem.

2.2.4 Optimization problem

For receding horizon control, we need to solve the following convex quadratic programming (QP) problem

$$\min_{z(k)} \frac{1}{2}\|W_z(z(k) - z_r)\|_2^2 \tag{2.8a}$$

$$\text{s.t. } Gz(k) - g(k) = 0 \tag{2.8b}$$

$$\underline{z}(k) \leq z(k) \leq \bar{z}(k) \tag{2.8c}$$

at each time step k, where $W_z \in \mathbb{R}^{(N_\mathrm{u}\cdot n_u + N_\mathrm{p}\cdot n_y)\times(N_\mathrm{u}\cdot n_u + N_\mathrm{p}\cdot n_y)}$ is a block diagonal matrix constructed by diagonally stacking the weights on inputs and outputs according to the arrangement of elements in z. Vector z_r contains the steady-state references for the decision variables and $\underline{z}$, $\bar{z}$ denote the lower and upper bounds, respectively, obtained from (2.7).

By using a quadratic penalty function to relax the equality constraints in (2.8), we reformulate problem (2.8) as the following BVLS problem

$$\min_{\underline{z}(k)\leq z(k)\leq \bar{z}(k)} \frac{1}{2}\|W_z(z(k) - z_r)\|_2^2 + \frac{\rho}{2}\|Gz(k) - g(k)\|_2^2$$

or, equivalently,

$$\min_{\underline{z}(k)\leq z(k)\leq \bar{z}(k)} \frac{1}{2}\left\| \begin{bmatrix} W_z \\ \sqrt{\rho}G \end{bmatrix} z(k) - \begin{bmatrix} W_z z_r \\ \sqrt{\rho}g(k) \end{bmatrix} \right\|_2^2 \tag{2.9}$$

where the penalty parameter $\rho > 0$ is a large weight.
The reformulation based on quadratic penalty function is done for the following reasons:

(i) Penalizing the equality constraints makes problem (2.9) always feasible;

(ii) No dual variables need to be optimized to handle the equality constraints;

(iii) No additional slack decision variables are introduced for softening output constraints, which would lead to linear inequalities of general type (cf. Section 2.3.1);

(iv) The BVLS problem (2.9) may be simpler and computationally cheaper to solve than the constrained QP (2.8).

We note that the tuning weights that comprise W_z were only assumed to be positive semidefinite, which could result in a non-strictly convex QP that is harder to solve in general. However, while solving (2.9), the Hessian of the equivalent QP problem remains positive definite as long as the matrix $\begin{bmatrix} W_z \\ \sqrt{\rho}G \end{bmatrix}$ has full rank. Hence, the matrix of tuning weights can be semidefinite as long as the aforementioned condition is satisfied, which is less restrictive than the typical case in which all the weights must be positive for strict convexity of the optimization problem.

Relaxing the equality constraints as in (2.9) also has an engineering justification: As the prediction model (2.1) is only an approximate representation of the real system dynamics, (opportunistic) violations of the linear dynamic model equations will only affect the quality of predictions, depending on the magnitude of the violation. As shown in the next toy example, we can make the violation small enough by appropriately tuning ρ, so that the violation is negligible when problem (2.8) is feasible, and performance is comparable to that of the soft-constrained MPC approach in case of infeasibilities (cf. Section 2.3).

2.2.5 Example

Figure 2.1 shows a SISO Linear Time-Invariant (LTI) system in which the position y of a sliding mass $m = 1.5$ kg is controlled by an external input force F against the action of a spring with stiffness $\kappa = 1.5\ \mathrm{N{\cdot}m^{-1}}$ and a damper with damping coefficient $c = 0.4\ \mathrm{N{\cdot}s{\cdot}m^{-1}}$. The continuous-time model

$$m\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + c\frac{\mathrm{d}y}{\mathrm{d}t} + \kappa y(t) = F(t)$$

can be converted to the following ARX form (2.2) with a sampling time of 0.1 s and zero-order hold on the input

$$y(k+1) = 1.9638y(k) - 0.9737y(k-1) + 0.0033(u(k) + u(k-1)), \quad (2.10)$$

where the input variable $u = F$.

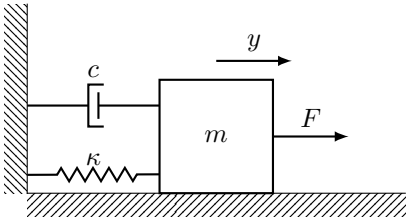


Figure 2.1: Mass-spring-damper system.

For MPC we set $N_\mathrm{p} = 10$, $N_\mathrm{u} = 5$, $W_y = 10$, $W_u = 1$, $\sqrt{\rho} = 10^3$. The maximum magnitude of the input force is 2 N, while the mass is constrained to move within 0.4 m to the right and 0.1 m to the left. The output set-point y_r is 0.2 m to the right which implies that the steady-state input reference u_r is 0.3 N. The initial condition is $y(k) = 0.1$ m, $y(k-1) = 0$ and $u(k-1) = 0$.

Figure 2.2 shows that offset-free tracking is achieved while satisfying the constraints, and that the controller performance is not compromised by relaxing the dynamic constraints. The bottom plot in Figure 2.2 shows that the violation of equality constraints is minimal during the transient and zero at steady-state, when there is no incentive in violating the equality constraints. Finally, Figure 2.3 analyzes the effect of ρ on the resulting error introduced in the model equations.

2.3 Infeasibility handling

Infeasibility may arise while solving (2.8) because output constraints (2.7b) are not satisfiable at a given sample time, due for instance to unexpected disturbances, modeling errors, or to an excessively short prediction horizon N_p. This section investigates the way infeasibility is handled by the BVLS approach as compared to a more standard soft-constraint approach applied to the MPC formulation based on an I/O model.

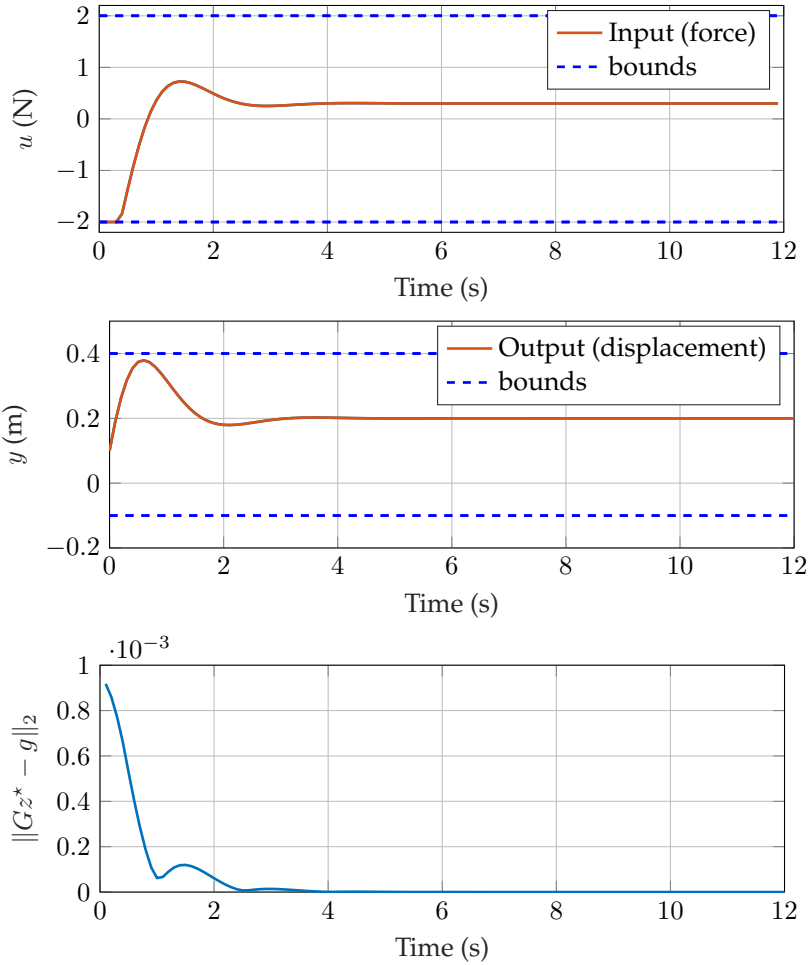


Figure 2.2: Closed-loop simulation: controller performance.

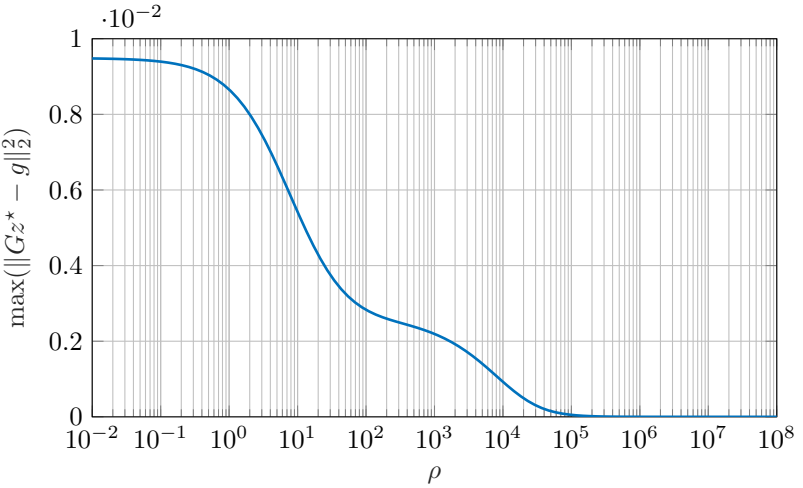


Figure 2.3: Maximum perturbation introduced in the linear dynamics as a function of the penalty ρ.

2.3.1 Soft-constrained MPC

We call the "standard approach" when an exact penalty function is used in the formulation to penalize slack variables, which relaxes the output constraints [10, Sect. 13.5], therefore getting the following QP

$$\min_{u(\cdot),y(\cdot),\epsilon} J(k) + \sigma_1 \cdot \epsilon + \sigma_2 \cdot \epsilon^2 \tag{2.11a}$$

$$\text{s.t. } y(k+l) = \sum_{j=1}^{n_\text{a}} A_j y(k-j+l) + \sum_{j=1}^{n_\text{b}} B_j u(k-j+l), \tag{2.11b}$$

$$\underline{y}(k+l) - \epsilon \underline{V} \leq y(k+l) \leq \bar{y}(k+l) + \epsilon \bar{V}, \forall l \in \{1,2,\ldots,N_\text{p}\}, \tag{2.11c}$$

$$\underline{u}(k+l) \leq u(k+l) \leq \bar{u}(k+l), \forall l \in \{0,1,\ldots,N_\text{u}-1\}, \tag{2.11d}$$

$$\epsilon \geq 0, \tag{2.11e}$$

where ϵ denotes the scalar slack variable, $\underline{V}$ and $\bar{V}$ are vectors with all elements > 0, and $J(k)$ as defined in (2.3). The penalties σ_1 and σ_2 are chosen such that σ_1 is greater than the infinity norm of the vector of optimal Lagrange multipliers of (2.11), and σ_2 is a small penalty included in

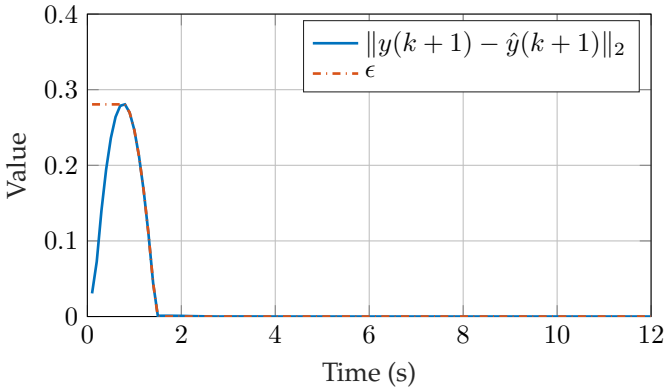


Figure 2.4: Value of slack variable ϵ on solving the soft constrained problem (2.11) and violation of the equality constraint (2.10) at each time step where $\hat{y}$ is obtained from z by solving problem (2.9). $\epsilon > 0$ indicates time instants with output constraint relaxation.

order to have a smooth function. This ensures that the output constraints are relaxed only when no feasible solution exists.

2.3.2 Comparison of BVLS and soft-constrained MPC formulations

The BVLS approach takes a different philosophy in perturbing the MPC problem formulation to handle infeasibility: instead of allowing a violation of output constraints as in (2.11), the linear model (2.2) is perturbed as little as possible to make them satisfiable.

We compare the two formulations (2.9) and (2.11) on the mass-spring-damper system example of Section 2.2.5. In order to test infeasibility handling, harder constraints are imposed such that the problem (2.8) is infeasible, with the same remaining MPC tuning parameters: the maximum input force magnitude is constrained to be 1.2 N and the spring cannot extend more than 0.2 m. Figures 2.4 and 2.5 demonstrate the analogy between the two formulations in handling infeasibility. From Figure 2.4 it is clear that the BVLS approach relaxes the equality constraints

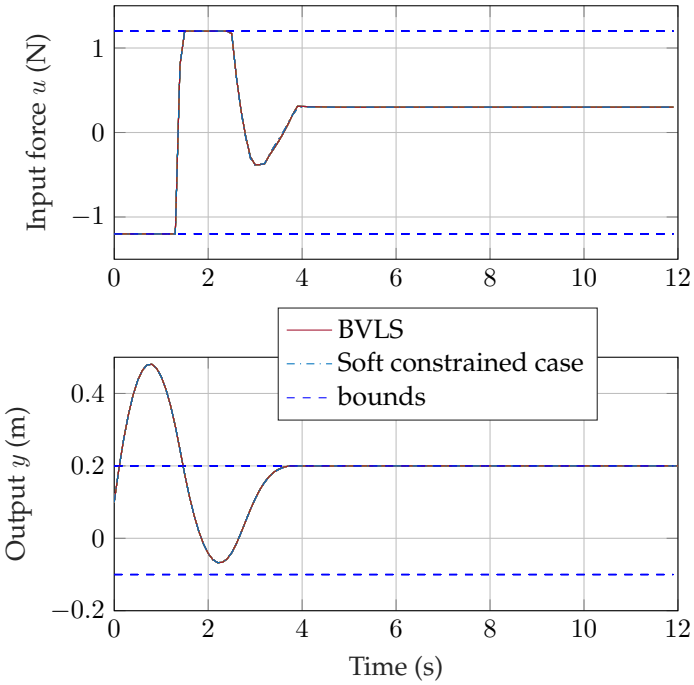


Figure 2.5: Closed-loop simulation of mass-spring-damper system with soft-constrained MPC and BVLS formulations.

only when the problem is infeasible, the same way the soft-constraint approach activates a nonzero slack variable. As a result, even though the two problem formulations are different, for this example the trajectories are almost indistinguishable during times of infeasibility. In order to compare the influence of the relaxations in both cases, we assess the input and output trajectories as shown in Figure 2.5. In general, the input and output trajectories in the two cases might not be identical as this also depends on how much the equality constraints are relaxed in the BVLS case. In the soft constrained case, the equality constraints are strictly satisfied and only the output inequality constraints are relaxed. In the BVLS case, the box constraints are actually strictly satisfied while solving the problem, however, the outputs violate bounds in reality due

to the prediction error caused by relaxation of equality constraints.

2.4 Optimality and stability analysis

We analyze the effects of introducing the quadratic penalty function for softening the dynamic constraints (2.4). First, we explore the analogy between the QP and the BVLS problem formulations described earlier. Then, we derive the conditions for closed-loop stability of the BVLS formulation. For simplicity, we consider a regulation problem without inequality constraints, that is we analyze local stability around zero when $\underline{y}(k) < 0 < \bar{y}(k)$ and $\underline{u}(k) < 0 < \bar{u}(k)$. Problem (2.8) becomes

$$\min_{z} \frac{1}{2}\|W_z z\|_2^2 \tag{2.12a}$$

$$\text{s.t. } Gz - g = 0 \tag{2.12b}$$

(the parentheses indicating the time step have been dropped for simplicity of notation). Note that the QP (2.12) has a unique minimizer, i.e. (2.12) is strictly convex, if W_z has full rank (implying a positive definite Hessian matrix: $W_z^\top W_z > 0$) and G has full row rank (implying linear independence constraint qualification [10]).

By moving the equality constraints (2.12b) in the cost function, we obtain the following unconstrained least-squares problem

$$\min_{z} \quad \frac{1}{2}\left\| \begin{bmatrix} \sqrt{\rho}G \\ W_z \end{bmatrix} z - \begin{bmatrix} \sqrt{\rho}g \\ \mathbf{0} \end{bmatrix} \right\|_2^2 \tag{2.13}$$

The convergence theory of QPM is well established (cf. [11, Section 17.1]), which clarifies that by using larger values of ρ, one can reduce the suboptimality caused due to the relaxation of the equality constraints.

Let $z_\rho^\star$ be the solution of the least-squares problem (2.13), then it can be expressed as

$$z_\rho^\star = (\underbrace{W_z^\top W_z}_{W} + G^\top \rho G)^{-1} G^\top \rho g, \tag{2.14}$$

where the matrix $(W + G^\top \rho G)$ is symmetric positive definite, and hence invertible, because it is the sum of a positive-definite matrix (W) with a

positive-semidefinite matrix $G^\top G$. The expression for $z_\rho^\star$ in (2.14) does not make the influence of ρ on the solution quite apparent. Hence, we next discuss Theorem 2.1, which shows that the solutions of (2.12) and (2.13) coincide when $\rho \to +\infty$, a special case of [11, Theorem 17.1], with a detailed alternative proof that provides an intuitive comparison of the analytical solutions of (2.12) and (2.13).

Theorem 2.1 *Let $z^\star$ and $z_\rho^\star$ denote the solutions of problem (2.12) and (2.13) respectively, with W_z assumed to have full rank and G as defined in (2.6). Then as $\rho \to +\infty$, $z_\rho^\star \to z^\star$.*

Proof: $G \in \underbrace{\mathbb{R}^{N_\mathrm{p} \cdot n_y}}_{m_G} \times \underbrace{(N_\mathrm{u} \cdot n_u + N_\mathrm{p} \cdot n_y)}_{n_G}$ always has lesser number of rows than columns because clearly, $n_G > m_G$. Hence, the equality constraint (2.12b) can be eliminated using the singular value decomposition (SVD)

$$G = U \begin{bmatrix} \Sigma & \mathbf{0} \end{bmatrix} \underbrace{\begin{bmatrix} V_1^\top \\ V_2^\top \end{bmatrix}}_{V^\top},$$

where U and V are orthogonal matrices and hence $V_1^\top V_2 = V_2^\top V_1 = \mathbf{0}$. As defined in (2.6), G has linearly independent rows i.e., G has full rank, which implies that it has m_G non-zero singular values. Hence, the diagonal matrix $\Sigma \in \mathbb{R}^{m_G \times m_G}$, which contains the square root of the non-zero eigen values of $G^\top G$ or $GG^\top$ in its diagonal entries, is invertible. Solving the system of linear equations (2.12b) gives the following linear system

$$U \begin{bmatrix} \Sigma & \mathbf{0} \end{bmatrix} V^\top z = g. \tag{2.15}$$

Let $z = V\nu = V_1\nu_1 + V_2\nu_2$. From (2.15) we get

$$\nu_1 = \Sigma^{-1} U^\top g$$

and problem (2.12) reduces to the unconstrained least-squares problem

$$\min_{\nu_2} \quad \frac{1}{2} \| W_z V_2 \nu_2 - (-W_z V_1 \nu_1) \|_2^2. \tag{2.16}$$

Since $V_2 \in \mathbb{R}^{n_G \times (n_G - m_G)}$ has orthonormal columns, $\operatorname{rank}(V_2) = n_G - m_G$ (full rank). We know that

$$\begin{aligned} &\operatorname{rank}(W_z V_2) \leq \min\left\{\operatorname{rank}(W_z), \operatorname{rank}(V_2)\right\}, \text{ i.e.,} \\ &\operatorname{rank}(W_z V_2) \leq n_G - m_G. \end{aligned} \tag{2.17}$$

From Sylvester's rank inequality,

$$\begin{aligned} \operatorname{rank}(W_z) + \operatorname{rank}(V_2) - n_G \leq \operatorname{rank}(W_z V_2), \text{ i.e.,} \\ n_G - m_G \leq \operatorname{rank}(W_z V_2). \end{aligned} \tag{2.18}$$

Comparing (2.17) and (2.18), we infer that $\operatorname{rank}(W_z V_2) = n_G - m_G$, which implies that $W_z V_2$ has full column rank i.e., $(W_z V_2)^\top W_z V_2 > 0$. Hence, the solution of (2.16) is unique and may be expressed as

$$\begin{aligned} \nu_2^\star &= -[(W_z V_2)^\top W_z V_2)]^{-1} V_2^\top \underbrace{W_z^\top W_z}_{W} V_1 \nu_1 \\ &= -(V_2^\top W V_2)^{-1} V_2^\top W V_1 \Sigma^{-1} U^\top g. \end{aligned}$$

Reconstructing z from ν_1 and ν_2 gives

$$z^\star = [I - V_2 (V_2^\top W V_2)^{-1} V_2^\top W] V_1 \Sigma^{-1} U^\top g. \tag{2.19}$$

Again, using the SVD of G, problem (2.13) can be rewritten as

$$\begin{aligned} &\min_z \quad \frac{1}{2} \left\| \begin{bmatrix} \sqrt{\rho} U \begin{bmatrix} \Sigma & \mathbf{0} \end{bmatrix} V^\top \\ W_z \end{bmatrix} z - \begin{bmatrix} \sqrt{\rho} g \\ \mathbf{0} \end{bmatrix} \right\|_2^2 \\ &= \min_z \quad \frac{1}{2} \left\| \begin{bmatrix} U \begin{bmatrix} \Sigma & \mathbf{0} \end{bmatrix} \\ \frac{1}{\sqrt{\rho}} W_z V \end{bmatrix} V^\top z - \begin{bmatrix} g \\ \mathbf{0} \end{bmatrix} \right\|_2^2, \end{aligned} \tag{2.20}$$

since V is orthogonal. The solution $z_\rho^\star$ of the above least-squares problem (2.20) is computed as follows:

$$V^\top z_\rho^\star = \underbrace{\begin{bmatrix} U\Sigma & \mathbf{0} \\ \frac{W_z}{\sqrt{\rho}} V_1 & \frac{W_z}{\sqrt{\rho}} V_2 \end{bmatrix}^\dagger}_{\Gamma^\dagger} \begin{bmatrix} g \\ \mathbf{0} \end{bmatrix},$$

where $\Gamma^\top\Gamma > 0$ because $W_z V$, and hence Γ, has full column rank, which can easily be proven via the same steps discussed earlier in proving that $W_z V_2$ has full column rank. Based on the fact that V and U are orthogonal, we obtain,

$$
\begin{aligned}
z_\rho^\star &= V\left(\Gamma^\top\Gamma\right)^{-1}\Gamma^\top\begin{bmatrix} g \\ \mathbf{0}\end{bmatrix} \\
&= V\underbrace{\left[\begin{array}{c|c} \Sigma^2 + V_1^\top\frac{W}{\rho}V_1 & V_1^\top\frac{W}{\rho}V_2 \\ \hline V_2^\top\frac{W}{\rho}V_1 & V_2^\top\frac{W}{\rho}V_2\end{array}\right]^{-1}}_{\left[\begin{array}{c|c} K & L \\ \hline M & N\end{array}\right]^{-1}}\begin{bmatrix}\Sigma U^\top g \\ \mathbf{0}\end{bmatrix}.
\end{aligned}
$$

Since $\Gamma^\top\Gamma$ is a symmetric positive-definite matrix, from the Schur complement condition for positive definiteness, we have that the matrices N and $(K - LN^{-1}M)$ are both positive definite and thus, invertible. Therefore, using the matrix inversion lemma, we can write

$$
\begin{aligned}
z_\rho^\star &= V\left[\begin{array}{c|c} (K-LN^{-1}M)^{-1} & * \\ \hline -N^{-1}M(K-LN^{-1}M)^{-1} & *\end{array}\right]\begin{bmatrix}\Sigma U^\top g \\ \mathbf{0}\end{bmatrix} \\
&= \begin{bmatrix} V_1 & V_2\end{bmatrix}\begin{bmatrix}(K-LN^{-1}M)^{-1}\Sigma U^\top g \\ -N^{-1}M(K-LN^{-1}M)^{-1}\Sigma U^\top g\end{bmatrix} \\
&= (V_1 - V_2N^{-1}M)(K-LN^{-1}M)^{-1}\Sigma U^\top g
\end{aligned}
$$

$$
\begin{aligned}
\implies z_\rho^\star = (V_1 - V_2(V_2^\top W V_2)^{-1}V_2^\top W V_1)\times \\
\left(\Sigma^2 + \frac{V_1^\top W V_1 - V_1^\top W V_2(V_2^\top W V_2)^{-1}V_2^\top W V_1}{\rho}\right)^{-1}\times \\
\Sigma U^\top g. \quad (2.21)
\end{aligned}
$$

Evaluating the limit $\rho \to +\infty$ on both sides of (2.21) leads to $z_\rho^\star \to [I - V_2(V_2^\top W V_2)^{-1}V_2^\top W]V_1\Sigma^{-1}U^\top g$, i.e., $\lim_{\rho\to+\infty} z_\rho^\star = z^\star$. □

Comparing $z^\star$ from (2.19) with $z_\rho^\star$ in (2.21) shows that a sufficiently large penalty ρ may result in negligible suboptimality. This also explains Figure 2.3 in which it was observed that the suboptimality introduced

by penalizing equality constraints monotonically decreases in magnitude with increase in the penalty parameter ρ.

Considering that increasing ρ influences numerical conditioning of problem (2.9), as is the case when using penalty functions in soft-constrained MPC, its value must be tuned to be not too large, depending on the available computing precision. This issue is discussed further in Section 3.5.3 through a nonlinear system example, which is more challenging as compared to the one discussed earlier in Section 2.2.5. Next Theorem 2.2 proves the existence of a lower bound on the penalty parameter ρ such that, if the MPC controller is stabilizing under dynamic equality constraints, it remains stable under the relaxation via a quadratic penalty function reformulation as in (2.12).

Theorem 2.2 *Consider the regulation problem* (2.12) *and let*

$$\zeta(k+1) = \mathcal{A}\zeta(k) + \mathcal{B}u(k)$$

be a state-space realization of (2.2)*, which we assume to be stabilizable, such that* $\zeta(k) \triangleq [(y^\top(k-n+1)\cdots y^\top(k))\ (u^\top(k-n+1)\cdots u^\top(k-1))]^\top \in \mathbb{R}^{n_\zeta}$ *and* $n = \max(n_\mathrm{a}, n_\mathrm{b}-1), n_\zeta = n\cdot n_y + (n-1)\cdot n_u$*. The receding horizon control law can then be described as*

$$\begin{aligned} u(k) &= \Lambda z^\star(k) \\ &= \underbrace{\Lambda[I - V_2(V_2^\top W V_2)^{-1}V_2^\top W]V_1\Sigma^{-1}U^\top S}_{K}\zeta(k) \end{aligned} \tag{2.22}$$

where $\Lambda = \begin{bmatrix} I & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix} \in \mathbb{R}^{n_u\times(N_\mathrm{u}\cdot n_u + N_\mathrm{p}\cdot n_\zeta)}$*,* $g(k) = S\zeta(k)$ *such that* $S = \begin{bmatrix}\mathcal{A}^\top & \mathbf{0}\end{bmatrix}^\top \in \mathbb{R}^{N_\mathrm{p}\cdot n_\zeta\times n_\zeta}$*, and* $K \in \mathbb{R}^{n_u\times n_\zeta}$ *is the feedback gain. Similarly, for problem* (2.13)*, referring* (2.14)*, the control law is*

$$u_\rho(k) = \Lambda z_\rho^\star(k) = \underbrace{\Lambda(W + G^\top\rho G)^{-1}G^\top\rho S}_{K_\rho}\zeta(k) \tag{2.23}$$

Assuming that the control law (2.22) *is asymptotically stabilizing, there exists a finite value* ρ^* *such that the control law* (2.23) *is also asymptotically stabilizing* $\forall\rho > \rho^\star$*.*

Proof: Let $m \triangleq \max(|\mathrm{eig}(\mathcal{A}+\mathcal{B}K)|)$, where $\mathrm{eig}()$ denotes the set of eigenvalues of its argument. By the asymptotic closed-loop stability property of the control law (2.22) we have that

$$0 \leq m < 1$$

Let $\sigma = \frac{1}{\rho}$. The continuous dependence of the roots of a polynomial on its coefficients implies that the eigenvalues of a matrix depend continuously on its entries. The continuity property of linear, absolute value, and $\max$ functions implies that $\max(|\mathrm{eig}(\mathcal{A}+\mathcal{B}K_{\frac{1}{\sigma}})|)$ is also a continuous function of σ and is equal to m for $\sigma = 0$. Therefore,

$$\forall \gamma > 0 \; \exists \delta > 0 : \; \left|\max(|\mathrm{eig}(\mathcal{A}+\mathcal{B}K_{\frac{1}{\sigma}})|) - m\right| \leq \gamma, \forall 0 \leq \sigma \leq \delta \qquad (2.24)$$

In particular, for any γ such that $0 < \gamma < 1-m$ we have that $\max(|\mathrm{eig}(\mathcal{A}+\mathcal{B}K_{\frac{1}{\sigma}})|) < 1$. Let for example $\gamma = \frac{1-m}{2}$ and define $\rho^* = \frac{1}{\delta}$ for any δ satisfying (2.24). Then for any $\rho > \rho^*$ the corresponding MPC controller is asymptotically stabilizing. □

From Theorem 2.2 we can thus state the theoretical lower bound on the penalty parameter ρ to be the solution of the following optimization problem

$$\begin{aligned} &\min \quad \rho \\ &\text{s.t. } \max(|\mathrm{eig}(\mathcal{A}+\mathcal{B}\Lambda(W+G^\top \rho G)^{-1}G^\top \rho S)|) < 1. \end{aligned}$$

A way to start with an asymptotically stabilizing (non-relaxed) MPC controller is to adopt the approach described in [24]. As proved in [24], including the following terminal constraint

$$\zeta(N_\mathrm{p}+n-1) = 0_\chi = \zeta_r \qquad (2.25)$$

guarantees closed-loop stability, where $\zeta_r = [\underbrace{(y_r^\top \cdots y_r^\top)}_{n \text{ times}} \; \underbrace{(u_r^\top \cdots u_r^\top)}_{n-1 \text{ times}}]^\top \in \mathbb{R}^{n_\zeta}$ and provided that $N_\mathrm{p} \geq n$. For the regulation problem, $0_\chi = \mathbf{0}$. By substituting (2.2) in the above terminal constraint (2.25), $n \cdot n_y$ equality constraints of the form $G_1 z = g_1$ are obtained which can be included in (2.5). Theorem 2.2 allows us to relax

such equality constraints by penalizing their violation and still guarantee closed-loop asymptotic stability, provided that ρ is a sufficiently large penalty as in Theorem 2.2.

2.5 Comparison with state-space model based approach

We compare our BVLS-based approach (2.9) against the conventional condensed QP approach [10] based on a state-space realization of the ARX model and condensed QP problem

$$\min_{\xi} \quad \frac{1}{2}\xi^\top H\xi + f^\top \xi \tag{2.26a}$$

$$\text{s.t.} \quad \Pi\xi \leq \theta \tag{2.26b}$$

where $\xi \in \mathbb{R}^{n_\xi}$ is the vector of decision variables (i.e., predicted inputs and slack variable for soft constraints), $n_\xi = N_\text{p} \cdot n_u + 1, \Pi \in \mathbb{R}^{(2N_\text{p}\cdot(n_u+n_y)+1\times n_\xi)}$, and $\theta \in \mathbb{R}^{2N_\text{p}\cdot(n_u+n_y)+1}$ are such that (2.26b) imposes box constraints on the input and output variables, and non-negativity constraint on the slack variable.

We consider the open-loop unstable discrete-time transfer function and state-space model of the AFTI-F16 aircraft [27] under the settings of the demo `afti16.m` in [28]. The system under consideration has 4 states, 2 inputs and 2 outputs. The tuning parameters are the same for both MPC formulations (2.9) and (2.26a) in order to compare the resulting performances. As the main purpose here is only to compare the BVLS formulation versus the condensed QP approach based on state-space models (QPss), we use MATLAB's interior-point method for QP in `quadprog` to solve the QPss (2.26a), and its box-constrained version to solve [2] BVLS (2.9). A comparison using state-of-the-art QP solvers for the same example is included in Chapter 4, which discusses the proposed

[2] The MPC problems have been formulated and solved in MATLAB R2015b using sparse matrix operations where applicable for both cases in order to compare most efficient implementations. The code has been run on a Macbook Pro 2.6 GHz Intel Core i5 with 8GB RAM.

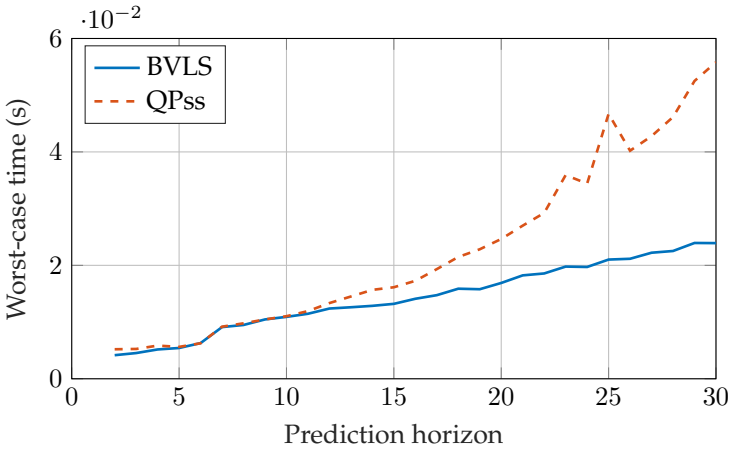


Figure 2.6: Simulation results of the AFTI-F16 aircraft control problem: Worst-case CPU time to solve the BVLS problem based on I/O model and condensed QP (QPss) based on state-space model. The BVLS problem has $4N_\mathrm{p}$ variables with $8N_\mathrm{p}$ constraints whereas the QP problem has $2N_\mathrm{p}+1$ variables with $8N_\mathrm{p}+1$ constraints.

BVLS solver in detail. It is worth noting that for the considered example, due to unstable open-loop dynamics, the Hessian matrix in (2.26a) tends to become ill-conditioned with increase in the prediction horizon and beyond a certain large value it can even cause numerical overflow. This is clearly because the Hessian contains higher powers of the system matrix, therefore scaling its eigen values with the prediction horizon.

Figure 2.6 shows that even though the BVLS problem has almost twice the number of primal variables to be optimized, it is solved faster due to simpler constraints. However, this benefit due to simpler constraints strongly depends on the solution algorithm. This also motivates the research on solution algorithms that best exploit the simplicity of the optimization problem, which is discussed in Chapter 4.

Fewer computations are involved in constructing the BVLS problem (these are online computations in case of linear models that change in real time) as compared to the condensed QP one, as shown in Figure 2.7. This makes the BVLS approach a better option in the LPV setting,

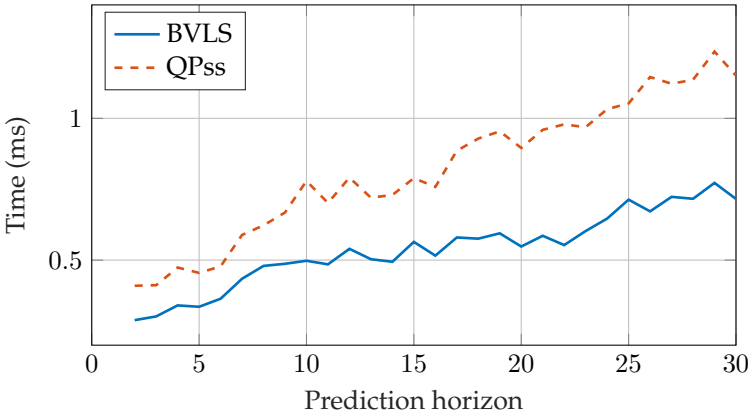


Figure 2.7: Simulation results of the AFTI-F16 aircraft control problem: Comparison of the CPU time required to construct the MPC problems (2.9) and (2.26a) against prediction horizon.

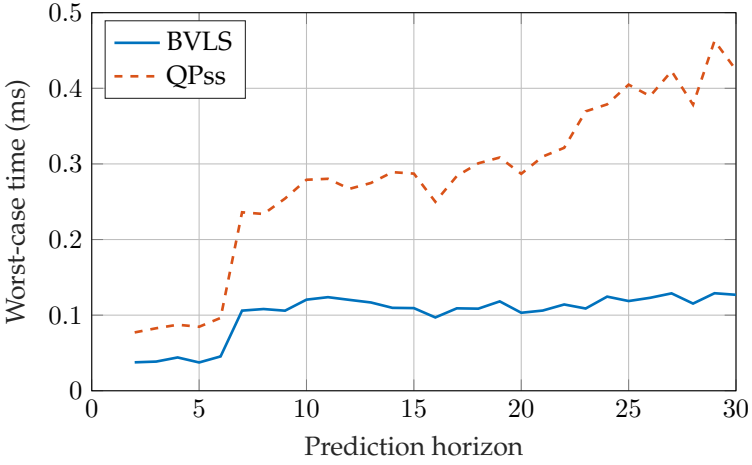


Figure 2.8: Simulation results of the AFTI-F16 aircraft control problem: Comparison of the worst-case CPU time required to update the MPC problem before passing to the solver at each step.

where the problem is constructed on line. Moreover, even in the LTI case one has to update vectors θ, g on line. Figure 2.8 shows that the BVLS approach requires fewer computations for such a type of update. Note also that the computations required for state estimation (including constructing the observer matrices in the LPV case) that is needed by the condensed QP approach have not been taken into account, which would make the BVLS approach even more favorable.

2.6 Conclusions

In this chapter we have proposed an MPC approach based on linear I/O models and BVLS optimization. The obtained results suggest that the BVLS approach may be favorable in both the LTI and adaptive (or LPV) case, and especially for the latter case it may considerably reduce the on-line computations. A potential drawback of the BVLS approach is the risk of numerical ill-conditioning due to the use of large penalty values, an issue that could appear also in soft-constrained MPC formulations. This issue is addressed in Chapter 4 which describes a numerically robust BVLS solver that is well-suited for the considered problems. Furthermore, an implementation that exploits structure of the proposed MPC problem formulation and is efficient in terms of both memory and computations is discussed in Chapter 6.

Chapter 3

Nonlinear model predictive control problem formulations

3.1 Introduction

Nonlinear Model Predictive Control (NMPC) is a popular control strategy which is able to deal with constrained nonlinear systems. However, a common obstacle is the need for solving a nonconvex optimization problem within a stipulated sampling period. A common approach in tackling this issue is developing efficient algorithms tailored to NMPC problems. Often, a suboptimal solution which can be computed fast and efficiently is preferred over a precise one that requires longer computational times [5]. Many different tools have been developed for this purpose, see e.g. [29–35]. In this work instead, we follow a different approach and formulate the NMPC problem in a simple way such that it becomes possible to employ existing fast optimization algorithms.

The quadratic penalty method (QPM) [11] converts a constrained nonlinear optimization problem to a sequence of unconstrained ones, which are solved suboptimally such that the penalty parameter is increased at each instance, until a solution that satisfies termination criteria

is achieved. Using a large enough value of the penalty parameter yields an approximately optimal solution in a single iteration. The need of adequately selecting the penalty parameter to balance accuracy of the solution and ill-conditioning of the problem made this approach not the most appealing for general-purpose solvers. In NMPC, however, small constraints violations are typically negligible compared to model inaccuracy and external perturbations acting on the system. Therefore, the quadratic penalty method is very appealing for such applications due to the possibility of developing efficient implementations. Similar to the linear MPC case, we propose to use a single iteration of the quadratic penalty method which keeps simple bounds on the decision variables as such and relaxes the equality constraints via a quadratic penalty function. The obtained problem is bound-constrained nonlinear least-squares (NLLS), which can be solved by employing the Gauss-Newton method [21] and a Bounded-Variable Least-Squares (BVLS) solver, which is both efficient and numerically robust.

As opposed to standard infeasibility handling approaches which relax the output constraints [19, 30], we relax the equality constraints related to the model of the system. As discussed earlier, the main motivations for such a choice are that 1) it preserves feasibility of the optimization problem, similarly to the output constraint relaxation, and 2) the available model is an approximate representation of the true system.The benefits of the proposed method are demonstrated and summarized with an example that is commonly considered in the NMPC literature. In particular, we show that the constraint violation is not significant unless the original problem becomes infeasible, and therefore, the control performance is not deteriorated.

We first define in Section 3.2 the class of models, perfomance index, and constraints that are typically considered for formulating NMPC problems. Based on that, the benchmark NMPC problem formulations which we consider are described in Section 3.3, whereas the proposed NMPC formulation based on an iteration of the quadratic penalty method is described in Section 3.4, including a brief discussion on an alternative approach based on the augmented Lagrangian method [11].

Numerical results and their discussion are presented in Section 3.5 with concluding remarks in Section 3.6.

Notation. For a vector $a \in \mathbb{R}^m$, its p-norm is $\|a\|_p$, j^{th} element is $a(j)$. The set of integers in the closed interval between a and b is denoted as $[a, b]$. The remaining notation is same as described in Chapter 2.

3.2 Preliminaries

We describe the system dynamics by using the following discrete-time input-output model

$$f(Y_k, U_k, V_k) = \mathbf{0}, \tag{3.1}$$

where we define the inputs and outputs, respectively, at a given discrete-time step k as $U_k = (u_{k-n_\mathrm{b}}, \ldots, u_{k-1})$, $u_k \in \mathbb{R}^{n_u}$, $Y_k = (y_{k-n_\mathrm{a}}, \ldots, y_k)$, $y_k \in \mathbb{R}^{n_y}$. Vector $V_k = (v_{k-n_\mathrm{c}}, \ldots, v_{k-1})$, $v_k \in \mathbb{R}^{n_v}$ defines a measured disturbance and $\mathbf{0}$ represents a zero vector. Function $f : \mathbb{R}^{n_\mathrm{a} n_y} \times \mathbb{R}^{n_\mathrm{b} n_u} \times \mathbb{R}^{n_\mathrm{c} n_v} \to \mathbb{R}^{n_y}$ is in general nonlinear, where $n_\mathrm{a}, n_\mathrm{b}$ and n_c define the model order. We assume f to be differentiable. The class of nonlinear models of the form (3.1) includes, for instance, state-space models and the noise-free polynomial NARX (nonlinear autoregressive exogenous) models [36].

As discussed in the case of linear MPC, we consider a convex quadratic performance index 'P', which is separable in time and a typical choice for regulation and reference tracking in MPC:

$$\begin{aligned}\mathrm{P}(k) = \sum_{j=1}^{N_\mathrm{p}} \frac{1}{2} \|W_y^{\frac{1}{2}} (y_{k+j} - \bar{y}_{k+j})\|_2^2 + \sum_{j=0}^{N_\mathrm{u}-2} \frac{1}{2} \|W_u^{\frac{1}{2}} (u_{k+j} - \bar{u}_{k+j})\|_2^2 \\ + \frac{1}{2}(N_\mathrm{p} - N_\mathrm{u} + 1) \cdot \|W_u^{\frac{1}{2}} (u_{k+N_\mathrm{u}-1} - \bar{u}_{k+N_\mathrm{u}-1})\|_2^2,\end{aligned} \tag{3.2}$$

where N_p and N_u denote the prediction and control horizon respectively. Matrices $W_y \in \mathbb{R}^{n_y \times n_y}$ and $W_u \in \mathbb{R}^{n_u \times n_u}$ are positive semidefinite tuning weights, and $\bar{y}$, $\bar{u}$ denote the references for outputs and inputs, respectively. In general, the proposed NMPC approach described in Section 3.4 is not limited to the above-mentioned performance index. Depending on the control objectives, any cost function that results in a

sum of squares of linear or differentiable nonlinear functions may be considered in order to formulate the NMPC problem.

In order to exploit an efficient solution algorithm, the only inequality constraints that we consider are simple bounds on the optimization variables. General *soft* inequality constraints (3.3) can be converted to the considered setting by introducing slack variables $\nu \in \mathbb{R}^{n_\mathrm{i}}$ having non-negativity constraints such that

$$\begin{aligned} g(w_k) \leq \mathbf{0} \quad &\text{becomes,} \\ g(w_k) + \nu_k = \mathbf{0} \quad &\text{and} \quad \nu_k \geq 0, \end{aligned} \tag{3.3}$$

where $w_k = (u_k, \ldots, u_{k+N_\mathrm{u}-1}, y_{k+1}, \ldots, y_{k+N_\mathrm{p}})$, $z_k = (w_k, \nu_k)$; $g : \mathbb{R}^{(n_z - n_\mathrm{i})} \to \mathbb{R}^{n_\mathrm{i}}$ is assumed to be differentiable, while n_z and n_i denote the number of decision variables and general inequality constraints, respectively. We summarise all NMPC constraints at each time step k as

$$h(z_k, \phi_k) = \mathbf{0} \tag{3.4}$$

$$p_k \leq z_k \leq q_k \tag{3.5}$$

where p_k, q_k are vectors defining bounds on the input and output variables, and non-negativity constraint on the slack variables. Vector $\phi_k = (u_{k-n_\mathrm{b}+1}, \ldots, u_{k-1}, y_k, \ldots, y_{k-n_\mathrm{a}+1})$ denotes the initial condition. Some components of z_k may be unbounded and in that case those bounds are passed to the proposed solver as the largest negative or positive floating-point number in the computing platform. Hence, practically, $p_k, q_k \in \mathbb{R}^{n_z}$. Owing to the construction of equalities (3.4), it is worth noting that the Jacobian matrix of h w.r.t. z evaluated at any given z_k is sparse, with its sparsity pattern depending on the chosen model (3.1).

3.3 Conventional formulations

3.3.1 Constrained NLP

Employing the performance index (3.2) and the constraint set defined by (3.4) and (3.5), the NMPC problem can be defined as the following

constrained NLP

$$\min_{z_k} \frac{1}{2}\|W(z_k - \bar{z})\|_2^2 \tag{3.6a}$$

$$\text{s.t. } h(z_k, \phi_k) = \mathbf{0}, \tag{3.6b}$$

$$p_k \leq z_k \leq q_k, \tag{3.6c}$$

where, $W \in \mathbb{R}^{(n_z - n_i)\times n_i}$ is block-sparse and is constructed from the square root of the tuning weights (W_y, W_u) defined in (3.2), $\bar{z}$ is constructed from the input and output references such that the elements corresponding to slack variables are zero. While one usually avoids the use of slack variables in the definition of the general (nonlinear) inequalities (3.3), we abide by this definition, since it is required by the formulation proposed in Section 3.4.

3.3.2 Soft-constrained NLP

In practice, due to unmodeled dynamics and perturbations acting on the system, the imposed constraints might become impossible to satisfy such that Problem (3.6) becomes infeasible. In order to preserve feasibility, a common approach is to introduce additional slack variable(s) ϵ in the problem to relax the output constraints, which then become *soft constraints*. Details regarding this approach based on exact penalty functions may be referred in [10,18]. For completeness, we provide a possible relaxation of (3.6) using an exact penalty approach.

$$\min_{z_k, \epsilon} \frac{1}{2}\|W(z_k - \bar{z})\|_2^2 + \frac{\sigma_1}{2}\|\epsilon\|_2^2 + \sigma_2\epsilon \tag{3.7a}$$

$$\text{s.t. } h(z_k, \phi_k) = \mathbf{0}, \tag{3.7b}$$

$$p_k - V\epsilon \leq z_k \leq q_k + V\epsilon, \tag{3.7c}$$

$$\epsilon \geq 0, \tag{3.7d}$$

where V is a matrix with all elements non-negative such that only the output constraints are relaxed and the remaining constraints are strictly satisfied. The tuning weights σ_1, σ_2 are such that $\sigma_1 \geq 0$ is a small weight, and $\sigma_2 > 0$ is a large weight which ensures that $\epsilon > 0$ only when

Problem (3.6) becomes infeasible and $\epsilon = 0$ otherwise. Problem (3.7), which we will refer to as 'NLP-soft' can be solved by an NLP solver. In this thesis, however, we propose to use a different reformulation of (3.6) which allows us to use simple and fast optimization algorithms.

3.4 Eliminating equality constraints

Handling equality constraints via penalty functions, or an augmented Lagrangian method, has proven to be effective for efficiently solving constrained optimization problems [11, 37]. This section shows how similar methods can be applied to efficiently solve MPC problems of the form (3.6). In order to employ fast solution methods, the general constrained problem (3.6) can be simplified as a box-constrained nonlinear least-squares (NLLS-box) problem by using a quadratic penalty function and consequently eliminating the equality constraints (3.4) such that (3.6) becomes

$$\min_{p_k \leq z_k \leq q_k} \frac{1}{2} \left\| \begin{array}{c} \frac{1}{\sqrt{\rho}} W_k (z_k - \bar{z}_k) \\ h_k(z_k,\ \phi_k) \end{array} \right\|_2^2 \equiv \min_{p_k \leq z_k \leq q_k} \frac{1}{2} \| r_k(z_k) \|_2^2, \tag{3.8}$$

where the penalty parameter ρ is a positive scalar and $r : \mathbb{R}^{n_z} \to \mathbb{R}^{n_r}$ denotes the vector of residuals. We propose the reformulation (3.8) of problem (3.6) for the following reasons:

1. Penalizing the violation of equality constraints makes problem (3.8) always feasible;

2. While solving (3.8), since we do not include additional slack variables to soften constraints, the function h_k does not need to be analytic beyond bounds, which is discussed in further detail in Section 5.3 (cf. Remark 5.2);

3. No dual variables need to be optimized to handle equality constraints;

4. Problem (3.8) is simpler to solve as compared to (3.6), for instance, when using SQP algorithms (cf. Chapter 5), initializing a feasible

guess is straightforward, the subproblems are feasible even with inconsistent linearizations of h_k, and convergence impeding phenomena such as the Maratos effect [11] are implicitly avoided.

Unlike QPM [11, Framework 17.1], in which a sequence of problems are solved with gradually increasing values of ρ in each iteration, we propose to solve only one problem with a large value of ρ for solving (3.6), owing to the fact that a good initial guess is often available in MPC. It has been proven in Theorem 2.1 that for a quadratic cost (3.6a) subject to only consistent linear equality constraints, a single QPM iteration with sufficiently large penalty ρ may result in negligible suboptimality. This has been clearly demonstrated by numerical examples in Chapter 2 for the linear MPC case and later in Section 3.5 for the general case. A practical upper bound on ρ depends on the computing precision and numerical robustness of the optimization solver such that the Jacobian of the vector of residuals in (3.8) is numerically full-rank. The parameter ρ is tuned based on the fact that a higher value results in lower suboptimality at the cost of problem scaling which may affect the convergence rate of the adopted solution methods. A theoretical lower bound on ρ exists and has been derived in Section 2.4 for the case of LTI systems based on closed-loop stability conditions. The extension of such a result to the general case is not immediate and will not be tackled in this thesis.

An alternative approach to solve the optimization problem (3.6) without the equality constraints (3.4) is the bound-constrained Lagrangian method (BLM) [11, Algorithm 17.4], which can efficiently be solved by iteratively using the nonlinear gradient projection algorithm [11]. At each iteration (i) of the BLM, one solves

$$z_k^{(i+1)} = \arg \min_{p_k \le z_k \le q_k} \frac{1}{2} \left\| \begin{array}{c} W_k(z_k - \bar{z}_k) \\ \sqrt{\rho^{(i)}} h_k(z_k,\ \phi_k) \end{array} \right\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k) \quad (3.9)$$

where Λ denotes the vector of Lagrange multipliers corresponding to the equality constraints, and updates the estimates $\Lambda^{(i)}$ and $\rho^{(i)}$, until convergence (cf. [11, Algorithm 17.4]).

Proposition 3.1 *The optimization problem (3.9) is equivalent to the NLLS-box problem*

$$z_k^{(i+1)} = \arg\min_{p_k \le z_k \le q_k} \frac{1}{2} \left\| \begin{array}{c} \frac{1}{\sqrt{\rho^{(i)}}} W_k(z_k - \bar{z}_k) \\ h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \end{array} \right\|_2^2 \tag{3.10}$$

Proof: We have that problem

$$\arg\min_{p_k \le z_k \le q_k} \frac{1}{2} \left\| \begin{array}{c} W_k(z_k - \bar{z}_k) \\ \sqrt{\rho^{(i)}} h_k(z_k,\ \phi_k) \end{array} \right\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k)$$

$$\text{and } \arg\min_{p_k \le z_k \le q_k} \frac{1}{2} \|W_k(z_k - \bar{z}_k)\|_2^2 + \mathcal{H}(z_k),$$

where $\mathcal{H}(z_k)$

$$\begin{aligned}
&= \frac{\rho^{(i)}}{2} \|h_k(z_k,\ \phi_k)\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k) + \frac{\left\|\Lambda_k^{(i)}\right\|_2^2}{2\rho^{(i)}} \\
&= \frac{\rho^{(i)}}{2} \left(\|h_k(z_k,\ \phi_k)\|_2^2 + \frac{2\Lambda_k^{\top(i)} h_k(z_k, \phi_k)}{\rho^{(i)}} + \left\| \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right\|_2^2 \right) \\
&= \frac{\rho^{(i)}}{2} \left(h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right)^\top \left(h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right) \\
&= \frac{\rho^{(i)}}{2} \left\| h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right\|_2^2, \text{ are equivalent.}
\end{aligned}$$

Scaling by the constant $1/\rho^{(i)}$ yields the result. □

Remark 3.1 *Proposition 3.1 holds for any sum-of-squares cost function with (3.6a) as the special case, for instance $\|\mathcal{S}(z_k)\|_2^2$, where $\mathcal{S}$ is any vector-valued function.*

Proposition 3.1 shows that we can employ the same NLLS-box solvers to solve (3.9), which may be more efficient and numerically robust (cf.

Chapters 5,6) as compared to the use of other NLP solvers. When using BLM, sequences of $z_k^{(i)}$ and $\Lambda_k^{(i)}$ respectively converge to their optimal values $z_k^\star$ and $\Lambda_k^\star$ whereas $h_k(z_k^\star, \phi_k) \approx \mathbf{0}$, numerically. Then via Proposition 3.1, we note that for a fixed value of $\rho \gg \|\Lambda_k^\star\|_\infty$ in the equality-constrained case, we obtain $h_k\left(z_k^{(i+1)}, \phi_k\right) \approx \Lambda_k^{(i+1)}/\rho \approx \mathbf{0}$ [11, Chapter 17], which is simply the solution obtained using a single iteration of QPM for the same ρ and is consistent with the special case described via Theorem 2.1.

Although with BLM it is possible to initialize ρ to aribtrarily low values and solve numerically easier problems, which is its main advantage over QPM, the final value of ρ is not guaranteed to remain low. A main disadvantage of BLM over QPM is that it needs problem (3.6) to be feasible, otherwise the problem must be formulated with *soft constraints* on output variables [18], which typically results in the use of penalty functions with large values of the penalty parameter and non-box inequality constraints, making the problems relatively more difficult to solve. Moreover, even if the feasibility of (3.6) is given, it may take significantly longer to solve multiple instances of (3.9) as compared to a single iteration of QPM with a large penalty, which is more suitable for MPC problems where slight suboptimality may be preferable to a longer computation time. However, in the presence of *hard* general (nonlinear) inequality constraints where QPM might not be applicable, using BLM for feasible problems with the proposed solver and sparsity exploiting methods described in Chapter 6 may be an efficient alternative. BLM is not discussed further as the scope of this paper is limited to MPC problems with box constraints on decision variables.

3.5 Numerical example

In this section, the proposed NMPC formulation is tested for performance in simulations. It is compared against the performance of benchmark NLP formulations (3.6) and (3.7) in terms of quality of control.

3.5.1 Simulation setup

For the purpose of illustration, we consider the Continuous Stirred Tank Reactor (CSTR) [38], which is common in the process industry and is an open-loop unstable system with highly nonlinear dynamics. All simula-

tions have been performed in MATLAB R2015b[3] using the continuous-time model of the CSTR system present in the model predictive control toolbox as a reference for the true system. Discretizing the model equations using the forward Euler method with a sampling period $t_\mathrm{s} = 6$ seconds and substituting the model parameters, the following nonlinear discrete-time prediction model of the form (3.1) is obtained which represents the continuous-time model accurately enough:

$$T_{k+1} = T_k + t_\mathrm{s}(T_{\mathrm{f}_k} - 1.3T_k + \kappa_1 C_{\mathrm{A}_k} e^{\frac{-5963.6}{T_k}} + 0.3T_{\mathrm{j}_k}), \tag{3.11a}$$

$$C_{\mathrm{A}_{k+1}} = C_{\mathrm{A}_k} + t_\mathrm{s}(C_{\mathrm{Af}_k} - \kappa_2 C_{\mathrm{A}_k} e^{\frac{-5963.6}{T_k}} - C_{\mathrm{A}_k}), \tag{3.11b}$$

where constants $\kappa_1 = 416375136, \kappa_2 = 34930800$; T, T_j and T_f respectively denote the temperatures of the reactor, jacket coolant and the feedstream in K; C_A and C_Af respectively denote the concentration of reagent in the reactor and the feedstream in kgmol/m^3. The control objective here is to manipulate the input T_j in order to keep C_A at the desired set-point in the presence of measured disturbances T_f and C_Af. The simulation setup here is kept similar to the CSTR demo in the MPC toolbox of MATLAB. For testing the controller in reference tracking of the concentration C_A, a ramp reference as shown in Figure 3.1 is considered. The corresponding temperature references for the jacket coolant and reactor are computed from (3.11) by assuming the measured disturbances to be constant in prediction. In order to test the controller's performance in regulation, the feedstream temperature is fluctuated as a sinusoid with white measurement noise acting on all temperature states. The constraints[4] that must be satisfied are upper and lower bounds on the variables and input rate. In order to handle the input rate constraints while having only box constraints on variables, only the bounds on the first input increment in predictions are imposed. This is done by replacing the lower bound u_{k_min} by $\max\{u_{k_\mathrm{min}}, u_{k-1} + \Delta u_\mathrm{min}\}$ and the upper bound u_{k_max} by $\min\{u_{k_\mathrm{max}}, u_{k-1} + \Delta u_\mathrm{max}\}$ where u represents the input, Δu its increment with limits indicated by the subscript.

The MPC problems are formulated as described in Sections 3.3-3.4. The NLP problems (3.6), (3.7) are solved using MATLAB's 'fmincon' solver with SQP.

[3]The codes have been run on a Macbook Pro 2.6 GHz Intel Core i5 with 8GB RAM.

[4]The reactor's temperature is constrained in the range of 300-400 K whereas its concentration lies within 0-10 kgmol/m^3. The input is constrained within the range of 240-360 K and its rate is limited within $\pm$2 K/min.

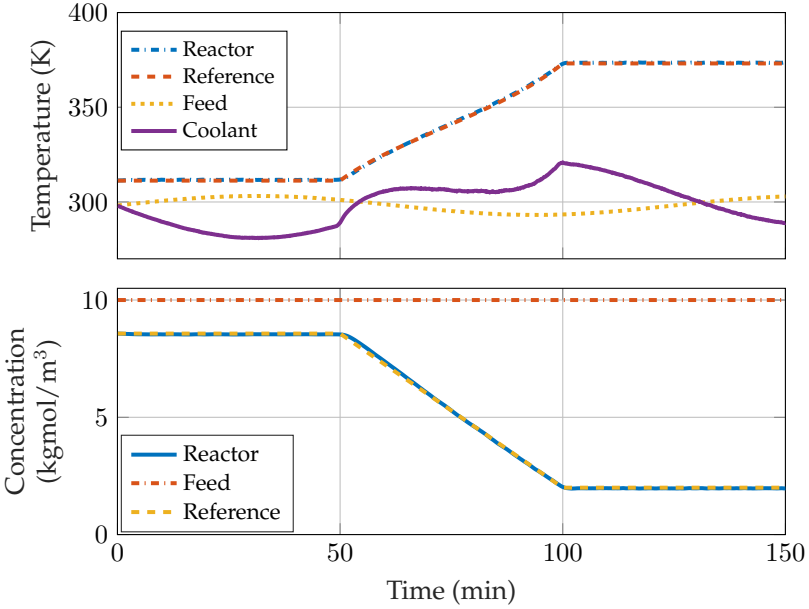


Figure 3.1: Performance in tracking and regulation using formulation (3.8).

3.5.2 Control performance comparison

Figure 3.1 demonstrates the quality of performance in reference tracking and regulation of the proposed NMPC approach. The prediction horizon is set to 10 steps and the control horizon to 1 step. The weight on the coolant's and reactor's temperature tracking error is set to 1 and 10 respectively, whereas the weight on reactor concentration's tracking error is set to 100. The value of the penalty parameter ($\sqrt{\rho}$) in (3.8) is set to 10^4. The solvers are initialized with the feasible initial guess $z = 0.5(p + q)$ and future warmstarts are derived by using the shift-initialization technique [5, 39] such that the next initial guess is a shifted sequence of the previous one and projected in the set of box constraints.
We compare the cost function values achieved with the considered formulations in Figure 3.2. We also observe that the equality constraints are almost strictly satisfied by the proposed NMPC approach even though they were relaxed with a quadratic penalty function. In conclusion, the same quality of performance is achieved as compared to the NLP formulation (3.6) due to a negligible difference between the optimal value of

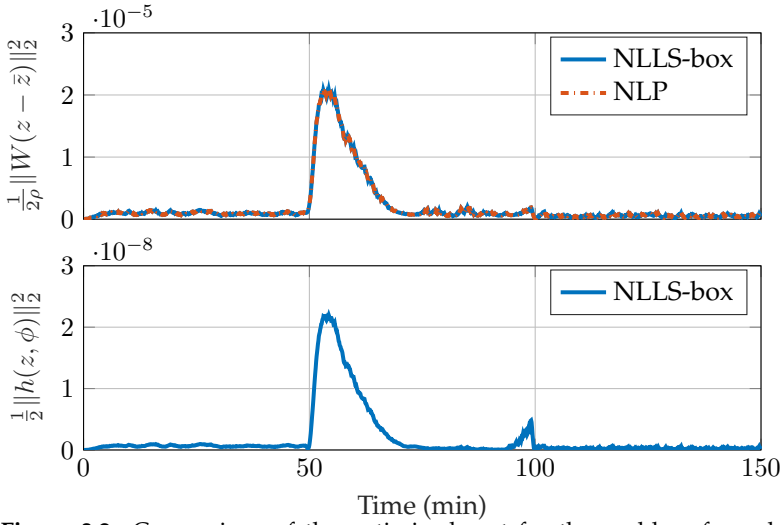


Figure 3.2: Comparison of the optimized cost for the problem formulations (3.6), (3.8).

the cost functions achieved in the two cases.

3.5.3 Choice of the penalty parameter

Recall that the penalty parameter ρ must be tuned to be large enough, such that the model mismatch is within a tolerable limit as observed in the above example. An immediate question that arises is, *what is a large enough value?* Figure 3.3 shows the variation in the nonlinear equality constraint violation w.r.t. $\sqrt{\rho}$ for the considered example. Although we infer that practically there are no strict rules to tune ρ within a broad range, the following main issues must be considered before tuning ρ in order to avoid extreme values:

1. An increase in ρ beyond a certain value raises risk of numerical ill-conditioning and may result in slower convergence of the optimization algorithm, depending on its type. In fact, as observed in Figure 3.3, beyond a certain threshold, instead of a reduction in the nonlinear equality constraint violation, an increase in ρ will increase the constraint violation due to numerical errors. Hence, the numerical robustness of the solution algorithm must be considered

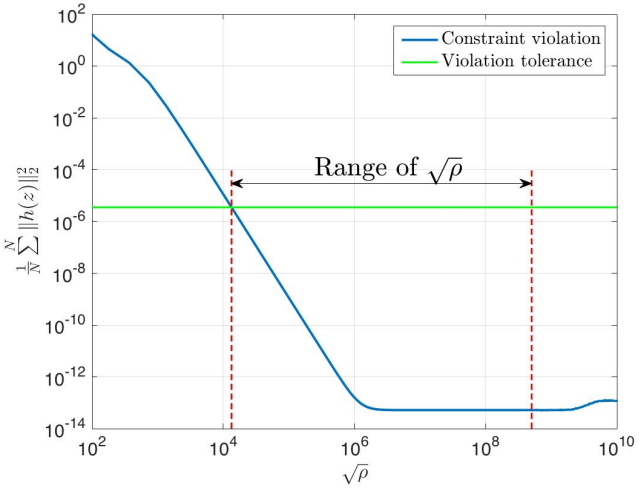


Figure 3.3: Equality constraint violations versus ρ for NMPC of CSTR simulations in double precision floating-point arithmetic[5]. Duration of each simulation was 1500 steps of 6 seconds.

before tuning ρ. If the algorithm is sensitive to problem scaling, the value of ρ must be chosen to suitably trade-off control performance with numerical robustness. For MPC formulations employing penalty functions, it is thus more practical and preferable to use those numerically robust methods that are scantly sensitive to problem scaling, which is an issue addressed in Chapter 4.

2. The floating-point precision of the computing platform is a hard limiting factor for the choice of ρ because a loss of precision is equivalent to a worse numerical conditioning of the problem. Thus, the *largeness* of ρ is relative to the available computing precision as numerical errors become larger in reduced precision even without altering the penalty parameter.

3. In order to have strictly convex QP subproblems, the condition number of the resulting Hessian must be lower than the reciprocal of machine precision. As this condition number is directly scaled

[5]The machine epsilon of the computer used for this simulation was $\approx 2 \times 10^{-16}$ in double precision floating-point arithmetic and $\approx 10^{-7}$ in single precision.

by ρ, a failure of the aforementioned condition makes the occurence of a numerical failure likely, as the Hessian would be numerically semi-definite even though theoretically it may not be the case and upper bounds on numerical error with any arithmetic operations involving the Hessian would be too large.

In Figure 3.3, the NLLS-box problems were solved using the algorithm proposed in Chapter 5 (BVNLLS), based on the Gauss-Newton method, which has linear least-squares subproblems instead of QPs. The Hessian matrix, which has a squared condition number in comparison to the matrix in the equivalent least-squares problem, is neither created nor factorized as will be clarified then. Thus, based on the issues discussed earlier, an SQP algorithm with QP subproblems, such as fmincon, could potentially start suffering from numerical errors for much lower values of ρ, approximately the square root of what was observed in the case with BVNLLS in Figure 3.3. The same would become even worse with a reduced floating-point precision. This topic which is focused on numerical methods will be discussed in further detail in Chapters 4 and 5.

3.5.4 Infeasibility handling performance comparison

Finally, the way in which infeasibility is handled by the formulations (3.7) and (3.8) is compared through a simulation scenario with the CSTR system, such that the reactor temperature reference (373.13 K) exceeds the reduced upper bound (370 K). As shown in Figure 3.4, given the short prediction horizon ($N_p = 10, N_u = 1$), formulation (3.6) becomes inapplicable as the constraints become too strict to satisfy when the reactor temperature almost reaches its limit and the coolant temperature cannot reach fast enough in time to a value that would satisfy the constraints. The slack variable ϵ in (3.7) is only active when (3.6) is infeasible (cf. Figure 3.5). Comparing this against the equality constraints' relaxation in the NLLS-box case, it is clear that the constraint violations occur noticeably and essentially so, only when the NLP problem (3.6) becomes infeasible. Figure 3.4 shows that the trajectories in both cases coincide due to actuator saturation, which leads to the same inputs computed during instances of infeasibility. In general, one might expect different trajectories if the infeasibilities arise when the actuator limits are far. This is because in the NLLS-box case a model mismatch is introduced during times of infeasibility while satisfying bounds, whereas in the other case the output bounds are relaxed while strictly satisfying the

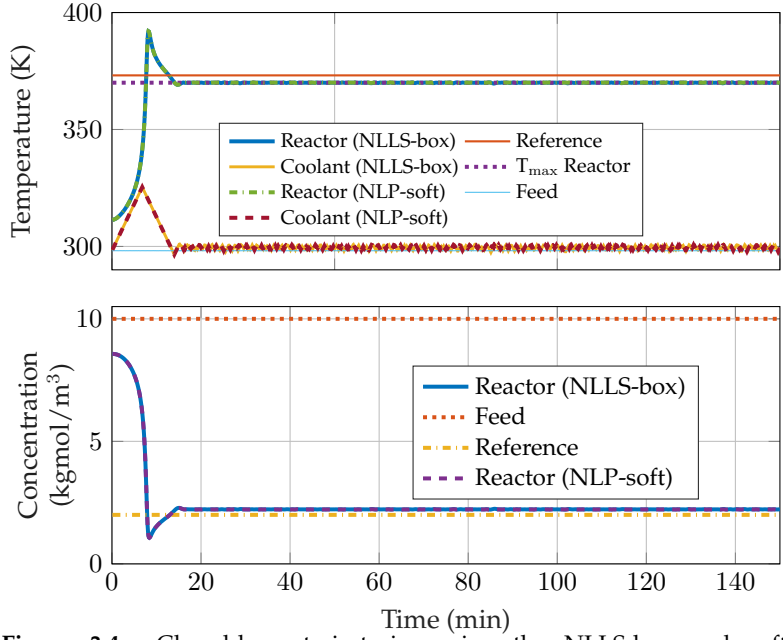


Figure 3.4: Closed-loop trajectories using the NLLS-box and soft-constrained NLP approaches.

equality constraints that represent an approximate future behaviour of the system. This fact is observed in the bottom part of Figure 3.5 where we compare the value of the performance index, which is a function of the predicted sequence of inputs and outputs. The way infeasibility is handled in both cases is similar to the linear MPC case, which was thoroughly discussed in Chapter 2.

3.6 Conclusions

This chapter discussed an NMPC approach based on simple box-constrained nonlinear least-squares formulations, by extending the ideas discussed for linear MPC in the previous chapter. The main idea proposed was to employ a quadratic penalty function, which eliminates the nonlinear equality constraints from the problem and makes it always feasible. It was also shown that by simply completing the squares, the

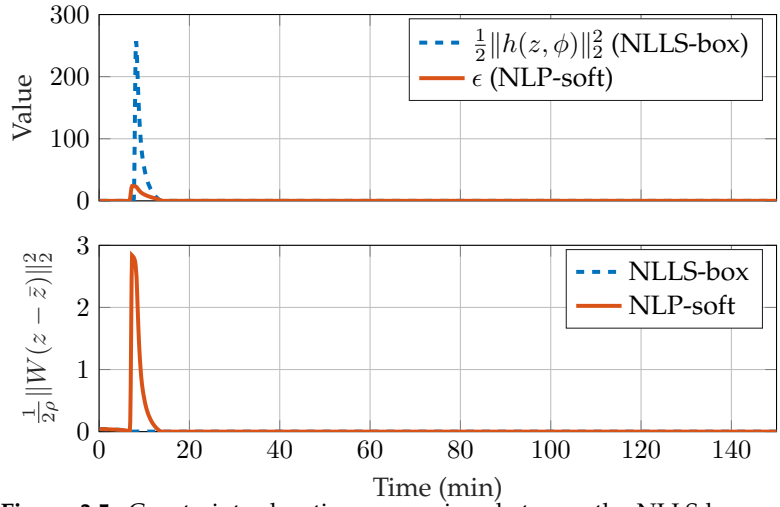


Figure 3.5: Constraint relaxation comparison between the NLLS-box and soft-constrained NLP approaches.

augmented-Lagrangian formulation can also be set to have a nonlinear least-squares cost function. Results suggest that the approach can be an appealing alternative in practical applications where simple and fast optimization solvers are preferable. The following chapters describe solution algorithms in order to efficiently solve MPC problems that are formulated as proposed earlier. In Chapter 6, the NMPC formulation proposed in this chapter is applied in a more general setting based on parameter-varying nonlinear models.

Chapter 4

Bounded-variable least squares solver

4.1 Introduction

Linear least-squares (LS) problems with simple bounds on variables arise in several applications of considerable importance in many areas of engineering and applied science [20]. Several algorithms exist that efficiently solve bounded-variable least-squares (BVLS) or quadratic programming (QP) problems including the ones encountered in model predictive control (MPC) and are based on first-order methods such as Nesterov's fast gradient-projection method [22], the accelerated dual gradient-projection method (GPAD) [40], the alternating direction method of multipliers (ADMM) [41,42]. Because of the penalty functions often used in MPC for practical feasibility guarantee [10,18,19] through constraint relaxations, however, first-order methods require suitable preconditioners due to poor scaling of the problem. This makes active-set methods an attractive alternative for faster solution of small to medium-sized optimization problems arising in embedded MPC, due to their scarce sensitivity to problem scaling. For an overview of state-of-the-art methods in solving QP problems for MPC the reader is referred to the recent papers [7,8].

In primal active-set methods such as BVLS [20] and nonnegative least-squares (NNLS) [43], at each iteration a change in the working active set corresponds to adding or removing a column from the matrix

used to solve an unconstrained linear least-squares (LS) subproblem. By recursively updating the matrix factors, computations are significantly reduced. However, round-off error may accumulate, for example due to imperfect orthogonalization when using QR factorization. The main contribution of this chapter includes methods which aim at overcoming such limitations without compromising numerical robustness through an effective application of stable QR update routines [44] for BVLS.

This work is mainly motivated by the application of BVLS in fast linear and nonlinear MPC as discussed in Chapters 2 and 3, in order to solve optimization problems of the form

$$\min_{l \leq x \leq u} \frac{1}{2}\|f(x)\|_2^2, \text{ s.t. } h(x) = \mathbf{0}, \tag{4.1}$$

where x is the vector of decision variables with bounds l and u, and $h(x) = \mathbf{0}$ represents constraints arising from the prediction of system dynamics and general inequality constraints transformed to equalities using non-negative slack variables. The functions f and h are assumed to be first-order differentiable. Problem (4.1) can be efficiently solved using BVLS through a single iteration of the quadratic penalty method [11], by transforming (4.1) to the following simple form:

$$\min_{l \leq x \leq u} \frac{1}{2}\|f(x)\|_2^2 + \frac{\rho}{2}\|h(x)\|_2^2,$$

where the quadratic penalty parameter $\rho > 0$ is large. The resulting LS subproblems may have a high condition number and require QR update routines that are robust against numerical errors due to potential poor conditioning. The implementation of stable QR update methods described here for BVLS is straightforwardly applicable to the special case of NNLS, an algorithm which can also be applied to solve general QPs for embedded MPC as shown in [45].

This chapter is organized as follows. Section 4.2 briefly describes the baseline BVLS algorithm of [20] and its scope for improvement. The proposed algorithm, which employs robust and recursive QR factorization, is described in Section 4.4. In Section 4.5, a new approach to recursively update the right hand side (RHS) of the triangular system of equations obtained when solving BVLS via the primal active-set method is described while summarizing the stable QR update procedures. Comparison results with competitive solution methods and the performance of the proposed solver in single precision are presented in Section 4.6. Concluding remarks are included in Section 4.7.

Notation. We denote the set of real vectors of dimension m as $\mathbb{R}^m$; a real matrix with m rows and n columns as $A \in \mathbb{R}^{m\times n}$, its transpose as $A^\top$, its inverse as A^{-1} and its j^{th} column as A_j. For a vector $a \in \mathbb{R}^m$, its p-norm is $\|a\|_p$, its j^{th} element is a_j, and $\|a\|_2^2 = a^\top a$. A vector or matrix of appropriate dimension(s) with all its elements zero is represented by $\mathbf{0}$. An identity matrix of appropriate dimensions is denoted by I. An empty matrix is denoted by [ ]. If $\mathcal{F}$ denotes a set of indices, then its cardinality is denoted by $|\mathcal{F}|$ and its j^{th} element as $\mathcal{F}_j$. $A_{\mathcal{F}}$ is a matrix formed from the columns of A corresponding to the indices in the index set $\mathcal{F}$ and $a_{\mathcal{F}}$ forms a vector (or set) with elements of the vector (or set) a as indexed in $\mathcal{F}$.

4.2 Baseline BVLS algorithm

This section recalls the algorithm based on the primal active-set method of [11, Algorithm 16.3] for solving the following bounded variable least-squares problem

$$J(x) = \min_{l\le x\le u} \quad \frac{1}{2}\|Ax - b\|_2^2, \tag{4.2}$$

where $A \in \mathbb{R}^{m\times n}, m \ge n$, $b \in \mathbb{R}^m$, and $l, u \in \mathbb{R}^n$ represent consistent lower and upper bounds respectively on the vector of decision variables $x \in \mathbb{R}^n$. To keep the algorithm description simple, we assume that x has finite bounds, although the algorithm can be easily extended to handle components that have no lower and/or upper bound.

Assumption 4.1 *A is full column rank and $l \le u$.*

For example, Assumption 4.1 is satisfied when (4.2) is obtained from an MPC problem as shown in Chapter 2. Stark and Parker's version[6] of the BVLS algorithm [20], which is outlined via Algorithm 4.1, has two main loops, as is typical of primal active-set methods. The inner loop (Steps 9-18) of Algorithm 4.1 runs until primal feasibility (assessed in Step 10) is achieved in a finite number of iterations and the outer loop (Steps 3-8) runs until the remaining convergence criteria assessed in Steps 5-6 are satisfied. The Lagrange multipliers are simply derived from the gradient vector w and the index ($t^\star$) corresponding to the most negative one is

[6] We refer to the pseudo-code in [20] and their referenced Fortran 77 implementation retrieved from `http://lib.stat.cmu.edu/general/bvls`.

Algorithm 4.1 BVLS Algorithm of Stark and Parker [20]

Inputs: LS matrices A, b; lower bounds l; upper bounds u; initial guess $x \in \mathbb{R}^n$ (warm-start), otherwise $x = l$ (cold-start);

1: $\mathcal{L} \leftarrow \{j | x(j) \leq l(j)\}$;
$\mathcal{U} \leftarrow \{j | x(j) \geq u(j)\}$;
$\mathcal{F} \leftarrow \{j | l(j) < x(j) < u(j)\}$;
if warm-start **then** execute Steps 2 - 4 and then go to Step 9;
2: $x_{\mathcal{F}} \leftarrow (l_{\mathcal{F}} + u_{\mathcal{F}})/2$; $x_{\mathcal{L}} \leftarrow l_{\mathcal{L}}$; $x_{\mathcal{U}} \leftarrow u_{\mathcal{U}}$;
3: $p \leftarrow (b - Ax)$; $w \leftarrow A^{\top} p$;
4: $p \leftarrow p + A_{\mathcal{F}} x_{\mathcal{F}}$;
5: $t^{\star} \leftarrow \min(\arg\max_{t \in \mathcal{L} \cup \mathcal{U}} s_t w_t)$, where $s_t = 1$ if $t \in \mathcal{L}$ and $s_t = -1$ if $t \in \mathcal{U}$; $\tau \leftarrow \max_{t \in \mathcal{L} \cup \mathcal{U}} s_t w_t$;
6: **If** $\mathcal{L} = \mathcal{U} = \emptyset$ **or** $\tau \leq 0$, **then go to** Step 19; (KKT test)
7: $\mathcal{F} \leftarrow \mathcal{F} \cup t^{\star}$; **if** $t^{\star} \in \mathcal{L}$, $\mathcal{L} \leftarrow \mathcal{L} \setminus t^{\star}$ **else** $\mathcal{U} \leftarrow \mathcal{U} \setminus t^{\star}$;
8: $p \leftarrow p + A_{t^{\star}} x_{t^{\star}}$;
9: **If** $\mathcal{F} \neq \emptyset$, **then** $z \leftarrow \arg\min_z \|A_{\mathcal{F}} z - p\|_2^2$;
10: **If** $l_{\mathcal{F}_j} \leq z_j \leq u_{\mathcal{F}_j} \forall j \in \{1, \cdots, |\mathcal{F}|\}$ **or** $\mathcal{F} = \emptyset$, **then** $x_{\mathcal{F}} \leftarrow z$ **and go to** Step 3;
11: $\mathcal{I} \leftarrow \{\mathcal{F}_j | l_{\mathcal{F}_j} > z_j$ or $z_j > u_{\mathcal{F}_j}, j \in \{1, ..., |\mathcal{F}|\}\}$;
12: $\alpha \leftarrow \min \left\{ 1, \min_{j \in \{1, \cdots, |\mathcal{F}|\}, \mathcal{F}_j \in \mathcal{I}} \left\{ \left| \frac{l_{\mathcal{F}_j} - x_{\mathcal{F}_j}}{z_j - x_{\mathcal{F}_j}} \right|, \left| \frac{u_{\mathcal{F}_j} - x_{\mathcal{F}_j}}{z_j - x_{\mathcal{F}_j}} \right| \right\} \right\}$;
13: $x_{\mathcal{F}} \leftarrow x_{\mathcal{F}} + \alpha (z - x_{\mathcal{F}})$;
14: $\mathcal{L}' \leftarrow \left\{ \mathcal{F}_j | x_{\mathcal{F}_j} \leq l_{\mathcal{F}_j}, j \in \{1, \cdots, |\mathcal{F}|\} \right\}$;
15: $\mathcal{U}' \leftarrow \left\{ \mathcal{F}_j | x_{\mathcal{F}_j} \geq u_{\mathcal{F}_j}, j \in \{1, \cdots, |\mathcal{F}|\} \right\}$;
16: $\mathcal{F} \leftarrow \mathcal{F} \setminus (\mathcal{L}' \cup \mathcal{U}')$;
17: $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}'$; $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{U}'$; $p \leftarrow p - A_{\mathcal{L}'} x_{\mathcal{L}'} - A_{\mathcal{U}'} x_{\mathcal{U}'}$;
18: **Go to** Step 9;
19: $x^{\star} \leftarrow x$;
20: **end.**

Outputs: Solution $x^{\star}$ of (4.2), active set of lower and upper bounds $\mathcal{L}$ and $\mathcal{U}$ respectively, set of free variables $\mathcal{F}$.

introduced in the index set of free variables ($\mathcal{F}$) in order to initialize the inner loop. In [20], the unconstrained LS problems in Algorithm 4.1 are solved by computing the QR factorization with column pivoting from scratch at each iteration in order to enforce numerical stability, at the price of an increase in computational effort.

4.3 Solving unconstrained least-squares problems

Due to a sum-of-squares cost function, and constraints in the form of simple bounds, the equality constrained QP subproblems that arise in the primal active-set method [11, Algorithm 16.3] simplify to unconstrained linear least-squares while solving BVLS problems. The LS subproblems can be solved efficiently by decomposing the coefficient matrix through QR factorization, or by forming and solving normal equations through the Cholesky or $\mathrm{LDL}^\top$ factorization, or by the conjugate gradient method [21, 46]. This is due to the fact that the R factor from the QR factorization of a tall matrix A is theoretically equivalent to the Cholesky factor of the corresponding Hessian matrix $A^\top A$ [21]. Even though normal equations could be computationally cheaper if the equivalent QP matrices are provided, it has been proven to be an unstable method for LS problems [47], and QR factorization is preferred in general. This is clear due to the fact that the QP equivalent of LS problem (4.3) is always worse-conditioned [48] because the condition number of the factorized matrix i.e. the Hessian of the QP is squared as compared to that of the coefficient matrix of the LS problem.

In Algorithm 4.1, a change in the working active set in each iteration corresponds to adding or removing a column from the matrix that is factorized in the previous iteration. Hence, it is possible to exploit the information from the previously computed factors in order to update them in a recursive manner [45]. Even though this may significantly reduce computations, it is prone to round-off error accumulation, for instance, due to imperfect orthogonalization of the Q factor in a recursive QR factorization method. Hence, numerically stable update routines are required for applications at the cost of additional computations. The problems of computing and updating matrix factorizations for LS problems have been addressed in exhaustive detail in the literature, for instance in [21, 46, 49] and [43]. There are three main methods to solve LS by

QR factorization: the Householder method, Givens rotation method, and the modified Gram-Schmidt (MGS) orthogonalization procedure [46]. In order to compute the solution of the single LS problem

$$L(x) = \min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 \tag{4.3}$$

with full-rank matrix $A = QR, A \in \mathbb{R}^{m \times n}, m \geq n$, it suffices to compute only the R factor ($R \in \mathbb{R}^{n \times n}$) for which the Householder or Givens method is typically employed and requires $2n^2(m - n/3)$ or $3n^2(m - n/3)$ floating-point operations (*flops*), respectively [46]. For the same purpose, the MGS method needs $2mn^2$ *flops* while it also explicitly computes the Q factor of the "economic" or "thin" QR decomposition $A = QR$ where $Q \in \mathbb{R}^{m \times n}$ and $R \in \mathbb{R}^{n \times n}$. All the three QR methods are numerically stable and provide an equivalent solution provided that orthogonalization errors are avoided, especially in the case of MGS by computing the factors for the augmented system $\begin{bmatrix} A & b \end{bmatrix}$ instead, as described by [47, Theorem 19.2]. The QR update procedures described in [46] when inserting a column in the corresponding matrix, and for the NNLS algorithm of [43], rely on the assumption that the "full" QR factorization is available beforehand. However, for solving a sequence of related LS problems, explicitly computing the first k columns of the full orthogonal matrix factor ($Q \in \mathbb{R}^{m \times m}$) by the Householder method requires an additional $4mnk - 2(m + k)n^2 + (4/3)n^3$ *flops* [46]. This computational burden can be avoided by using instead the more efficient algorithms for stable QR updates described in [44], which is based on Gram-Schmidt (GS) orthogonalization. Not only does this method save the explicit computation of the full Q factor, but it also is effective in terms of memory since at most n columns of Q are stored. Moreover, it provides numerical robustness when the columns of A are nearly dependent, by reorthogonalizing the added column to the orthonormal basis until orthogonality is achieved upto working precision or a tolerance specification. This makes the procedures for QR updates described in [44] very suitable for application in primal active-set algorithms such as BVLS and NNLS.

4.4 Robust BVLS solver based on QR updates

In this section we propose a variant of Algorithm 4.1 that aims at minimizing computations while maintaining numerical stability. The approach is summarized in Algorithm 4.2 and the reader is referred to [20] for an

Algorithm 4.2 Robust BVLS based on recursive QR updates

Inputs: Matrices A, b, l, u; feasibility tolerance $\gamma > 0$.

1: $x \leftarrow (l+u)/2$; $\mathcal{F} \leftarrow \{1, \cdots, n\}$; $\mathcal{U}, \mathcal{L} \leftarrow \emptyset$; $\iota \leftarrow 0$; $t' \leftarrow -1$; $t'' \leftarrow -2$; $t''' \leftarrow -3$; $t^\circ \leftarrow 0$; $p \leftarrow b - A_{(\mathcal{L}\cup\mathcal{U})} x_{(\mathcal{L}\cup\mathcal{U})}$;
2: $\{Q, R, d\} \leftarrow \texttt{gs}(A, p, \mathcal{F})$; **Go to** Step 14;
3: $p \leftarrow b - A_{(\mathcal{L}\cup\mathcal{U})} x_{(\mathcal{L}\cup\mathcal{U})}$;
4: $w \leftarrow p - A_{\mathcal{F}} x_{\mathcal{F}}$; $w_{(\mathcal{L}\cup\mathcal{U})} \leftarrow A_{(\mathcal{L}\cup\mathcal{U})}^\top w$;
5: **If** $(\mathcal{L} \cup \mathcal{U} = \emptyset)$ **or** $(\max(w_{\mathcal{L}}) \leq \gamma$ **and** $\min(w_{\mathcal{U}}) \geq -\gamma)$, **go to** Step 36;
6: $t^\star \leftarrow \min(\arg\max_{t \in \mathcal{L}\cup\mathcal{U}} s_t w_t)$, where $s_t = 1$ if $t \in \mathcal{L}$ and $s_t = -1$ if $t \in \mathcal{U}$;
7: **if** $t''' = t'$ **and** $t'' = t^\star$, **then**
8: **if** $t^\circ = 1$, **go to** Step 36 **end if**;
9: $w_{t^\star}, w_{t'} \leftarrow 0$; $t^\circ \leftarrow 1$; **go to** Step 5
10: **end if**
11: $t''' \leftarrow t''$; $t'' \leftarrow t'$; $t' \leftarrow t^\star$; **if** $t^\star \in \mathcal{L}$, $\iota \leftarrow -1$ **else** $\iota \leftarrow 1$;
12: $\mathcal{F} \leftarrow \mathcal{F} \cup \{t^\star\}$; **if** $t^\star \in \mathcal{L}$, $\mathcal{L} \leftarrow \mathcal{L} \setminus \{t^\star\}$ **else** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{t^\star\}$;
13: $\{Q, R, d\} \leftarrow \texttt{qrinsert}(Q, R, d, p, A_{t^\star}, x_{t^\star}, \mathcal{F}, t^\star)$;
14: **If** $\mathcal{F} \neq \emptyset$, **then** $z \leftarrow$ `solve_triu`(R, d);
15: **if** $(\iota = -1$ **and** $z_j < l_{t^\star})$ **or** $(\iota = 1$ **and** $z_j > u_{t^\star})$, where $\mathcal{F}_j = t^\star$, **then**
16: $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{t^\star}, \mathcal{F}, t^\star)$;
17: **if** $\iota = -1$, $\mathcal{L} \leftarrow \mathcal{L} \cup \{t^\star\}$; **else** $\mathcal{U} \leftarrow \mathcal{U} \cup \{t^\star\}$; **end if**
18: $\mathcal{F} \leftarrow \mathcal{F} \setminus \{t^\star\}$; $w_{t^\star} \leftarrow 0$; **go to** Step 5;
19: **end if**
20: **If** $(l_{\mathcal{F}_j} - \gamma) \leq z_j \leq (u_{\mathcal{F}_j} + \gamma)\ \forall j \in \{1, \cdots, |\mathcal{F}|\}$ **or** $\mathcal{F} = \emptyset$, **then** $x_{\mathcal{F}} \leftarrow z$ **and go to** Step 3;
21: $\iota \leftarrow 0$; $\mathcal{I} \leftarrow \{\mathcal{F}_j | l_{\mathcal{F}_j} > z_j$ or $z_j > u_{\mathcal{F}_j}, j \in \{1, ..., |\mathcal{F}|\}\}$;
22: $\alpha \leftarrow \min\left\{1, \min_{j \in \{1,\cdots,|\mathcal{F}|\}, \mathcal{F}_j \in \mathcal{I}} \left(\left| \frac{l_{\mathcal{F}_j} - x_{\mathcal{F}_j}}{z_j - x_{\mathcal{F}_j}} \right|, \left| \frac{u_{\mathcal{F}_j} - x_{\mathcal{F}_j}}{z_j - x_{\mathcal{F}_j}} \right| \right) \right\}$;
23: $\kappa \leftarrow \arg_j \alpha$; $k \leftarrow |\mathcal{F}|$;

24: **for** $j \in \{1, \cdots, k\}$ **do** $x_{\mathcal{F}_j} \leftarrow x_{\mathcal{F}_j} + \alpha(z_j - x_{\mathcal{F}_j})$;
25: **if** $x_{\mathcal{F}_j} \leq l_{\mathcal{F}_j} + \gamma$ **or** $(j = \kappa$ **and** $z_\kappa < l_{\mathcal{F}_\kappa})$ **then**
26: $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{\mathcal{F}_j}, \mathcal{F}, \mathcal{F}_j)$;
27: $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{F}_j$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_j$;
28: **else**
29: **if** $x_{\mathcal{F}_j} \geq u_{\mathcal{F}_j} - \gamma$ **or** $(j = \kappa$ **and** $z_\kappa > u_{\mathcal{F}_\kappa})$ **then**
30: $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{\mathcal{F}_j}, \mathcal{F}, \mathcal{F}_j)$;
31: $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{F}_j$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_j$;
32: **end if**
33: **end if**
34: **end for**;
35: **Go to** Step 14;
36: **end.**

37: **procedure** $\texttt{gs}(A, p, \mathcal{F})$
38: $Q \leftarrow [\,]$; $R \leftarrow [\,]$;
39: **for** $j \in \mathcal{F}$ **do**
40: $\{q, r, \rho\} \leftarrow \texttt{orthogonalize}(A_j, Q, R)$; (Algorithm 4.3)
41: $Q \leftarrow \begin{bmatrix} Q & q \end{bmatrix}$; $R \leftarrow \begin{bmatrix} R & r \\ \mathbf{0} & \rho \end{bmatrix}$;
42: **end for**
43: $\{\sim, d, \sim\} \leftarrow \texttt{orthogonalize}(p, Q, R)$;
44: **end procedure**

Outputs: Primal solution x of (4.2), Active set of lower and upper bounds $\mathcal{L}$ and $\mathcal{U}$ respectively, set $\mathcal{F}$ of free variables.

---

easy understanding of the main steps. The idea is to employ a different approach for QR factorization and recursive updates that cost only a fraction of the computations required to solve an LS problem from scratch. At initialization, all variables are placed in the free set (Step 1) unless an initial guess is provided. Next, a thin QR factorization (cf. Theorem 4.1 in Section 4.5) is computed in Step 2 (unless provided) by using the Gram-Schmidt orthogonalization procedure `gs`, before subsequent updates in the inner (Steps 14-35) and outer (Steps 3-13) loops through the stable update procedures `qrinsert` (cf. Lemma 4.1 in Section 4.5) and `qrdelete` (cf. Lemma 4.2 in Section 4.5). The unconstrained least-squares problem solved at each iteration is

$$\min_{z} \quad \frac{1}{2}\|A_{\mathcal{F}}z - p\|_2^2, \tag{4.4}$$

where p, the RHS of the linear system, can be computed by the relation in Step 3 of Algorithm 4.2. Using thin QR factorization, $A_{\mathcal{F}} = QR$, solving (4.4) reduces to solving the linear triangular system

$$Rz = Q^\top p = d \tag{4.5}$$

in Step 14 by the back-substitution procedure `solve_triu`. Unlike [20], or approaches in which the RHS of the triangular system (d) to solve the least-squares problem is explicitly computed through $\mathcal{O}(mn)$ operations [46], our approach recursively updates vector d (cf. Propositions 4.1 and 4.2 in Section 4.5) and avoids computing the matrix-vector product $Q^\top p$ in (4.5), at each iteration of the algorithm. In order to avoid introducing orthogonalization errors due to the product with $Q^\top$ in (4.5), vector d is initialized (Step 43) instead from the thin QR factors of the augmented system as described in [47, Ch. 19]. Unlike Algorithm 4.1, in which the whole negative gradient vector is computed, we only compute elements of the negative gradient corresponding to the active set of constraints as the Lagrange multipliers corresponding to the free set are zero by construction. Moreover, Step 4 of Algorithm 4.1 is not executed; by performing Steps 3-4 of Algorithm 4.2 instead, which saves an additional $4m|\mathcal{F}|$ *flops* over Algorithm 4.1 each time the outer loop is executed.

4.4.1 Initialization

Cold-start

Algorithm 4.2 is initialized with all variables in the free set (Step 1) as the initial guess. This is done in order to have a thin QR factorization of

matrix A (Step 2) before subsequent updates. In practice this approach is more likely[7] to work faster than the case in which all variables are initialized at a bound because: a) The outer loop (Steps 3-13) iterations in which the QR factorization is incrementally updated costs more *flops* on an average than the inner loop (Steps 14-35) ones in which the QR factors are reduced in dimension; b) the outer loop iterations involve matrix vector multiplications (Steps 3, 4), which in total comprise the most time-consuming part of the algorithm after Step 2; c) having a thin QR factorization of the whole matrix A ensures that the number of outer loop iterations are minimal. Besides that, the inner loop operations are numerically less sensitive than the outer loop ones and hence make the choice of a full free-set initialization a preferred approach.

Warm-start

Following the warm-starting procedure of Algorithm 4.1, it is possible to warm-start Algorithm 4.2 by initializing the active sets based on the entries in the initial guess for the primal solution. Next, after executing Steps 3, one needs to compute the thin QR factorization of matrix $\begin{bmatrix} A_{\mathcal{F}} & p \end{bmatrix}$ by the modified Gram-Schmidt method as described in [47] or by recursive QR factorization (successive calls to Algorithm 4.3 `orthogonalize` i.e., the reorthogonalization based GS procedure `gs`) for better numerical robustness and then skip to Step 14. The rest of the algorithm follows the same flow as Algorithm 4.2. However, this might not improve the efficiency of the solver unless the warm-start has most of its entries in the optimal active set. It is worth noting that the algorithm can also benefit from the QR factorization associated with the guess solution, which is an important feature in the context of MPC as detailed in Remark 4.1. This prevents recomputing the QR factors of A in Step 2, which reduces most of the computational burden while computing the solution.

Remark 4.1 *In the case of embedded MPC of linear time-invariant systems using BVLS, the QR factors of A can be precomputed offline for cold-start. Moreover, during successive calls to the solver, since the matrix A does not change, the previously computed solution and its associated QR factors can be used for warm-starting in order to significantly reduce the computational burden. Note that if the bounds vary, the components of the initial guess must be*

[7] In general, the execution time with both approaches for initialization i.e. full or empty free set, differs and depends on the cardinality of the free set at the optimal solution.

modified accordingly in order to have its active sets unchanged from the previous solution. As an additional precaution against error accumulation due to such reuse of the QR factors, they can be reset by not providing them to the solver at every N^{th} call, where N can be tuned depending on the problem size, desired accuracy, and computing precision.

4.4.2 Finite termination and anti-cycling procedures

The convergence proof of Algorithm 4.2 follows the ones of [11, Algorithm 16.3] and BVLS in [20]. However, due to rounding errors in finite-precision computations (especially when working in single-precision floating-point arithmetic), the inner loop (Steps 14-35) may not terminate when the computed α (cf. Step 22) results in no component of $\mathcal{F}$ entering an active set of bounds. Numerical error in the gradient (Step 4) may cause failure in satisfying dual feasibility, and cycling of the outer loop. Moreover, when the columns of A are nearly linearly dependent, the algorithm may cycle [20]. As a possible consequence:

1. a component $t^\star$ inserted in an active set of bounds is introduced in the free set in the immediate next step;

2. as a result, after the least-squares step, another component $\hat{t}$ gets inserted in an active set of bounds;

3. in the next iteration, $\hat{t}$ immediately gets inserted in the free set and causes $t^\star$ to be inserted back in the active set, causing a cycle.

We summarize below the measures included in the proposed algorithm in order to avoid the above described cycles, extending the ideas described in [20, 43]. Convergence of the inner loop is guaranteed by including a feasibility tolerance γ (typically between machine precision and 10^{-6}) and by moving at least the index κ (Step 23) in the respective active set at each iteration, such that κ accounts for the value of α as done in [43]. As suggested in [20], unnecessary failure in satisfying dual feasiblity conditions is detected by Step 15 which signals if the component most recently introduced in the free set would immediately enter an active set of bounds. Steps 16-18 set the Lagrange multiplier of the corresponding component to zero and the algorithm steps back to a termination check (Step 5). Cycling between a pair of free components is detected by storing (Step 11) and comparing the previous three values of the index $t^\star$ introduced in the free set as shown in Step 7 of Algorithm 4.2.

The Lagrange multipliers corresponding to the cycling components $t^\star$ and $\hat{t}$ are set to zero in Step 9 and the algorithm then steps back to termination check. If the termination check fails and no other change in $\mathcal{F}$ results in termination, we infer that the algorithm will not converge any further. So, the outer loop terminates having the test in Step 7 satisfied for a second instance (Step 9) due to the cycling of a pair of components. Hence, the algorithm itself terminates. In any case, in all the practical applications in which the solver is used on line, such as in embedded MPC, a bound on the maximum number of iterations is enforced to guarantee termination, where this bound depends on problem size and real-time requirements.

4.5 Recursive thin QR factorization

In this section, we recall the stable QR update procedures of [44] which are adopted in Algorithm 4.2, in the context of active-set changes. Based on that, we derive the recursive relations which update the RHS of (4.5).

Theorem 4.1 (Thin QR factorization) *Let $A \in \mathbb{R}^{m\times n}$ be a full rank matrix with $m > n$. Let $A_{\mathcal{F}} \in \mathbb{R}^{m\times|\mathcal{F}|}$ be formed from a subset of the columns of A, indexed by the set of indices $\mathcal{F} \subseteq \{1, 2, \cdots, n\}$, $|\mathcal{F}|\leq n$. Then there exists a matrix $Q \in \mathbb{R}^{m\times|\mathcal{F}|}$ with orthonormal columns, and a full rank upper-triangular matrix $R \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ such that $A_{\mathcal{F}} = QR$.*

Proof: We prove the theorem by induction on the cardinality $|\mathcal{F}|$ of $\mathcal{F}$. We first prove the theorem for the trivial case of $|\mathcal{F}|= 1$, where we can write

$$\begin{aligned} A_{\mathcal{F}_1} &= Q^{(1)}R^{(1)}, \\ Q^{(1)} &= A_{\mathcal{F}_1}/\|A_{\mathcal{F}_1}\|_2, \\ \text{and } R^{(1)} &= \|A_{\mathcal{F}_1}\|_2. \end{aligned}$$

Assume that the theorem holds $\forall\mathcal{F}$ of cardinality $|\mathcal{F}|= k-1$. For a generic set $\mathcal{F}$ of k indices, if $\mathcal{F}'$ denotes its first $k-1$ indices, we can write

$$A_{\mathcal{F}} = \begin{bmatrix} A_{\mathcal{F}'} & A_{\mathcal{F}_k} \end{bmatrix}.$$

Considering that $|\mathcal{F}'|= k-1$, using the induction hypothesis we can state that $\exists\ Q^{(k-1)}$ orthonormal and $R^{(k-1)}$ full rank upper triangular,

such that $A_{\mathcal{F}'} = Q^{(k-1)}R^{(k-1)}$. Since A is full rank (Assumption 4.1), via Gram-Schmidt orthogonalization [44] we can find an orthonormal vector $q \in \mathbb{R}^m$ ($q^\top Q^{(k-1)} = \mathbf{0}$), a vector $r \in \mathbb{R}^{(k-1)}$, and a scalar $\rho \neq 0$ such that

$$A_{\mathcal{F}_k} = Q^{(k-1)}r + \rho q.$$

Thus, we can rewrite

$$A_{\mathcal{F}} = \begin{bmatrix} A_{\mathcal{F}'} & A_{\mathcal{F}_k} \end{bmatrix} = \begin{bmatrix} Q^{(k-1)}R^{(k-1)} & Q^{(k-1)}r + q\rho \end{bmatrix} = Q^{(k)}R^{(k)},$$

where

$$Q^{(k)} = \begin{bmatrix} Q^{(k)} & q \end{bmatrix}$$
$$\text{and } R^{(k)} = \begin{bmatrix} R^{(k-1)} & r \\ \mathbf{0} & \rho \end{bmatrix}.$$

Matrix $Q^{(k)}$ has orthonormal columns and $R^{(k)}$ is full rank upper triangular by inspection, which proves the theorem for $|\mathcal{F}| = k$. □

Remark 4.2 *In [44], in order to enforce precise orthogonality the Gram-Schmidt procedure on any vector is repeated in situations when numerical cancellation is detected and is termed reorthogonalization. If the ratio $\|A_{\mathcal{F}_k} - Q^{(k-1)}r\|_2 / \|A_{\mathcal{F}_k}\|_2$ is small or if ρ is extremely small, loss of orthogonality is likely and the same ratio is used to determine whether reorthogonalization must be performed [44]. This argument yields Algorithm 4.3 that iteratively "refines" the computed vectors by reducing orthogonality loss. The parameter η in Algorithm 4.3 sets an upper bound on the loss of orthogonality as shown by the detailed error analysis in [44]. Increasing η increases the chance of reorthogonalization and tightens the tolerance on orthogonality closer to the machine precision. We use $\eta = 1/\sqrt{2}$ and $k_{max} = 4$ in Algorithm 4.3 based on*

Algorithm 4.3 Orthogonalization procedure [44]

1: $r^0 \leftarrow \mathbf{0}$, $v^0 \leftarrow v$, $0 << \eta < 1$;
2: **for** $k = 1, 2, \cdots,$ until $\|v^k\|_2 > \eta \|v^{k-1}\|_2$ or $k = k_{\max}$ **do**:
3: $\quad s^k \leftarrow Q^\top v^{k-1}$;
4: $\quad r^k \leftarrow r^{k-1} + s^k$;
5: $\quad v^k \leftarrow v^{k-1} - Q s^k$;
6: **end for**
7: $r \leftarrow r^k$; $\rho \leftarrow \|v^k\|_2$; $q \leftarrow v^k / \rho$;
8: **end.**

the analysis in [44]. Note that the iterations shown in Algorithm 4.3 refer to classical Gram-Schmidt orthogonalization, but for a numerically robust implementation we use the theoretically equivalent Modified Gram-Schmidt iterations (cf. [21, Algorithm 2.4.4], [47, Algorithm 8.1]). For double precision computations, reorthogonalizations may not be performed at all for small to medium sized problems, unless they are ill-conditioned.

Based on Theorem 4.1, Lemma 4.1 and 4.2 respectively delineate the recursive relations that update the thin QR factorization when an index is either inserted or deleted arbitrarily from the set $\mathcal{F}$.

Lemma 4.1 *Given $A_{\mathcal{F}} = QR$, if $A_{\bar{\mathcal{F}}} = \bar{Q}\bar{R}$ denotes the thin QR factorization of $A_{\bar{\mathcal{F}}}$ for $\bar{\mathcal{F}} := \mathcal{F} \cup \{t^\star\}$ and $|\bar{\mathcal{F}}| = k+1$ with $k < n$, then there exists an orthogonal matrix G, two vectors q, r, and a scalar ρ, such that*

$$\begin{aligned} \bar{Q} &= \begin{bmatrix} Q & q \end{bmatrix} G^\top, \\ \bar{R} &= G \begin{bmatrix} R_{\mathcal{F}'} & r & R_{\mathcal{F}''} \\ \mathbf{0} & \rho & \mathbf{0} \end{bmatrix}, \\ \text{where } \mathcal{F}' &:= \{j | \mathcal{F}_j < t^\star, j \in \{1, \cdots, k\}\} \\ \text{and } \mathcal{F}'' &:= \{j | \mathcal{F}_j > t^\star, j \in \{1, \cdots, k\}\}. \end{aligned}$$

Proof: With $A_{\mathcal{F}'} \in \mathbb{R}^{m \times |\mathcal{F}'|}$, $A_{\mathcal{F}''} \in \mathbb{R}^{m \times (k - |\mathcal{F}'|)}$, and $A_{t^\star}$ as the inserted column, considering $R = \begin{bmatrix} R_{\mathcal{F}'} & R_{\mathcal{F}''} \end{bmatrix}$, we have the following relations

$$A_{\bar{\mathcal{F}}} = \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{t^\star} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = \begin{bmatrix} Q & A_{t^\star} \end{bmatrix} \begin{bmatrix} R_{\mathcal{F}'} & \mathbf{0} & R_{\mathcal{F}''} \\ \mathbf{0} & 1 & \mathbf{0} \end{bmatrix}.$$

By orthogonalizing $A_{t^\star}$ via Algorithm 4.3, i.e., by the Gram-Schmidt procedure [44], we obtain

$$A_{t^\star} = Qr + \rho q, \text{ and hence} \tag{4.6}$$

$$A_{\bar{\mathcal{F}}} = \underbrace{\begin{bmatrix} Q & q \end{bmatrix}}_{\tilde{Q}} \underbrace{\begin{bmatrix} R_{\mathcal{F}'} & r & R_{\mathcal{F}''} \\ \mathbf{0} & \rho & \mathbf{0} \end{bmatrix}}_{\tilde{R}}, \tag{4.7}$$

where q, r and ρ are computed such that $\tilde{Q}$ has orthonormal columns and $\tilde{R}$ has subdiagonal elements in its $(|\mathcal{F}'|+1)^{\text{th}}$ column. These subdiagonal elements can be zeroed-out by successive application of Givens matrices [46], which converts $\tilde{R}$ to the upper triangular matrix

$$\bar{R} = G\tilde{R}, \tag{4.8}$$

where $G \in \mathbb{R}^{|\bar{\mathcal{F}}|\times|\bar{\mathcal{F}}|}$ denotes the product of the Givens matrices, which are orthogonal by definition. Hence, $G^\top G = I$, and from (4.7)-(4.8), $A_{\bar{\mathcal{F}}} = \tilde{Q}G^\top G\tilde{R} = \tilde{Q}G^\top \bar{R}$, which implies that $\bar{Q} = \tilde{Q}G^\top$. □

Lemma 4.2 *Given $A_{\mathcal{F}} = QR$, if $A_{\hat{\mathcal{F}}} = \hat{Q}\hat{R}$ denotes the thin QR factorization of $A_{\hat{\mathcal{F}}}$ for $\hat{\mathcal{F}} \coloneqq \mathcal{F} \setminus \{t^\star\}$, then there exists a matrix H such that*

$$\hat{Q} = QH^\top$$
$$\text{and } \hat{R} = HR_{\tilde{\mathcal{F}}},$$

where $\tilde{\mathcal{F}} \coloneqq \{j | \mathcal{F}_j \neq t^\star, j \in \{1, \cdots, |\mathcal{F}|\}\}$.

Proof: Given $A_{\mathcal{F}} = QR$, with $\mathcal{F}'$ and $\mathcal{F}''$ as defined in Lemma 4.1, we have

$$\begin{aligned} A_{\mathcal{F}} &= \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{t^\star} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = Q \begin{bmatrix} R_{\mathcal{F}'} & \tilde{r} & R_{\mathcal{F}''} \end{bmatrix}, \\ A_{\hat{\mathcal{F}}} &= \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = Q \begin{bmatrix} R_{\mathcal{F}'} & R_{\mathcal{F}''} \end{bmatrix} = QR_{\tilde{\mathcal{F}}}, \end{aligned} \tag{4.9}$$

where $R_{\tilde{\mathcal{F}}}$ is upper Hessenberg [46]. Zeroing the off-diagonal elements of $R_{\tilde{\mathcal{F}}}$ using Givens rotations, we obtain

$$G'R_{\tilde{\mathcal{F}}} = \begin{bmatrix} \hat{R} \\ \mathbf{0} \end{bmatrix},$$

where $G' \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ denotes the product of the Givens matrices. Hence,

$$\hat{R} = EG'R_{\tilde{\mathcal{F}}} = HR_{\tilde{\mathcal{F}}},$$

where

$$E = \begin{bmatrix} I & \mathbf{0} \end{bmatrix}^\top$$
$$\text{and } H = EG'.$$

Since G' is orthogonal,

$$G'^\top E^\top EG' = I,$$

which implies that $H^\top H = I$. As

$$A_{\mathcal{F}} = \hat{Q}\hat{R} = QR_{\tilde{\mathcal{F}}} = QH^\top HR_{\tilde{\mathcal{F}}},$$

we obtain $\hat{Q} = QH^\top$. □

Remark 4.3 *Updating the thin QR factorization when inserting or deleting a column as above requires* $4lm(k+1)+6|\mathcal{F}''|(m+|\mathcal{F}''|/2)$ *and* $6|\mathcal{F}''|(m+|\mathcal{F}''|/2)$ *flops (floating-point operations excluding indexing and looping overhead) respectively, where* l *denotes the number of orthogonalizations performed and the remaining terms are defined in Lemma 4.1. The update routines are numerically stable as precise orthogonality is retained during the recursions.*

Proposition 4.1 *Consider* $d = Q^\top p$ *as defined in (4.5) where* $A_\mathcal{F} = QR$*, and the updated factorization* $A_{\bar{\mathcal{F}}} = \bar{Q}\bar{R}$ *for* $\bar{\mathcal{F}} = \mathcal{F} \cup \{t^\star\}$ *as in Lemma 4.1. The update* $\bar{d}$ *of* d *can be obtained recursively as* $\bar{d} = G\tilde{d}$*, where*

$$\tilde{d} = \begin{bmatrix} d + rx_{t^\star} \\ q^\top p + \rho x_{t^\star} \end{bmatrix}$$

and $x_{t^\star}$ *denotes the* $t^\star$*th element of solution vector* x *at the current iteration of Algorithm 4.2 when solving (4.2).*

Proof: From (4.5), we have

$$d = Q^\top p, \text{ and} \tag{4.10}$$

$$\bar{d} = \bar{Q}^\top \bar{p}, \tag{4.11}$$

where $\bar{p}$ denotes the vector p of (4.5) after an index $t^\star$ is inserted in the free set $\mathcal{F}$. Then by its definition,

$$\bar{p} = p + A_{t^\star} x_{t^\star}. \tag{4.12}$$

Using Lemma 4.1 and substituting (4.12) in (4.11) gives

$$\bar{d} = G \begin{bmatrix} Q & q \end{bmatrix}^\top (p + A_{t^\star} x_{t^\star}). \tag{4.13}$$

On substituting (4.6) in (4.13), we get

$$\begin{aligned} \bar{d} &= G \begin{bmatrix} Q & q \end{bmatrix}^\top (p + Qrx_{t^\star} + \rho q x_{t^\star}) \\ &= G \begin{bmatrix} Q^\top p + Q^\top Q r x_{t^\star} + \rho Q^\top q x_{t^\star} \\ q^\top p + q^\top Q r x_{t^\star} + \rho q^\top q x_{t^\star} \end{bmatrix}. \end{aligned} \tag{4.14}$$

Since Q has orthonormal columns and q is orthogonal to the span of Q, we have $Q^\top Q = I$, $q^\top q = 1$, $Q^\top q = \mathbf{0}$ and $q^\top Q = \mathbf{0}$. Hence, by substituting these relations and (4.10), (4.14) simplifies as

$$\bar{d} = G \begin{bmatrix} d + rx_{t^\star} \\ q^\top p + \rho x_{t^\star} \end{bmatrix}. \tag{4.15}$$

□

Proposition 4.1 shows through (4.15) that computing $\tilde{d}$ from d only needs $2(|\mathcal{F}|+m+1)$ *flops* and $\bar{d}$ is obtained by using $6|\mathcal{F}''|$ *flops* for applying the Givens rotations [46] on $\tilde{d}$. Updating d to $\bar{d}$ without the recursive relation (4.15) would cost $2m(|\mathcal{F}|+2)$ *flops* instead, due to the computations in (4.12) and (4.11), at each iteration of the outer loop of Algorithm 4.2. Proposition 4.2 establishes the recursive relation to update d for the case in which an index is removed from the free set $\mathcal{F}$.

Proposition 4.2 *Consider $d = Q^\top p$ as defined in (4.5) where $A_\mathcal{F} = QR$, and the updated factorization $A_{\hat{\mathcal{F}}} = \hat{Q}\hat{R}$ for $\hat{\mathcal{F}} = \mathcal{F} \setminus t^\star$ as in Lemma 4.2, then the update $\hat{d}$ of d can be obtained recursively as $\hat{d} = H(d - \tilde{r}x_{t^\star})$.*

Proof: Let $\hat{p}$ denote the vector p of (4.5) after an index $t^\star$ is removed from $\mathcal{F}$, then

$$\hat{p} = p - A_{t^\star}x_{t^\star}, \tag{4.16}$$

where from (4.9) the deleted column

$$A_{t^\star} = Q\tilde{r}. \tag{4.17}$$

From (4.5), (4.16), (4.17), and Lemma 4.2,

$$\hat{d} = \hat{Q}^\top\hat{p} = HQ^\top(p - Q\tilde{r}x_{t^\star}).$$

On substituting $Q^\top Q = I$ and (4.10) in the above equation, we get

$$\hat{d} = H(d - \tilde{r}x_{t^\star}). \tag{4.18}$$

□

Updating d to $\hat{d}$ recursively via (4.18) needs only $2|\mathcal{F}|+6(|\mathcal{F}''|-1)$ *flops* instead of $2m|\mathcal{F}|$ *flops* for computing $\hat{Q}^\top\hat{p}$ and (4.16), in each iteration of the inner loop of Algorithm 4.2.

Remark 4.4 *Even though the vector d is initialized with machine precision accuracy, as a precaution to avoid the error accumulated over potentially several recursive updates in large sized problems it is recommended to reinitialize d via Step 43 in Algorithm 4.2 after every N iterations, with N chosen according to the available computing precision.*

4.6 Numerical results

4.6.1 Random BVLS problems

This section describes the numerical results obtained in MATLAB by testing the proposed solver (Algorithm 4.2) and various others on random BVLS problems, which are generated as explained in Appendix 4.8.2. In order to test for numerical robustness and performance while dealing with ill-conditioned or nearly rank-deficient problems, the condition number of the A matrix is set to 10^8 ($\equiv 10^{16}$ for the Hessian of the equivalent QP $\min_{l \leq x \leq u} \frac{1}{2} x^\top A^\top A x - b^\top A x$).

The following solvers are considered for the numerical tests:

1. BVLS_SP - Stark and Parker's BVLS algorithm [20] implemented in embedded MATLAB (cf. Algorithm 4.1);

2. RBVLS - proposed BVLS solver (Algorithm 4.2) coded in plain C without any external libraries and interfaced with MATLAB using a standard compiler in single and double precision;

3. BVLS2 - a variant of Algorithm 4.2 in which the number of orthogonalizations in QR update procedures is restricted to one for faster execution;

4. QPoases_C - box-constrained variant of the open source QP solver QPOASES version 3.2.0 [50] with C backend and main setting *reliable*, where we also enable settings that avoid additional computational overhead while solving nearly rank-deficient problems;

5. OSQP - solver version 0.3.0 of the QP method [42] based on ADMM using sparse matrices and 5000 maximum iterations to limit its maximum execution time;

6. Gurobi - the dual simplex algorithm of Gurobi 7.5.2 [51] was chosen for the tests as it performed best amongst its other available algorithms;

7. fastGP - Algorithm 4.4 (the fast gradient projection algorithm [22]) implemented in embedded MATLAB, with restart in every 50 off 3000 maximum iterations and termination criterion based on [52, Equation 6.18], cf. Appendix 4.8.1;

8. ODYSQP - ODYS QP solver[8] [53];

9. CPLEX - primal simplex algorithm (method `cplexlsqlin`) of CPLEX 12.6.3 [54].

For all solvers, the feasibility or optimality tolerance was set to 10^{-9} with all computations in double precision (machine precision $\epsilon \approx 10^{-16}$), except for the single precision ($\epsilon \approx 10^{-7}$) version RBVLS_single of Algorithm 4.2, where the same tolerance was set to 10^{-6}. The tolerances for all solvers are relaxed to 10^{-6} for the MPC example in Section 4.6.2. For first-order methods OSQP and fastGP, the tolerances are internally relaxed based on problem-specific termination criteria [42], [52]. All the numerical tests have been run on a Macbook Pro 2.6 GHz Intel Core i5 with 8GB RAM. All matrix operations have been computed through plain C code interfaced to MATLAB R2015b. The execution time of each solver is obtained from its internally measured time in order to avoid counting the time for interfacing it with MATLAB.

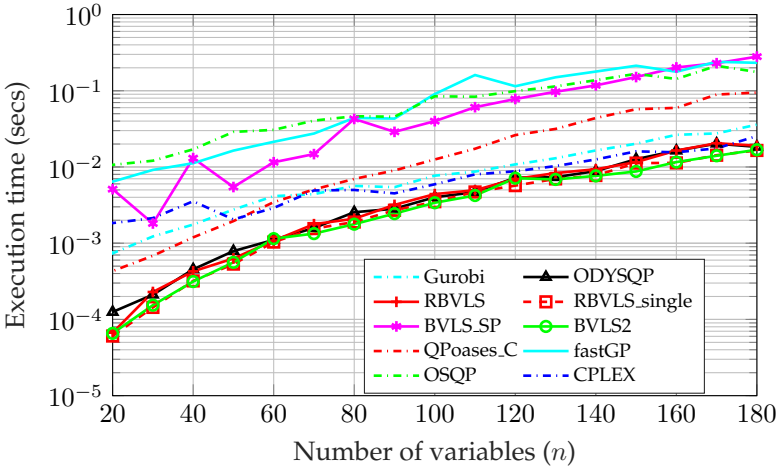
In Figure 4.2, the cost function values compare the quality of the solution instead of the solution vector obtained from different solvers because in the presence of tolerances and poor-conditioning, the solution values obtained may differ while numerically yielding the same value of the cost function. The following conclusions may be drawn while referring Figures 4.1-4.2:

1. The proposed algorithm performs well in terms of both computational efficiency and accuracy as compared to the benchmark solvers irrespective of the numerical conditioning of the problems.

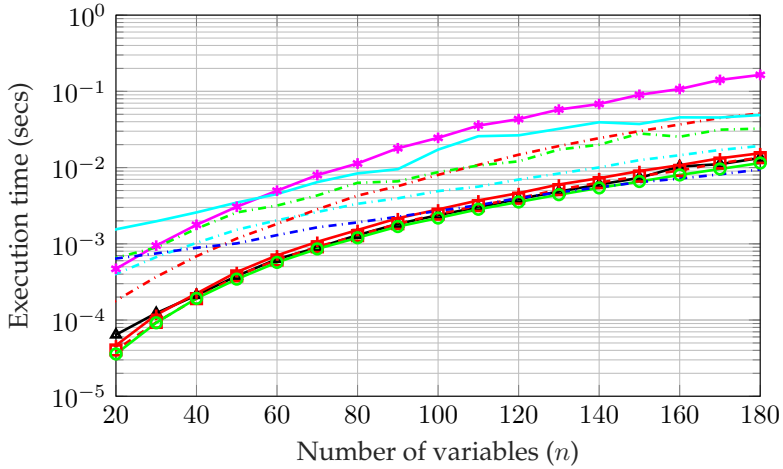
2. The proposed solver with a single orthogonalization in QR updates (BVLS2) has same accuracy as the Stark and Parker's BVLS algorithm (BVLS_SP) in double precision and is faster by a significant margin. By performing multiple orthogonalizations when required, the RBVLS algorithm computes a solution with high accuracy in all cases.

3. The accuracy of the propsed algorithm (RBVLS_single) in single precision depends on the loss of precision due to numerical conditioning. However, it computes a solution which results in an error of low order of magnitude $\approx 10^{-6}$ even when solving problems

[8] ODYS QP solver version "General purpose" 2017 has been used for all tests reported in this chapter.

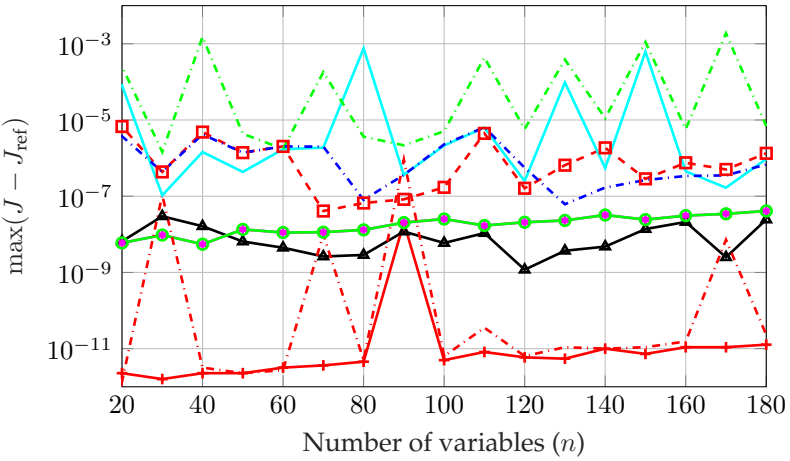


(a) Worst-case computational time.

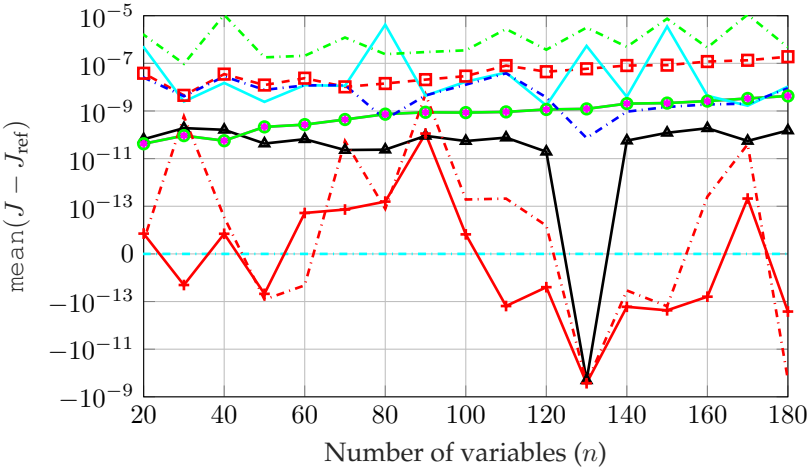


(b) Average computational time.

Figure 4.1: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10^8$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set.



(a) Worst-case difference from the benchmark cost.



(b) Average difference from the benchmark cost.

Figure 4.2: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10^8$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set. The optimal cost J of (4.2) obtained using each solver is compared with the benchmark value J_{ref} obtained using Gurobi. The legend for the plots is same as in Figure 4.1a and is not included here for clarity by avoiding overlap with the plots.

with the reciprocal of a condition number comparable to the working machine precision.

4. The numerical results demonstrate that the proposed BVLS solver is suitable for applications, such as embedded MPC, where numerical robustness and computational efficiency are necessary requirements.

Additionally, results for test problems having better conditioned A matrix with respective condition numbers $10^1, 10^4$ are included in Appendix 4.8.3 in order to further demonstrate computational competitiveness, especially in comparison with solvers based on first-order methods (cf. Figures 4.9a-4.11a). These results demonstrate that the error in computing the solution increases for first-order methods with increase in the condition number. This also influences their convergence rate, resulting in a performance slower than active-set method based solvers for problems with large condition numbers. In order to demonstrate this, details on the number of iterations taken by each solver for the test problems are included in Appendix 4.8.4. Figures 4.13-4.15 show that active-set method based solvers are significantly less sensitive to the condition number of the problem as compared to first-order methods.

4.6.2 Application: embedded linear model predictive control

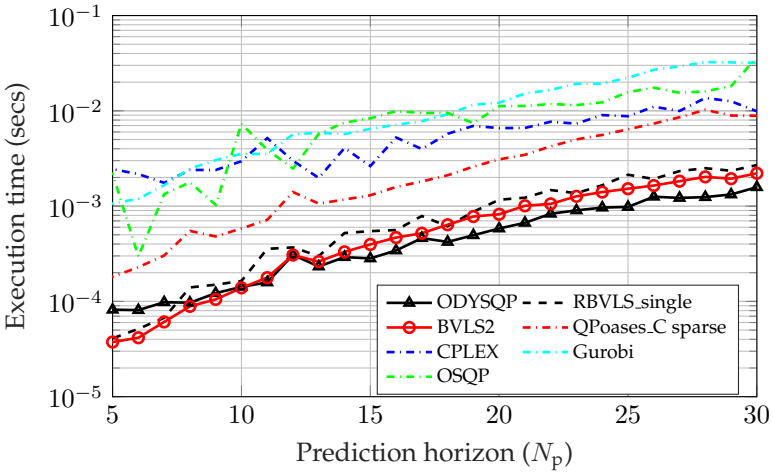
We now consider BVLS problems that arise when solving the MPC problem for control of an AFTI-F16 aircraft based on a linear time-invariant (LTI) model described in Chapter 2. These problems have a high condition number, a property typically encountered in MPC problems involving the use of penalty functions and open-loop unstable prediction models, which hinders the convergence of first-order methods such as fastGP. In order to count only the time required for *online* computations, for all the QP solvers, the time required for computing the Hessian matrix and its factors is not accounted, as it can be done *offline* once and stored. For the same reason, the QPoases_C and ODYSQP solvers are provided with pre-computed dense matrix factorizations of the Hessian, and in case of OSQP only *solve time* was measured. Since the considered MPC optimization problem is numerically sparse, the solvers OSQP, Gurobi, CPLEX and QPoases_C are provided with sparse matrices for a faster performance. The proposed algorithm is warm-started with

the previous solution (cf. Remark 4.1 in Section 4.4) and all other solvers are warm-started from the shifted previous solution from the second instance onward, except for ODYSQP which is always cold-started. Figure 4.3a shows that the proposed algorithm is competitive in computational efficiency as compared to the benchmark solvers, whereas Figure 4.3b shows that it considerably exploits warm-starts. Although in embedded MPC applications the worst-case CPU time is the most relevant measure, as it is used to guarantee meeting hard real-time constraints, the average time may still be of interest when the same CPU is shared with other tasks.

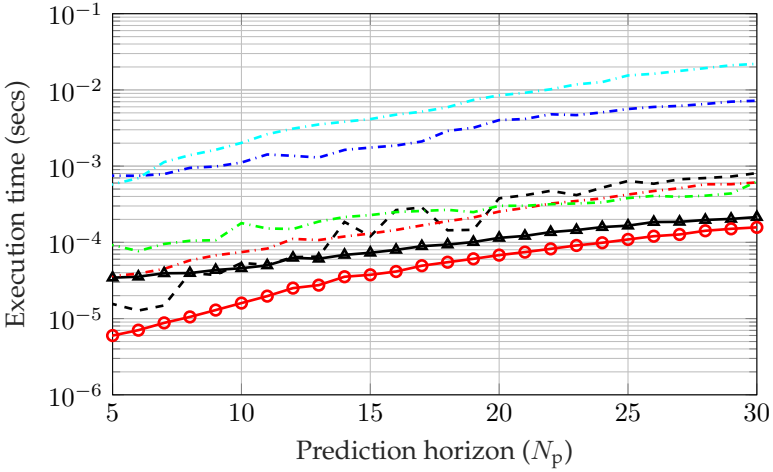
Recalling the comparison of the BVLS problem formulation based on an I/O model against the condensed QP one based on state-space models in Section 2.5, Figure 4.4 shows the same comparison as in Figure 2.6 but with modern solvers. It was noted earlier in Section 2.5 that the benefits of the BVLS formulation depend on the solution algorithm. In Figure 4.4, we observe that: 1) there is a decrease in CPU time when using the BVLS formulation with QPoases_C as compared to the condensed QP one, and 2) the proposed solver is more efficient in exploiting the simple structure of the optimization problem as compared to QPoases_C. Moreover, there is room for improvement as the comparisons only included a dense implementation of BVLS2. In order to justify the usage of the sparse variant of QPoases_C in the comparisons in Figure 4.3, additional results using the dense variant are included in Figure 4.4. Solvers exploiting the block-sparse structure of MPC problems have a computational complexity that scales linearly with N_p [55, 56], and over a certain value of N_p they may outperform *dense* solvers, for which the computations scale quadratically, as seen in Figure 4.3. This aspect of adapting the proposed BVLS algorithm for embedded MPC applications with a reduced requirement for memory and computations, by exploiting the specific structure of the resulting BVLS problems, is addressed in Chapter 6.

4.6.3 Hardware implementation on a programmable logic controller

Programmable logic controllers (PLCs) are one of the most widely employed embedded control devices in industrial automation. This section briefly describes a framework for implementing the MPC approach based on the BVLS solver on a standard industrial PLC. Preliminary results based on hardware-in-the-loop (HiL) tests with a quadruple tank



(a) Worst-case computational time.



(b) Average computational time.

Figure 4.3: Solver performance comparison for BVLS formulation based tracking-MPC simulation of AFTI-F16 aircraft. The number of decision variables and box constraints each $= 4N_p$ resulting in matrix $A \in \mathbb{R}^{6N_p \times 4N_p}$. For each N_p, the simulation is run for 100 time instances.

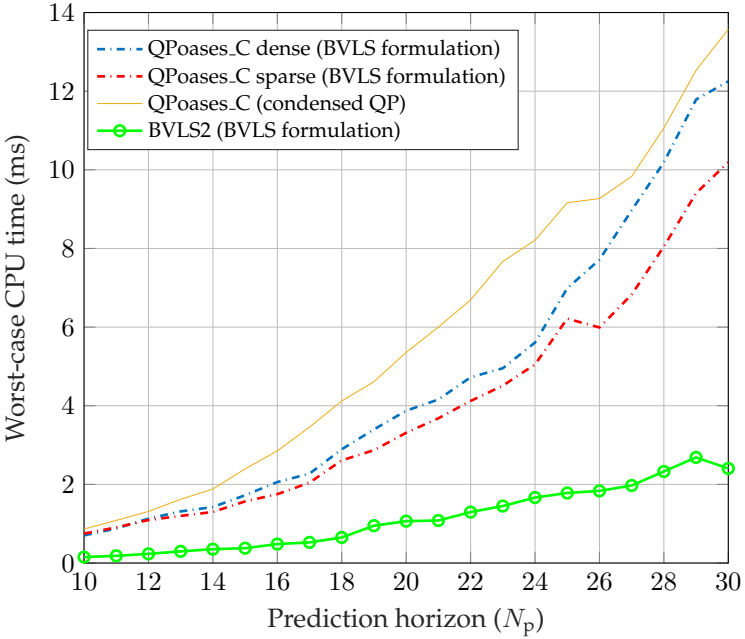


Figure 4.4: Comparison of the performance (solver time) of QPoases_C with BVLS (2.9) and condensed QP (2.26a) formulations for the AFTI-F16 LTI MPC problem. QPoases_C sparse indicates the variant of QPoases_C with Hessian input in sparse format. The times are averaged over 50 instances of the same simulation. The BVLS formulation has $4N_\mathrm{p}$ variables with $8N_\mathrm{p}$ constraints whereas the condensed QP formulation has $2N_\mathrm{p} + 1$ variables with $8N_\mathrm{p} + 1$ constraints (cf. Section 2.5).

system are included for illustration. The numerical results described earlier in this chapter were based on a computer implementation of the algorithms, which is much easier than implementation on industrial embedded hardware platforms. Embedded optimization on an industrial PLC poses some challenges that are not encountered in a computer implementation. Some of the challenges that must be considered before a practical application are:

1. A computer can execute 'easy-to-code' programs written in a wide range of high-level languages using appropriate software whereas in a PLC, programs must be written using the programming lan-

Figure 4.5: Modicon M340 PLC of Schneider Electric used for the HiL tests. The main specifications are 256kB data RAM, $1.16 \mu s/flop$ [57].

guages of the IEC[9] 61131-3 standard, such as 'Structured text' (ST) or 'functional block diagrams' (FBD). Structured text is relatively a low-level programming language which has similarities to basic programming languages such as C but with very limited functionalities.

2. The entire source-code of the optimization algorithm must be a single set of simple instructions without 'functions' that are commonly used in high-level programming languages. However, this is platform dependent.

3. Optimization algorithms must be designed to be robust against numerical errors that may be caused by precision loss because typically in PLCs, computations are carried out in single-precision floating-point arithmetic.

4. Characteristics such as low random-access memory (RAM) and slow computations are expected from standard industrial PLCs (see Figure 4.5).

Thanks to the library-free implementation of the BVLS solver, it was simple to code the same in ST. Porting problem data such as matrices and

[9]IEC is the abbreviation for International Electrotechnical Commission.

working vectors for the solver, as well as the ST code into an FBD in the PLC is a non-trivial issue. For this purpose we use the code-generation library reported in [58].

Case study: Linear MPC of a quadruple tank system

We consider a quadruple water tank system where two of the tanks are assembled above the other two. Two pumps control the water flows q_a and q_b (i.e. the control inputs u), which are redirected to the tanks as shown in Figure 4.6. Further details on its operation may be referred in [58, Section 7A]. The control objective is to steer the heights (h_1, h_2) of the lower tanks, which are the system outputs y, to the desired levels y_r.

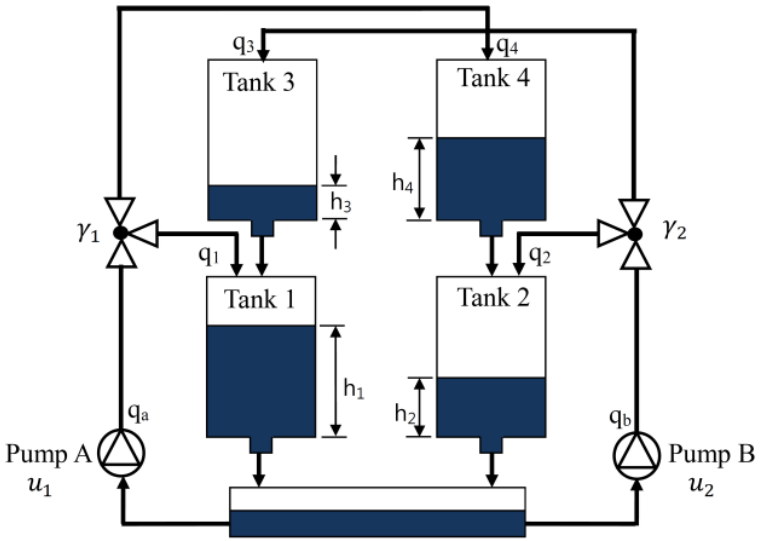


Figure 4.6: Quadruple tank system [59].

The nonlinear system dynamics is governed by the following ordinary differential equations,

$$A_t \frac{dh_1}{dt} = -a_1\sqrt{2gh_1} + a_3\sqrt{2gh_3} + \gamma_1 \frac{q_a}{3600} \tag{4.19a}$$

$$A_t \frac{dh_2}{dt} = -a_2\sqrt{2gh_2} + a_4\sqrt{2gh_4} + \gamma_2 \frac{q_b}{3600} \tag{4.19b}$$

$A_t\ (m^2)$	$a_1\ (m^2)$	$a_2\ (m^2)$	$a_3\ (m^2)$	$a_4\ (m^2)$	γ_1	γ_2
0.03	$1.3 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$9.3 \cdot 10^{-5}$	$8.8 \cdot 10^{-5}$	0.3	0.4

Table 4.1: Quadruple tank system parameters.

$$A_t \frac{dh_3}{dt} = -a_3\sqrt{2gh_3} + (1-\gamma_2)\frac{q_\mathrm{b}}{3600} \tag{4.19c}$$

$$A_t \frac{dh_4}{dt} = -a_4\sqrt{2gh_4} + (1-\gamma_1)\frac{q_\mathrm{a}}{3600}, \tag{4.19d}$$

where the values of the parameters are given in Table 4.1 and $g = 9.81$. The system is subject to the following input and output constraints,

$$\begin{aligned} 0 \leq q_\mathrm{a}, q_\mathrm{b} \leq 3\ \mathrm{m}^3/\mathrm{h} \\ 0 \leq h_1, h_2 \leq 1.2\ \mathrm{m} \end{aligned} \tag{4.20}$$

Based on linearization around the operating point (see Table 4.2) and a zero-order hold discretization scheme with a sampling period of 5 s, we obtain the following discrete-time linear ARX model

$$y(k) = \sum_{j=1}^{n_\mathrm{a}} A^{(j)} y(k-j) + \sum_{j=1}^{n_\mathrm{b}} B^{(j)} u(k-j),$$

where $n_\mathrm{a} = n_\mathrm{b} = 4$ and

$$A^{(1)} = 3.8086I,\ A^{(2)} = -5.4392I,\ A^{(3)} = 3.4523I,\ A^{(4)} = -0.8217I,$$

$$B^{(1)} = \begin{bmatrix} 0.0135 & 0.0006 \\ 0.0005 & 0.0179 \end{bmatrix}, \quad B^{(2)} = \begin{bmatrix} -0.0387 & -0.0005 \\ -0.0005 & -0.0515 \end{bmatrix},$$

$$B^{(3)} = \begin{bmatrix} 0.0369 & -0.0005 \\ -0.0005 & 0.0493 \end{bmatrix}, \quad B^{(4)} = \begin{bmatrix} -0.0117 & 0.0005 \\ 0.0005 & -0.0157 \end{bmatrix}.$$

Based on the LTI model described above, the BVLS problem matrices in (2.9) are constructed and stored along with the QR factors of the coefficient matrix with single precision floating-point numbers. We set MPC

q_a^0	q_b^0	h_1^0	h_2^0	h_3^0	h_4^0
1.9	2.0	0.7175	0.7852	0.6594	0.8950

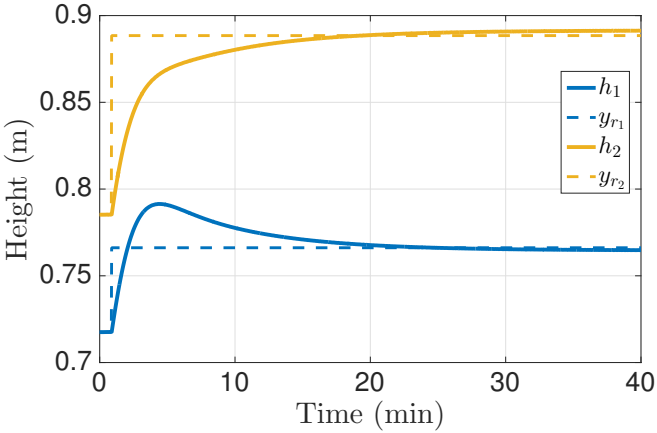
Table 4.2: Operating point parameters for the quadruple tank system.

parameters $N_\mathrm{p} = 9$, $N_\mathrm{u} = 2$, $\sqrt{\rho} = 900$, $W_u = I$ and $W_y = 5I$. Based on the implementation of RBVLS for single precision computations, the HiL test was setup with the MPC algorithm embedded in the (real) PLC shown in Figure 4.5 using the framework of [58] such that the sensors and actuators are interfaced with a high-fidelity quadruple tank system simulator. The MPC approach was tested through reference tracking and disturbance rejection scenarios in HiL simulation. For the tracking problem, the output setpoints are set as $y_r = \begin{bmatrix} 0.7662 \\ 0.8885 \end{bmatrix} m$. For the disturbance rejection test, an impulsive disturbance (a bucket of water) is added in tank 1, which suddenly raises its height, while the quadruple tank's I/O variables must be maintained at the operating point (cf. Table 4.2). The real-time closed-loop control results[10] are shown in Figures 4.7 and 4.8.

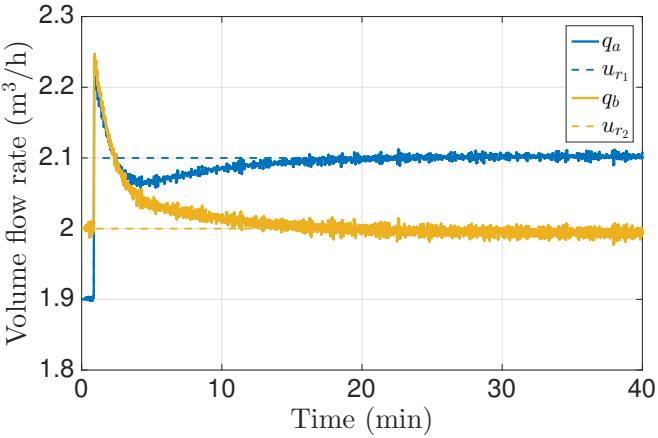
We infer from Figure 4.7 that near-perfect reference tracking was achieved with a small offset because of model mismatch, which occurs as the reference is not at the operating point. Moreover, the input references were computed using the LTI model approximation at the operating point, and discretization errors further contribute to model mismatch. From the results of the disturbance rejection test shown in Figure 4.8, it is seen that the disturbance is eliminated without steady-state offset, since the reference is the same as the operating point. The main purpose of this disturbance rejection test was to demonstrate a scenario in which some of the box constraints become active (cf. Figure 4.8b), thereby testing all routines of the active-set method based solver within the MPC algorithm.

It was noted that the MPC algorithm took 20-21 ms execution time, which also includes internal PLC tasks and interface with system emulation, at each sampling interval of 5 s. Based on times reported for internal PLC tasks and tests performed with plain execution of the solver on the PLC, the execution time of the BVLS solver was 16-18ms for the problem size of 22 decision variables such that the initial QR factorization at the first instance was provided *offline*. With online computation of the QR factorization from scratch, the worst-case time at the first time step increases to around 160ms. This gives a hint on the worst-case time for the linear parameter varying MPC case, where the models would be updated at each step. The total memory consumption of the control algorithm including BVLS i.e. the code memory occupied in the PLC was around 116.08 kilo-bytes (kB), which is about 3.02% of the total 3840 kB. The

[10]The data in this subsection are courtesy of Pablo Krupa (University of Seville), who setup and conducted the HiL experiment.

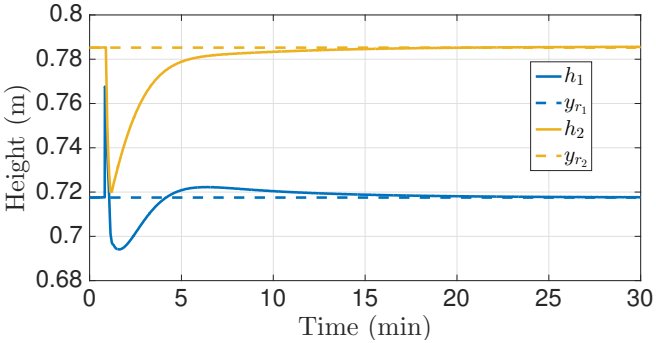


(a) Outputs: heights of tank 1 and 2.

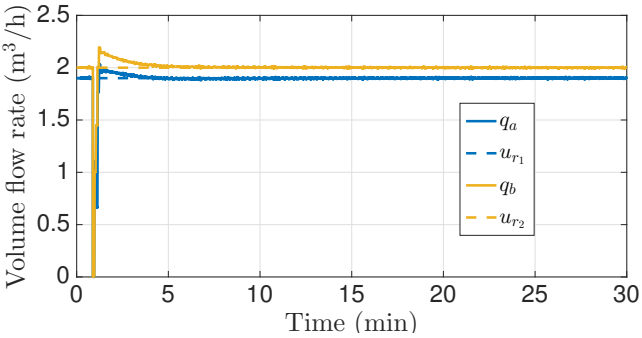


(b) Inputs: water flow rates.

Figure 4.7: Closed-loop trajectories of I/O variables during tracking MPC of the quadruple tank system using PLC.



(a) Outputs: heights of tank 1 and 2.



(b) Inputs: water flow rates.

Figure 4.8: Closed-loop trajectories of the quadruple tank system's I/O variables during disturbance rejection tests using RBVLS based MPC on PLC.

code memory requirement does not change with the problem. The data memory occupied in the PLC for the tests described earlier was 11.06 kB which is about 4.23% of the total 256 kB.

The preliminary results discussed above encourage future work with further practical considerations such as inclusion of a disturbance estimator or integral action for offset-free tracking. Implementation of methods described in Chapter 6 for nonlinear MPC with sparsity exploitation, on the PLC platform, is planned for future work, from which we expect the execution times to be potentially an order of magnitude lower as compared to the numerically *dense* implementation while reducing memory consumption.

4.7 Conclusions

In this chapter we presented a new method for solving BVLS problems. The algorithm is numerically robust, is computationally efficient, and is competitive with respect to state-of-the-art algorithms. The numerical results demonstrate its competitiveness against fast solution methods in solving general small to medium sized BVLS problems such as those arising in embedded MPC, that may also be nearly rank deficient. Numerical stability has been observed even in single precision floating-point arithmetic due to the proposed stable QR update procedures. The hardware implementation of the robust BVLS solver on an industrial embedded hardware platform, such as a PLC, illustrated that several engineering challenges can be successfully addressed by employing the proposed numerical methods.

4.8 Appendix

4.8.1 Fast gradient projection algorithm

The implementation of the variant of Nesterov's fast gradient projection algorithm [22] to solve BVLS problems (4.2) considered in this thesis is described via the following pseudo-code. Specifically, this implementation includes features such as 'restart' after every N_{r} iterations (set to 50 for the tests) with current iterate as new starting point for faster convergence, and automatic termination.

Algorithm 4.4 fastGP algorithm

Inputs: Matrices A, b, l, u; initial guess x_0; feasibility tolerance $\gamma > 0$; maximum number of iterations m_{i}; approximate Lipschitz constant $\tau = \max\left(\mathrm{eig}\left(A^\top A\right)\right)$; $\mu = \min\left(\mathrm{eig}\left(A^\top A\right)\right)$; restart parameter N_{r}.

$x \leftarrow x_0$; $y \leftarrow x_0$; $i \leftarrow 0$; $j \leftarrow 0$; (Initialization)
$\tau' \leftarrow \frac{2\gamma}{(\frac{1}{\mu} - \frac{1}{\tau})}$; (termination parameter)
while $i < m_{\mathrm{i}}$ **do**
  $g \leftarrow A^\top(Ay - b)$; (Gradient computation)
  $x \leftarrow y - \tau g$;
  $\forall k, x_k \leftarrow \min(\max(x_k, l_k), u_k)$; (Projection step)
  $i \leftarrow i + 1$;
  $x' \leftarrow x - x_0$;
  if $\|y - x\|_2^2 < \tau'$ **then** (Stopping criterion [52])
    Go to Step 19; (Terminate algorithm)
  end if
  if $\mathtt{round}(i/N_{\mathrm{r}}) = i/N_{\mathrm{r}}$ **and** $(y - x)^\top x' > 0$ **then** (Restart criterion)
    $y \leftarrow x$; $j \leftarrow 0$; (Restart algorithm)
  else
    $j \leftarrow j + 1$;
    $y \leftarrow x + \frac{j-1}{j+2} x'$
  end if
  $x_0 \leftarrow x$;
end while

Outputs: Primal solution x of (4.2).

4.8.2 Generation of random BVLS test problems

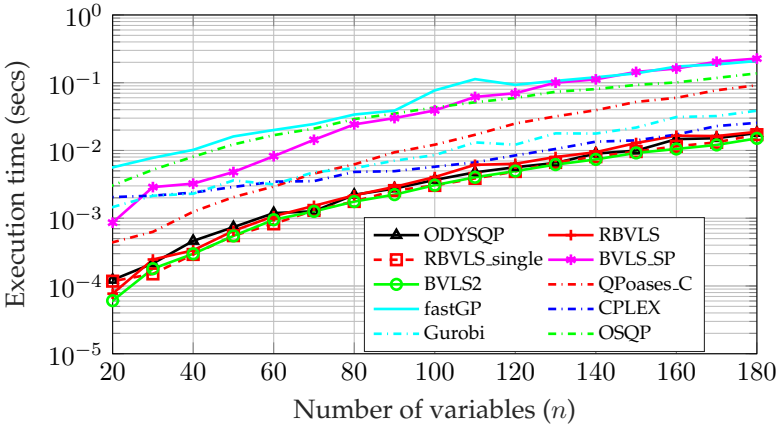
Through Algorithm 4.5, which is described via the following pseudo-code, a set of 180 random problems is generated for each value of problem size considered. The test problems are generated with all possible cardinality values (except full) of the optimal active set. The whole set of problems is generated for each of the three different condition number values of $A : 10, 10^4$ and 10^8. All the matrices are generated such that they have the same numbers in both double and single precision. The number of iterations or computations of the proposed active-set method is directly related to the cardinality of the optimal active-set and hence, we consider test problems including a whole range of cardinality values (50% bounds active at the optimum on average). This gives a fair estimate of the expected worst-case and average performance of the solver for computing the solution of similarly conditioned random BVLS problems

Algorithm 4.5 Random BVLS test problems in MATLAB (pseudo-code)

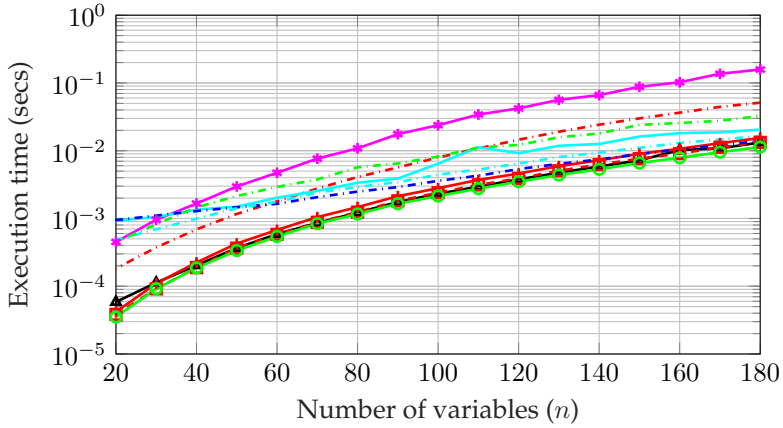
Set the range of number of decision variables:
$\mathcal{R} \leftarrow \{10, 20, \cdots, 180\}$;
Generate a set of random problems for each case in $\mathcal{R}$:
for $i \in \mathcal{R}$ **do**
  for $j \in \{1, 2, \cdots, 180\}$ **do**
    $n \leftarrow i$; (number of decision variables)
    $m \leftarrow \texttt{ceil}(1.5 \cdot n)$; (integer number of rows of A)
    $c_r \leftarrow$ reciprocal of condition number of A;
    $\texttt{rng}(j)$; (set seed of the random number generator)
    Generate a random dense matrix A of given size and reciprocal of its condition number c_r:
    $A^{ij} \leftarrow \texttt{full}(\texttt{sprand}(m, n, 0.99, c_r))$;
    $x \leftarrow 100 \cdot \texttt{rand}(n, 1)$; (Random vector of length n)
    $\bar{u} \leftarrow \max(x)$; $\bar{l} \leftarrow \min(x)$; (Bounds)
    Set an unconstrained solution such that fixed number of bounds are active at the optimum:
    $x_{\{1,3,\cdots,\texttt{mod}(j,i)\}} \leftarrow \bar{l} - 20$; $x_{\{2,4,\cdots,\texttt{mod}(j,i)\}} \leftarrow \bar{u} + 20$;
    $b^{ij} \leftarrow A^{ij}x$;
    Output: Random BVLS problem $\min\limits_{l \le x \le u} \|A^{ij}x - b^{ij}\|_2^2$
  end for
end for

not included in the test set. In order to test for numerical robustness and performance while dealing with ill-conditioned or nearly rank-deficient problems, the condition number of the A matrix may be set to 10^8 ($\equiv 10^{16}$ for the Hessian of an equivalent QP).

4.8.3 Numerical comparisons for random BVLS problems with lower condition numbers

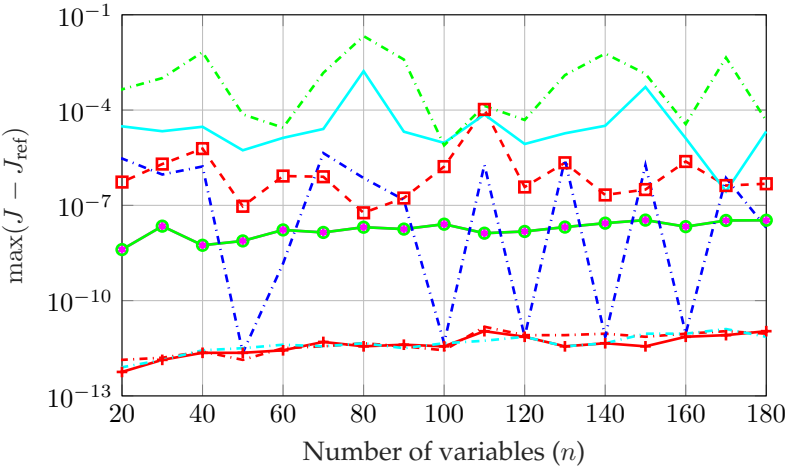


(a) Worst-case computational time.

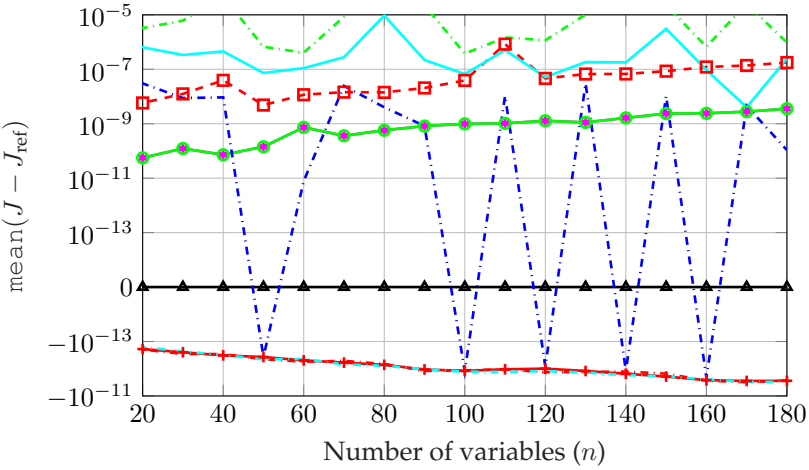


(b) Average computational time.

Figure 4.9: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10^4$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set.

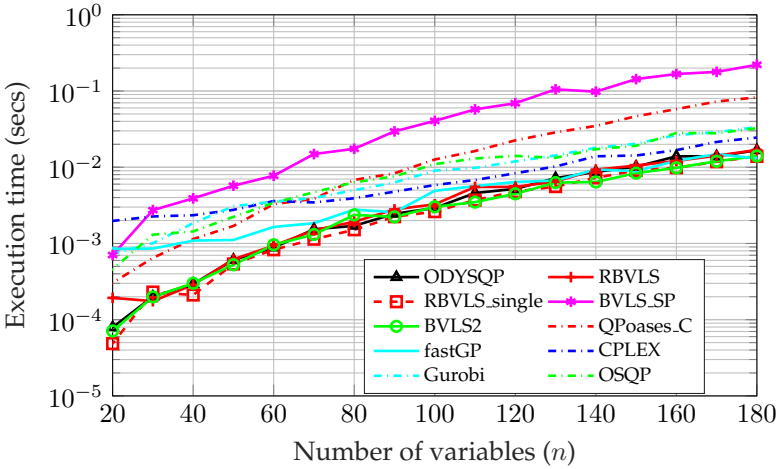


(a) Worst-case difference from the benchmark cost.

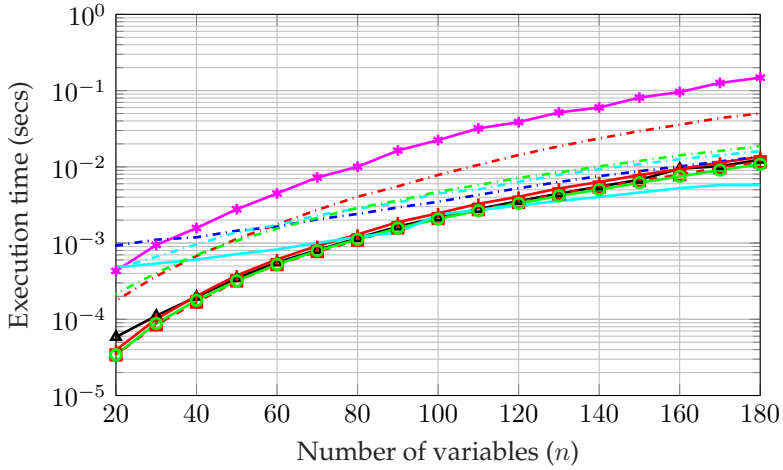


(b) Average difference from the benchmark cost.

Figure 4.10: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10^4$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set. The optimal cost J of (4.2) obtained using each solver is compared with the benchmark value J_{ref} obtained using ODYSQP. The legend for the plots is same as in Figure 4.9a and is not included here for clarity by avoiding overlap with the plots.

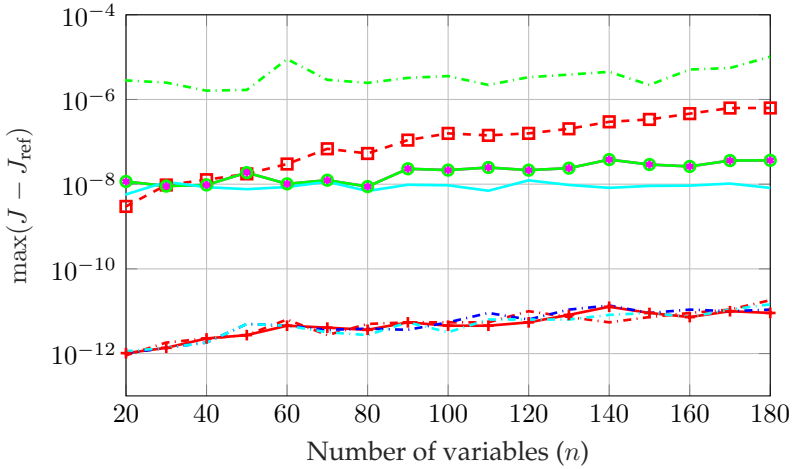


(a) Worst-case computational time.

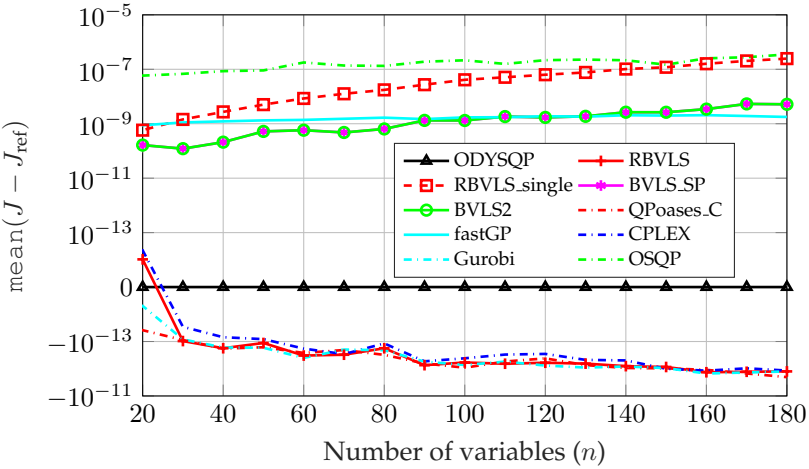


(b) Average computational time.

Figure 4.11: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set.



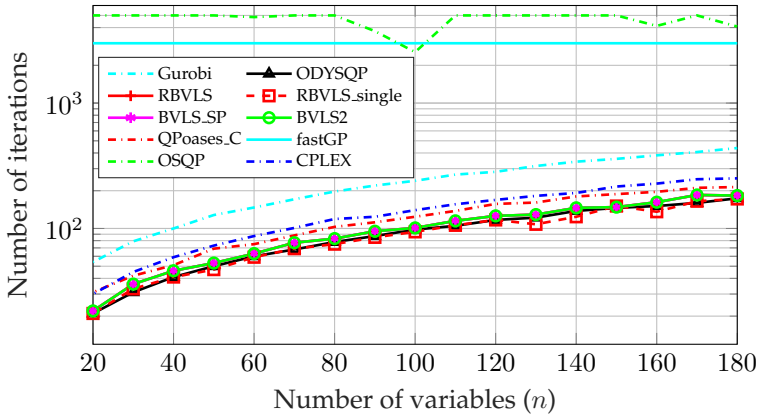
(a) Worst-case difference from the benchmark cost.



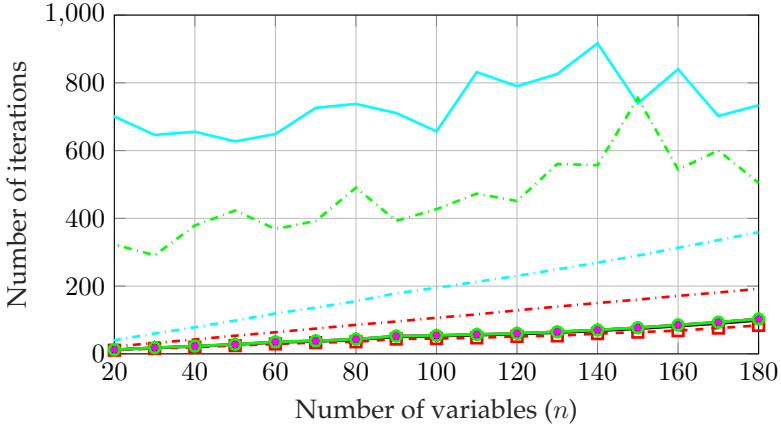
(b) Average difference from the benchmark cost.

Figure 4.12: Solver performance comparison for BVLS test problems with condition number of matrix $A = 10$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set. The optimal cost J of (4.2) obtained using each solver is compared with the benchmark value J_{ref} obtained using ODYSQP.

4.8.4 Iteration count of solvers in the numerical comparisons

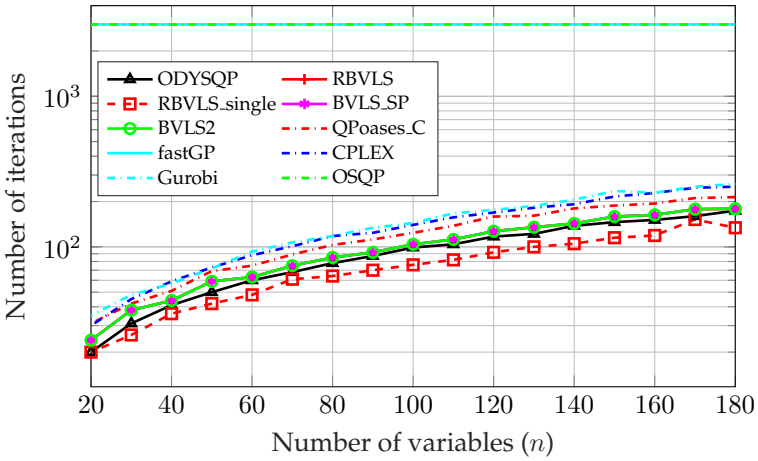


(a) Worst-case number of iterations.

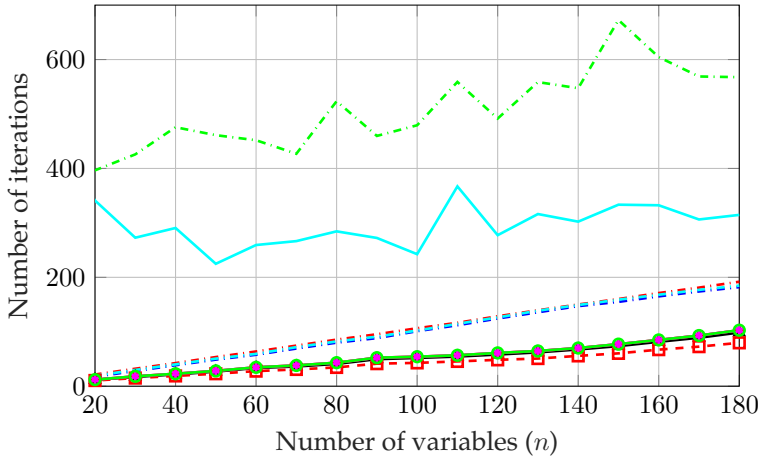


(b) Average number of iterations.

Figure 4.13: Number of iterations of each solver while solving the random BVLS test problems with condition number of matrix $A = 10^8$, referring the comparision in Figure 4.1.

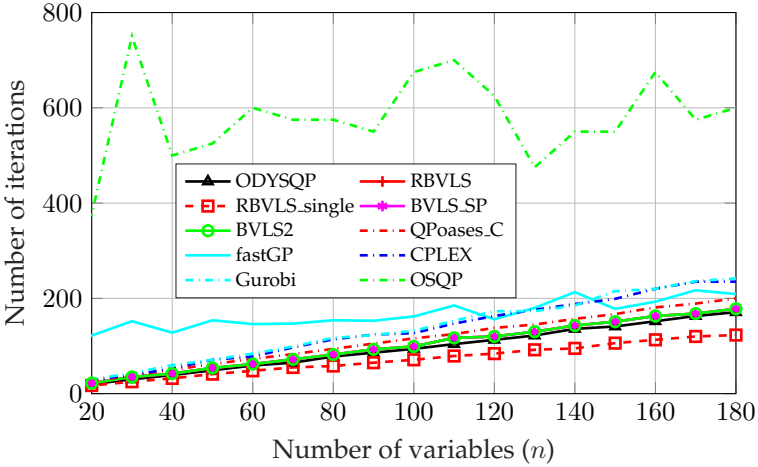


(a) Worst-case number of iterations.

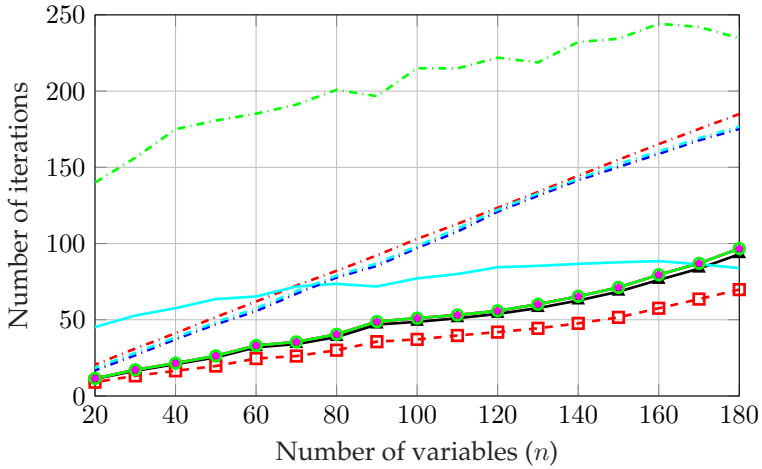


(b) Average number of iterations.

Figure 4.14: Number of iterations of each solver while solving the random BVLS test problems with condition number of matrix $A = 10^4$, referring the comparision in Figure 4.9.



(a) Worst-case number of iterations.



(b) Average number of iterations.

Figure 4.15: Number of iterations of each solver while solving the random BVLS test problems with condition number of matrix $A = 10$, referring the comparision in Figure 4.11.

Chapter 5

Bounded-variable nonlinear least squares

5.1 Introduction

In order to efficiently solve nonlinear MPC problems of the form (3.8), it is desirable to have a solution method that benefits from warm-starting information, is robust to problem scaling, and exploits the structure of the problem. The bounded-variable nonlinear least-squares (BVNLLS) method we propose in Algorithm 5.1 addresses such features. It can be seen as either an *ad hoc* primal-feasible line-search SQP algorithm [11] or an extension of the Gauss-Newton method [21, Section 9.2] to handle box-constraints. The Gauss-Newton approximation of the Hessian is effective for nonlinear least-squares cost functions and it only needs first-order information of the residual. Although the global convergence property of BVNLLS follows that of line-search methods for problems with simple bounds [37], we provide an alternative proof specific to BVNLLS for an insightful overview, which also justifies the backtracking rule (Steps 6-10 of Algorithm 5.1) analogous to the *Armijo* condition [11] for the choice of step-size α.

The BVNLLS algorithm is described in Section 5.2 followed by a theoretical discussion on its global convergence in Section 5.3. Numerical results comparing computational performance with other benchmark solvers are included in Section 5.4 and are based on a typical nonlinear MPC example in simulation.

Notation. The jth element of a vector a is denoted in this chapter as $a(j)$. The set of real numbers between a and b, excluding a and including b is denoted as $(a, b]$ or $[b, a)$. The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\bar{x} \in \mathbb{R}^n$ is either denoted by $\nabla_x f(x)|_{\bar{x}}$ or $\nabla_x f(\bar{x})$, the Hessian matrix by $\nabla_x^2 f(\bar{x})$; the Jacobian of a vector function $g : \mathbb{R}^n \to \mathbb{R}^m$ by $\mathrm{J}_x g(x)|_{\bar{x}}$ or $\mathrm{J}g(\bar{x})$.

5.2 Optimization algorithm

Problem (3.8) can be efficiently solved using the bounded-variable nonlinear least-squares (BVNLLS) algorithm described in Algorithm 5.1. The

Algorithm 5.1 Bounded-Variable Nonlinear Least Squares (BVNLLS) solver

Inputs: Bounds $p,\ q \in \mathbb{R}^{n_z}$, feasible initial guess $z \in \mathbb{R}^{n_z}$, $b = r(z)$, optimality tolerance $\gamma \geq 0$, $c \in (0, 0.5)$, $\tau \in (0,\ 1)$.

1: $J \leftarrow \mathrm{J}_z r$ (Linearization);
2: $\mathcal{L} \leftarrow \{j | z(j) \leq p(j)\}; \mathcal{U} \leftarrow \{j | z(j) \geq q(j)\}$;
3: $d \leftarrow J^\top b$ (Compute gradient of the cost function);
$\lambda_p(j) \leftarrow d(j), \forall j \in \mathcal{L}$; $\lambda_q(j) \leftarrow -d(j), \forall j \in \mathcal{U}$;
4: **if** $\lambda_p(j) \geq -\gamma, \forall j \in \mathcal{L}$ **and** $\lambda_q(j) \geq -\gamma, \forall j \in \mathcal{U}$ **and** $|d(j)| \leq \gamma, \forall j \notin \mathcal{L} \cup \mathcal{U}$ **then go to** Step 12 (Stop if converged to a first-order optimal point);
5: $\Delta z \leftarrow \arg \min_{p-z \leq \Delta z \leq q-z} \|J\Delta z + b\|_2^2$ (Line search);
6: $\alpha = 1$; $\theta \leftarrow c\alpha d^\top \Delta z$; $\psi \leftarrow b^\top b$; $b \leftarrow r(z + \Delta z)$; $\Phi \leftarrow b^\top b$;
7: **while** $\Phi > \psi + \theta$ **do** (Backtracking line search)
8: $\alpha \leftarrow \tau\alpha$; $\theta \leftarrow \alpha\theta$;
9: $b \leftarrow r(z + \alpha\Delta z)$; $\Phi \leftarrow b^\top b$;
10: **end while**
11: $z \leftarrow z + \alpha\Delta z$; **go to** Step 1 (Update the iterate);
12: $z^\star \leftarrow z$; $\lambda_p(j) \leftarrow 0, \forall j \notin \mathcal{L}$; $\lambda_q(j) \leftarrow 0, \forall j \notin \mathcal{U}$;
13: **end.**

Outputs: Local or global optimum $z^\star$ of (3.6), objective function value Φ at $z^\star$, and Lagrange multiplier vectors λ_p and λ_q corresponding to lower and upper bounds respectively.

main idea is to solve the NLLS-box problem by an approach that follows the steps of a Gauss-Newton Hessian approximation based Sequential Quadratic Programming (SQP) [11] while exploiting the special structure of (3.8). The Gauss-Newton method [21, Sec. 9.2] is a well known approach to solve Nonlinear Least Squares (NLLS) problems and is based on a sequence of linear approximations of the residual function $r(z)$ in (3.8). BVNLLS is an extension of the Gauss-Newton method to handle box-constraints, in which a box-constrained Least Squares (BVLS) problem is solved in each iteration until termination criteria are met. Based on the Jacobian of the residual evaluated in Step 1 of Algorithm 5.1, we can compute the step Δz in Step 5 using the efficient BVLS algorithm discussed in Chapter 4. Convergence of the BVNLLS algorithm that involves Gauss-Newton step computations is ensured by including a backtracking line-search method [11, Section 3.1] based on the Armijo-Goldstein condition [21, Section 9.2.1], which forms Steps 6-10 of the BVNLLS algorithm. Necessary and sufficient conditions for the termination of the algorithm are described as follows based on first-order optimality conditions that must be satisfied by a local or global minimum of (3.8). We first note that Algorithm 5.1 starts with an initial guess (cold-start or a warm-start) that satisfies the bounds. Moreover, since Step 5 of Algorithm 5.1 generates a step such that the next iterate also satisfies the bounds, the primal feasibility condition of problem (3.8) is satisfied at any iteration (cf. Lemma 5.1). Let us denote the Lagrange multipliers for lower and upper bounds respectively as λ_l and λ_u, and the sets of active lower and upper bounds respectively as $\mathcal{L}$ and $\mathcal{U}$, as defined in Step 2.
In Algorithm 5.1, we denoted the gradient of the NLLS cost function at iterate i as

$$d := J^\top r(z).$$

The following KKT conditions, which certify first-order optimality, are satisfied by a local minimum of (3.8):

$$d - \lambda_l + \lambda_u = \mathbf{0}, \tag{5.1a}$$
$$\lambda_l(j) = d(j), \qquad \forall j \in \mathcal{L}, \tag{5.1b}$$
$$\lambda_u(j) = -d(j), \qquad \forall j \in \mathcal{U}, \tag{5.1c}$$
$$d(j) = \mathbf{0}, \qquad \forall j \in \{[1, n_z]\} \setminus \{\mathcal{L} \cup \mathcal{U}\}, \tag{5.1d}$$
$$d(j) \geq \mathbf{0}, \qquad \forall j \in \mathcal{L}, \tag{5.1e}$$
$$d(j) \leq \mathbf{0}, \qquad \forall j \in \mathcal{U}. \tag{5.1f}$$

where we used $\mathcal{L} \cap \mathcal{U} = \emptyset$. This defines the termination criterion

evaluated in Step 4 of Algorithm 5.1. The BVLS problem only has simple bounds, which allows the corresponding convex QP to be solved by a primal active-set method [11] by applying efficient linear algebra routines to the smaller-dimensional space of the primal variables. For generic QPs, instead, one needs to work in a higher dimensional space which also includes the Lagrange multipliers. In BVLS, the Lagrange multipliers corresponding to the active-set are obtained simply from respective entries in the gradient vector of the least-squares cost. The same characteristic is also present in BVNLLS, implied by (5.1) and Step 3 of Algorithm 5.1. Moreover, for generic QPs, active-set methods require one to form and factorize the Hessian matrix $J^\top J$, which has a condition number squared as compared to that of the Jacobian matrix J. In our setting, we expect to have a high condition number due to the necessity of choosing the penalty parameter ρ large enough. This can cause ill-conditioning while solving QPs based on the Hessian matrix even when it does not occur in BVLS, in which the matrix that is factorized is the Jacobian.

In summary, the above mentioned characteristics make BVNLLS computationally efficient, and comparatively numerically robust. It is also interesting to note that a single full Gauss-Newton step of Algorithm 5.1 would generate a solution that is equivalent to the one produced by the Real-Time Iteration (RTI) scheme [5], a special case of linear time-varying MPC [39]. This solution can then be used to provide a warmstart for the next time instant (cf. [32,39]).

5.3 Global convergence

This section describes the global convergence property of BVNLLS when solving nonlinear MPC problems of the form (3.8). At the ith iteration of Algorithm 5.1, the search direction $\Delta z^{(i)}$ is computed at Step 5 as

$$\Delta z^{(i)} = \arg\min_{\bar{p}\leq\Delta\hat{z}\leq\bar{q}} \|J\Delta\hat{z} + b\|_2^2, \tag{5.2}$$

where the Jacobian matrix $J = \mathrm{J}_z r\left(z^{(i-1)}\right)$ is full rank, $b = r\left(z^{(i-1)}\right)$, $\bar{p} = p - z^{(i-1)}$ and $\bar{q} = q - z^{(i-1)}$, and p, q are the bounds ($p < q$) on z. Since the lower and upper bound of any component of Δz cannot both be active at the same time, the optimal set of active constraint gradients of (5.2) are always linearly independent. Hence, linear independence constraint qualification holds. Based on this and the fact that the Hessian

$J^\top J > 0$, problem (5.2) always has a unique set of optimal primal and dual variables.

Lemma 5.1 (Primal feasibility) *Consider that $z^{(i)} = z^{(i-1)} + \alpha \Delta z^{(i)}$ as in Step 11 at the ith iteration of Algorithm 5.1 with any $\alpha \in (0,\, 1]$. If $p \leq z^{(0)} \leq q$ and $\bar{p} \leq \Delta z^{(i)} \leq \bar{q}$, then $p \leq z^{(i)} \leq q$ at all iterations i.*

Proof: We prove the lemma by induction. The lemma clearly holds for $i = 0$, as by assumption the initial guess z^0 is feasible, $p \leq z^{(0)} \leq q$. Consider the ith iteration of Algorithm 5.1. From Step 5 we have that $p - z^{(i-1)} \leq \Delta z^{(i)} \leq q - z^{(i-1)}$, which multiplied by α, $\alpha > 0$, gives

$$\alpha p - \alpha z^{(i-1)} \leq \alpha \Delta z^{(i)} \leq \alpha q - \alpha z^{(i-1)}. \tag{5.3}$$

By adding $z^{(i-1)}$ to each side of the inequalities in (5.3) we get

$$\alpha p + (1-\alpha) z^{(i-1)} \leq z^{(i)} \leq \alpha q + (1-\alpha) z^{(i-1)}. \tag{5.4}$$

By induction, let us assume that $p \leq z^{(i-1)} \leq q$. Since $\alpha \leq 1$, we get the further inequalities

$$\alpha p + (1-\alpha) p \leq z^{(i)} \leq \alpha q + (1-\alpha) q$$

or $p \leq z^{(i)} \leq q$. □

Lemma 5.2 *The search direction $\Delta z^{(i)}$ given by (5.2) is a descent direction for the cost function $f(z) = \frac{1}{2}\|r(z)\|_2^2$ in (3.8).*

Proof: If $\mathcal{D}\left(f(z),\, \Delta z\right)$ denotes the directional derivative of $f(z)$ in the direction Δz, then $\Delta z^{(i)}$ is a descent direction if $\mathcal{D}\left(f\left(z^{(i-1)}\right),\, \Delta z^{(i)}\right) < 0$. By definition of directional derivative [11, Appendix A],

$$\mathcal{D}\left(f\left(z^{(i-1)}\right),\, \Delta z^{(i)}\right) = \nabla_z f\left(z^{(i-1)}\right)^\top \Delta z^{(i)}. \tag{5.5}$$

By substituting

$$\nabla_z f\left(z^{(i-1)}\right) = \mathrm{J}_z r\left(z^{(i-1)}\right)^\top r\left(z^{(i-1)}\right) = J^\top b \tag{5.6}$$

in (5.5) we get

$$\mathcal{D}\left(f\left(z^{(i-1)}\right),\, \Delta z^{(i)}\right) = b^\top J \Delta z^{(i)}. \tag{5.7}$$

Since $\Delta z^{(i)}$ solves the convex subproblem (5.2), the following Karush-Kuhn-Tucker (KKT) conditions [10] hold:

$$J^\top \left(J\Delta z^{(i)} + b \right) + \Lambda_{\bar{q}} - \Lambda_{\bar{p}} = \mathbf{0} \tag{5.8a}$$

$$\Delta z^{(i)} \geq \bar{p} \tag{5.8b}$$

$$\Delta z^{(i)} \leq \bar{q} \tag{5.8c}$$

$$\Lambda_{\bar{q}}, \Lambda_{\bar{p}} \geq \mathbf{0} \tag{5.8d}$$

$$\Lambda_{\bar{q}}(j) \left(\Delta z^{(i)}(j) - \bar{q}(j) \right) = 0 \; \forall j \tag{5.8e}$$

$$\Lambda_{\bar{p}}(j) \left(\bar{p}(j) - \Delta z^{(i)}(j) \right) = 0 \; \forall j, \tag{5.8f}$$

where $\Lambda_{\bar{q}}$ and $\Lambda_{\bar{p}}$ denote the optimal Lagrange multipliers of subproblem (5.2). From (5.8a) we have,

$$b^\top J\Delta z^{(i)} = (\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)} - \Delta z^{(i)^\top} J^\top J\Delta z^{(i)}. \tag{5.9}$$

By substituting $\bar{p} = p - z^{(i-1)}$ and $\bar{q} = q - z^{(i-1)}$ in the complementarity conditions (5.8e)-(5.8f), we can write

$$\Lambda_{\bar{q}}^\top \left(\Delta z^{(i)} - q + z^{(i-1)} \right) + \Lambda_{\bar{p}}^\top \left(p - z^{(i-1)} - \Delta z^{(i)} \right) = 0,$$
$$\text{i.e., } (\Lambda_{\bar{q}} - \Lambda_{\bar{p}})^\top \Delta z^{(i)} = \Lambda_{\bar{q}}^\top (q - z^{(i-1)}) + \Lambda_{\bar{p}}^\top (z^{(i-1)} - p).$$

From (5.8b)-(5.8d) we have $\Lambda_{\bar{q}}, \Lambda_{\bar{p}} \geq \mathbf{0}$, and by Lemma 5.1 $q - z^{(i-1)} \geq \mathbf{0}$ as well as $z^{(i-1)} - p \geq \mathbf{0}$, which implies that

$$(\Lambda_{\bar{q}} - \Lambda_{\bar{p}})^\top \Delta z^{(i)} \geq \mathbf{0}, \text{ i.e.,}$$
$$(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)} \leq \mathbf{0}. \tag{5.10}$$

Since J is full rank, $J^\top J > 0$. Using this fact and Lemma 5.4 along with (5.10) in (5.9) gives

$$b^\top J\Delta z^{(i)} < 0. \tag{5.11}$$

Considering (5.11) and (5.7), we have that the directional derivative for the considered search direction is negative, which proves the lemma. □

Remark 5.1 *In the proof of Lemma 5.2, the pair of Lagrange multipliers were considered to be unique at optimality, which is theoretically true based on the given problem (5.2). However, in general, the uniqueness of dual variables is not required for (5.10) and Lemma 5.2 to hold.*

Remark 5.2 *We infer from Lemma 5.1 and (5.8b)-(5.8c) that BVNLLS is a primal-feasible method, which is an important property when the function $r(z)$ is not analytic beyond bounds [60].*

Given this, we provide proof for the following lemmas which help to prove global convergence via Theorem 5.1.

Lemma 5.3 *If the solution of (5.2) is $\Delta z^{(i)} = \mathbf{0}$, then $z^{(i-1)}$ is a stationary point satisfying the first-order optimality conditions of problem (3.8).*

Proof: Given $\Delta z^{(i)} = \mathbf{0}$, we need to prove that $z^{(i-1)}$ satisfies the following first-order optimality conditions for problem (3.8):

$$\mathrm{J}_z r(z)^\top r(z) + \lambda_q - \lambda_p = \mathbf{0} \tag{5.12a}$$

$$p \leq z \leq q \tag{5.12b}$$

$$\lambda_q, \lambda_p \geq \mathbf{0} \tag{5.12c}$$

$$\lambda_q(j)(z(j) - q(j)) = \lambda_p(j)(p(j) - z(j)) = 0, \ \forall j, \tag{5.12d}$$

where the optimal Lagrange multipliers are denoted by λ_p and λ_q for the lower and upper bounds, respectively.

By substituting $\Delta z^{(i)} = \mathbf{0}$ in (5.8), and recalling $\bar{q} = q - z^{(i-1)}$ and $\bar{p} = p - z^{(i-1)}$, we obtain

$$J^\top b + \Lambda_{\bar{q}} - \Lambda_{\bar{p}} = \mathbf{0}, \tag{5.13a}$$

$$p \leq z^{(i-1)} \leq q, \tag{5.13b}$$

$$\Lambda_{\bar{q}}(j)(z^{(i-1)}(j) - q(j)) = 0 \ \forall j, \tag{5.13c}$$

$$\Lambda_{\bar{p}}(j)(p(j) - z^{(i-1)}(j)) = 0 \ \forall j. \tag{5.13d}$$

Clearly, considering (5.8d) along with the definitions of J, b, and (5.13), we conclude that $z^{(i-1)}$, $\Lambda_{\bar{q}}$ and $\Lambda_{\bar{p}}$ solve the KKT system (5.12). □

Lemma 5.4 *In Algorithm 5.1, $\Delta z^{(i)} \neq \mathbf{0}$ at any iteration.*

Proof: We prove this lemma by contradiction. Assume that Algorithm 5.1 reaches an iteration i where Step 5 is executed and returns $\Delta z^{(i)} = \mathbf{0}$. This implies that $z^{(i-1)}$ is a stationary point satisfying the first-order optimality conditions of nonlinear problem (3.8), as shown in Lemma 5.3. Then, the termination criterion in Step 4 would end the algorithm without further computations, so that iteration i is never reached, a contradiction. Note that in particular, if the initial guess $z^{(0)}$ is optimal, $\Delta z^{(i)}$ is never computed. □

Theorem 5.1 (Global convergence of BVNLLS) *Consider the optimization problem (3.8) and define the scalar cost function* $f(z) = \frac{1}{2}\|r(z)\|_2^2$. *At each iteration* i *of Algorithm 5.1, there exists a scalar* $\alpha \in (0, 1]$ *such that*

$$f\left(z^{(i-1)} + \alpha\Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) < c\alpha\nabla f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} \tag{5.14}$$

with $0 < \alpha \leq 1$ *and* $0 < c < 1$, *where* $z^{(i)} = z^{(i-1)} + \alpha\Delta z^{(i)}$.

Proof: Consider the Taylor series expansion of $f\left(z^{(i)}\right)$

$$f\left(z^{(i-1)} + \alpha\Delta z^{(i)}\right) = f\left(z^{(i-1)}\right) + \alpha\nabla_z f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} + \frac{\alpha^2}{2}\Delta z^{(i)^\top}\nabla_z^2 f\left(z^{(i-1)}\right)\Delta z^{(i)} + \mathcal{E}(\|\alpha\Delta z^{(i)}\|^3), \tag{5.15}$$

where the term $\mathcal{E}\left(\|\cdot\|^3\right)$ represents the third order error. Also,

$$\nabla_z^2 f\left(z^{(i-1)}\right) = \sum_{j=1}^{n_r} r_j\left(z^{(i-1)}\right)\nabla_z^2 r_j\left(z^{(i-1)}\right) + \mathrm{J}_z r\left(z^{(i-1)}\right)^\top \mathrm{J}_z r\left(z^{(i-1)}\right) = H + J^\top J, \tag{5.16}$$

where r_j denotes the jth element of the residual vector. By substituting the relations (5.6) and (5.16) in (5.15) we get

$$f\left(z^{(i-1)} + \alpha\Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) = \alpha b^\top J\Delta z^{(i)} + \frac{\alpha^2}{2}\Delta z^{(i)^\top}\left(H + J^\top J\right)\Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha\Delta z^{(i)}\right\|^3\right). \tag{5.17}$$

Using (5.9), Equation (5.17) can be simplified as

$$f\left(z^{(i-1)} + \alpha\Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) = -\frac{\alpha(2-\alpha)}{2}\Delta z^{(i)^\top}J^\top J\Delta z^{(i)} + \alpha(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top\Delta z^{(i)} + \frac{\alpha^2}{2}\Delta z^{(i)^\top}H\Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha\Delta z^{(i)}\right\|^3\right). \tag{5.18}$$

Referring (5.6) and (5.9), on subtracting $c\alpha\nabla f(z^{(i-1)})^\top \Delta z$ from both sides of (5.18) we get

$$f\left(z^{(i-1)}+\alpha\Delta z^{(i)}\right)-f\left(z^{(i-1)}\right)-c\alpha\nabla f\left(z^{(i-1)}\right)^\top\Delta z^{(i)}$$
$$=-\frac{\alpha(2-2c-\alpha)}{2}\Delta z^{(i)^\top}J^\top J\Delta z^{(i)}+\alpha(1-c)(\Lambda_{\bar{p}}-\Lambda_{\bar{q}})^\top\Delta z^{(i)}$$
$$+\frac{\alpha^2}{2}\Delta z^{(i)^\top}H\Delta z^{(i)}+\mathcal{E}\left(\left\|\alpha\Delta z^{(i)}\right\|^3\right). \quad (5.19)$$

Let

$$\bar{N}=-\tfrac{(2-2c-\alpha)}{2}\Delta z^{(i)^\top}J^\top J\Delta z^{(i)}+(1-c)(\Lambda_{\bar{p}}-\Lambda_{\bar{q}})^\top\Delta z^{(i)}.$$

From (5.10), Lemma 5.4, and from the facts that $\alpha\in(0,\ 1]$, $c\in(0,1)$, and that matrix J has full rank ($J^\top J>0$), we infer that $\bar{N}$ must be negative for sufficiently small α. Let

$$\bar{M}=\frac{1}{2}\Delta z^{(i)^\top}H\Delta z^{(i)}+\mathcal{E}\left(\left\|\alpha\Delta z^{(i)}\right\|^3\right)$$

Then (5.19) can be written as

$$f\left(z^{(i-1)}+\alpha\Delta z^{(i)}\right)-f\left(z^{(i-1)}\right)-c\alpha\nabla f\left(z^{(i-1)}\right)^\top\Delta z^{(i)}$$
$$=\alpha\bar{N}+\alpha^2\bar{M}. \quad (5.20)$$

Let $\alpha\bar{N}+\alpha^2\bar{M}+\epsilon=0$, or $\epsilon=\alpha\left(-\alpha\bar{M}-\bar{N}\right)$. Clearly, since $\bar{N}<0$, there exists a value of $\alpha>0$ such that $\epsilon>0$. This proves that there exists a positive value of α such that $\alpha\bar{N}+\alpha^2\bar{M}<0$. Hence from (5.20),

$$f\left(z^{(i-1)}+\alpha\Delta z^{(i)}\right)-f\left(z^{(i-1)}\right)-c\alpha\nabla f\left(z^{(i-1)}\right)^\top\Delta z^{(i)}<0,$$

for a sufficiently small positive value of α. □

Remark 5.3 *In the case of linear MPC i.e., when $h_k(z_k,\ \phi_k)$ is linear in (3.8), the bounded-variable least-squares (BVLS) problem (5.2) is solved only once as the KKT conditions (5.12) coincide with (5.8). Moreover, the backtracking steps are not required as the higher order terms in (5.15) are zero and Theorem 5.1 holds with $\alpha=1$ for any $c\in(0,\ 1)$.*

Remark 5.4 *Referring to* (5.19)*, the value of* c *is practically kept below* 0.5 *in Algorithm 5.1 in order to enforce fast convergence with full steps and is typically chosen to be as small as* 10^{-4} *[11]. As seen in* (5.19)*, since we only need the matrix* J *to be full rank for convergence of BVNLLS, the matrix* W_k *in* (3.8) *may be rank-deficient as long as* J *is full rank.*

Remark 5.5 *Suboptimality in solving the BVLS subproblems may result in a smaller decrease in the cost between BVNLLS iterations than the theoretical decrease indicated by Theorem 5.1. Hence, it is essential to have an accurate BVLS solver in order to have fast convergence. For this reason, we advocate the use of active-set methods to solve BVLS problems.*

Each iteration of BVNLLS corresponds to solving a linear MPC problem, a special case of (3.8). This allows one to have a common framework for linear and nonlinear MPC in our approach. The BVLS problem (5.2) can be solved efficiently and accurately by using a primal active-set algorithm as shown in Chapter 4, which uses numerically robust recursive QR factorization routines to solve the LS subproblems.

5.4 Numerical performance

The MPC problems are formulated as described in Section 3.3 for the considered example described in Section 3.5. BVNLLS uses a MATLAB-interfaced[11] C implementation of the BVLS algorithm based on recursive QR updates. The NLP problems (3.6), (3.7) are solved using MATLAB's 'fmincon' solver with SQP and for comparisons, also the sparse NLP solver 'IPOPT' [61] compiled in MATLAB with the 'MA57' linear systems solver. The solver IPOPT is considered with manually supplied exact sparse Jacobian evaluation functions including sparsity pattern, and an initial guess for the primal and dual variables in order to incorporate all benefits of the tool. The fmincon SQP solver of MATLAB is also provided with gradient evaluation functions and a warmstart for faster convergence. All the NLPs are solved until convergence so that each solver gives the same quality of solution (cf. Section 3.5.2).

Figure 5.1 shows that the BVNLLS solver is considerably faster for small to medium sized problems (30 to 150 decision variables and same

[11] The codes have been run in MATLAB R2015b on a Macbook Pro 2.6 GHz Intel Core i5 with 8GB RAM. The time measurements only account for the solvers' execution time by excluding the time spent in interfacing data with MATLAB where applicable.

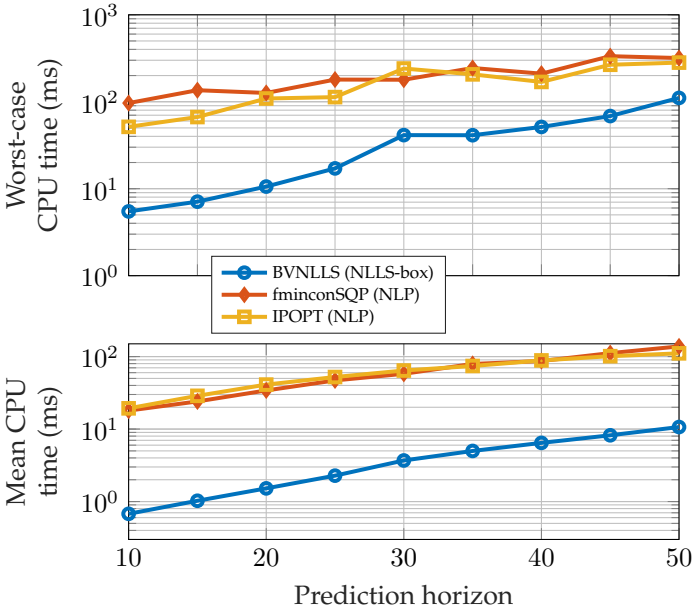


Figure 5.1: CPU time for each solver during closed-loop simulation of the CSTR w.r.t. prediction horizon ($N_\mathrm{p} = N_\mathrm{u}$, number of variables and box constraints $= 3N_\mathrm{p}$, number of equality constraints $= 2N_\mathrm{p}$).

number of bilateral bound constraints). Note that the BVNLLS and fmincon solvers have a numerically dense implementation whereas the IPOPT solver uses sparse numerical methods. Observing these comparisons demonstrate that the BVNLLS based approach is an attractive practical alternative to solve NMPC problems in real-time embedded applications where computational complexity may restrict the use of MPC. Moreover, the solver can be coded specifically to exploit problem-dependent sparse structure of the matrices involved, which is a result of the proposed NMPC formulation. Handling the sparse problem directly would not only further speed up computations but also reduce the memory requirements, which are characteristics suitable for a real-time embedded implementation. This issue is addressed in detail in the next chapter, which also includes further numerical comparisons.

Chapter 6

Methods and tools for efficient non-condensed model predictive control

6.1 Introduction

A usual practice in MPC is to first formulate an optimization problem based on the prediction model and MPC tuning parameters, before passing it in a standard form to an optimization solver. Such a problem construction step can be performed *offline* when the prediction model of the system is time-invariant, e.g. linear time-invariant (LTI) models, whereas it needs to be repeated at each instance in case of parameter-varying models, such as nonlinear models linearized at the current operating point, or changes in MPC tuning parameters (such as prediction horizon, control horizon, tuning weights, or sampling time). Often, constructing the optimization problem requires a computational effort comparable to that required for solving the optimization problem itself. The same occurs in the recently proposed data-driven MPC scheme [62] where, due to potentially time-varying model and/or tuning parameters, re-constructing the MPC optimization problem online becomes necessary, which significantly increases the computational load. Notwithstanding these limitations of MPC, scarcely any effort has been made to date to design a real-time MPC approach which does not need (re-

)construction of the optimization problem with varying model and/or MPC tuning parameters. Approaches which partially address this aspect for a limited class of linear parameter-varying (LPV) models with a fixed MPC problem structure include [9, 63].

The methods proposed in this chapter aim at reducing the computational complexity of MPC while eliminating the optimization problem construction step even for the general case of nonlinear parameter-varying systems, through algorithms that can adapt to changes in the model and/or tuning parameters at runtime. The main ideas employed for this purpose are: 1) a structured and sparse formulation of the MPC problem through a quadratic penalty function in order to exploit simple and fast solution methods, 2) replacing matrix instances via abstract operators that map the model and tuning parameters to the result of the required matrix operations in the optimization algorithm. The contributions of this chapter also include methods to exploit problem sparsity and efficiently implement the algorithms proposed in earlier chapters for MPC based on box-constrained (nonlinear) least-squares.

In Chapter 4, it has been shown for the numerically *dense* case that, as compared to solving an LS problem from scratch, employing a recursive QR factorization scheme that exploits the relation between successive LS problems can significantly increase computational speed without compromising numerical robustness, even without using advanced linear-algebra libraries. In the *sparse* case, even though very efficient approaches exist for solving a single LS problem using direct [64] or iterative [65] methods with sparse linear algebra libraries, to the best of the author's knowledge no methods have been reported for recursively updating the sparse QR factorization of a matrix. A recursive approach for sparse LU factorization has been described in [66]; however, such an approach not only needs the storage of the matrix and its sparsity pattern, which requires constructing the MPC problem and forming the normal equations that could be numerically susceptible, but it also relies on linear-algebra packages that could be cumbersome to code, especially in an embedded control platform. In this chapter, we present novel methods for numerically stable sparse recursive QR factorization based on Gram-Schmidt orthogonalization, which are easy to implement and are very efficient even for small-size problems, therefore extending the dense approach discussed in Chapter 4. Although the proposed methods are designed for the specific MPC application, i.e., to solve the sparse LS problems having a specific parameter-dependent structure without forming the matrix that is factorized, they may be applicable for other LS

problems with block-sparse matrices having similar special structures.

This chapter is organized as follows. Section 6.2 describes the considered general class of discrete-time models and MPC problem formulation. A parameterized implementation of the BVNLLS algorithm for solving MPC problems without the construction phase and relying on the abstraction of matrix instances is described in Section 6.3. Methods for sparse recursive thin QR factorization are described in Section 6.4. Section 6.5 briefly reports numerical results based on a nonlinear MPC (NMPC) benchmark example that clearly demonstrate the excellent computational performance of the proposed methods against other methods. Finally, Section 6.6 concludes the chapter.

Notation. For a vector $a \in \mathbb{R}^m$, its jth element is $a(j)$. If $\mathcal{F}$ denotes a set of indices, $A_{\mathcal{F}}$ denotes a matrix formed from columns of A corresponding to the indices in $\mathcal{F}$. Given N square matrices $A_1, \ldots, A_N$, of possible different orders, $\mathrm{blockdiag}(A_1, \ldots, A_N)$ is the block diagonal matrix whose diagonal blocks are $A_1, \ldots, A_N$.

For scalars a and b, $\min(a, b)$ and $\max(a, b)$ denote, respectively, the minimum and maximum of the two values. Depending on the context, $(a, b]$ or $[b, a)$ represent either the set of real numbers or integers between a and b, excluding a and including b.

The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\bar{x} \in \mathbb{R}^n$ is either denoted by $\nabla_x f(x)|_{\bar{x}}$ or $\nabla_x f(\bar{x})$, the Hessian matrix by $\nabla_x^2 f(\bar{x})$; the Jacobian of a vector function $g : \mathbb{R}^n \to \mathbb{R}^m$ by $\mathrm{J}_x g(x)|_{\bar{x}}$ or $\mathrm{J}g(\bar{x})$.

Finite sets of elements are represented by curly braces containing the elements; $\emptyset$ denotes the empty set. If a set $\mathcal{A}$ is a subset of set $\mathcal{B}$ (i.e., if $\mathcal{B}$ is the superset of $\mathcal{A}$), it is written as $\mathcal{A} \subseteq \mathcal{B}$ (or alternatively $\mathcal{B} \supseteq \mathcal{A}$). The symbols $\cup, \cap$, and $\setminus$ between two sets denote, respectively, set union, intersection, and difference. The summation notation for sets is denoted by $\bigcup$. The number of elements of a finite set $\mathcal{A}$ is denoted by $|\mathcal{A}|$.

6.2 Nonlinear parameter-varying model

For maximum generality, the prediction model we use in MPC is described by the following discrete-time multivariable nonlinear parameter-varying dynamical model equation

$$\mathcal{M}(Y_k,\ U_k,\ S_k) = \mathbf{0}, \tag{6.1}$$

where k denotes the current sample step, $U_k = (u_{k-n_\mathrm{b}}, \ldots, u_{k-1})$ with $u \in \mathbb{R}^{n_u}$ the input vector, and $Y_k = (y_{k-n_\mathrm{a}}, \ldots, y_k)$ with $y \in \mathbb{R}^{n_y}$ the output vector. Vector $S_k = (s_{k-n_\mathrm{c}}, \ldots, s_{k-1})$, where $s \in \mathbb{R}^{n_s}$, $n_s \geq 0$, contains possible exogenous signals, such as measured disturbances.

We assume that function $\mathcal{M} : \mathbb{R}^{n_\mathrm{a} n_y} \times \mathbb{R}^{n_\mathrm{b} n_u} \times \mathbb{R}^{n_\mathrm{c} n_s} \to \mathbb{R}^{n_y}$ is differentiable, where $n_\mathrm{a}, n_\mathrm{b}$ and n_c denote the model order. Special cases include deterministic nonlinear parameter-varying auto-regressive exogenous (NLPV-ARX) models, state-space models (y = state vector, $n_a = n_b = n_c = 1$), neural networks with a smooth activation function, discretized first-principles models and differential algebraic equations. Designing the MPC controller based on the input-output (I/O) difference equation (6.1) has several benefits such as: 1) data-based black-box models which are often identified in I/O form do not need a state-space realization for control, 2) a state estimator is not required when all output and exogenous variables are measured, 3) input delays can easily be incorporated in the model by simply shifting the sequence in U_k backwards in time.

The proposed MPC solution methods described later in this chapter use a linearized version of the nonlinear dynamics about a computed sequence of inputs and outputs in each iteration. That is why the model (6.1) was assumed to be differentiable, and we describe its linearized version as follows. Linearizing (6.1) w.r.t. a sequence of inputs $\hat{U}$ (that is, $U_k = \hat{U} + \Delta U$) and outputs $\hat{Y}$ ($Y_k = \hat{Y} - \Delta Y$) gives

$$\mathcal{M}(\hat{Y},\, \hat{U},\, S_k) + \left(\mathrm{J}_{Y_k} \mathcal{M}(Y_k,\, U_k,\, S_k)\big|_{\hat{Y},\, \hat{U}}\right) \Delta Y + \left(\mathrm{J}_{U_k} \mathcal{M}(Y_k,\, U_k,\, S_k)\big|_{\hat{Y},\, \hat{U}}\right) \Delta U = \mathbf{0},$$

which is equivalently written as the following affine parameter-varying I/O model, i.e.,

$$-A\,(S_k)_0\,\Delta y_k = \sum_{j=1}^{n_\mathrm{a}} A\,(S_k)_j\,\Delta y_{k-j} + \sum_{j=1}^{n_\mathrm{b}} B\,(S_k)_j\,\Delta u_{k-j} + \mathcal{M}(\hat{Y},\, \hat{U},\, S_k), \quad (6.2)$$

where the Jacobian matrices

$$A\,(S_k)_j = \mathrm{J}_{y_{k-j}} \mathcal{M}(Y_k,\, U_k,\, S_k)\Big|_{\hat{Y},\,\hat{U}} \in \mathbb{R}^{n_y \times n_y},\ \forall j \in [0, n_a],$$
$$B\,(S_k)_j = \mathrm{J}_{u_{k-j}} \mathcal{M}(Y_k,\, U_k,\, S_k)\Big|_{\hat{Y},\,\hat{U}} \in \mathbb{R}^{n_y \times n_u},\ \forall j \in [1, n_b].$$

Note that for the special case of LTI models in ARX form, in (6.2), A_0 would be an identity matrix whereas S_k would be absent and $\mathcal{M}(\hat{Y},\, \hat{U}) = \mathbf{0}$, $\hat{Y} = \mathbf{0}$, $\hat{U} = \mathbf{0}$.

We recall the following performance index (P) for reference tracking in MPC:

$$\mathrm{P}_k = \sum_{j=1}^{N_\mathrm{p}} \frac{1}{2} \|W_{y_{k+j}}(y_{k+j} - \bar{y}_{k+j})\|_2^2 + \sum_{j=0}^{N_\mathrm{u}-2} \frac{1}{2} \|W_{u_{k+j}}(u_{k+j} - \bar{u}_{k+j})\|_2^2 + \frac{1}{2}(N_\mathrm{p} - N_\mathrm{u} + 1) \cdot \|W_{u_{k+j}}(u_{k+N_\mathrm{u}-1} - \bar{u}_{k+N_\mathrm{u}-1})\|_2^2, \quad (6.3)$$

where vectors $\bar{y}$, $\bar{u}$ denote output and input references, respectively. The methods described later in this chapter can straightforwardly be extended to any performance index which is a sum of squares of linear or C^1 nonlinear functions. We will refer to the MPC optimization problem formulation as described by (3.6) based on the cost function (6.3) subject to constraints on the decision variables. In this chapter we will consider equality constraints that arise from the prediction model (6.1), and restrict inequality constraints to only simple bounds on input and output variables.

6.3 Abstracting matrix instances

6.3.1 Problem structure

The sparse structure of matrices W_k and $\nabla_z h_k(z_k,\, \phi_k)^\top$, which form the Jacobian J of the residual in (3.8), completely depends on the MPC tuning parameters, model order, and the ordering of the decision variables. By ordering the decision variables in vector z_k as follows

$$z_k = \left[u_k^\top\ y_{k+1}^\top\ u_{k+1}^\top\ y_{k+2}^\top\ \cdots\ u_{k+N_\mathrm{u}-1}^\top\ y_{k+N_\mathrm{u}}^\top \,\middle|\, y_{k+N_\mathrm{u}+1}^\top\ \cdots\ y_{k+N_\mathrm{p}-1}^\top\ y_{k+N_\mathrm{p}}^\top\right]^\top \quad (6.4)$$

we get the matrix structure described in (6.5), where the superscript of matrices in parentheses denote the output prediction step the matrices refer to.

$$\mathrm{J}h_k(z) = \nabla_z h_k(z_k,\ \phi_k)^\top =$$

$$\left[\begin{array}{ccccccccccc|ccccc}
B_1^{(1)} & A_0^{(1)} & \mathbf{0} & \mathbf{0} & \cdots & & & & & & & & \cdots & & \mathbf{0} \\
B_2^{(2)} & A_1^{(2)} & B_1^{(2)} & A_0^{(2)} & \mathbf{0} & \cdots & & & & & & & & \cdots & \mathbf{0} \\
 & \vdots & & \ddots & & & & & \ddots & & & & & & \vdots \\
B_{N_\mathrm{u}}^{(N_\mathrm{u})} & A_{N_\mathrm{u}-1}^{(N_\mathrm{u})} & & & & \cdots & & & B_1^{(N_\mathrm{u})} & A_0^{(N_\mathrm{u})} & & \mathbf{0} & \cdots & & \mathbf{0} \\
\hline
B_{N_\mathrm{u}+1}^{(N_\mathrm{u}+1)} & A_{N_\mathrm{u}}^{(N_\mathrm{u}+1)} & & \cdots & & B_3^{(N_\mathrm{u}+1)} & A_2^{(N_\mathrm{u}+1)} & \sum_{i=1}^{2} B_i^{(N_\mathrm{u}+1)} & A_1^{(N_\mathrm{u}+1)} & & & A_0^{(N_\mathrm{u}+1)} & \mathbf{0} & \cdots & \mathbf{0} \\
B_{N_\mathrm{u}+2}^{(N_\mathrm{u}+2)} & A_{N_\mathrm{u}+1}^{(N_\mathrm{u}+2)} & \ddots & & \cdots & B_4^{(N_\mathrm{u}+2)} & A_3^{(N_\mathrm{u}+2)} & \sum_{i=1}^{3} B_i^{(N_\mathrm{u}+2)} & A_2^{(N_\mathrm{u}+2)} & & & A_1^{(N_\mathrm{u}+2)} & A_0^{(N_\mathrm{u}+2)} & \mathbf{0} & \cdots & \mathbf{0} \\
 & \vdots & & \ddots & & \ddots & & \vdots & & & & \vdots & & \ddots & & \mathbf{0} \\
B_{N_\mathrm{p}}^{(N_\mathrm{p})} & A_{N_\mathrm{p}-1}^{(N_\mathrm{p})} & & \cdots & & B_{N_\mathrm{p}-N_\mathrm{u}+2}^{(N_\mathrm{p})} & A_{N_\mathrm{p}-N_\mathrm{u}+1}^{(N_\mathrm{p})} & \sum_{i=1}^{N_\mathrm{p}-N_\mathrm{u}+1} B_i^{(N_\mathrm{p})} & A_{N_\mathrm{p}-N_\mathrm{u}}^{(N_\mathrm{p})} & & & A_{N_\mathrm{p}-N_\mathrm{u}-1}^{(N_\mathrm{p})} & & & A_1^{(N_\mathrm{p})} & A_0^{(N_\mathrm{p})}
\end{array}\right] \tag{6.5}$$

Note that we dropped the parentheses (S_k) in (6.5) to simplify the notation and, as defined in (6.2), $A(S_k)_j = \mathbf{0}$, $\forall j > n_\mathrm{a}$, and $B(S_k)_j = \mathbf{0}$, $\forall j > n_\mathrm{b}$. Clearly, the Jacobian matrix $\mathrm{J}h_k$ of equality constraints only consists of entries from the sequence of linear models of the form (6.2) linearized around the initial guess trajectory. Considering the model parameters $n_\mathrm{a}, n_\mathrm{b}$ to be smaller than N_p in (6.5), as illustrated in Figure 6.1, we observe that the top-left part of $\mathrm{J}h_k$ is block sparse, the bottom-right part has a block-banded structure, the bottom-left part has dense columns corresponding to $u_{k+N_\mathrm{u}-1}$, whereas the top-right part is a zero matrix with $n_y N_\mathrm{u}$ rows and $n_y \cdot (N_\mathrm{p} - N_\mathrm{u})$ columns. If $n_\mathrm{a}, n_\mathrm{b}$ are greater than N_p, then $\mathrm{J}h_k$ would instead have its bottom-left part to be dense with block lower-triangular structure in its top-left and bottom-right parts. All in all, the sparsity pattern of $\mathrm{J}h_k$ is completely defined by the model parameters n_u, n_y, n_a, n_b, and MPC horizons N_u, N_p. Clearly, evaluating $\mathrm{J}h_k$ only requires the sequence of linear models and the sparsity pattern information. Note that in case the linear models are computed by a linearization function, a memory/throughput tradeoff can be chosen here, as they can be either computed once and stored (lowest throughput), or evaluated by the linearization each time they are required (lowest memory allocation). Finally, recalling (3.8), we obtain the full Jacobian matrix

$$J = \begin{bmatrix} W_k \\ \mathrm{J}h_k \end{bmatrix} \tag{6.6}$$

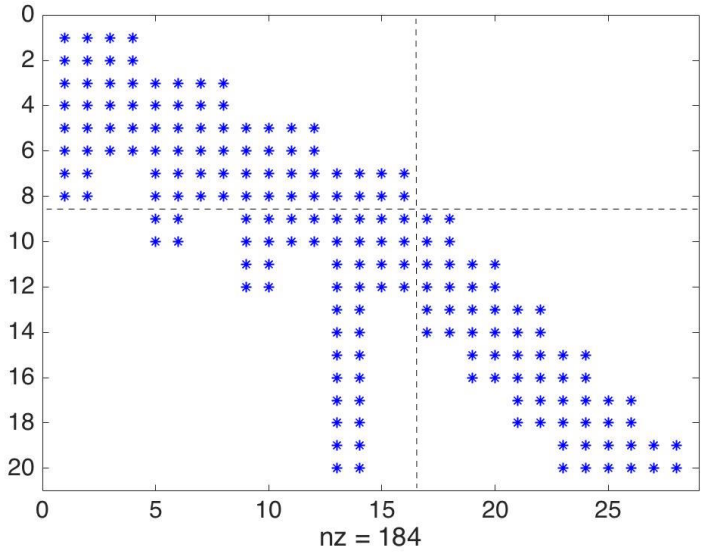


Figure 6.1: Sparsity pattern[12] of $\mathrm{J}h_k$ for a random model with $N_\mathrm{p} = 10$, $N_\mathrm{u} = 4$, $n_\mathrm{a} = 2$, $n_\mathrm{b} = 4$, $n_u = 2$ and $n_y = 2$.

required in Algorithm 5.1, where W_k is the block diagonal matrix

$$\begin{aligned} W_k \quad = \quad & \mathrm{blockdiag}(W_{u_k}, W_{y_{k+1}}, W_{u_{k+1}}, W_{y_{k+2}}, \ldots, \\ & W_{u_{k+N_\mathrm{u}-1}}, W_{y_{k+N_\mathrm{u}}}, W_{y_{k+N_\mathrm{u}+1}}, \ldots, W_{y_{k+N_\mathrm{p}}}) \end{aligned}$$

In the sequel we assume for simplicity that all matrices $W_{u_{(\cdot)}}, W_{y_{(\cdot)}}$ are diagonal, so that W_k is actually a diagonal matrix.

6.3.2 Abstract operators

All matrix-vector operations involving J in Algorithm 5.1 and in the BVLS solver (Algorithm 4.2), including the matrix factorization routines

[12]The figure was originally generated using the `spy` function in MATLAB R2015b.

that will be described in Section 6.4, only need the product of a column-subset of J or a row-subset of $J^\top$ with a vector. Hence, rather than explicitly forming and storing J, all the operations involving J can be represented by two operators `Jix` (ith column of J times a scalar x) and `JtiX` (ith column of J times a vector X) defined by Algorithms 6.1 and 6.2, respectively. The basic principle of both Algorithms 6.1 and 6.2 is to extract non-zero entries indexed in J from the corresponding model coefficients

Algorithm 6.1 Operator `Jix`

Inputs: Output memory $v = \mathbf{0} \in \mathbb{R}^{n_z+N_\mathrm{p}n_y}$; vector w storing diagonal elements of W_k; scalar x; column number i; parameters $n_a,\ n_b,\ n_u,\ n_y,\ N_\mathrm{u}$ and N_p.

$v(i) \leftarrow w(i) \cdot x$;
Find integers $\beta \in [0, N_\mathrm{p})$ and $\eta \in [1,\ n_u + n_y]$ such that $i = \beta n_y + n_u \cdot \min(\beta,\ N_\mathrm{u} - 1) + \eta$;
$\bar{n} \leftarrow N_\mathrm{u}n_u + (N_\mathrm{p} + \beta)n_y$; $m \leftarrow N_\mathrm{u}n_u + 2N_\mathrm{p}n_y$; $j \leftarrow 0$;
if $\beta \neq N_\mathrm{u} - 1$ **or** $\eta > n_u$ **then**
  if $\eta > n_u$, $\bar{m} \leftarrow \bar{n} + n_a n_y$ **else** $\bar{m} \leftarrow \bar{n} + n_b n_y$;
  for $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, \min(\bar{m}, m) - n_y\}$ **do**
    if $\eta > n_u$ **then** $\forall j'' \in \{1, \cdots, n_y\}$,
      $v(j' + j'') \leftarrow x \cdot A_j^{(\beta+j+1)}(j'',\ \eta - n_u)$;
    else
      $v(j' + j'') \leftarrow x \cdot B_{j+1}^{(\beta+j+1)}(j'',\ \eta)$;
    end if
    $j \leftarrow j + 1$;
  end for
else
  for $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, m - n_y\}$ **do**
    $j \leftarrow j + 1$;
    $\bar{B}(j'') \leftarrow \sum_{i'=1}^{\min(j,\ n_b)} B_{i'}^{\beta+j}(j'',\ \eta), \forall j'' \in [1, n_y]$;
    $v(j' + j'') \leftarrow x \cdot \bar{B}(j''), \forall j'' \in [1, n_y]$;
  end for
end if
end.

Output: Vector $v = i$th column of J in (5.2) scaled by x.

Algorithm 6.2 Operator `JtiX`

Inputs: Vector w storing diagonal elements of W_k; vector X; column number i; parameters $n_a,\ n_b,\ n_u,\ n_y,\ N_\mathrm{u}$ and N_p.

1: $v' \leftarrow w(i) \cdot X(i)$;
2: Steps 2-3 of Algorithm 6.1;
3: **if** $\beta \neq N_\mathrm{u} - 1$ **or** $\eta > n_u$ **then**
4: **if** $\eta > n_u, \bar{m} \leftarrow \bar{n} + n_a n_y$ **else** $\bar{m} \leftarrow \bar{n} + n_b n_y$;
5: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, \min(\bar{m}, m) - n_y\}$ **do**
6: **if** $\eta > n_u$ **then** $\forall j'' \in \{1, \cdots, n_y\}$,
7: $v' \leftarrow v' + X(j' + j'') \cdot A_j^{(\beta+j+1)}(j'',\ \eta - n_u)$;
8: **else**
9: $v' \leftarrow v' + X(j' + j'') \cdot B_{j+1}^{(\beta+j+1)}(j'',\ \eta)$;
10: **end if**
11: $j \leftarrow j + 1$;
12: **end for**
13: **else**
14: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, m - n_y\}$ **do**
15: Steps 16-17 of Algorithm 6.1;
16: $v' \leftarrow v' + X(j' + j'') \cdot \bar{B}(j''), \forall j'' \in [1, n_y]$;
17: **end for**
18: **end if**
19: **end.**

Output: $v' =$ inner product of ith row of $J^\top$ in (5.2) and X.

based on the given model and MPC tuning parameters. Since the top part W_k of J is a diagonal matrix, the first non-zero entry in any column of J is obtained from the vector of weights (cf. Step 1 of `Jix` and `JtiX`). The remaining steps only concern evaluating $\mathrm{J}h_k$ as in (6.5), in which the coefficients in each column match the corresponding element in z_k as in (6.4). Referring to the sparsity pattern of $\mathrm{J}h_k$ in (6.5), each of its columns only contains either model coefficients related to the input or to the output, and in the columns corresponding to the inputs $u_{k+N_\mathrm{u}-1}$ some of the input coefficients are summed due to the finite control horizon $N_\mathrm{u} < N_\mathrm{p}$. The location of the first non-zero term in each column of $\mathrm{J}h_k$ depends on the corresponding stage of the input or output variable in prediction, whereas the last entry depends on n_a or n_b and N_p. Hence,

in Step 2 of Algorithm 6.1, the integer β is computed such that $\beta n_y + 1$ is the index of the first non-zero entry in $\mathrm{J}h_k(z)$ (cf. Steps 3, 6 and 15). The integer η computed in the same step denotes the input or output channel to which the column corresponds, in order to accordingly index and extract the coefficients to be scaled as shown in Steps 8, 10 and 17 of Algorithm 6.1. Depending on the column index i of J, computing β and η only needs a trivial number of integer operations including at most one integer division, for instance, if $i \leq N_{\mathrm{u}}(n_u + n_y)$, β is obtained by an integer division of i by $(n_u + n_y)$ and $\eta = i - \beta(n_u + n_y)$. The same computation is straightforward for the only other possible case in which $i > N_{\mathrm{u}}(n_u + n_y)$.

Clearly, since the rows of $J^\top$ are the columns of J, Algorithm 6.2 differs from Algorithm 6.1 only in Steps 7, 9 and 16 in which the scaled coefficient is accumulated to the resulting inner product instead of a plain assignment operation. It is possible to easily extend Algorithm 6.2 for the special case in which X in `JtiX` is the ith column of J i.e., to efficiently compute the 2-norm of the ith column of J, which may be required in the linear-algebra routines. Replacing the instances of J by `Jix` and `JtiX` in the BVNLLS and in the inner BVLS solver has the following advantages:

1. The problem construction step in MPC is eliminated, as matrix J is neither formed nor stored.

2. The code of the two operators does not change with any change in the required data or dimensions as all the indexing steps are parameterized in terms of MPC tuning parameters, i.e., known data. Hence, the resulting optimization solver does not need to be code-generated with a change in problem dimensions or data. The same fact also allows having a real-time variation in the MPC problem data and tuning parameters without any change in the solver. A structural change in the BVNLLS optimization problem formulation, such as the type of performance index, is already decided in the MPC design phase and can be simply accommodated by only modifying Algorithms 6.1 and 6.2.

3. Unlike basic sparse-matrix storage schemes [65] which would store the non-zeros of J along with indexing information, we only store the sequence of linear models at most, resulting in a significantly lower memory requirement. Alternatively, as mentioned earlier, even the coefficients $A_*^{(*)}$, $B_*^{(*)}$ can be generated during the execution of Algorithms 6.1- 6.2 using linearization functions applied on

the current trajectory.

4. The number of floating-point operations (*flops*) involving instances of J, both in the BVNLLS and the BVLS solvers, is minimal and is reduced essentially to what sparse linear-algebra routines can achieve.

5. A matrix-free implementation can be achieved when using Algorithm 4.4 i.e. fast gradient projection [22] on primal problem to solve (5.2) in BVNLLS, as the operators `Jix` and `JtiX` can be used for computing the gradient. In addition, considering that even the model coefficients are optional to store, the resulting NMPC algorithm will have negligible increase in memory requirement w.r.t. the prediction horizon.

As mentioned above, in applications with very restrictive memory requirements, using the aforementioned methods with Algorithm 4.4, one may employ a matrix-free solver similar to [67] and its references. However, when using the gradient-projection algorithm, its low memory usage may come at the cost of slow convergence due to its sensitivity to problem scaling. The next section shows how the sparsity of the Jacobian J can be further exploited for faster computations in the linear algebra methods within the proposed BVLS solver i.e. Algorithm 4.2.

6.4 Sparse recursive thin QR factorization

The primal active-set method for solving BVLS problems proposed in Chapter 4 efficiently solves a sequence of related LS problems using recursive thin QR factorization. The reader is referred to Chapter 4 and references [44, 46] for an overview on thin QR factorization and the recursive update routines in the context of the BVLS solver. This section shows how the sparsity of matrix J can be exploited for significantly reducing the computations involved in the recursive updates of its QR factors, without the use of sparse-matrix storage or conventional sparse linear-algebra routines. The main idea is to have the location of nonzeros in the matrix factors expressed in terms of model and MPC tuning parameters, as described above. We first analyze how the sparse structure of column-subsets of J is reflected in their thin QR factors based on Gram-Schmidt orthogonalization, then characterize the recursive update routines.

6.4.1 Gram-Schmidt orthogonalization

Recall that $J \in \mathbb{R}^{m \times n}$, where $n = N_{\mathrm{u}} n_u + N_{\mathrm{p}} n_y$ and $m = n + N_{\mathrm{p}} n_y$, i.e., $m > n$ (see (3.8), (5.2), (6.4) and (6.5)). Let $J_{\mathcal{F}}$ denote the matrix formed from those columns of J with indices in the set $\mathcal{F}$. Then there exists a unique thin QR factorization [46, Theorem 5.2.3] of $J_{\mathcal{F}}$ which may be expressed via the Gram-Schmidt orthonormalization procedure $\forall i \in [1, |\mathcal{F}|]$ as

$$Q_i' = J_{\mathcal{F}_i} - \sum_{j=1}^{i-1} Q_j Q_j^\top J_{\mathcal{F}_i}, \tag{6.7a}$$

$$Q_i = Q_i' / \left\| Q_i' \right\|_2, \tag{6.7b}$$

$$R(j,i) = Q_j^\top J_{\mathcal{F}_i}, \forall j \in [1, i-1], \tag{6.7c}$$

$$R(i,i) = \left\| Q_i' \right\|_2, \tag{6.7d}$$

where $Q \in \mathbb{R}^{m \times |\mathcal{F}|} \coloneqq [Q_1, Q_2, \cdots, Q_{|\mathcal{F}|}]$ has orthonormal columns, $R \in \mathbb{R}^{|\mathcal{F}| \times |\mathcal{F}|}$ is upper triangular and $J_{\mathcal{F}} = QR$. In (6.7), with a slight abuse of notation, the subscripts denote column number, i.e., Q_i denotes the ith column of Q, whereas $\mathcal{F}_i$ denotes the ith index in $\mathcal{F}$. As shown in (6.7a), starting from the first column of $J_{\mathcal{F}}$, the procedure constructs an orthogonal basis by sequentially orthogonalizing the subsequent columns w.r.t. the basis. The orthogonalization procedure shown in (6.7a) is referred to as the classical Gram-Schmidt (CGS) method [46, Section 5.2.7]. Since the CGS method is practically prone to numerical cancellation due to finite-precision arithmetic, we use the modified Gram-Schmidt (MGS) method [46, Section 5.2.8] in which the orthogonalization is performed using the working value of Q_i' instead of $J_{\mathcal{F}_i}$ in each iteration of the procedure. When applying MGS to solve the linear system before recursive updates, we also orthogonalize the right hand side (RHS) of the equations, i.e., we use an augmented system of equations in order to compensate the orthogonalization error (cf. [47, Chapter 19]). Moreover, for numerical robustness in limited precision, in the proposed MGS procedure a reorthogonalization step is automatically performed which iteratively refines the QR factors for reducing the orthogonalization error in case it exceeds a given threshold (cf. Algorithm 4.3, [44]).

6.4.2 Sparsity analysis

In order to avoid redundant *flops* due to multiplying zero entries while solving the LS problems without sparse storage schemes, we first determine the sparsity pattern of Q and R approximately, based on the relations described in (6.7). While doing so, the following notions will be used.

Definition 6.1 (Non-zero structure) *We define the non-zero structure of a vector x to be the set of indices $\mathcal{S}(x)$ such that $x(i) \neq 0$, $\forall i \in \mathcal{S}(x)$, and $x(j) = 0, \forall j \notin \mathcal{S}(x)$.*

Definition 6.2 (Predicted non-zero structure) *If $\hat{\mathcal{S}}(x)$ denotes the predicted non-zero structure of a vector x, then $x(j) = 0$ $\forall j \notin \hat{\mathcal{S}}(x)$ i.e., $\hat{\mathcal{S}}(x) \supseteq \mathcal{S}(x)$.*

Based on the Definition 6.1, $x = x'$ implies

$$\mathcal{S}(x) = \mathcal{S}(x'). \tag{6.8}$$

$$\mathcal{S}(x' + x'') \subseteq \{\mathcal{S}(x') \cup \mathcal{S}(x'')\}, \tag{6.9}$$

which holds with equality, i.e., $\mathcal{S}(x' + x'') = \{\mathcal{S}(x') \cup \mathcal{S}(x'')\}$, if and only if the set
$\{i | x'(i) + x''(i) = 0, x'(i) \neq 0, x''(i) \neq 0\} = \emptyset$. Likewise,

$$\mathcal{S}(\kappa x) \subseteq \mathcal{S}(x), \kappa \in \mathbb{R},$$

because $\mathcal{S}(\kappa x) = \emptyset$ for $\kappa = 0$, whereas

$$\mathcal{S}(\kappa x) = \mathcal{S}(x), \forall \kappa \in \mathbb{R} \setminus \{0\}. \tag{6.10}$$

Theorem 6.1 *Consider an arbitrary sparse matrix $M \in \mathbb{R}^{n_1 \times n_2}$ of full rank such that $n_1 \geq n_2$ and let $\tilde{Q}$ denote the Q-factor from its thin QR factorization i.e., $M = \tilde{Q}\tilde{R}$. The non-zero structure of each column $\tilde{Q}_i$ of $\tilde{Q}$ satisfies*

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right), \forall i \in [1, n_2], \tag{6.11a}$$

$$\text{and } \mathcal{S}\left(\tilde{Q}_1\right) = \mathcal{S}\left(M_1\right). \tag{6.11b}$$

Proof: We consider the Gram-Schmidt orthogonalization procedure described in (6.7) applied to M with $\mathcal{F} = [1, n_2]$ (this simplifies the notation, i.e., $\mathcal{F}_i = i$). Referring to (6.7b), since $\tilde{Q}'$ represents an orthogonal basis of the full rank matrix M with real numbers, $1/\left\|\tilde{Q}'_i\right\| \neq 0 \; \forall i$, and hence from (6.10),

$$\mathcal{S}\left(\tilde{Q}_i\right) = \mathcal{S}\left(\tilde{Q}'_i\right), \forall i. \tag{6.12}$$

From (6.7a),

$$\tilde{Q}'_1 = M_1. \tag{6.13}$$

Thus, considering (6.13) with (6.8) and (6.12) proves (6.11b). Again, considering (6.7a) with (6.12) and (6.8),

$$\mathcal{S}\left(\tilde{Q}_i\right) = \mathcal{S}\left(M_i - \sum_{j=1}^{i-1} \tilde{Q}_j \tilde{Q}_j^\top M_i\right) = \mathcal{S}\left(M_i + \sum_{j=1}^{i-1} \tilde{Q}_j \kappa_j\right) \tag{6.14}$$

where $\kappa_j = -\tilde{Q}_j^\top M_i \in \mathbb{R}, \forall j \in [1, i-1]$, as κ_j represents the result of an inner product of two real vectors. From (6.9) and (6.14),

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \left\{\mathcal{S}\left(M_i\right) \cup \left\{\bigcup_{j=1}^{i-1} \mathcal{S}\left(\tilde{Q}_j\right)\right\}\right\}. \tag{6.15}$$

Applying (6.15) recursively,

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \left\{\left\{\bigcup_{j=2}^{i} \mathcal{S}\left(M_j\right)\right\} \cup \mathcal{S}\left(\tilde{Q}_1\right)\right\}. \tag{6.16}$$

Thus, substituting (6.11b) in (6.16) completes the proof. □

Corollary 6.1 *Given $i \in [1, n_2]$ and $j' \in [1, n_2]$,*

$$\textit{if } \left\{\bigcup_{j=1}^{j'} \mathcal{S}\left(M_j\right)\right\} \cap \mathcal{S}\left(M_i\right) = \emptyset, \textit{then } \tilde{R}(j, i) = 0 \; \forall j \in [1, j']. \tag{6.17}$$

Proof: Based on result (6.11a) of Theorem 6.1, we can say that $\bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right)$ is a predicted non-zero structure of $\tilde{Q}_i$ i.e.,

$$\bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right) = \hat{\mathcal{S}}\left(\tilde{Q}_i\right), \tag{6.18}$$

and hence

$$\hat{\mathcal{S}}\left(\tilde{Q}_i\right) = \mathcal{S}\left(M_i\right) \cup \hat{\mathcal{S}}\left(\tilde{Q}_{i-1}\right), \forall i \in [1, n_2]. \tag{6.19}$$

If $\mathcal{S}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset$, then $\tilde{Q}_j$ and M_i have disjoint non-zero structures and hence, referring to (6.7c),

$$\mathcal{S}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset \implies R(j,i) = \tilde{Q}_j^\top M_i = 0. \tag{6.20}$$

From (6.19) we have that

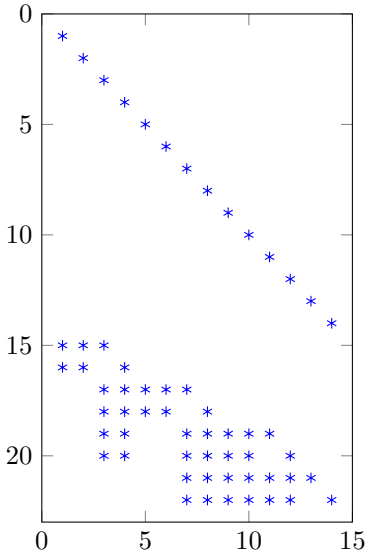
$$\hat{\mathcal{S}}\left(\tilde{Q}_i\right) \supseteq \hat{\mathcal{S}}\left(\tilde{Q}_{i'}\right), \forall i' < i. \tag{6.21}$$

From (6.18), (6.21) and Definition 6.2, i.e., $\hat{\mathcal{S}}\left(\tilde{Q}_i\right) \supseteq \mathcal{S}\left(\tilde{Q}_i\right)$, it follows that $\left\{\bigcup_{j=1}^{j'} \mathcal{S}\left(M_j\right)\right\} \cap \mathcal{S}\left(M_i\right) = \emptyset$ implies $\hat{\mathcal{S}}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset$, $\forall j < j'$. The corollary result is then immediate given (6.20). □

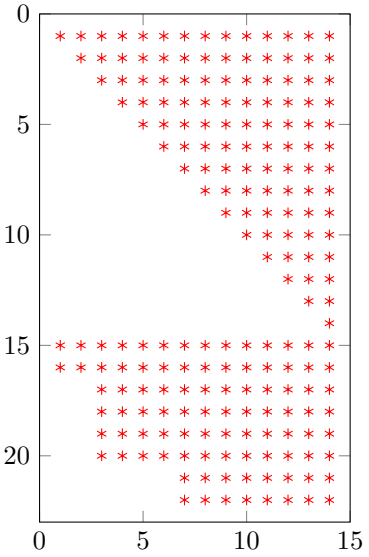
Theorem 6.1 and Corollary 6.1 establish rigorous upper bounds on the non-zero structure of the QR factors based on the non-zero structure of the factorized matrix. Figure 6.2 shows a graphical illustration through which it is easily possible to verify by inspection the relations derived for estimating the sparsity pattern of the thin QR factorization via Theorem 6.1 and Corollary 6.1.

Since the non-zero structure of $J_{\mathcal{F}}$ is completely determined in terms of model and tuning parameters as shown in Section 6.3.2, the predicted non-zero structure of its QR factors consequently depends only on them, as will be shown in the remaining part of this section.

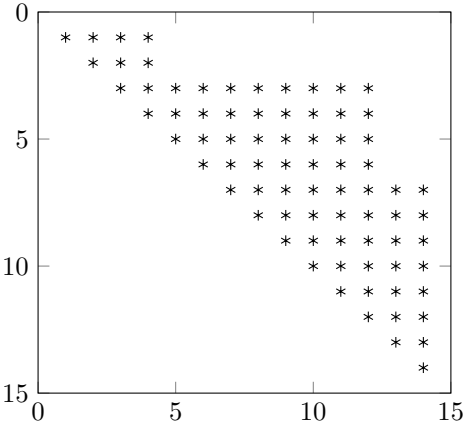
Corollary 6.2 *Consider the matrix $J \in \mathbb{R}^{m \times n}$ whose first n rows form a diagonal matrix and the last $m - n$ rows contain $Jh_k(z)$ as shown in (6.5). Let $J_{\mathcal{F}}$ denote the matrix formed from the columns of J indexed in the index set $\mathcal{F}$ such that $\mathcal{F}_{i+1} > \mathcal{F}_i, \forall i \in [1, |\mathcal{F}|]$. If $Q \in \mathbb{R}^{m \times |\mathcal{F}|}$ denotes the Q-factor from the thin QR factorization of $J_{\mathcal{F}}$, then $\forall i \in [2, |\mathcal{F}|]$, $\left\{\bigcup_{j=1}^{i} \{\mathcal{F}_j\}\right\} \cup \left(\bar{n}_{\mathcal{F}_1}, \max\left(\mathcal{B}_{i-1}, \min\left(\bar{m}_{\mathcal{F}_i}, m\right)\right)\right] = \hat{\mathcal{S}}\left(Q_i\right)$, where the positive integers $\bar{n}_{j'}$, $\bar{m}_{j'}$ respectively denote the values of $\bar{n}$, $\bar{m}$ computed in Steps 2-5 of Algorithm 6.1 for j'th column of J, and $\mathcal{B}$ is an index set such that its ith element stores the largest index of $\hat{\mathcal{S}}\left(Q_i\right)$.*



(a) Sparse matrix $J = QR$.



(b) Q factor of J.



(c) R factor of J.

Figure 6.2: Sparsity pattern of Jacobian J and its thin QR factors, referring (6.6), (6.5), for a random NARX model with non-zero coefficients, non-zero diagonal tuning weights, and parameters $n_y = 2$, $n_u = 2$, $n_\mathrm{a} = 2$, $n_\mathrm{b} = 1$, $N_\mathrm{p} = 4$, $N_\mathrm{u} = 3$. The asterisks mark non-zero entries.

Proof: Considering the structure of matrix J, Definition 6.1 and the fact that $\min(\bar{m}_j, m) > \bar{n}_j \geq n \geq |\mathcal{F}|, \forall j$ by construction, we have that

$$\mathcal{S}(J_{\mathcal{F}_i}) = \{\mathcal{F}_i\} \cup (\bar{n}_{\mathcal{F}_i}, \min(\bar{m}_{\mathcal{F}_i}, m)]. \tag{6.22}$$

From (6.18) we note that $\bigcup_{j=1}^{i} \mathcal{S}(J_{\mathcal{F}_j}) = \hat{\mathcal{S}}(Q_i)$, and using (6.22) we can rewrite

$$\begin{aligned}
\hat{\mathcal{S}}(Q_i) &= \bigcup_{j=1}^{i} \mathcal{S}(J_{\mathcal{F}_j}) \\
&= \left\{ \bigcup_{j=1}^{i} \{\mathcal{F}_j\} \right\} \cup \left\{ \bigcup_{j=1}^{i} (\bar{n}_{\mathcal{F}_j}, \min(\bar{m}_{\mathcal{F}_j}, m)] \right\}, \\
&= \left\{ \bigcup_{j=1}^{i} \{\mathcal{F}_j\} \right\} \cup (\bar{n}_{\mathcal{F}_1}, \mathcal{B}_i], \qquad (6.23)
\end{aligned}$$

because observing (6.5), $\mathcal{F}_{j+1} > \mathcal{F}_j$ implies $\bar{n}_{\mathcal{F}_j} \leq \bar{n}_{\mathcal{F}_{j+1}}$. From result (6.19), (6.22) and definition of set $\mathcal{B}$,

$$\mathcal{B}_i = \max(\mathcal{B}_{i-1}, \min(\bar{m}_{\mathcal{F}_i}, m)), \tag{6.24}$$

which on substitution in (6.23) completes the proof. □

Note that from (6.22) and result (6.11b),

$$\mathcal{S}(Q_1) = \mathcal{S}(J_{\mathcal{F}_1}) = \{\mathcal{F}_1, (\bar{n}_{\mathcal{F}_1}, \min(\bar{m}_{\mathcal{F}_1}, m)]\}. \tag{6.25}$$

By definition of set $\mathcal{B}$ we have $\mathcal{B}_1 = \min(\bar{m}_{\mathcal{F}_1}, m)$ from (6.25), and hence $\mathcal{B}_i$ can be determined $\forall i$ by using (6.24).

Corollary 6.3 *$Q(i,j) = 0 \; \forall i \in [1,n] \setminus \mathcal{F}, \forall j \in [1, |\mathcal{F}|]$. Also, $\forall j' \in [1, |\mathcal{F}|]$, $Q(i,j) = 0 \; \forall j \in [1, j')$ such that $i = \mathcal{F}_{j'}$.*

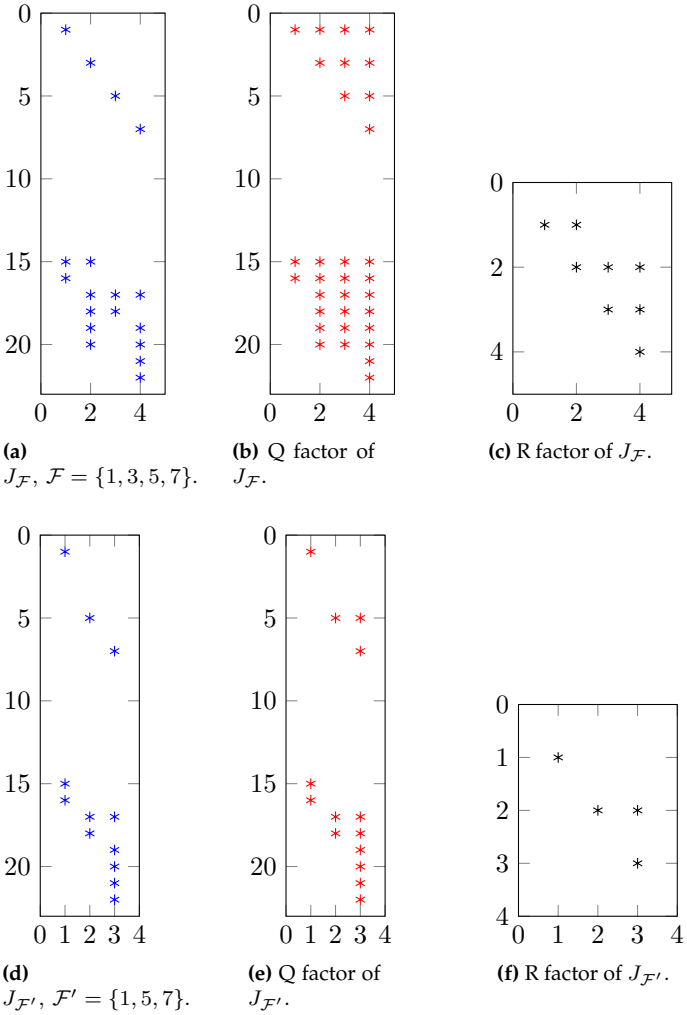
Proof: Let Q'' denote the submatrix formed from the first n rows of Q. Since $\bar{n}_{\mathcal{F}_1} > n$, from Corollary 6.2 we can write $\bigcup_{j=1}^{i} \{\mathcal{F}_j\} = \hat{\mathcal{S}}(Q''_i)$. Thus, referring this relation and Definition 6.2, if an index is not in the set $\mathcal{F}$, the corresponding row of Q'' and hence Q has no non-zero element. The latter part is proved by (6.19) considering the facts that J is diagonal and $\mathcal{F}_{i+1} > \mathcal{F}_i$. □

The relations derived in Corollaries 6.2, 6.3 can be visualized through the graphical illustrations in Figures 6.3, 6.4. For instance, comparing Figures 6.4a-6.4b or 6.4d-6.4e clearly shows that non-zero elements in the first n rows of the thin Q factor are only in those ones indexed in $\mathcal{F}$, as indicated by Corollary 6.3.

From Corollaries 6.2 and 6.3 we infer that the non-zero structure of all the $|\mathcal{F}|$ columns of Q can be stored using a scalar for $\bar{n}_{\mathcal{F}_1}$ and two integer vectors of dimension $|\mathcal{F}|$ containing the index sets $\mathcal{F}$ and $\mathcal{B}$, where $\mathcal{B}_i = \max\left(\min\left(\bar{m}_i, m\right), \bar{m}_{i-1}\right)$. In order to only compute the non-zeros of R, while constructing each of its column, we need to find and store a scalar j' as shown in Corollary 6.1. This is done by using the relations described in Theorem 6.1, Corollary 6.1 and (6.23). Specifically, when computing the ith column of R ($i > 1$), j' is found by counting the number of times $B_j < \bar{n}_{\mathcal{F}_i}$ for increasing values of $j \in \left(\hat{j}, i\right)$ until the condition is not satisfied, where $\hat{j}$ denotes the value of j' for the $(i-1)$th column of R.

6.4.3 Recursive updates

In the primal active-set method, a change in the active-set corresponds to an index inserted in or deleted from the set $\mathcal{F}$. We exploit the uniqueness of thin QR factorization in order to update the structure indicating sets $\mathcal{F}$ and $\mathcal{B}$. When an index t is inserted in the active-set of bounds, the last column of Q and the ith column of R are deleted such that $t = \mathcal{F}_i$, and the QR factorization is updated by applying Givens rotations that triangularize R. In this case $\mathcal{F}$ is simply updated to $\mathcal{F}' = \mathcal{F} \setminus \{t\}$ and $\mathcal{B}$ is updated such that (6.24) is satisfied after removing its ith index. Morover, using Corollary 6.3, the Givens rotations are not applied on the tth row of Q which is simply zeroed. Figure 6.3 shows an example with the exact sparsity pattern of the matrices before and after an update in the active-set. On the other hand, when an index t is removed from the active-set of bounds, $\mathcal{F}$ is updated to $\mathcal{F} \cup \{t\}$ such that $\mathcal{F}_{j+1} > \mathcal{F}_j$, $\forall j$. If t is inserted in $\mathcal{F}$ in the jth position, an index is inserted in the jth position of $\mathcal{B}$ using (6.24) and the elements with position greater than j are updated to satisfy (6.24). Figure 6.4 shows an example where these recursive updates rules can be applied to estimate the sparsity pattern. Since the sparse structure of the updated QR factors is known during recursive updates, using $\mathcal{F}$, $\mathcal{B}$ and Corollary 6.3, the *flops* for applying Givens rotations on rows of Q and matrix-vector multiplications in the Gram-Schmidt (re)orthogonalization procedure are performed only on



(a) $J_{\mathcal{F}}$, $\mathcal{F} = \{1, 3, 5, 7\}$.

(b) Q factor of $J_{\mathcal{F}}$.

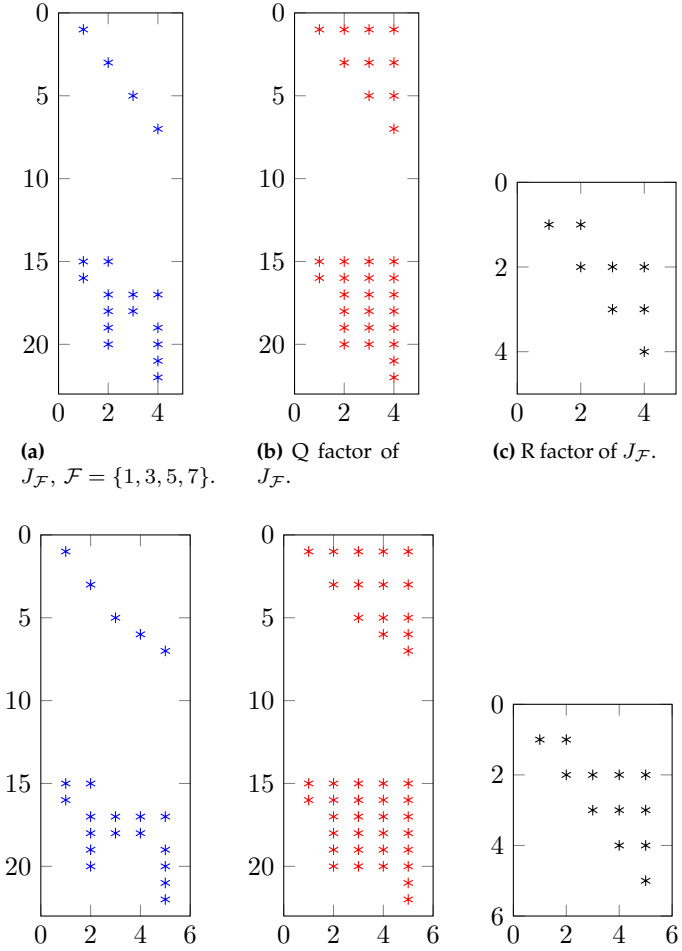
(c) R factor of $J_{\mathcal{F}}$.

(d) $J_{\mathcal{F}'}$, $\mathcal{F}' = \{1, 5, 7\}$.

(e) Q factor of $J_{\mathcal{F}'}$.

(f) R factor of $J_{\mathcal{F}'}$.

Figure 6.3: Illustration showing changes in sparsity pattern of $J_{\mathcal{F}}$ formed from columns of the Jacobian matrix J in Figure 6.2a, and its thin QR factors (obtained without reorthogonalization) when an index is removed from the set $\mathcal{F}$, which is updated to $\mathcal{F}' = \mathcal{F} \setminus \{3\}$. Referring Corollary 6.2, $\bar{n}_{\mathcal{F}_1} = \bar{n}_{\mathcal{F}'_1} = 15$, and the set $\mathcal{B}$ can be recursively updated from $\{16, 20, 20, 22\}$ to $\{16, 20, 22\}$. The asterisks mark non-zero entries.



(a) $J_{\mathcal{F}}$, $\mathcal{F} = \{1, 3, 5, 7\}$.

(b) Q factor of $J_{\mathcal{F}}$.

(c) R factor of $J_{\mathcal{F}}$.

(d) $J_{\mathcal{F}''}$, $\mathcal{F}'' = \{1, 3, 5, 6, 7\}$.

(e) Q factor of $J_{\mathcal{F}''}$.

(f) R factor of $J_{\mathcal{F}''}$.

Figure 6.4: Illustration showing changes in sparsity pattern of $J_{\mathcal{F}}$ formed from columns of the Jacobian matrix J in Figure 6.2a, and its thin QR factors when an index is inserted in the set $\mathcal{F}$, which is updated to $\mathcal{F}'' = \mathcal{F} \cup \{6\}$. Referring Corollary 6.2, $\bar{n}_{\mathcal{F}_1} = \bar{n}_{\mathcal{F}''_1} = 15$, and the set $\mathcal{B}$ can be recursively updated from $\{16, 20, 20, 22\}$ to $\{16, 20, 20, 20, 22\}$. The asterisks mark non-zero entries.

non-zero elements. This makes the QR update routines significantly faster as is reflected in the numerical results described in Section 6.5.

6.4.4 Advantages and limitations

The predicted non-zero structure of the Q-factor via (6.19) is exact if and only if the set relation (6.11a) holds with equality. For (6.11a) to hold with equality for Q, $Q_j^\top J_{\mathcal{F}_i}$ must be non-zero for all pairs of indices i and j referring the CGS orthogonalization in (6.7a) and moreover the summation of non-zeros in the RHS of (6.7a) must result in a non-zero. Even though theoretically this may not be the case for the matrices that we consider, due to finite precision computations which disallow perfect orthogonality, and the use of MGS with potentially multiple orthogonalizations to compute columns of Q, the predicted non-zero structure of columns of Q via Corollary 6.2 rarely contains indices of zero elements, i.e., numerically it is an accurate estimate and often the exact non-zero structure. Referring to Corollary 6.1 and Algorithm 4.3, the same fact leads to the conclusion that if multiple orthogonalizations (for numerical robustness) are performed, in the worst case, the upper-triangular part of the R factor may have no zero elements. Nevertheless, the initial sparsity in R before reorthogonalization is still exploited in its construction but the worst-case fill-in makes it necessary to use R as a *dense* upper-triangular matrix when solving the triangular system by back-substitution to compute the solution of the underlying LS problem.

From Theorem 6.1, we observe that the predicted non-zero structure of columns $Q_j, \forall j \geq i$, would contain at least the indices of non-zero elements in the ith column of $J_{\mathcal{F}}$. Hence, in case $N_\mathrm{u} < N_\mathrm{p}$, referring the analysis in Section 6.4.2, the fill-in of Q can be reduced by a re-ordering of the decision variable vector in (6.4) such that the columns of J corresponding to the variables $u_{k+N_\mathrm{u}-1}$ are moved to become its last columns. Note that even though this re-ordering does not optimize the fill-in of Q, for which dedicated routines exist in literature (cf. [64]), it still allows a relatively simple and a computationally effective implementation of recursive thin QR factorization for the matrix of interest through a straightforward extension of the methods described in Section 6.4.3.

In order to benefit computationally from the recursive updates, a full storage of the thin QR factors is required. This causes greater memory requirement beyond a certain large problem size where a sparse-storage scheme would need smaller memory considering that with conventional sparse linear algebra, one would only compute and store the R factor

while always solving the LS problem from scratch instead. However, the latter approach could turn out to be computationally much more expensive. Using the techniques discussed in Sections 6.4.2 and 6.4.3 with a sparse-storage scheme could address this limitation specific to large-scale problems for memory-constrained applications but it needs a much more intricate implementation with cumbersome indexing, that is beyond the scope of this thesis.

6.5 Numerical results

6.5.1 Software framework

In order to implement the (nonlinear) MPC controller based on formulation (3.8) or (3.10), one only needs the code for Algorithm 5.1. The inner BVLS solver may be alternatively replaced by another algorithm that exploits sparsity via the abstract operators for instance the fast gradient projection algorithm we mentioned earlier i.e., Algorithm 4.4. Besides, routines that evaluate the model (6.1) and the Jacobian matrices i.e., the model coefficients in (6.2) are required from the user in order to evaluate the residual and perform the linearization step (or alternatively finite-differences) in BVNLLS. Note that an optimized self-contained code for these routines can easily be generated or derived by using symbolic tools such as those of MATLAB or the excellent open-source software CasADi [68]. This signifies that, except for the user-defined model and tuning parameters, the software does not need any code-generation, as for a given class of performance indices the code for Algorithms 5.1, 6.1-6.2 does not change with the application. The user is only required to provide the MPC tuning parameters and a symbolic expression for the model (6.1), which eases the deployment of the proposed MPC solution algorithm in embedded control hardware.

6.5.2 Computational performance

The results presented in this section are based on a library-free C implementation of BVNLLS based on Algorithms 6.1 and 6.2, and the BVLS solver based on sparse recursive thin QR factorization routines discussed in Section 6.4. The reader is referred to Section 5.4 for details on simulation settings and benchmark solvers related to the following discussion. All the non-convex optimization problems in the simulations referred

below were solved until convergence, on a Macbook Pro equipped with 8GB RAM and 2.6 GHz Intel Core i5 processor. Figure 6.5 shows that the proposed methods allow BVNLLS to outperform its *dense* linear algebra based variant even on small-sized test problems by almost an order of magnitude on average. As compared to other solvers which are instead applied to the benchmark formulation (3.6), i.e. the SQP solver (`fmincon`) of MATLAB and the interior-point solver (IPOPT) of [61], a reduction in computational time by around two orders of magnitude is observed for the small-sized test problems. This reduction can be credited to the fact that IPOPT, which is based on sparse linear-algebra routines, is more effective for large-sized problems, and that BVNLLS exploits warmstarts based on the previously computed solution which is provided from the second instance onwards. Figure 6.6 suggests that

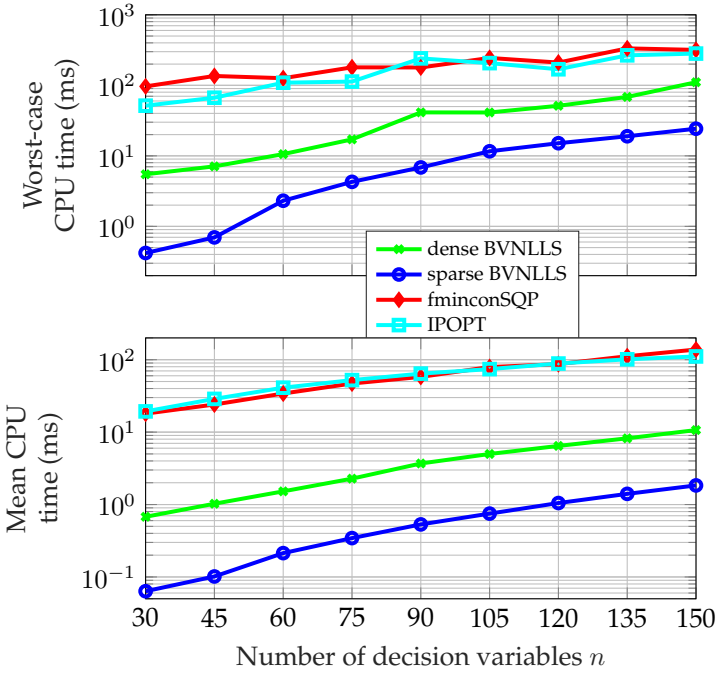


Figure 6.5: Computational time spent by each solver during NMPC simulation of CSTR for increasing values of $N_\mathrm{p} = N_\mathrm{u} = n/3$, n set of box-constraints and $2N_\mathrm{p}$ equality constraints.

despite being based on an active-set algorithm, the proposed sparsity-exploiting methods empower BVNLLS to significantly outperform the benchmarks even for large-sized problems.

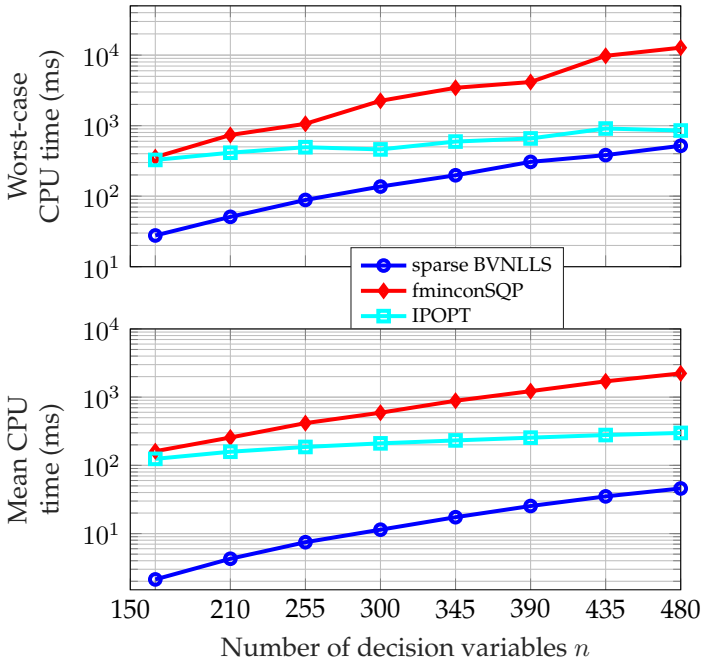


Figure 6.6: Computational time spent by each solver during NMPC simulation of CSTR for large values of $N_\mathrm{p} = N_\mathrm{u} = n/3$ with n set of box-constraints and $2N_\mathrm{p}$ equality constraints.

6.6 Conclusions

This chapter presented a new approach to solving constrained linear and nonlinear MPC problems that, by relaxing the equality constraints generated by the prediction model into quadratic penalties, allows the use of a very efficient bounded-variable nonlinear least squares solver. The linear algebra behind the latter has been specialized in detail to take into account the particular structure of the MPC problem, so that the resulting required memory footprint and throughput are minimized for efficient

real-time implementation, without the need of any external advanced linear algebra library.

Chapter 7

Conclusions

The main objective of this thesis was to address issues that limit the application of model predictive control (MPC) in fast-sampled systems with nonlinear and/or varying dynamics, resource constraints, and particular requirements for embedded implementation. Considering the significant applicability of data-driven or black-box linear models which are often identified in input/output (I/O) form, we developed an MPC approach that directly employs them for greater computational efficiency, unlike modern linear MPC design approaches which rely on their state-space realization. The results presented in this thesis suggest that formulating the MPC optimization problems in a structured non-condensed manner that allows the application of fast solution algorithms, can give rise to implementations that may be significantly faster than existing approaches for linear and nonlinear MPC. Research on algorithms that exploit sparsity in the MPC optimization problems has been mainly on first-order and interior-point solvers in literature, with scant attention to problems with varying model and MPC tuning parameters. In this thesis, we have shown that by implementing active-set algorithms which take advantage of warmstarts and the sparse problem structure by efficiently updating matrix factorizations, it is possible to achieve a better numerical performance and adaptation to MPC parameters compared to many other existing algorithms, even for large problem sizes. Moreover, this is achieved without using external linear algebra packages, and with numerical robustness in single precision, which is often necessary to have for code deployment in industrial embedded hardware platforms. These results are thus expected to broaden

the scope of MPC in applications where resource constraints may have formerly restricted its use.

7.1 Summary of contributions

- Chapter 2 presented two main ideas related to linear MPC. The first idea was to show how the MPC optimization problem can be formulated based on a linear I/O model in order to achieve a faster performance as compared to the conventional state-space based approach. The second idea focussed on formulating the problem in a way that grants the application of fast and simple optimization solvers while handling infeasibility. We also showed how stability enforcing constraints can be imposed with such a formulation based on I/O models. The results demonstrated that having a simple sparse formulation allows fast construction of the optimization problem which is suitable for linear parameter-varying MPC.

- The results for linear MPC were extended to the nonlinear case in Chapter 3 where we discussed simple box-constrained nonlinear least-squares problem formulations that result in a special sparse structure of the matrices involved. The main idea was to use penalty functions that eliminate nonlinear equality constraints. The same chapter also included a discussion on the relation between an iteration of the quadratic penalty method (QPM) with the bound-constrained augmented-Lagrangian approach. It was shown via a numerical example that using the proposed approach that is a single iteration of the QPM, infeasibility can be implicitly handled and at the same time the solution can be achieved with negligible suboptimality by appropriately tuning the penalty parameter.

- The latter part of the thesis focussed on fast solution algorithms that maximize the benefits of the proposed formulations. Chapter 4 presented a primal active-set method based solver, which is very efficient due to a novel implementation of linear algebra methods. Specifically, numerically stable methods for recursive thin QR factorization were discussed in context of the primal active-set algorithm. A numerical comparison of the solver with existing methods was included along with a discussion on hardware implementation aspects.

- Chapter 5 focussed on the proposed algorithm for solving box-constrained nonlinear least-squares problems. Specifically, we proposed a simple line-search algorithm that employs the BVLS solver discussed in Chapter 4 for computing the search directions. It was shown with a theoretical proof that the proposed algorithm is globally convergent. Numerical results showed that with the proposed implementation, a superior computational performance can be achieved w.r.t. the benchmark solvers.

- The solvers discussed in Chapters 4-5 focussed on algorithmic development without taking into account the sparse nature of the non-condensed MPC problems. This issue was addressed in Chapter 6, where we developed new methods for exploiting problem sparsity using the proposed solvers. It was shown that while doing so, the problem construction step in MPC design can be eliminated by abstracting matrix instances in the solution algorithm, which is parameterized in terms of model and tuning parameters. This allows for significantly fast implementations which are also very suitable for applications where model and tuning parameters may vary in real time. These methods thus make it possible to have a stand-alone algorithmic framework free from code-generation requirements for linear, nonlinear and adaptive MPC variants based on a given problem formulation. Moreover, the same framework allows to trade-off memory and computations by appropriately choosing the BVLS solver and fine-tuning the code of abstract matrix operators.

- Another important novel contribution of this thesis was the development of efficient methods for sparse recursive thin QR factorization, for their application in the proposed active-set algorithms in relevance to the considered non-condensed MPC formulations. In active-set methods for box-constrained least-squares problems, the overdetermined linear system solved in each iteration has varying size and sparsity pattern, making it complicated to exploit sparsity. We theoretically analyzed the sparsity pattern and its variation in each iteration by using the matrix relations from Gram-Schmidt orthogonalization, and proposed easy to implement tricks through which maximum benefits of problem sparsity can be derived. This content was also included in Chapter 6 where numerical results demonstrate that these techniques result in considerably faster im-

plementations of the proposed solvers, which may significantly outperform existing benchmarks.

7.2 Open problems for future research

Considering that the methods discussed in this thesis add contributions to the existing literature on optimization for MPC, in this section we discuss possible extensions and relevant problems that remain to be addressed, for which further research is highly encouraged.

- **Broader implementation:** Although the proposed MPC approach is based on a very broad class of models, the discussion in this thesis was limited to a quadratic performance index for reference tracking problems and to simple bounds on variables as constraints. It was shown that the same methods can be extended for problems with different sum-of-squares cost functions and with inequality constraints of general type if their slight relaxation is tolerable. However, such a generic implementation needs further effort, which may be motivated in future by applications with such specifications.
- **Matrix-free implementation:** Methods for achieving an NMPC solver with scant memory usage were discussed in this thesis but not implemented. A comparison of such methods with existing implementations would give a broader insight on the trade-off between memory and computations. The author plans to test such algorithms in future on industrial embedded hardware platforms.
- **Methods to avoid ill-conditioning:** The quadratic penalty function based formulation of MPC problems inherently causes the linear systems solved in the proposed BVLS algorithm to have a high condition number. For this issue, numerically stable linear algebra methods were discussed in Chapter 4. Specifically, orthogonal decomposition methods were discussed, for which the condition number of the factorized matrix is square root as compared to the matrix of normal equations. However, using the techniques described in [11, Section 17.1], one can reformulate the normal equations to a well-conditioned system of linear equations in higher dimension (cf. [11, Equations 17.20-17.21]) through the introduction of as many additional variables as the number of equality constraints (that are eliminated through the quadratic penalty).

Using this technique with the methods proposed in this thesis, the resulting reformulated linear systems would need the (recursive) factorization of a symmetric positive definite matrix, which would change in BVLS iterations only by insertion or deletion of a given row and column. This can be efficiently solved via recursive Cholesky factorization or alternatively by recursive QR factorization. The advantage of numerical robustness with this approach, which was not discussed in this thesis, comes at the cost of solving a larger linear system in each iteration of the BVLS solver. Moreover, development of methods to avoid potential numerical error accumulation if one uses recursive Cholesky factorization and methods to exploit sparsity while using this approach, remain as unsolved challenges. Thorough research is required to quantify the pros and cons of this approach, which could be a useful alternative for the linear algebra involved in BVLS for QPM based MPC in embedded platforms relying on low precision computing.

- **Bound-constrained augmented-Lagrangian method (BLM):** Even though we discussed in Chapter 3 that this method has limitations concerning computational efficiency, and convergence in case of problem infeasibility, it is an interesting approach considering that general (nonlinear) constraints may be handled without relaxation. This framework was barely explored and was not tested in this thesis but given its ability to handle such constraints precisely, it is expected to perform well in cases where the QPM based approach cannot be applied, for instance, to generate hot-starts for mixed-integer problems or hybrid MPC problems [69], by considering the integer variables as (real) continuous ones subject to strict quadratic equality constraints. This is based on the fact that for sum-of-squares cost functions, the BLM subproblems can be formulated as box-constrained nonlinear least-squares problems as shown in Chapter 3 in order to benefit from the efficient solution methods already described in this thesis (cf. Chapters 5 and 6).

- **Stability analysis for nonlinear MPC:** The focus of this thesis was partial towards efficient solution methods for MPC and discussion on stability analysis was limited to the linear time-invariant MPC case, where we proved that under certain conditions stability can be enforced despite of relaxtion of the equality constraints. Unfortunately, this issue was not addressed for the nonlinear MPC case. However, considering that the relaxation of the nonlinear equality

constraints due to suboptimality can also be considered as an uncertainty in the prediction model, there are relevant articles in the literature which might tackle this theoretical issue; specifically, the reader is referred to the work of D. Limon et al., cf. [70, 71].

- **Numerical analysis and exact complexity certification:** Limited numerical precision in embedded platforms can be a challenging problem from an implemention perspective. This thesis partially addressed this issue in Chapter 4 by using numerically stable linear algebra routines for BVLS, which may work well in practice. However, a thorough complexity certification analysis is required for establishing practical guarantees by knowing the worst case number of iterations through which worst-case numerical error bounds may also be derived. Several research articles existing in the recent literature address this problem based on complexity certification of the optimization algorithm and aid in deriving bounds on numerical error for a fixed-point implementation. In [52, 72–74] such results have been established for first-order methods whereas interestingly, in [75, 76] these results are discussed for active-set QP algorithms. An extension of such results to the proposed BVLS algorithm would significantly increase its appeal from a practical viewpoint. However, this topic needs much further research in order to verify whether such an extension is possible.

References

[1] M. Rafal and W. Stevens, "Discrete dynamic optimization applied to on-line optimal control," *AiChE Journal*, vol. 14, no. 1, pp. 85–91, 1968. 1

[2] S. Joe Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733 – 764, 2003. 1

[3] A. Bemporad, D. Bernardini, R. Long, and J. Verdejo, "Model predictive control of turbocharged gasoline engines for mass production," in *WCX^{TM}: SAE World Congress Experience*, Detroit, MI, USA, Apr. 2018. 1

[4] S. Di Cairano and I. V. Kolmanovsky, "Real-time optimization and model predictive control for aerospace and automotive applications," in *Proc. Annual American Control Conference (ACC)*, Milwaukee, WI, 2018, pp. 2392–2409. 1

[5] M. Diehl, H. Bock, and J. Schlöder, "A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005. 1, 32, 42, 93

[6] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014. 1

[7] S. J. Wright, *Efficient Convex Optimization for Linear MPC*. Cham: Springer International Publishing, 2019, pp. 287–303. [Online]. Available: https://doi.org/10.1007/978-3-319-77489-3_13 1, 48

[8] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, "Recent advances in quadratic programming algorithms for nonlinear model predictive control," *Vietnam Journal of Mathematics*, Sept. 2018. 1, 48

[9] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010. 1, 12, 102

[10] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017. 2, 8, 19, 22, 28, 36, 48, 95

[11] J. Nocedal and S. Wright, *Numerical Optimization*. 2nd ed. Springer, 2006. 2, 8, 22, 23, 32, 33, 37, 38, 40, 49, 50, 52, 58, 90, 92, 93, 94, 99, 129

[12] N. Saraf and A. Bemporad, "Fast model predictive control based on linear input/output models and bounded-variable least squares," in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1919–1924. 2, 4

[13] N. Saraf, M. Zanon, and A. Bemporad, "A fast NMPC approach based on bounded-variable nonlinear least squares," in *Proc. 6th IFAC Conference on Nonlinear Model Predictive Control*, Madison, WI, August 2018, pp. 337–342. 2, 4, 5

[14] N. Saraf and A. Bemporad, "A bounded-variable least-squares solver based on stable QR updates," *IEEE Transactions on Automatic Control*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8747522 2, 5

[15] N. Saraf and A. Bemporad, "An efficient non-condensed approach for linear and nonlinear model predictive control with bounded variables," in arXiv preprint arXiv:1908.07247, 2019. [Online]. Available: https://arxiv.org/abs/1908.07247 2, 4, 5, 6

[16] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4, pp. 667–682, 1999. 7

[17] J. K. Huusom, N. K. Poulsen, S. B. Jørgensen, and J. B. Jørgensen, "Tuning of methods for offset free MPC based on ARX model representations," in *Proc. American Control Conference (ACC)*, 2010, pp. 2355–2360. 7

[18] E. Kerrigan and J. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," in *Proc. UKACC International Conference (Control)*, Cambridge, UK, 2000. 8, 36, 40, 48

[19] P. Scokaert and J. Rawlings, "Feasibility issues in linear model predictive control," *AIChE J.*, vol. 45, no. 8, pp. 1649–1659, 1999. 8, 33, 48

[20] P. Stark and R. Parker, "Bounded-variable least-squares: An algorithm and applications," *Computational Statistics*, vol. 10, pp. 129–141, 1995. 8, 48, 49, 50, 51, 52, 53, 56, 58, 65

[21] A. Björck, *Numerical Methods for Least Squares Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1996. 8, 33, 52, 61, 90, 92

[22] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Dordrecht, The Netherlands: Kluwer Academic, 2004. 8, 48, 65, 80, 111

[23] E. Mosca, J. M. Lemos, and J. Zhang, "Stabilizing I/O receding horizon control," in *Proc. 29th IEEE Conference on Decision and Control*, vol. 4, 1990, pp. 2518–2523. 8

[24] A. Bemporad, L. Chisci, and E. Mosca, "On the stabilizing property of SIORHC," *Automatica*, vol. 30, no. 12, pp. 2013 – 2015, 1994. 8, 9, 27

[25] L. Ljung, *System Identification : Theory for the User*. 2nd ed. Prentice Hall, 1999. 9

[26] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides, "A condensed and sparse QP formulation for predictive control," in *Proc. 50th IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 5217–5222. 12

[27] P. Kapasouris, M. Athans, and G. Stein, "Design of feedback control systems for unstable plants with saturating actuators," in *Proc. IFAC Symp. Nonlinear Contr. Syst. Design*, 1989. 28

[28] A. Bemporad, "Hybrid Toolbox - User's Guide," http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox, Dec. 2003. 28

[29] M. Cannon, "Efficient nonlinear model predictive control algorithms," *Annual Reviews in Control*, vol. 28, no. 2, pp. 229–237, 2004. 32

[30] B. Houska, H. Ferreau, and M. Diehl, "ACADO toolkit - An opensource framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011. 32, 33

[31] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A Simple and Efficient Algorithm for Nonlinear Model Predictive Control," in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1939–1944. 32

[32] M. Diehl, H. Ferreau, and N. Haverbeke, "Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation," in *Nonlinear Model Predictive Control. Lecture Notes in Control and Information Sciences*, L. Magni, D. Raimondo, and F. Allgöwer, Eds. Berlin, Heidelberg: Springer, 2009, vol. 384, pp. 56–98. 32, 93

[33] T. Ohtsuka, "A continuation/GMRES method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563–574, 2004. 32

[34] W. Li and L. Biegler, "A Multistep, Newton-type control strategy for constrained, nonlinear processes," in *1989 American Control Conference*, Pittsburgh, PA, USA, 1989, pp. 1526–1527. 32

[35] V. Zavala and L. Biegler, "The advanced step NMPC controller: Optimality, stability and robustness," *Automatica*, vol. 45, no. 1, pp. 86–93, 2009. 32

[36] I. Leontaritis and S. Billings, "Input-output parametric models for nonlinear systems. Part i: deterministic non-linear systems," *International Journal of Control*, vol. 41, pp. 303–328, 1985. 34

[37] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996. 37, 90

[38] D. Seborg, T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*. 2nd ed. Wiley, 2004. 40

[39] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear MPC: bridging the gap via the real-time iteration," *International Journal of Control*, pp. 1–19, 2016. 42, 93

[40] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, pp. 18–33, 2014. 48

[41] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011. 48

[42] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1906–1911. 48, 65, 66

[43] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, ser. Classics in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1995. 48, 52, 53, 58

[44] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, "Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization," *Mathematics of Computation*, vol. 30, no. 136, pp. 772–795, 1976. 49, 53, 59, 60, 61, 111, 112

[45] A. Bemporad, "A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1111–1116, 2016. 49, 52

[46] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, MD: 4th ed. The John Hopkins University Press, 2013. 52, 53, 56, 61, 62, 64, 111, 112

[47] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997. 52, 53, 56, 57, 61, 112

[48] J. Stoer, "On the numerical solution of constrained least-squares problems," *SIAM Journal on Numerical Analysis*, vol. 8, no. 2, pp. 382–411, 1971. 52

[49] K. Schittkowski and J. Stoer, "A factorization method for the solution of constrained linear least squares problems allowing subsequent data changes," *Numerische Mathematik*, vol. 31, pp. 431–463, 1979. 52

[50] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014. 65

[51] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com 65

[52] S. Richter, *Computational complexity certification of gradient methods for real-time model predictive control*. ETH Zurich, Switzerland: Ph.D. dissertation, 2012. 65, 66, 80, 131

[53] G. Cimini, A. Bemporad, and D. Bernardini, "ODYS QP Solver," ODYS S.r.l. (https://odys.it/qp), Sept. 2017. 66

[54] IBM ILOG CPLEX V12.1: User's Manual for CPLEX, International Business Machines Corporation, 2009. 66

[55] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated algorithms for nonlinear model predictive control on long and on short horizons," in *52nd IEEE Conference on Decision and Control*, Florence, Italy, 2013, pp. 5113–5118. 70

[56] R. Quirynen, A. Knyazev, and S. Di Cairano, "Block structured preconditioning within an active-set method for real-time optimal control," in *Proc. 2018 European Control Conference (ECC)*, Limassol, Cyprus, 2018, pp. 1154–1159. 70

[57] "Product datasheet BMXP3420302," Schneider Electric SE, France, 2019, accessed on September 5, 2019. [Online]. Available: https://www.schneider-electric.com/en/product/download-pdf/BMXP3420302 73

[58] P. Krupa, D. Limon, and T. Alamo, "Implementation of Model Predictive Controllers in Programmable Logic Controllers using IEC 61131-3 standard," in *Proc. European Control Conference (ECC)*, Limassol, Cyprus, 2018, pp. 1–6. 74, 76

[59] W. A. Altabey, "Model optimal control of the four tank system," *International Journal of Systems Science and Applied Mathematics*, vol. 1, no. 4, pp. 30–41, 2016. 74

[60] S. Bellavia, M. Macconi, and B. Morini, "An affine scaling trust-region approach to bound-constrained nonlinear systems," *Applied Numerical Mathematics*, vol. 44, no. 3, pp. 257 – 280, 2003. 96

[61] A. Wächter and L. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006. 99, 123

[62] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven MPC design," *IEEE control systems letters*, vol. 3, no. 3, pp. 577–582, 2019. 101

[63] L. Cavanini, G. Cimini, and G. Ippoliti, "Computationally efficient model predictive control for a class of linear parameter-varying systems," *IET Control Theory & Applications*, vol. 12, no. 10, pp. 1384–1392, 2018. 102

[64] T. Davis, *Direct Methods for Sparse Linear Systems*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2006. 102, 121

[65] Y. Saad, *Iterative Methods for Sparse Linear Systems*. 2nd ed. Society for Industrial and Applied Mathematics, 2003. 102, 110

[66] J. Dongarra, V. Eijkhout, and P. Łuszczek, "Recursive approach in sparse matrix LU factorization," *Sci. Program.*, vol. 9, no. 1, pp. 51–60, 2001. [Online]. Available: http://dx.doi.org/10.1155/2001/569670 102

[67] S. Diamond and S. Boyd, *Matrix-Free Convex Optimization Modeling*. In: Goldegorin B. (eds) Optimization and Its Applications in Control and Data Sciences. Springer Optimization and Its Applications, vol 115. Springer, Cham, 2016, pp. 221–264. [Online]. Available: https://doi.org/10.1007/978-3-319-42056-1_7 111

[68] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. 122

[69] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407 – 427, 1999. 130

[70] D. Limon, T. Alamo, and E. F. Camacho, "Input-to-state stable MPC for constrained discrete-time nonlinear systems with bounded additive uncertainties," in *Proc. 41st IEEE Conference on Decision and Control*, vol. 4, 2002, pp. 4619–4624. 131

[71] D. Limon, T. Alamo, D. M. Raimondo, D. Muñoz de la Peña, J. M. Bravo, A. Ferramosca, and E. F. Camacho, *Input-to-State Stability: A Unifying Framework for Robust Model Predictive Control*. Berlin, Heidelberg: Springer, 2009, pp. 1–26. [Online]. Available: https://doi.org/10.1007/978-3-642-01094-1_1 131

[72] V. Nedelcu, I. Necoara, and Q. Tran-Dinh, "Computational Complexity of Inexact Gradient Augmented Lagrangian Methods: Application to Constrained MPC," *SIAM Journal on Control and Optimization*, vol. 52, no. 5, pp. 3109–3134, 2014. 131

[73] I. Necoara, A. Patrascu, and F. Glineur, "Complexity of first-order inexact Lagrangian and penalty methods for conic convex programming," *Optimization Methods and Software*, vol. 34, no. 2, pp. 305–335, 2019. 131

[74] M. Rubagotti, P. Patrinos, A. Guiggiani, and A. Bemporad, "Real-time model predictive control based on dual gradient projection: Theory and fixed-point FPGA implementation," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 15, pp. 3292–3310, 2016. 131

[75] G. Cimini and A. Bemporad, "Complexity and convergence certification of a block principal pivoting method for box-constrained quadratic programs," *Automatica*, vol. 100, pp. 29 – 37, 2019. 131

[76] G. Cimini and A. Bemporad, "Exact Complexity Certification of Active-Set Methods for Quadratic Programming," *IEEE Transactions on Automatic Control*, vol. 62, no. 12, pp. 6094–6109, 2017. 131