Imperial College London

Numerical Methods for Model Predictive Control

Ian Scott McInerney

March 2022

Supervisors: Prof. Eric C. Kerrigan
Prof. George A. Constantinides

Department of Electrical and Electronic Engineering
Imperial College London

This thesis is submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy of Imperial College London

Declaration of originality

The work presented hereafter is based on research carried out by the author at Imperial College London. It is the sole work of the author, except where otherwise acknowledged, and was conducted under the supervision of their PhD supervisors. The reuse of the author's own published works are acknowledged according to the publishers' guidelines.

For the IEEE copyrighted works that are contained inside this thesis, the IEEE does not endorse any of Imperial College London's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

Copyright declaration

Abstract

There has been an increased interest in controlling complex systems using Model Predictive Control (MPC). However, the use of resource-constrained computing platforms in these systems has slowed the adoption of MPC. This thesis focuses on increasing the efficiency of numerical methods for MPC in terms of resource usage and solution time, while also simplifying the design process.

We first show how block Toeplitz operators can be used to link the linear MPC matrices to the transfer function of the predicted system, resulting in horizon-independent bounds on the condition number of the condensed Hessian and the upper iteration bound for the Fast Gradient Method (FGM). We derive a horizon-independent preconditioner that produces up to a 9x speedup for the FGM while reducing the preconditioner computation time by up to 50,000x compared to an existing preconditioner. We propose a new method for computing the minimum number of fractional bits needed to ensure the FGM with fixed-point arithmetic is stable, with an example showing decreases of up to 77% in resource usage and 50% in the computational energy when using this method on a Field Programmable Gate Array.

Finally, we present a framework using the derivative-free Mesh Adaptive Direct Search method to solve nonlinear MPC problems with non-differentiable features or quantized variables without the need for complex or costly reformulations. We augment the system dynamics with additional states to compute the Lagrange cost term and the violation of the path constraints along the state trajectory, and then perform a structured search of the input space using a single-shooting simulation of the system dynamics. We demonstrate this framework on a robust Goddard rocket problem with a non-differentiable cost and a quantized thrust input, where we achieve an altitude within 40 m of the target, while other methods are unable to get closer than 180 m.

Acknowledgements

This thesis and the four years I have spent at Imperial College London have been shaped by many people, many more than I can name individually, but I will try nonetheless.

First and foremost, I would like to thank my two supervisors, Prof. Eric Kerrigan and Prof. George Constantinides, for all their time, encouragement, patience, and support over these last four years. From the short meetings to the sometimes hour-long discussions we have had, every moment has helped me to tackle the various challenges that arose and to develop into a better researcher and instructor.

Next, I would like to thank the HiPEDS CDT[†] and all the faculty and staff involved with it. I feel fortunate to have been funded by HiPEDS these last four years, to have had the opportunity to participate in the many CDT activities and events (both professional and social), and to have been part of a wonderful and supportive cohort of students.

I would also like to thank all of my colleagues in the Department of Electrical and Electronic Engineering, and especially my colleagues in the two research groups I was part of while at Imperial, for all the insightful and enjoyable discussions over these years. Special thanks are due to Dr Omar Faqir, Dr Yuanbo Nie, Dr Marta Zagorowska, Dr Paola Falugi, Dr James Davis, Dr He Li, Dr Erwei Wang, Lucian Nita and Eduardo Vila. I would especially like to thank Prof. Eric Kerrigan, Prof. George Constantinides, Dr Marta Zagorowska and Dr Yuanbo Nie for reading and commenting on early drafts of this thesis.

Finally, I would like to thank all of my family, friends, and teachers in the United States, since without their support I would not have ended up where I am now. I am especially grateful to two teachers during my schooling, Joe Muskin at Next Generation Middle School in Champaign IL and Jim Gerry at the Illinois Mathematics and Science Academy (IMSA) in Aurora IL, for providing the opportunities and encouragement for me to explore science and engineering and setting me on the path to where I am today.

[†]The Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS), EPSRC Grant Reference EP/L016796/1.

I am deeply grateful for the support and encouragement of my parents throughout my life — from allowing me to move three hours away so that I could attend high school at IMSA when I was 15 to encouraging me to move across the Atlantic to pursue my PhD at Imperial, they have always been there for me no matter how far away I was. I am also grateful to my brother Ross, who guided me towards many wonderful opportunities at IMSA and Iowa State University. Finally, to my Fiancée Sinem, for her understanding, support, and encouragement during my PhD, especially during the final few months while I was writing this thesis.

Ian McInerney

London, November 2021

To Ross.

Contents

Declaration of originality 1

Copyright declaration 3

Abstract 5

Acknowledgements 7

I Introduction and Background Information 23

1 Introduction 25
1.1 Motivation . . . 25
1.1.1 Computation vs performance trade-off . . . 26
1.1.2 Derivative-free MPC solvers . . . 27
1.2 Contributions . . . 28
1.3 Organization and detailed contributions . . . 29
1.3.1 Background information . . . 29
1.3.2 Block Toeplitz operators in linear MPC . . . 30
1.3.3 Derivative-free optimization for NMPC . . . 31
1.4 Mathematical notation . . . 31
1.5 List of publications . . . 32

2 Introduction to Model Predictive Control 35
2.1 The Model Predictive Control problem . . . 35
2.1.1 The Linear Quadratic Regulator problem . . . 37
2.2 Nonlinear MPC algorithms . . . 40
2.2.1 DFO formulations . . . 41

2.3 Linear MPC formulations . . . 44
2.3.1 Implicit MPC formulations . . . 44
2.3.2 Numerically robust MPC . . . 46
2.4 Implementation of linear MPC . . . 48
2.4.1 Quadratic programming algorithms . . . 48
2.4.2 Number representation . . . 51
2.4.3 Parallelization opportunities . . . 55
2.4.4 Memory organization . . . 57
2.4.5 Overall trends . . . 58
2.5 First-order optimization solvers for linear MPC . . . 59
2.5.1 Fast Gradient Method . . . 60
2.5.2 Dual Gradient Projection . . . 62
2.6 Example systems . . . 63
2.7 Conclusions . . . 65

3 An introduction to block Toeplitz matrices 67
3.1 Examples of block Toeplitz matrices . . . 68
3.1.1 Queueing theory . . . 69
3.1.2 Signal processing . . . 70
3.1.3 Information theory . . . 71
3.1.4 Control theory . . . 72
3.1.5 2D boundary value problems . . . 72
3.2 Block Toeplitz matrices and their matrix symbols . . . 74
3.2.1 The matrix symbol . . . 75
3.2.2 Forming the matrix symbol . . . 75
3.2.3 Circulant matrices . . . 76
3.3 Structure preserving operations . . . 77
3.4 Spectral properties . . . 81
3.4.1 Spectrum of a block circulant matrix . . . 83
3.4.2 Spectrum of a block Toeplitz matrix . . . 84
3.4.3 Szegö's limit theorem . . . 86
3.5 Conclusions . . . 87

II Toeplitz Operators in Linear MPC 89

4 Spectral properties for the condensed problem 91
4.1 System prediction matrix . . . 92
4.2 Primal Hessian . . . 95
4.2.1 Matrix symbol . . . 95
4.2.2 Spectral bounds . . . 101
4.2.3 Inclusion of a state-input cross term . . . 104
4.3 Inequality constraint matrix . . . 107
4.4 Dual Hessian . . . 109
4.5 Conclusions . . . 112

5 Computational complexity and system performance 115
5.1 Preliminaries . . . 115
5.2 Extremal eigenvalues . . . 118
5.2.1 Primal Hessian . . . 118
5.2.2 Dual Hessian . . . 119
5.3 Condition number . . . 120
5.3.1 Discrete-time weights . . . 121
5.3.2 Discretization of continuous-time weight matrices . . . 123
5.4 Computational complexity . . . 124
5.4.1 Fast Gradient Method . . . 124
5.4.2 Dual Gradient Projection . . . 125
5.5 Conclusions . . . 126

6 Preconditioner design 129
6.1 Analysis of the preconditioned Hessian . . . 129
6.2 Preconditioner design . . . 131
6.3 Numerical experiments . . . 133
6.3.1 Matrix conditioning . . . 134
6.3.2 Performance of the Fast Gradient Method . . . 134
6.4 Conclusions . . . 138

7 Fixed-point round-off error analysis 139
7.1 Matrix pseudospectrum . . . 139
7.2 Choosing the fractional precision . . . 140
7.2.1 Generic rounding model . . . 143
7.2.2 Parametric rounding model . . . 144
7.3 Numerical experiments . . . 149
7.4 Conclusions . . . 151

III Derivative-free Optimization for NMPC 153

8 Mesh Adaptive Direct Search algorithm 155
8.1 The MADS algorithm . . . 156
8.2 Constraint handling . . . 159
8.2.1 Extremal barrier constraints . . . 159
8.2.2 Progressive barrier constraints . . . 159
8.3 Meshing . . . 161
8.3.1 Granular variables . . . 161
8.4 Convergence . . . 163
8.5 Conclusions . . . 163

9 DFO-NMPC problem formulation **165**
9.1 Proposed formulation . . . 165
9.1.1 Cost functional . . . 166
9.1.2 Path constraints . . . 166
9.1.3 Overall problem formulation . . . 167
9.2 Discussion and conclusions . . . 168
9.2.1 Advantages of using MADS . . . 168
9.2.2 Augmentation scheme . . . 169

10 DFO-NMPC for a robust Goddard rocket problem **171**
10.1 Example problem . . . 171
10.1.1 System dynamics . . . 172
10.1.2 Optimal control problem . . . 173
10.1.3 Transcription to DFO-NMPC . . . 175
10.2 Experimental configuration . . . 178
10.3 Solution without velocity path constraints . . . 179
10.4 Solution with velocity path constraints . . . 179
10.4.1 Penalty barrier implementation . . . 181
10.4.2 Progressive barrier implementation . . . 183
10.4.3 Comparison against particle swarm optimization . . . 185
10.5 Quantized input variables . . . 186
10.5.1 Quantized version of continuous results . . . 188
10.5.2 Using granular MADS . . . 190
10.5.3 Using quantized PSO . . . 192
10.6 Overdetermined problem formulation . . . 194
10.7 Conclusions . . . 195

IV Conclusions **199**

11 Conclusions and future work **201**
11.1 Toeplitz operators in linear MPC . . . 201
11.2 DFO-NMPC . . . 204

References **207**

List of Figures

2.1 Taxonomy of various optimization algorithms used to implement MPC on FPGAs. 49

2.2 Clock cycles required for the computations in FPGA implementations of linear MPC. 59

2.3 Configuration of the mass-spring-damper example system. 64

3.1 Block diagram of a 2nd order MIMO causal FIR filter. 70

3.2 5-point stencil for a 2D boundary value problem. 73

4.1 Singular values of the prediction matrix for the first Schur-stable system compared to the approximation from Proposition 4.1.1. 96

4.2 Spectral bounds for the condensed Hessian of the first Schur-stable example problem. 102

4.3 Eigenvalue spectrum for the condensed Hessian of the first Schur-stable example problem. 103

4.4 Spectral properties of the condensed primal Hessian for the mass-spring-damper system. 107

4.5 Maximum eigenvalue for the condensed dual Hessian of the first Schur-stable system. 112

5.1 Horizon-independent extremal eigenvalue and condition number bounds for scaled Q and R matrices with the first Schur-stable system. 120

5.2 Effect of scaled Q and constant R on the FGM solving the first Schur-stable system. 125

5.3 Effect of scaled Q and constant R on the DGP solving the first Schur-stable system. 127

5.4 Effect of scaled Q and constant R on the DGP solving the mass-spring-damper system. 128

6.1 Condition number versus the horizon length of the condensed Hessian for the first Schur-stable system. 135

6.2 Condition number versus the horizon length of the prestabilized condensed Hessian for the inverted pendulum system. 136

7.1 The inverse of the resolvent of H when $\tilde{\lambda}$ is constrained to the real line. 140
7.2 Interval containing the spectrum of $\hat{H}$ with different rounding modes. 141
7.3 Extremal values of $\lambda(\hat{H})$ with $N = 50$. 142
7.4 The minimum number of fractional bits required for a given horizon length when using the generic rounding model from Theorem 7.2.1. 145
7.5 Value of $\|[\mathbf{H}]_i\|$ as the diagonal number grows. 146
7.6 The matrix norms from Theorem 7.2.2, with round to nearest, and round to zero. 149
7.7 The effect of scaling the cost function on the rounding margin and fixed-point representation. 150
7.8 Percent error of fixed-point versus double-precision floating-point implementations of FGM using round to zero and the fractional lengths from Theorem 7.2.2. . . . 151

8.1 The mesh and frame for GPS and MADS after one successful iteration. 157

10.1 Robust tube for the robust Goddard rocket problem. 173
10.2 Open-loop solution with no velocity path constraints. 180
10.3 Open-loop solution with velocity path constraints using penalty barriers in the cost. 182
10.4 Open-loop solution with velocity path constraints using the progressive barrier method. 184
10.5 Open-loop solution using PSO with velocity path constraints as penalty barriers in the cost. 187
10.6 Open-loop solution using the quantized input trajectory in Table 10.6b. 189
10.7 Open-loop solution with quantized input variables solved with granular MADS. . 191
10.8 Open-loop solution with quantized inputs solved with PSO. 193
10.9 Open-loop solution including t_{f_l} and t_{f_u} in the MADS optimization problem. . . 196

List of Tables

2.1 FPGA implementation details for selected linear MPC solvers. 50

2.2 Rounding modes in fixed-point arithmetic data types and their maximum round-off error. 53

2.3 Size of the fields (in bits) and the exponent bias in IEEE-754 standard floating-point formats. 53

2.4 Works containing results on the stability under roundoff errors, variable bounds, and computational complexity certification of selected algorithms. 54

6.1 Condition number of the preconditioned condensed Hessian. 135

6.2 Iterations required for cold-start convergence of the preconditioned Fast Gradient Method to ϵ=10^{-5}. 137

6.3 Time required for computing the preconditioners. 137

7.1 Resource usage for FGM implemented using ProtoIP [95] on a Zynq 7020 at 100MHz with $N = 20, i = 5$. 152

10.1 Physical parameters for the rocket dynamics (10.1). 172

10.2 Results for MADS DFO-NMPC with no velocity path constraints. 181

10.3 Results for MADS DFO-NMPC with velocity path constraints using penalty barriers in the cost. 183

10.4 Results for MADS DFO-NMPC with velocity path constraints using the progressive barrier method. 185

10.5 Results for DFO-NMPC solved with PSO and with velocity path constraints. . . 186

10.6 Results using a quantized version of the input trajectory in Table 10.4b. 190

10.7 Results with quantized inputs solved using granular MADS. 192

10.8 Results with quantized inputs solved using PSO. 194
10.9 Results for MADS DFO-NMPC with t_{f_l} and t_{f_u} included in the optimization problem. 197

List of Algorithms

2.1 The Fast Gradient Method 60
2.2 The Dual Gradient Projection Method 62
8.1 The Mesh Adaptive Direct Search with progressive barrier constraints 158
9.1 Function evaluation for the DFO formulation of the NMPC problem 167

Nomenclature

Abbreviations

(C)LQR	(Constrained) Linear Quadratic Regulator
(N)MPC	(Nonlinear) Model Predictive Control
ASM	Active Set Method
BRAM	Block Random Access Memory (Block RAM)
DARE	Discrete Algebraic Riccati Equation
DFO	Derivative-free Optimization
DSP	Digital Signal Processing
FF	Flip flops
FGM	Fast Gradient Method
FPGA	Field Programmable Gate Array
GPS	General Pattern Search
IPM	Interior Point Method
LUT	Lookup Table
MADS	Mesh Adaptive Direct Search
MILP	Mixed Integer Linear Program
OCP	Optimal Control Problem
PSO	Particle Swarm Optimization
QP	Quadratic Program
SDP	Semidefinite Program
UDB	Upper Dual Bound
UIB	Upper Iteration Bound

Mathematical Spaces

$\tilde{\mathcal{C}}_{2\pi}$	Continuous functions inside $\mathcal{L}^{\infty}_{2\pi}$
$\mathcal{L}^{\infty}_{2\pi}$	Banach space of matrix-valued essentially bounded functions that are 2π-periodic
l^2	Hilbert space of square summable sequences

Number sets

$\mathbb{C}$	Complex numbers
$\mathbb{G}$	Granular numbers ($\mathbb{G} \coloneqq \{a \times 10^b \colon a \in \{1, 2, 5\},\ b \in \mathbb{Z}\}$)
$\mathbb{N}^+$	Positive integers
$\mathbb{R}$	Real numbers
$\mathbb{R}_{\geq 0}$	Positive orthant of real numbers
$\mathbb{T}$	Complex unit circle ($\mathbb{T} \coloneqq \{z \in \mathbb{C} : \lvert z \rvert = 1\}$)
$\mathbb{Z}$	Integers

Part I

Introduction and Background Information

Chapter 1

Introduction

1.1 Motivation

Model Predictive Control (MPC) is an optimal control method that aims to optimize the closed-loop performance of a controlled system by solving an optimization problem at each sampling instant to compute the next control input while explicitly handling the system constraints (such as physical, regulatory, or safety constraints) in the controller computation. MPC originated in the process control industry in the 1980s/90s as a method of controlling power plants and the chemical reactions in petroleum refineries. The goal of MPC is to maximize the production output while reducing the running costs and satisfying the operational constraints. It is well-known that the point at which processes/plants operate the most economically is at the intersection of the constraints placed on the reactions/processes [146]. MPC is able to operate processes closer to their constraints than other more conservative controllers (e.g. PID or LQR), meaning that MPC is able to operate processes more economically than the other controllers.

During the early years of MPC, computing the solution to the optimization problem was very slow due to the limited computational resources available and the complex optimization algorithms used. The usage of MPC was therefore limited to the process control industries, where there were slower systems with minutes to hours between the control inputs, giving time for the optimization solver to run to completion before the next input was needed. Additionally, the limitation in the available computational power and the need for faster computation of the control input led to the use of the Constrained Linear Quadratic Regulator (CLQR) formulation of MPC (i.e. the use of an LQR cost function with linear dynamics and constraints) as an approximation for the full nonlinear economic MPC problem in many fields.

Since then, there has been enormous progress in reducing the computational burden of computing the MPC control law. A combination of increased computing power available at a lower cost and the development of more efficient real-time optimization algorithms has led to MPC achieving sampling times on the order of microseconds [83] and being applied to systems such as power electronics/motors [92, 152, 171], diesel engines [64, 72], quad/multi-rotors [129], and building environmental control [46]. This growth has sparked great interest in using MPC in more places, with a large segment of future applications being resource-constrained systems such as cyber-physical systems and internet of things devices [108].

1.1.1 Computation vs performance trade-off

The computer processors used in modern cyber-physical systems are routinely utilized for more than just control, with additional tasks such as communication, coordination, user-interface and data-collection becoming more widespread as designers adopt networked systems and the internet of things. At the same time, computational resources are being further constrained by the demand for low-power and low-cost designs. Guaranteeing the proper operation of the control system on resource constrained processors in environments with safety and operational constraints is important to guarantee the dependability of the cyber-physical system [85]. MPC has been highlighted in [108] as a control algorithm aptly suited to provide these operational guarantees at a functional level through the inclusion of the operational/safety constraints directly in the controller computation. This means that guaranteeing the dependable operation of the cyber-physical system now also requires analyzing the MPC algorithm to determine guarantees on the controller performance given the reduced computational resources present. To this end, the authors of [85] suggest that an important question to answer is: what effect do the reduced computational resources have on the system performance?

This thesis investigates the relation between the computational complexity and performance for the CLQR MPC formulation in Part II, and examines not just the performance of the system, but also that of the solver itself. In doing so, we seek to reverse the question and instead urge designers to consider system performance alongside the computational resources used by asking the question: what effect does the desired system performance have on the computational resources required? The performance requirements are usually given as bounds in a specification (e.g. “settling-time less than 1s” or “track this signal with less than 5% error”) rather than an exact criterion, creating a space of possible controllers that can satisfy the requirements. These controllers may then have different computational resource demands, opening up the opportunity to trade off the computational performance of the controller with the system performance.

1.1.2 Derivative-free MPC solvers

With the rise of data-based control and the trend towards using full nonlinear economic MPC instead of the CLQR formulation, more engineers are wanting to control systems described by higher-fidelity models that capture nonlinearities more accurately or systems that only have blackbox/data-based models. These systems may arise in areas such as robotics or autonomous vehicles, or be the result of legal/regulatory restrictions that prevent the sharing of the detailed models (e.g. only provide compiled simulation code and not the actual physics equations to protect trade secrets). The blackbox nature of these systems may pose a challenge for derivative-based methods (i.e. first and second-order methods), since these methods require derivative information for the system dynamics, which may not be available or reliable.

To overcome this, Derivative-free Optimization (DFO) methods can also be used with MPC. In DFO solvers, no knowledge of the derivatives of the cost or constraints is required, and instead the solver searches the feasible set by computing the cost function at specific locations and uses that information to compute new locations to search. To solve the Nonlinear MPC (NMPC) problem using DFO methods, the solver will perform simulations of the dynamics using candidate input trajectories and compute the cost of each trajectory to locate the trajectory with the lowest cost. These solvers can lead to embarrassingly parallel implementations, since all simulations in an iteration can be run in parallel.

Simulation-based solvers have been readily used in Finite Control Set (FCS) MPC algorithms in power electronics to determine switching times for the low-level hardware [99]. The derivative-free methods used to solve the FCS-MPC problem are inefficient for long time horizons though, since they usually perform an exhaustive search of the possible future input sequences, which leads to a combinatorial explosion in the search space size as the horizon grows. Instead, there has been prior work that has suggested working around this combinatorial explosion by randomly sampling a fixed number of points in the search space [90], or using methods such as pattern search [53], the Nelder-Mead simplex algorithm [157], Trust Region methods [42], or Particle Swarm Optimization [179].

In this thesis, we build on the pattern search method from [53] and use the Mesh Adaptive Direct Search (MADS) algorithm introduced in [6] with a new NMPC formulation that is more natural and easier to use for non-experts. The proposed NMPC formulation for derivative-free optimization solvers provides an easy and intuitive way to handle the path constraints and Lagrange cost term in the problem through augmenting the system dynamics to measure the constraint violation and cost value over the horizon. The use of MADS for the DFO solver makes the number of points evaluated around the current iterate grow linearly with the input dimension, instead of combinatorially like other solvers.

1.2 Contributions

In this thesis, we focus on making the numerical methods used for MPC simpler to use and more resource efficient. This is done in two stages, with the first focusing on deriving horizon-independent analysis and design methods for linear MPC, and the second focusing on the derivation of a DFO method for solving NMPC problems.

For linear MPC, we present several novel results, including:

- horizon-independent bounds on the eigenvalues and condition number for condensed linear MPC matrices that allow for the computation of a horizon-independent Upper Iteration Bound (UIB) for the Fast Gradient Method (FGM),
- analysis of the Hessian condition number and UIB for the FGM as the system performance is changed,
- an efficient horizon-independent preconditioner that is up to 50,000x faster to compute than an SDP preconditioner and that can provide a speedup of up to 9x for the FGM, and
- data-type sizing rules for the FGM in fixed-point arithmetic that reduce the hardware usage and solution time by up to 77% and 25%, respectively, for when the FGM is implemented on an FPGA.

Put together, these contributions allow for a control engineer to design and analyze parts of a linear MPC controller using only the matrices in the cost function and state-space description of the system, and without having to know the desired horizon length. Using horizon-independent design techniques simplifies the design process for the control engineer, allowing them to vary the horizon as a tuning parameter and not need to recompute if the controller meets timing requirements or redo the possibly expensive computation of a preconditioner or FPGA design.

It should be noted that the idea of the horizon-independent spectral bounds is not new, and similar results for the primal condensed Hessian and prediction matrix were reported in [153] and [54, Sect. 11]. The analysis in this thesis generalizes the results in those works by using the numerically robust formulation for linear MPC and allowing an arbitrary positive-definite state weighting matrix Q and a terminal weighting matrix P chosen as either Q or the solution to the Discrete Algebraic Riccati Equation (DARE).

Next, we derived a new DFO formulation for the NMPC problem using the MADS algorithm that provides several benefits, such as:

- the ability to handle nonsmooth or blackbox cost functions/dynamics/constraints,

- easy handling of other variable types such as quantized/granular, periodic or categorical variables,
- path constraint satisfaction across the entire horizon without a mesh refinement scheme, and
- more accurate computation of the stage cost value across the entire horizon.

Put together, these benefits simplify the problem setup for the control designer and allow for the implementation of controllers on more complex systems without the need for possibly complex or lossy reformulations of the problem. This is especially important for the design of robust controllers or controllers for systems with data-based models, where the natural expression of the problem may be incompatible with existing solver architectures.

This formulation makes it extremely simple to handle quantized variables with no modifications to the actual NMPC problem by simply informing MADS about the desired quantization level. The MADS solver then handles all the quantization of the variables, and as shown in the numerical example in Chapter 10, without a large increase in the solution time of the problem.

1.3 Organization and detailed contributions

This thesis is separated into three parts, the first containing background material, and the last two each focusing on numerical methods for linear and nonlinear MPC, respectively.

1.3.1 Background information

In the remainder of Part I, we present background information on both MPC and block Toeplitz operator theory.

In Chapter 2, we present a brief background on MPC and the associated solvers. This includes describing the various formulations of linear MPC, surveying the existing problem formulations for solving NMPC using DFO solvers, providing a survey of existing works that implement linear MPC on FPGAs, describing the Fast Gradient Method and Dual Gradient Projection method for solving linear MPC problems, and finally, presenting the five example systems we use for the numerical examples in Part II.

In Chapter 3 we provide an introduction to the theory behind block Toeplitz matrices, operations on them, and their spectral properties. The chapter summarizes the relevant mathematical research in the field into an accessible format that highlights the parts most applicable to the analysis we perform in Part II.

1.3.2 Block Toeplitz operators in linear MPC

In Part II we derive analysis techniques and design methods for the linear MPC problem that are horizon-independent and utilize the ideas of block Toeplitz operator theory that we presented in Chapter 3.

Chapter 4 contains the derivation of spectral properties for the prediction matrix and Hessian in both the primal and dual condensed linear MPC problems. These properties are horizon-independent, meaning the spectral bounds for the eigenvalues and condition number can be found using a single computation, and be valid for the matrices across all horizon lengths. We also present spectral properties for when the Hessian includes a state-input cross-term weighting matrix S, which can occur when trying to make the MPC controller match an existing controller [47, 160] or from discretizing the continuous-time LQR cost function [26]. Finally, we present some spectral properties for the dual condensed Hessian.

Chapter 5 examines how these spectral properties change when the cost function is changed by scaling the cost matrices. We show that the extremal eigenvalues of the condensed primal Hessian grow linearly when the cost matrices are scaled, and that if the cost matrices are all scaled by the same factor, the condition number of the Hessian remains the same. We also identify two regions in the bounds: one where the input weights affect the eigenvalues of the Hessian the most, and the other where the singular values of the predicted system affect the eigenvalues of the Hessian the most.

In Chapter 6 we derive a new preconditioner for the linear MPC problem that is able to speed up convergence of the FGM by up to 9x. This preconditioner is horizon-independent and is computed using only matrices with the number of states and inputs as their dimensions, but requires the terminal state penalty matrix P to be either the solution to the discrete Lyapunov equation for Schur-stable systems, or the DARE for unstable systems. We also show that the proposed preconditioner provides performance equivalent to an existing SDP preconditioner on several examples, while also providing a reduction in the computational effort required to compute the preconditioner.

Finally, in Chapter 7 we present a framework for computing the data type required to have stability of the FGM in fixed-point arithmetic. As part of this framework, we present a new measure that we call the rounding stability margin, based on the pseudospectrum of the Hessian in the Quadratic Program (QP) from linear MPC, to quantify how much round-off error can be experienced by the Hessian of the QP before the FGM becomes unstable. We then present two models for the round-off error introduced by moving the Hessian into a fixed-point representation. The first is a generic model that can be applied to any MPC problem formulated as a QP, but depends on the length of the prediction horizon. The second is a structure-exploiting parametric model that requires the predicted system to be Schur-stable, but provides a horizon-independent approximation of the round-off error. These round-off

error models are then combined with the rounding stability margin to compute the number of fractional bits required for algorithmic stability. We demonstrate that using the structure-exploiting parametric model reduces the number of fractional bits needed by 30–45%, and reduces the hardware usage and solution time by up to 77% and 25%, respectively, for an FPGA implementation of the FGM.

1.3.3 Derivative-free optimization for NMPC

In Part III we present a framework for solving the NMPC problem using DFO methods, specifically the MADS algorithm.

We begin by presenting an overview of the MADS algorithm in Chapter 8. This overview combines the ideas from several works into a single statement of the algorithm, and provides a discussion on the two techniques for implementing constraints in MADS and how MADS can natively handle granular (e.g. quantized) variables.

We next present the proposed framework for solving the NMPC problem using MADS in Chapter 9. Using MADS, the number of points evaluated around the current iterate grows linearly with the input dimension, instead of combinatorially like other derivative-free solvers. The NMPC formulation consists of a single-shooting simulation of the dynamics across the time horizon. The path constraints and Lagrange cost term are handled by augmenting the system dynamics with new states, so that the constraint violation and cost value are computed at the same time as the dynamics simulation. We then conclude this chapter by discussing the advantages and opportunities for future improvement of the proposed framework.

Finally, we apply the proposed DFO-NMPC framework to an example problem in Chapter 10. The example presented is a robust Goddard rocket problem, where the objective is to reach a target altitude while the system model has bounded uncertainty in some parameters. We show that the DFO-NMPC framework using MADS provides advantages over other DFO techniques, such as particle swarm optimization, which can become confused on problems that have discretized/granular variables.

1.4 Mathematical notation

In this thesis, we use the following mathematical notation.

A' and A^* denote the transpose and conjugate-transpose of the matrix A, respectively. $A \otimes B$ represents the Kronecker product of the matrix A with the matrix B. $\lambda_1 \leq \cdots \leq \lambda_k$ are the real eigenvalues of a Hermitian matrix A in sorted order, with the set of all eigenvalues

denoted by $\lambda(A)$ and the set of all singular values of A denoted by $\sigma(A)$. The p-norm is denoted by $\|\cdot\|_p$, with $\|A\|_2$ the matrix spectral norm, and $\|A\|_F$ the Frobenius norm. The condition number of a matrix is $\kappa(A) \coloneqq \|A\|_2\|A^{-1}\|_2$.

We use the notation from [185] to represent the transfer function matrix of the discrete-time linear system $\mathcal{G}$ with state space matrices A, B, C, D and $z \in \mathbb{C}$ as

$$\mathcal{G}(z) \coloneqq \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right].$$

$\mathcal{L}^\infty$ is the space of matrix-valued essentially bounded functions (i.e. matrix-valued functions that are measurable and have a finite Frobenius norm almost everywhere on their domain [125, §2]). $\tilde{\mathcal{C}}_{2\pi}$ is the space of continuous 2π-periodic functions inside $\mathcal{L}^\infty$. The set $\mathbb{T} \coloneqq \{z \in \mathbb{C} : |z| = 1\}$ is the complex unit circle. For an infinite-dimensional block Toeplitz matrix $\mathbf{T}$ with blocks of size $m \times n$, $\mathcal{P}_T(\cdot) : \mathbb{T} \to \mathbb{C}^{m\times n}$ represents the matrix symbol of $\mathbf{T}$ and $\mathbf{T}_N$ represents the truncated version of $\mathbf{T}$ after N block diagonals (where N is a positive integer). We provide more definitions relating to the spectrum of block Toeplitz matrices (e.g. λ_{min}, λ_{max}, σ_{min}, and σ_{max}) in Definitions 3.4.1 and 3.4.2 in Section 3.4

1.5 List of publications

The following publications were written during the PhD program. When papers were authored with collaborators other than the two supervisors, the details on paper contributions are provided. Items identified with a $\star$ contain material presented in this thesis.

Journal - Published

- $\star$ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Horizon-independent preconditioner design for linear predictive control. *IEEE Transactions on Automatic Control*, 2022. doi: 10.1109/TAC.2022.3145657. (*in press*).

- H. Li, I. McInerney, J. Davis, and G. A. Constantinides. Digit stability inference for iterative methods using redundant number representation. *IEEE Transactions on Computers*, 70(7):1074–1080, 2021. doi: 10.1109/TC.2020.3003529.
  Contributions: Derivation of theorem/proofs, manuscript preparation.

- J.-M. Rodriguez-Bernuz, I. McInerney, A. Junyent-Ferré, and E. C. Kerrigan. Design of a linear time-varying model predictive control energy regulator for grid-tied VSCs. *IEEE Transactions on Energy Conversion*, 36(2):1425–1434, 2021. doi: 10.1109/TEC.2021.3060319.
  Contributions: Implementation of optimization solver, manuscript preparation.

Conference - Published proceedings

⋆ I. McInerney, G. A. Constantinides, and E. C. Kerrigan. A survey of the implementation of linear model predictive control on FPGAs. In *6th IFAC Conference on Nonlinear Model Predictive Control*, pages 381–387, Madison, Wisconsin, US, 2018. IFAC. doi: 10.1016/j.ifacol.2018.11.063.

⋆ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Modeling round-off error in the fast gradient method for predictive control. In *58th IEEE Conference on Decision and Control (CDC)*, pages 4331–4336, Nice, France, 2019. IEEE. doi: 10.1109/CDC40024.2019.9029910.

⋆ I. McInerney, L. Nita, Y. Nie, A. Oliveri, and E. C. Kerrigan. Towards a framework for nonlinear predictive control using derivative-free optimization. In *7th IFAC Conference on Nonlinear Model Predictive Control*, Bratislava, Slovakia, 2021. IFAC.
Contributions: Framework derivation, numerical example tuning, manuscript preparation.

Conference - Extended abstract

- I. McInerney and E. C. Kerrigan. Automated project-based assessment in a predictive control course. In *2018 UKACC 12th International Conference on Control (CONTROL)*, page 443, Sheffield, UK, Aug. 2018. IEEE. doi: 10.1109/CONTROL.2018.8516728.

⋆ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Closed-form preconditioner design for linear predictive control. In *21st IFAC World Congress*, Berlin, Germany, Jul. 2020. IFAC.

Conference - Presentation only

⋆ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Modeling round-off error in the fast gradient method for predictive control. Presented at the *28th Biennial Numerical Analysis Conference*, Glasgow, Scotland, UK, Jun. 2019.

- ⋆ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Bounding computational complexity under cost function scaling in predictive control. Presented at the *International Conference on Industrial and Applied Mathematics (ICIAM)*, Valencia, Spain, Jul. 2019.

Working paper

- ⋆ I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Bounding computational complexity under cost function scaling in predictive control. *arXiv:1902.02221*, 2019.

Chapter 2

Introduction to Model Predictive Control[†]

2.1 The Model Predictive Control problem

Nonlinear Model Predictive Control (NMPC) is formulated using the Optimal Control Problem (OCP)

$$\min_{x,u,t_f} \Phi(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)\,\mathrm{d}t \tag{2.1a}$$

$$\text{s.t. } f(\dot{x}(t), x(t), u(t), t) = 0, \quad \forall t \in [t_0, t_f] \tag{2.1b}$$

$$g(x(t), u(t), t) \leq 0, \quad \forall t \in [t_0, t_f] \tag{2.1c}$$

$$h(x(t_0), u(t_0), t_0, x(t_f), u(t_f), t_f) = 0, \tag{2.1d}$$

where $x(\cdot) \in \mathbb{R}^n$ and $u(\cdot) \in \mathbb{R}^m$ are the continuous-time state and input trajectories, respectively. The cost function (2.1a) is composed of two terms: the Mayer cost $\Phi(\cdot)\colon \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$ that captures the cost at the final time point t_f, and the integral of the Lagrange cost $L(\cdot)\colon \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$ that captures the cost along the horizon. The equality constraints (2.1b) are the nonlinear continuous-time system dynamics $f(\cdot)\colon \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$, while the constraints (2.1c) and (2.1d) are the path constraints $g(\cdot)\colon \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^l$ and the boundary conditions $h(\cdot)\colon \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^j$, respectively.

[†]Some of the material presented in this chapter has been published in the following work:
I. McInerney, G. A. Constantinides, and E. C. Kerrigan. A survey of the implementation of linear model predictive control on FPGAs. In *6th IFAC Conference on Nonlinear Model Predictive Control*, pages 381–387, Madison, Wisconsin, US, 2018. IFAC. doi: 10.1016/j.ifacol.2018.11.063.

In the OCP (2.1), the possible state trajectories $x(\cdot)$ and input trajectories $u(\cdot)$ are given as functions of time, and so the optimization is done in what is known as an *infinite-dimensional function space*, where any mathematical function can be used to represent x and u. There are several ways to solve the infinite-dimensional OCP, with the most popular being to find an approximate numerical solution by using a transcription method to convert the infinite-dimensional OCP into a finite-dimensional optimization problem. There are several transcription methods that can be used, including shooting methods [22], collocation methods [93], and the integrated residual method [94].

NMPC can be implemented as either an open-loop or closed-loop controller. When used as an open-loop controller, the OCP (2.1) is solved a single time offline before the system runs, and then the control inputs of the system are generated online using the pre-computed input trajectory u sampled whenever a new input is needed. The open-loop controller will move the system to the objective assuming that the system always follows the exact state trajectory predicted by the dynamics model. If there are any unmodeled disturbances on the system that make the actual state trajectory different from the predicted trajectory, the open-loop controller may no longer be able to control the system to the objective or ensure constraint satisfaction.

Alternately, a closed-loop implementation of NMPC solves the OCP (2.1) every Δ_t seconds (for a controller running at a constant sampling rate). The current state $\hat{x}_0$ of the system is measured or estimated before computing the new controller, and that measurement is then used in the OCP by introducing a boundary condition $x(t_0) = \hat{x}_0$. This boundary condition ensures that the computed control trajectory is able to move the system to the objective even in the presence of disturbances and perturbations. The control applied to the system at every sample is then the value of the control trajectory $u(t)$ at $t = t_0$.

An important aspect of a closed-loop implementation is how the prediction and control horizons change between the sampling instants. In the simplest case, called *receding-horizon*, the horizon lengths remain constant between samples (so the next final time becomes $t_f + \Delta_t$) and the state and input trajectories are shifted to make $t_0 + \Delta_t$ the new initial time before being used as the initial condition for the next OCP. The receding-horizon method can be used with regulation-type problems, where the controller does not have a specific end-time and is instead running continuously.

Another method is *decreasing-horizon* control, where the final time t_f remains constant across all problems and t_0 is increased by Δ_t each sample, causing the overall horizon length to shrink. This method is used mainly when the system is being moved between two setpoints in a fixed amount of time, since it can guarantee that the system reaches the target point by the original t_f specified in the problem. An important consideration when implementing a decreasing-horizon controller is what control to use once t_f is reached and the horizon disappears —

since the OCP will no longer provide any control inputs. This could be as simple as providing $u = 0$ after t_f if the system is guaranteed to be stable, but a more common approach would be to use a simple regulating controller to regulate the system about the desired state (since this controller would then reject any disturbances that may appear).

2.1.1 The Linear Quadratic Regulator problem

The most common type of MPC problem solved using embedded systems has been the Constrained Linear Quadratic Regulator (CLQR) with a receding-horizon, since it is easily formulated as a convex Quadratic Program (QP). For the CLQR, the input sequence is usually assumed to be a piecewise-constant function, and the dynamics (2.1b) and constraints (2.1c) are linearized and discretized accordingly. The cost function (2.1a) is chosen to be a quadratic penalty on the deviation of the state and input trajectories from the origin.

For the discrete-time Linear Time-varying (LTV) CQLR, the OCP (2.1) is then transcribed and simplified into the QP

$$\min_{x,u} \ \frac{1}{2}x_N' P x_N + \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}' \begin{bmatrix} Q_k & S_k \\ S_k' & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{2.2a}$$

$$\text{s.t. } x_{k+1} = A_k x_k + B_k u_k, \ k = 0, \ldots, N-1 \tag{2.2b}$$

$$x_0 = \hat{x}_0$$

$$E_k^x x_k + E_k^u u_k \leq c_k, \ k = 0, \ldots, N-1 \tag{2.2c}$$

$$E_N^x x_N \leq c_N \tag{2.2d}$$

where N is the horizon length. This QP optimizes over the state and input trajectories $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, respectively, by discretizing the trajectories such that x_k and u_k are the value of the state and input trajectories, respectively, at each sampling instant k.

The LTV CLQR problem (2.2) regulates a discrete-time LTV system to the origin by imposing a quadratic penalty on any deviation of the states or inputs away from the origin, while allowing the cost function and constraints to vary across the horizon. The cost function is defined using the weight matrices $Q_k = Q_k' \in \mathbb{R}^{n\times n}$, $R_k = R_k' \in \mathbb{R}^{m\times m}$ for the system states and inputs, respectively, the positive semi-definite matrix $P = P' \in \mathbb{R}^{n\times n}$ for the system states at the final time point, and $S_k \in \mathbb{R}^{n\times m}$ for the state-input cross-term. By choosing the weighting matrices such that the R_k are positive-definite and $\begin{bmatrix} Q_k & S_k \\ S_k' & R_k \end{bmatrix}$ (or equivalently $Q_k - S_k R_k^{-1} S_k'$) is positive semi-definite, the resulting QP (2.2) is strictly convex.

The discrete-time system dynamics are incorporated by applying them as the linear equality constraints (2.2b) at every sampling instant, where $A_k \in \mathbb{R}^{n\times n}$ and $B_k \in \mathbb{R}^{n\times m}$ are the state-space matrices describing the linear discrete-time system at sampling instant k and $\hat{x}_0 \in \mathbb{R}^n$

is the current measured system state. The trajectory constraints are imposed as the linear inequality constraints (2.2c) at every sampling instant, where $E_k^u \in \mathbb{R}^{l\times m}$ and $E_k^x \in \mathbb{R}^{l\times n}$ are the constraint coefficient matrices, and the vector $c_k \in \mathbb{R}^l$ is the vector of bounds for the inequality constraints. The terminal state constraints are given by (2.2d), where $E_N^x \in \mathbb{R}^{j\times n}$ and $c_N \in \mathbb{R}^j$ are the constraint's coefficient matrix and vector of bounds, respectively.

Problem (2.2) can be further simplified for Linear Time-invariant (LTI) problems, forming the LTI CLQR. In these problems, the linear system is LTI and the constraints and weight matrices remain the same across the entire horizon. This means the matrices A_k, B_k, Q_k, R_k, S_k, E_k^u and E_k^x are no longer dependent on k and become A, B, Q, R, S, E^u and E^x, respectively. We then represent the transfer function matrix for the discrete-time LTI system in this problem as

$$\mathcal{G}(z) \coloneqq \left[\begin{array}{c|c} A & B \\ \hline I & 0 \end{array}\right].$$

Choosing the terminal weight matrix P in problem (2.2) can be crucial to the stability and performance of the closed-loop controller. The simplest choice is to set $P = Q$, so that final states are weighted the same as other states. This allows for the simple formation of the problem matrices, but does not provide stability guarantees for the closed-loop problem.

Instead, a possible choice for P is to either choose it to be the solution to the Discrete Algebraic Riccati Equation (DARE)

$$P = A'PA + Q - A'PB(B'PB + R)^{-1}B'PA, \tag{2.3}$$

or choose P to be the solution to the discrete Lyapunov equation

$$P = A'PA + Q, \tag{2.4}$$

where Q and R are the cost matrices from Problem (2.2), and A and B are the system matrices. Choosing P in these ways approximates the cost function value for the time after the horizon ends, and allows for the derivation of closed-loop stability guarantees for the controller [114].

Discretized continuous-time formulation

The CLQR formulation presented in (2.2) is built using a discrete-time system and a discrete-time cost function, however many systems and objectives are specified in continuous-time. The equivalent continuous-time formulation of the finite-horizon LTI problem is

$$\min_{x,u} \frac{1}{2}x'(T)P^c x(T) + \frac{1}{2}\int_0^T x'(\tau)Q^c x(\tau) + u'(\tau)R^c u(\tau)\,\mathrm{d}\tau \tag{2.5a}$$

$$\text{s.t. } \dot{x}(t) = A^c x(t) + B^c u(t) \tag{2.5b}$$
$$x(0) = \hat{x}_0$$
$$E^x x(t) + E^u u(t) \leq c, \quad \forall t \in [0, T) \tag{2.5c}$$
$$E_N^x x(T) \leq c_N, \tag{2.5d}$$

where T is the horizon length, A^c and B^c are the continuous-time state space matrices for the LTI dynamical system, and Q^c, R^c and P^c are the weight matrices.

To solve (2.5), the problem is discretized to convert it into the discrete-time LTI formulation (2.2). There are several ways to do this, with the simplest being to

1. represent the input as a zero-order hold between sampling instants,
2. discretize the system using an exponential integrator with sampling time Δ_t,
3. use the continuous-time cost matrices in the discrete-time problem (i.e. $Q = \Delta_t Q^c$, $R = \Delta_t R^c$, and $P = \Delta_t P^c$), and
4. apply the constraints only at the sampling instants.

While this may be the simplest way of discretizing the problem, it will not ensure the resulting discrete-time controller will be the same as the continuous-time controller.

In order to have the discrete-time controller achieve the same cost as the continuous-time controller, the weighting matrices must be generated from the continuous-time problem through the discretization procedure given in [26]. In this procedure, the discrete-time matrices are given by

$$\Phi(\Delta_t) \coloneqq e^{A^c \Delta_t}, \tag{2.6a}$$
$$\Gamma(\Delta_t) \coloneqq \int_0^{\Delta_t} e^{A^c(\Delta_t - \tau)} \, \mathrm{d}\tau B^c, \tag{2.6b}$$
$$Q \coloneqq \int_0^{\Delta_t} \Phi^*(\tau) Q^c \Phi(\tau) \, \mathrm{d}\tau, \tag{2.6c}$$
$$S \coloneqq \int_0^{\Delta_t} \Phi^*(\tau) Q^c \Gamma(\tau) \, \mathrm{d}\tau, \tag{2.6d}$$
$$R \coloneqq \Delta_t R^c + \int_0^{\Delta_\tau} \Gamma^*(\tau) Q^c \Gamma(\tau) \, \mathrm{d}\tau. \tag{2.6e}$$

with the sampling time Δ_t.

Note that with this discretization of the cost function, while the continuous-time formulation only has the state and input weight matrices Q^c and R^c, the resulting discrete-time formulation will also include a state-input cross term S matrix and may destroy the sparsity of the weight matrices.

2.2 Nonlinear MPC algorithms

There are many different algorithms for solving the nonlinear MPC optimization problem (2.1), with those that have been implemented on FPGAs shown in the algorithm taxonomy chart presented in Figure 2.1. Overall, these algorithms can be divided into four distinct groups:

- interior-point methods,
- Sequential Quadratic Programming (SQP) methods,
- learning/approximation methods, and
- derivative-free methods.

Interior-point and SQP methods generally solve a discrete-time finite-dimensional optimization problem transcribed from the continuous-time infinite-dimensional optimization problem (2.1) using a method such as direct collocation [93] or multiple shooting [22]. These methods require that the derivatives of the cost, dynamics and the constraints either be known beforehand (i.e. analytically computable by hand or by using automatic differentiation tools), or be found using online methods such as finite-difference approximation. This means that implementing problems containing nonsmooth components in the cost/dynamics/constraints or problems where the dynamics are a blackbox simulator requires a relaxation technique to be used to find the derivatives.

In the learning/approximation methods, the objective is to learn the NMPC control law using a neural network or other function approximator. This generally consists of an offline phase where many closed-loop simulations are performed using a different NMPC controller (e.g. an interior-point or SQP-based solver) to create a set of mappings from the current state to the next control input that are then used to train/fit a neural network/function approximation. At run-time, the controller is implemented using the trained neural network/function approximator, allowing for faster computation of the control input. This has some disadvantages though, namely that the learned/approximated controller no longer has the constraint satisfaction guarantees of the original NMPC controller, and the approximation could create non-optimal trajectories for parts of the state space that were not simulated in the offline training phase (e.g. if the learned controller experiences non-nominal trajectories or disturbances).

An alternative method is the use of Derivative-free Optimization (DFO) solvers to solve the NMPC problem (2.1). DFO solvers operate by selecting points in the search space of the optimization problem and then computing the cost at each point by evaluating the cost function. The main differences between the various DFO solvers are how the points

in the search space are selected and how the information about the cost function value and constraints are integrated into the solver. The use of these solvers for the NMPC problem (2.1) has several advantages, including

- solving the original continuous-time NMPC problem (2.1),
- handling nonsmooth cost functions/constraints/dynamics,
- handling blackbox/simulation-only dynamic models and objective functions, and
- embarrassingly-parallel algorithm architectures.

2.2.1 DFO formulations

In DFO, there are many different types of solvers that each have their own unique features and abilities. Many of these solvers have been used to solve the NMPC problem in the past, and here we present a brief summary of some notable implementations.

Nelder-Mead method

An early experiment done by [157] used the Nelder-Mead method to solve the NMPC problem (2.1). In their formulation, the search space that the optimizer uses is a vector of piecewise-constant control inputs to a discrete-time dynamical system. At each sampling point, a *single-shooting* simulation of the discrete-time dynamics is performed (i.e. the dynamics are simulated using a time-marching dynamics solver starting at t_0 and going to t_f). The constraints (2.1c) are enforced as penalty functions in the cost, with the constraint violation only computed at the time points where the control inputs can change. They found that the algorithm allowed for many different types of parallelizations, including during the evaluation of the dynamics and by evaluating multiple sampling points at the same time.

Time-domain simulation method

Another method proposed by [89, 90] consists of performing a fixed number of time-domain simulations on a discrete set of candidate input trajectories, and then choosing the input trajectory that has the lowest cost. In this method, the candidate piecewise-constant input trajectories are formed sample-by-sample by starting with a nominal input value for the first sampling instant and perturbing it by some small amounts. Then for each of those trajectories, the next input is generated by perturbing another nominal value by a small amount. In this way, the final simulations result in an exponentially growing tree, where starting with just 2 inputs and 3 possible perturbations for each of those inputs with a control horizon length of 2 leads to 81 total simulations. In this formulation, constraints are handled through barrier terms in the cost function and are enforced only at the time points where the control

inputs can change. After all the simulations are completed, the input trajectory with the lowest computed cost value is chosen as the input sequence to be applied to the system. While simple to implement and easy to parallelize, this formulation has the disadvantage that as the desired horizon length and number of inputs/perturbations increase, the number of trajectories simulated grows exponentially — making this method infeasible for long time scales.

Trust-region methods

A slightly different method that combines ideas from derivative-free optimization and sequential quadratic programming is the trust-region formulation proposed by [42]. In this method, the algorithm begins by sampling the cost function at a set of points in the search space of piecewise-constant input trajectories for the system. Then those samples are used to fit a quadratic function to approximate the mapping between the trajectories and the cost function, creating a quadratic surrogate model of the cost function. The quadratic surrogate model is then used as the objective for an SQP solver to find the next likely optimum point. Then, the optimum point from the SQP replaces one of the existing points used to fit the quadratic surrogate function, and the process repeats with the new set of points. This formulation passes the path constraints (2.1c) directly into the SQP subproblems as constraints on the SQP problem, discretizing them to be only enforced at the time points where the control inputs can change. While this formulation overcomes the exponential-growth of other methods and allows for nonsmooth cost and dynamics, it comes at a disadvantage because the path constraints are used directly in the SQP solver, and therefore both derivatives of the path constraints and an SQP solver that can handle the nonlinearities in the path constraints are needed.

Particle swarm methods

Another popular DFO method used for NMPC is Particle Swarm Optimization (PSO), which is an optimization method inspired by how swarms of animals behave. In PSO, the candidate trajectories are thought of as particles in a swarm, and they will move around the search space based on their previous position and the positions of other particles with a lower cost value. In each iteration, the particles compute a velocity using an update rule similar to

$$
\begin{aligned}
V^{k+1} &= \omega V^k + c_1 r_1 \left(P^k - X^k\right) + c_2 r_2 \left(P_g^k - X^k\right), \\
X^{k+1} &= X^k + V^{k+1},
\end{aligned}
$$

where X^k and V^k are the position and velocity, respectively, for the particle at iteration k, P^k is the position with the lowest cost value the particle has ever found, and P_g^k is the position with the lowest cost value found by the entire swarm so far. The trajectories of the particles can then be modified using the cognitive and social parameters c_1 and c_2, respectively, with r_1 and r_2 random numbers uniformly distributed in the range $[0, 1]$.

The NMPC formulation for PSO proposed by [179] has the particles searching over piecewise-constant input trajectories, and then computing the NMPC cost (2.1a) using a single-shooting simulation of the dynamics. The path constraints (2.1c) are implemented using a penalty term in the cost function that sums the constraint violations, enforcing them only at the time points where the control inputs can change. While conceptually simple, the behavior of PSO relies heavily on the constants c_1 and c_2 in the update rule, meaning they will need to be optimized/chosen for each problem.

Evolutionary methods

A relatively recent method explored for NMPC using DFO in [74] is to use an evolutionary optimization algorithm. Evolutionary optimization is modelled after the process of biological evolution, with the algorithm seeking to mimic the natural evolutionary behaviors genes go through when passed from parents to children. Evolutionary methods begin by selecting a set of parent points in the search space, and then evaluating the cost function at those points. Then one of two actions occurs to produce a new set of child points:

- *mating/crossover* - the parents with the lowest cost values are combined together, randomly taking each individual variable from each one to form the new children (i.e. mating 2 parents will form a new child with each parent contributing 50% of the elements); or
- *mutation* - Gaussian noise is added to the children to encourage more exploration.

Once the new set of children is formed, the cost function is evaluated at each of them, and then the children with the lowest cost values are used as the parents in the next iteration.

In the nonlinear evolutionary MPC proposed by [74], the evolutionary optimization algorithm is used to solve the NMPC problem. The cost (2.1a) is computed using a single-shooting simulation of the dynamics, and the path constraints are implemented by introducing penalty parameters to the cost function or by saturating the inputs/states in the simulation model for input-only or state-only constraints. In [74], the input space was parameterized as a piecewise-linear function with fixed knot points at the start, middle, and end of the horizon. The optimization solver then searched for the value of the input at those knot points.

Pattern search methods

The final method we highlight is the set of pattern search methods for DFO. These methods choose a set of poll points on a mesh surrounding the current iterate (usually in fixed directions, such as along the axes), and then evaluate the cost function at each of them to identify the ones with a lower cost value. The poll point with the lowest cost value becomes the new iterate for the next iteration, and the algorithm repeats. In the NMPC framework using pattern search proposed by [53], the input trajectory is assumed to be piecewise-constant and the cost function performs a single-shooting simulation of the dynamics for the entire horizon. Adding constraints to the problem is then done one of two ways:

- *input constraints* - implemented by limiting poll points to be only in the feasible set; and
- *state constraints* - implemented as saturations in the dynamics simulation done in the cost function.

2.3 Linear MPC formulations

There are two main MPC schemes used to solve the CLQR problem: implicit and explicit. In implicit linear MPC, the CLQR QP (2.2) is solved using a QP solver at every sampling instant — requiring a full QP solver to be implemented on the embedded device. This can be expensive in terms of both resource usage and time required, making it impractical for some systems that require very fast sampling times or that need to use very small processors.

Alternately, there is the explicit linear MPC scheme that was proposed in [20]. The basic premise of this scheme is that the state space for the initial condition $\hat{x}_0$ can be broken into polyhedral regions, and the solution of the QP (2.2) in each of those regions is then described by an affine function of $\hat{x}_0$. This allows for the computation to be split into two parts: the offline computation of all the polyhedral regions and the associated affine function parameters, and the online computation of locating the appropriate affine function parameters in memory and evaluating the affine function with $\hat{x}_0$. The major downside to this method though is that the amount of memory needed to store the static gain matrices for all the polyhedral regions grows rapidly with the size of the problem (both number of states and the horizon length), making it impractical to use on embedded devices for problems with a large state-space or long horizons.

2.3.1 Implicit MPC formulations

There are several different formulations of optimization problems proposed to solve the implicit linear MPC problem.

Sparse (uncondensed) form

The simplest formulation is known as the *sparse* (or *uncondensed*) formulation, and solves the full QP (2.2). This formulation results in sparse and banded matrices where the number of non-zero elements in them grows linearly with the horizon length. This can be advantageous in several situations, including (i) the QP solver used can exploit matrix sparsity, (ii) there are inequality constraints (2.2c) involving the state variables, or (iii) a long prediction horizon is needed.

Condensed form

Another popular formulation is the *condensed* formulation, which removes the state variables x_k from the QP by rewriting them as a function of only the inputs u_k. When the terminal state constraint has $E_N^x = E^x$ and $c_N = c$, this results in the inequality constrained QP

$$\min_u \ \frac{1}{2} u' H u + \hat{x}_0' J' u \tag{2.7a}$$

$$\text{s.t. } Gu \leq F\hat{x}_0 + g, \tag{2.7b}$$

with $u \coloneqq \begin{bmatrix} u_0' & u_1' & \dots & u_{N-1}' \end{bmatrix}'$, the condensed Hessian $H \coloneqq \Gamma' \bar{Q} \Gamma + \bar{S}' \Gamma + \Gamma' \bar{S} + \bar{R}$, and the matrices $J \coloneqq \Gamma' \bar{Q} \Phi$,

$$\Phi \coloneqq \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}, \quad \tilde{\Phi} \coloneqq \begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^{N-1} \end{bmatrix}, \quad \Gamma \coloneqq \begin{bmatrix} B & 0 & 0 & & 0 \\ AB & B & 0 & & 0 \\ A^2B & AB & B & & 0 \\ \vdots & & & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \cdots & B \end{bmatrix},$$

$$\bar{R} \coloneqq I_N \otimes R, \quad \bar{S} \coloneqq \begin{bmatrix} I_{N-1} \otimes S & 0 \\ 0 & 0_{n \times m} \end{bmatrix}, \quad \bar{Q} \coloneqq \begin{bmatrix} I_{N-1} \otimes Q & 0 \\ 0 & P \end{bmatrix}, \quad \tilde{\Gamma} \coloneqq \begin{bmatrix} 0_{n \times Nm} \\ \begin{bmatrix} I_{n(N-1)} & 0 \end{bmatrix} \Gamma \end{bmatrix},$$

$$G \coloneqq \bar{E}^x \tilde{\Gamma} + \bar{E}^u, \quad \bar{E}^x \coloneqq I_N \otimes E^x, \quad \bar{E}^u \coloneqq I_N \otimes E^u,$$

$$F \coloneqq -\bar{E}^x \tilde{\Phi}, \quad g \coloneqq \mathbf{1}_N \otimes c.$$

The condensed QP (2.7) has no equality constraints, and has a smaller number of variables than the sparse form. However, the Hessian H is full in general, and the number of entries in it grows quadratically with the horizon length and the number of inputs — making this formulation more suited for problems with fewer inputs or shorter horizons.

An important note about condensing the problem is that the structure of the constraint matrix G will depend on the types of linear inequality constraints present. If there are only inequality constraints on the inputs, then G will be block diagonal. Otherwise, if there are inequality constraints on both the states and inputs, G will be full and lower triangular.

The condensed form can be further modified into the *dual-condensed* formulation by using the following dual QP of (2.7)

$$y^*(\hat{x}_0) \coloneqq \underset{y}{\operatorname{argmin}} \ \frac{1}{2} y^T H_d y + (J_d \hat{x}_0 + g)^T y \tag{2.8a}$$

$$\text{s.t. } y \geq 0 \tag{2.8b}$$

where $H_d \coloneqq GH^{-1}G'$ is the dual-condensed Hessian, and $J_d \coloneqq GH^{-1}\Phi + F$. Since there is strong duality between (2.7) and (2.8), the control sequence can be recovered from a dual-optimal solution y^* through $u^* = -H^{-1}(G'y^*(\hat{x}_0) + J'\hat{x}_0)$.

In the dual-condensed form, the optimization variables, y, are the Lagrange multipliers for the inequality constraints, leading to only non-negativity constraints on every variable. However, the Hessian H_d is still a full matrix whose number of non-zero entries grows quadratically with the horizon length, in general.

Variable condensing forms

The uncondensed and condensed formulations given above can be seen as two ends of a condensing spectrum. Several methods for *variable condensing* have been proposed between the two extremes to give the designer control over the amount of sparsity in the resulting optimization problem.

One possible approach was proposed in [16], where instead of condensing the entire horizon into a single full matrix, the horizon is broken into smaller segments that are then condensed individually. An alternate approach was proposed in [80], where the open-loop dynamical system (2.2b) is replaced with a closed-loop system with a deadbeat linear state-feedback controller, creating a block banded structure in the resulting condensed Hessian.

2.3.2 Numerically robust MPC

When implementing a CLQR controller for unstable systems, the condensed formulation (2.7) becomes numerically unstable as the horizon length increases. This is due to the fact that unstable systems have $|\lambda_{max}(A)| > 1$, and so taking repeated powers of A to form Γ will then cause the condition number of H to grow monotonically with N.

To overcome this, a modification to (2.7) was proposed in [154] where instead of using the actual system A for computing the prediction matrix and optimal control, a prestabilized system $A-BK$ is used instead. This prestabilized system would guarantee that the prediction matrix entries do not cause the condition number of H to be unbounded with the horizon length, leading to better conditioning of the Hessian and the resulting optimization problem.

To formulate the prestabilized problem, a new system $\mathcal{G}_c$ is formed by introducing a state feedback controller K and a new input $v_k \in \mathbb{R}^m$, to make the original input $u_k = -Kx_k + v_k$ and giving the system

$$\mathcal{G}_c(z) \coloneqq \left[\begin{array}{c|c} A-BK & B \\ \hline I & 0 \end{array}\right], \tag{2.9}$$

where the controller $K \in \mathbb{R}^{m\times n}$ is chosen so that the closed-loop system $\mathcal{G}_c$ is Schur-stable (i.e. $|\lambda_{max}(A-BK)| < 1$). Note that because all eigenvalues of the system $\mathcal{G}_c$ need to be inside the unit circle, this transformation requires the pair (A, B) of the original system $\mathcal{G}$ to be stabilizable. There are several ways to compute a K for $\mathcal{G}_c$, however we focus on when K is chosen as the unconstrained infinite-horizon LQR controller computed using Q, R and S from (2.2a).

The computations are then done using the input space $v \coloneqq \begin{bmatrix} v_0' & v_1' & \cdots & v_{N-1}' \end{bmatrix}'$, turning the condensed QP (2.7) into

$$v^*(\hat{x}) \coloneqq \underset{v}{\operatorname{argmin}} \ \frac{1}{2} v' H_c v + \hat{x}' J_c' v \tag{2.10a}$$

$$\text{s.t. } G_c v \leq F_c \hat{x} + g \tag{2.10b}$$

with $A_c \coloneqq A - BK$, $Q_c \coloneqq Q + K'RK$, $\bar{K} \coloneqq I_N \otimes -K$,

$$\Phi_c \coloneqq \begin{bmatrix} A_c \\ A_c^2 \\ A_c^3 \\ \vdots \\ A_c^N \end{bmatrix}, \quad \tilde{\Phi}_c \coloneqq \begin{bmatrix} I \\ A_c \\ A_c^2 \\ \vdots \\ A_c^{N-1} \end{bmatrix}, \quad \Gamma_c \coloneqq \begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ A_cB & B & 0 & & 0 \\ A_c^2B & A_cB & B & & 0 \\ \vdots & & & \ddots & \vdots \\ A_c^{N-1}B & A_c^{N-2}B & A_c^{N-3}B & \cdots & B \end{bmatrix}, \tag{2.11}$$

$$\bar{Q}_c \coloneqq \begin{bmatrix} I_{N-1} \otimes Q_c & 0 \\ 0 & P \end{bmatrix}, \quad \tilde{\Gamma}_c \coloneqq \begin{bmatrix} 0_{n\times Nm} \\ \begin{bmatrix} I_{n(N-1)} & 0 \end{bmatrix} \Gamma_c \end{bmatrix}, \quad H_c \coloneqq \Gamma_c' \bar{Q}_c \Gamma_c + \Gamma_c' \bar{K}' \bar{R} + \bar{R}\bar{K}\Gamma_c + \bar{R},$$

$$J_c \coloneqq ((I + \Gamma_c'\bar{K}')\bar{R}\bar{K} + \Gamma_c'\bar{Q}_c)\Phi_c, \quad G_c \coloneqq (\bar{E}^x + \bar{E}^u\bar{K})\tilde{\Gamma}_c + \bar{E}^u, \quad F_c \coloneqq -(\bar{E}^x + \bar{E}^u\bar{K})\tilde{\Phi}_c. \tag{2.12}$$

The input applied to the original system is then $u_0 = -K\hat{x}_0 + v_0^*(\hat{x}_0)$.

2.4 Implementation of linear MPC

One of the key attributes that has led to the adoption of MPC as a viable control strategy in many applications has been the ability to implement the controller in real time on systems that require very fast sampling times. One key method of satisfying these real-time constraints is to use an FPGA with custom digital logic to solve the QP efficiently as the computational platform. In this thesis, we examine and analyze FPGA implementations of linear MPC in Part II, so in this section we will present several key ideas and considerations utilized when implementing QP solvers on an FPGA system.

2.4.1 Quadratic programming algorithms

There have been many types of optimization algorithms used to implement linear MPC on FPGAs, as shown in the taxonomy chart in Figure 2.1. This taxonomy chart shows there is a diverse ecosystem of solvers for nonlinear MPC, with each algorithm chosen for the specific application being targetted. However, when solving linear MPC there are three main classes of implicit QP algorithms: Interior-point Methods (IPMs), Active-set Methods (ASMs), and First-order Methods. In this overview, we focus on the properties of these implicit QP solvers for linear MPC.

Interior-point methods were some of the original methods applied to solve (2.2) in [103] and subsequent works. In IPM, the optimal solution of (2.2) is found by solving the nonlinear Karush-Kuhn-Tucker (KKT) system of equations using an iterative Newton's method, with each iteration in IPM consisting of three main steps:

1. update/linearize the KKT system,
2. solve the linear KKT system for a step direction, and then
3. update the current guess with the step direction.

The main computational burden is solving the linear system in Step 2. Experiments in [149] showed that exploiting the structure of the matrices arising in the LTI problem leads to faster and more scalable solvers.

The second type of algorithm that has been implemented on FPGA systems is the active-set method. In ASM, the algorithm finds the optimal point by solving a sequence of equality-constrained subproblems to locate the set of inequality constraints that are active at the optimal point. In each iteration, the algorithm chooses one or more inequality constraints to add/remove from the current active set being used based on whether the constraints were violated in previous iterations. In ASM solvers, the main computational burden is then solving a linear system representing the equality-constrained problem.

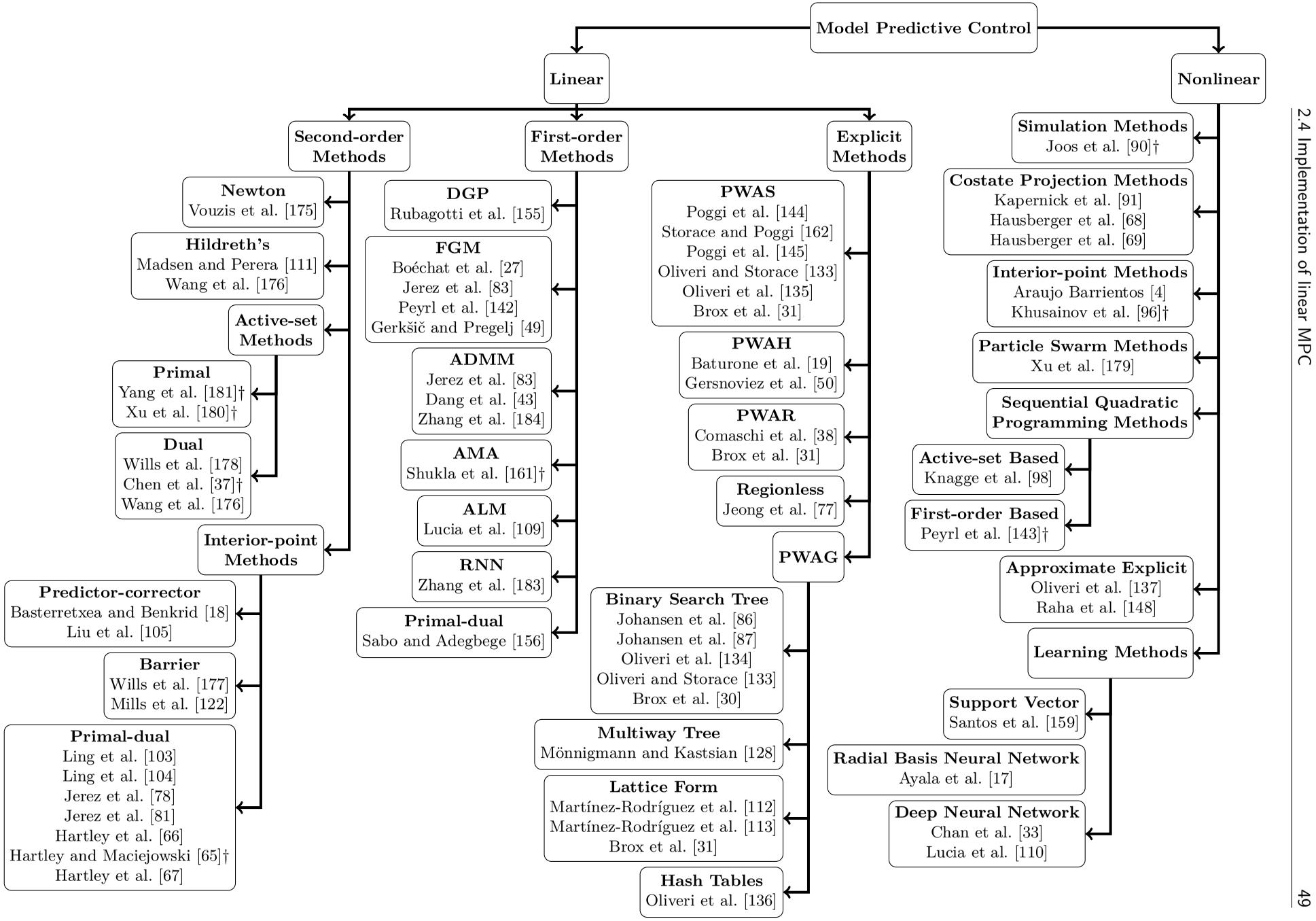


Figure 2.1: Taxonomy of various optimization algorithms used to implement MPC on FPGAs (adapted from [116]).
† Heterogeneous implementations

Table 2.1: FPGA implementation details for selected linear MPC solvers (adapted from [116]).

Implementation	QP[1]	Algorithm[2]	Number Format[3]	Design Entry	Clock (MHz)	Matrix[4]	QP Size[5]	Solver Time
Ling et al. [103]	C	IP/CHOL	float32	Handel-C	20	BRAM	3/0/60	23.7 ms[6]
Ling et al. [104]	C	IP/CHOL	float32	Handel-C	25	-	3/0/52	9.1 ms
Vouzis et al. [175]	C	Newton	LNS	Verilog	50	BRAM	2/0/4	688 μs
Basterretxea et al. [18]	C	IP/CHOL	fixed	AccelDSP	20	BRAM	3/0/6	120 μs
Wills et al. [177]	C	IP/CG	float16.6	VHDL	70	-	12/0/24	<200 μs
Yang et al. [181]	C	ASM	float32/fixed	Verilog	100	BRAM	5/0/-	500 μs[9]
Chen et al. [37]	C	DASM	float32	Verilog	150	-	3/0/12	468 μs
Wills et al. [178]	C	ASM	float7	VHDL	70	BRAM	12/0/24	<30 μs
Peyrl et al. [142]	C	FGM	fixed27.25	VHDL	120	-	15/0/30	0.49 μs
Jerez et al. [83]	C	FGM	fixed	VHDL	400/230	BRAM	40/0/80[7]	0.53/0.91 μs
Rubagotti et al. [155]	C	DGP	fixed32.16	Simulink	100	DiRAM	20/0/208	239 μs
Xu et al. [180]	C	ASM	float32/fixed	Verilog	100	BRAM	5/0/-	370 μs
Zhang et al. [183]	C	RNN	-	-	-	-	8/-/8	6.02 μs
Wang et al. [176]	C	ASM	float32	C[10]	10	BRAM	40/-/80[7]	1.85 ms
Wang et al. [176]	C	ASM	binary16	C[10]	10	BRAM	40/-/80[7]	1.28 ms
Wang et al. [176]	C	ASM	fixed	C[10]	10	BRAM	40/-/80[7]	860 μs
Liu et al. [105]	S	IP/CHOL	float32	VHDL	200	BRAM	300/-/600	4 ms
Hartley et al. [67]	S	IP/MINRES	float32	VHDL	250	BRAM	377/-/408	<12 ms
Jerez et al. [83]	S	ADMM	fixed	VHDL	400/230	BRAM	216/-/172[7]	4.9/8.52 μs
Dang et al. [43]	S	ADMM	fixed32.17	-	-	-	120/80/250	215 ms[6]
Shukla et al. [161]	S	AMA	float32	C[8]	100	BRAM	384/-/-	900 μs[6]
Zhang et al. [184]	S	ADMM	float32	VHDL	340	BRAM	204/-/300[7]	30.1 μs

- Indicates data that was not reported

[1] C - Condensed formulation, S - Uncondensed (Sparse) formulation

[2] ADMM - Alternating Direction Method of Multipliers, AMA - Alternating Minimization Algorithm, (D)ASM - (Dual) Active Set Method, CG - Conjugate Gradient solver, CHOL - Cholesky solver, DGP - Dual Gradient projection, (D)FGM - (Dual) Fast Gradient Method, IP - Interior-Point, MINRES - Minimum Residual solver, RNN - Recurrent Neural Network

[3] float32 - IEEE-754 single precision floating point, binary16 - IEEE-754 half precision floating point, LNS - Logarithmic Number System [166]

[4] BRAM - Block RAM, DiRAM - Distributed RAM

[5] Decision Variables/Equality Constraints/Inequality Constraints

[6] Fully sequential implementation

[7] Estimated from available data

[8] Using the Protoip tool by Suardi et al. [165]

[9] This value was actually reported in [180], but was run using the architecture from [181]

[10] Using Vivado HLS

A comparison between IPM and ASM on FPGAs was carried out in [101], with the results showing that for small problem sizes, ASM will take fewer iterations compared to the fixed computational cost of IPM. However, it appears that experimentally the number of interior-point iterations stays constant as the problem size grows, while the number of iterations required for ASM tends to grow approximately linearly with the number of optimization variables and constraints. This leads to the conclusion that for larger problems, IPM is better to use than ASM.

More recently, many FPGA implementations have started to use first-order methods, such as Nesterov's Fast Gradient Method (FGM) or the Alternating Direction Method of Multipliers (ADMM), instead of the second-order interior-point or active-set methods. These methods utilize only first-order information (i.e. the gradient) in their computations, and generally have three main steps

1. compute a search direction,
2. compute the step size and apply the search direction, and then
3. project onto the feasible set.

The main linear algebra operation in first-order methods is a matrix-vector multiplication in Step 1 (as opposed to the linear system solve in IPM or ASM). However, the projection operation in Step 3 can become computationally complex depending on the feasible set. For both primal gradient methods with a constraint set (2.2c) of purely upper and lower bounds and dual gradient methods, the projection operation is a simple variable saturation. However, if E_k^x and E_k^u represent a more complex constraint set, the projection operation for a primal gradient method may require the solution of its own QP through techniques such as a multi-parametric QP as proposed in [121] (which will essentially perform the projection operation using the same digital hardware as an explicit MPC solver).

2.4.2 Number representation

An important feature of FPGAs compared to CPUs is their support for non-standard data types in the calculations. This allows the designer to create hardware using custom data types, which can then reduce the amount of device area consumed by the individual computational resources and allow for more parallelism [40].

In this subsection we will introduce fixed-point and floating-point number formats, and then describe two types of numerical errors that are encountered: computational errors and representation errors.

Storage format

There are two main storage formats for decimal numbers in digital systems: fixed-point and floating-point.

Fixed-point numbers utilize the standard 2's complement number representation (used for normal integers) to represent both positive and negative numbers, but introduce an implied binary point into the numbers. The format of the number is characterized by two values: the number of integer bits and the number of fractional bits, with the sum of these two lengths called the word length. For example, the number 14.375 would be 00001110.01100 in the 8.5 format (where there are 8 integer bits and 5 fractional bits). The binary point is an abstraction created by the designer, with the hardware having no knowledge of where it is actually located. This means 14.375 is actually stored in the fixed-point 8.5 format as if it were the integer 460:

$$\underbrace{00001110}_{\text{integer}}\,\underbrace{01100}_{\text{fraction}}$$

To actually perform the operations (e.g. addition, multiplication, etc.) on fixed-point numbers, the standard integer arithmetic units are used. This allows for the computations to be mapped to the Digital Signal Processing (DSP) units of the FPGA to accelerate them.

The number of bits in each section of a fixed-point number is fixed at design-time, forcing a fixed range and precision on the numbers represented. Any values larger than the number of integer bits will overflow, and have the information to the left of the integer portion lost. Any values with fractional components smaller than the precision of the number will have those parts rounded to the precision of the fixed-point number, with the difference between the represented number and the actual number termed the round-off error. There are several rounding modes possible, with the common rounding modes shown in Table 2.2 along with the largest possible round-off error they could introduce.

An alternative representation for numbers is called floating-point, and is the commonly used format when performing computations on desktop and server computers. This format is characterized by three main parts: the sign bit, the mantissa, and the exponent. The number is effectively stored in scientific notation, where the mantissa provides the fractional part of the coefficient (there is normally an implied 1 before it) and the exponent provides the location of the binary point. The original number can then be retrieved as follows

$$y = (-1)^{\text{sign}} \times 1.(\text{mantissa}) \times 2^{\text{exponent}-\text{bias}}$$

The most common field sizes for the numbers correspond to the IEEE-754 standard, and are given in Table 2.3, but designers can also create custom formats by choosing other mantissa and exponent sizes.

Table 2.2: Rounding modes in fixed-point arithmetic data types and their maximum round-off error.

Rounding mode	**Maximum Round-off Error with f fractional bits**
Round to $+\infty$	$\epsilon_f = 2^{-f}$
Round to 0 (truncation to 0)	$\epsilon_f = 2^{-f}$ for negative numbers $\epsilon_f = -2^{-f}$ for positive numbers
Round to $-\infty$ (truncation)	$\epsilon_f = -2^{-f}$
Round to ∞	$\epsilon_f = -2^{-f}$ for negative numbers $\epsilon_f = 2^{-f}$ for positive numbers
Round towards the nearest value (convergent rounding)	$\epsilon_f = \pm 2^{-(f+1)}$

Table 2.3: Size of the fields (in bits) and the exponent bias in IEEE-754 standard floating-point formats.

Precision	**Total**	**Mantissa**	**Exponent**	**Bias**
Half	16	10	5	15
Single	32	23	8	127
Double	64	52	11	1023
Quad	128	112	15	16383

There are several important considerations when choosing between fixed-point and floating-point representation for implementations, including

- the range of possible values that must be represented,
- the amount of precision needed, and
- the available hardware components.

In general, floating-point numbers have a larger dynamic range than a fixed-point number using the same number of total bits (since floating-point numbers have an exponent term). On the other hand, fixed-point numbers have a precision that is constant for all representable numbers, while the precision available for floating-point numbers will vary depending on the magnitude of the number being represented. Fixed-point is also favored on low-power embedded systems, since many embedded microcontrollers or small FPGAs have no built-in hardware for efficiently performing floating-point operations. However, most higher-end CPUs include native hardware support for either the single or double precision format, and some newer FPGAs now include specialized blocks for floating-point operations as well.

Computational errors

Computational errors can manifest themselves in three ways:

Table 2.4: Works containing results on the stability under roundoff errors, variable bounds, and computational complexity certification of selected algorithms.

Algorithm	Stability under Roundoff Errors	Bounds on Variables/ Data-Type Size Rules	Finite Termination/ Upper Iteration Bound
Fast Gradient Method (FGM)	Jerez et al. [82]	Jerez et al. [82]	Richter et al. [151]
Alternating Direction Method of Multipliers (ADMM)	Jerez et al. [83]	Jerez et al. [83] Dang et al. [43]	-[1]
Dual Gradient Projection (DGP)	Patrinos et al. [140]	Patrinos et al. [140]	Patrinos et al. [140]
Accelerated Dual Gradient-Projection (GPAD)	-	-	Patrinos and Bemporad [138]
Proximal Newton Method	Guiggiani et al. [56]	Guiggiani et al. [56]	Guiggiani et al. [56]
MINRES/Lanczos	Greenbaum [55, §4]	Jerez et al. [84]	Greenbaum [55, §4]

- No results could be located

[1] While much work has been done to prove convergence rates of splitting methods and ADMM, these results require a number of assumptions that are not satisfied for MPC problems in general. Individual MPC problems may be able to meet the assumptions though, in which case iteration bounds do exist.

- *overflow*, where the magnitude of the requested number is larger than the maximum representable number;
- *underflow,* where the magnitude of the requested number is smaller than the smallest representable number; and
- *round-off,* which are errors that occur because the number requires more fractional places than available.

The generic optimization algorithms discussed in Section 2.4.1 were developed using either infinite precision or the standard data types (e.g. IEEE-754 floating-point) for execution on normal computers. When they are then implemented on embedded/FPGA systems, different data types are chosen, which then introduce new errors in the computation that could degrade the algorithm's stability or convergence. For instance, changing from floating-point to fixed-point introduces different round-off errors in the computations due to the differing dynamic-range and precisions. To guarantee the algorithms work with the new data types, new proofs of convergence and stability are needed. An overview of some of the relevant literature containing these proofs for the common algorithms used with linear MPC can be found in Table 2.4.

An additional concern, particularly for fixed-point implementations, is overflow errors. In general, finding the largest possible value each vector in an algorithm could contain is difficult due to the iterative nature of the computations. However, bounds do exist for some methods,

such as the fast gradient method [82] and dual gradient projection [139]. These bounds can then be used to size the integer and fractional components of fixed-point numbers in order to guarantee no overflow errors occur inside the algorithms.

Other algorithms, such as IPM, do not have bounds for every variable in the computation. Instead, bounds on variables in specific parts of the algorithm have been created. For instance, when the Minimum Residual (MINRES) solver is used for Step 2 of the IPM, [79] presented a scaling method that guarantees all vector elements in the MINRES solver will stay below precomputed bounds, allowing for a fixed-point implementation of MINRES inside an IPM.

Representation errors

In addition to examining the effect of the data type on the convergence of the algorithm, designers must be conscious of the effect the data type has on the optimization problem itself. For instance, the choice of the representation could cause the optimization problem to lose convexity or feasibility with respect to the original problem, or simply degrade the resulting optimal solution to an unsatisfactory level.

For example, experiments in [106] show that the discretization is affected by the data type, and that choosing the wrong combination of discretization method and data type can lead to severe performance problems. Their experiments show that the usual discretization method (known as the shift-form) performs very poorly when used inside an interior-point method with small data types (e.g. 5-bit mantissa floating-point). They traced the source of this problem to the merging of the state transition dynamics and the discrete integrator into a single value in the resulting matrices. In this case, the integrator forces a 1 on the matrix diagonal, and if the system has additional small-scale transition dynamics on the diagonal, they could become truncated by the number representation due to the fewer decimal places available near 1 of the low-precision floating-point numbers. To overcome this, they propose using the delta-domain discretization method to represent the discrete-time system to effectively separate the state integrator from the state transition computation. This then provides closed-loop control using 5-bit mantissa floating-point that is equivalent to the control using double precision (52-bit mantissa) floating-point.

2.4.3 Parallelization opportunities

Opportunities for parallelization of computations can occur at two distinct levels: algorithmic and computational. Algorithmic parallelizations are those inherent in the structure of the algorithm, while computational level parallelizations exist at the level of the arithmetic computations.

Algorithmic level

We use the term algorithmic level parallelization to refer to the ability to parallelize the overall algorithm steps inside a single iteration. The existence of these parallelizations is highly dependent upon the structure of the problem, and of the algorithm itself.

In IPM implementations on FPGAs (such as [67, 81, 105]), the main computational bottleneck is located inside Step 2 (the linear system solver), so attempting to parallelize the computations in Steps 1 and 3 provided no noticeable performance increase. Therefore, these implementations left Steps 1 and 3 with minimal parallelization and focused the research and development effort on the computational level parallelization available inside Step 2.

More recent FPGA implementations have examined the parallelizations possible in first-order methods, such as FGM or ADMM. Work in [83] demonstrates that, given an appropriate projection operation, there is no data dependence between the optimization variables in an iteration of these methods. This allows for the processing of each variable in parallel by duplicating the computational component containing a Matrix-vector Multiplier (MVM), projection block, and associated glue logic.

This parallelization is highly dependent upon the projection operator though, and the reported architectures utilize box constraints on the variables to allow for this exploitation. The introduction of more general affine constraints will introduce data dependencies in the projection operation, and make the parallelization structure highly dependent upon the structure of the constraint set. Additional work examined how to add soft constraints to the MPC problem, with [83] finding that in order to maximize the parallelization in the ADMM implementation there had to be one slack variable per soft constraint (e.g. a 1-norm soft constraint formulation).

Computational level

Many of the parallelizations that have been exploited in the FPGA implementations have been at the computational level rather than the algorithmic level. This level consists of the low-level linear algebra routines that compose the algorithm, such as the linear system solvers, matrix factorizations and matrix-vector multiplies.

An initial survey of parallelizing the linear system solvers was done in [107], where they explored the implementation of the conjugate gradient method using deeply pipelined inner-product units. Subsequent implementations in [67, 81] explored the implementation of the MINRES solver to speed-up Step 2 inside the interior-point method by parallelizing and pipelining the operations.

Parallelization of the MVM was used in nearly every implementation, but there were two main methods used: column-sweep and row-sweep. Column-sweep parallelism was implemented in [155], and consisted of a multiply-accumulate (MAC) module for each row of the matrix. This processes a column at a time, and will not produce a complete result until n clock cycles for an $m \times n$ matrix.

Row-sweep parallelism was implemented in [81, 83, 177], and consists of multiple units that compute the dot products between rows of the matrix and the vector. There is one multiplier per vector element, followed by an adder tree to combine all the results. This adder tree can have registers inserted between levels to reduce the critical path and allow for pipelining of different operations. A fully pipelined implementation (registers between every level) of a single MVM unit can produce the complete result in $m + \lceil \log_2 n \rceil$ clock cycles.

2.4.4 Memory organization

The architecture of the memory system consists of two main components: locality and the data storage pattern. Locality refers to how close the memory cells are to the computational elements, while the data storage pattern is how the data/matrices are arranged inside the memory cells.

Locality

There are three major memory regions that are available for FPGA implementations: Distributed RAM (DiRAM), Block RAM (BRAM), and off-chip memory. In the linear MPC implementations shown in Figure 2.1, none of them utilized off-chip memory, and all but one utilized the BRAM to store the main coefficient matrix (the Hessian), and the DiRAM for various computational products.

The remaining implementation, [155], stores the Hessian in DiRAM to have high locality with the MVM elements. For the active-set implementation done in [178], they experimented with placing the Hessian in the DiRAM and the BRAM. They reported results that show switching from BRAM to DiRAM will almost double the resource usage of the FPGA.

Storage pattern

The storage pattern for the matrices and vectors that an implementation utilizes is driven by both the choice of the linear MPC problem formulation and the computational architecture used. When designing the storage pattern in relation to the computational architecture, the primary objective is to present the data to the computational units with the smallest latency. For instance, for the column-sweep MVM implementation in [155], the Hessian is broken into

rows for storage at each MAC (each row in separate memory areas), allowing for each MAC to be fed with a new coefficient simultaneously. Alternately, the work [81] places each column of the Hessian in different BRAM units to efficiently feed the coefficients to the row-sweep MVM unit.

The chosen linear MPC problem formulation plays a key role in the scaling of the storage pattern as the problem size changes. For instance, in the FPGA implementations using the condensed formulation with a full Hessian, the memory usage of the Hessian scales quadratically with the horizon length.

Alternately, choosing the sparse formulation allows for the usage of the Compressed Diagonal Storage (CDS) method to represent the Hessian. As detailed in [28], CDS exploits the banded structure of an $m \times n$ matrix to only store the diagonals that contain values (those inside the bandwidth r). This is accomplished by turning each diagonal of the original matrix into a column of the CDS matrix, which then has dimensions $m \times r$. The CDS matrix then grows linearly with the horizon length. The CDS storage method was utilized in [81], where it decreased memory usage by at least an order of magnitude.

Further structure in the sparse LTI MPC problem can be exploited to realize gains of over 75% compared to the normal CDS implementation. Specifically, [81] shows that the CDS matrix for the LTI case contains many identical rows (with mostly zeros and ones). By storing only one copy of that row and then mapping the appropriate memory addresses to it when needed, memory usage can be reduced. Additional savings can also be found by exploiting the block structure of the matrix, so only one copy of a block is stored and then accessed through an appropriately designed memory system.

2.4.5 Overall trends

The implementation of linear MPC on FPGAs has seen substantial improvement over the past 15 years, with a diverse set of implementations that have pushed the computational time to sub-microsecond ranges. By examining the implementation details of many different solvers (provided in Table 2.1), several patterns emerge.

The first is that the recent speed-up in solver times has been accompanied by a switch to first-order methods, such as FGM for the condensed problem and ADMM for the sparse problem. This switch shows an order of magnitude reduction in the number of clock cycles required for the computation, as can be seen in the comparison presented in Figure 2.2.

Another pattern that emerges is that, unlike many initial implementations that were based on a high-level language, such as Simulink or Handel-C, the more performant implementations have been coded directly in a low-level Register-Transfer Level (RTL) language, such as VHDL or Verilog.

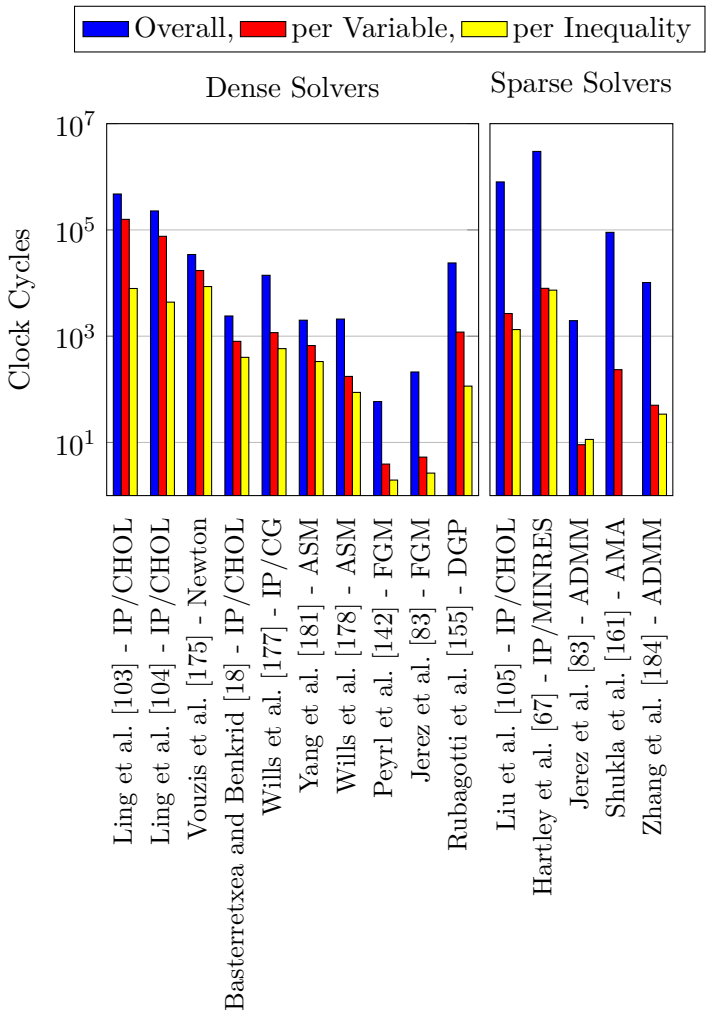


Figure 2.2: Clock cycles required for the computations in FPGA implementations of linear MPC. Per variable (inequality) values are the total clock cycles divided by the number of variables (inequality constraints). ©2018 IFAC.

2.5 First-order optimization solvers for linear MPC

As mentioned in Section 2.4.5, the current trend in embedded implementations of linear MPC seems to be heading towards first-order methods as the solvers of choice to meet strict timing demands. Therefore, we focus on the analysis and implementation of first-order methods for the CLQR problem in Part II, and briefly introduce in this section the two first-order methods we will use in Part II.

Algorithm 2.1 The Fast Gradient Method [83]

Input: $\hat{x}_0$ ▷ Current state measurement
Output: $u^{(I_{max})}$ ▷ Optimal control sequence

Let: L, μ be the maximum and minimum eigenvalue of H, respectively.
Let: $\beta \coloneqq \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$
Let: $y^{(0)} = 0$

for $i = 0$ to $I_{max} - 1$ **do**
  $t^{(i)} \coloneqq (I - \frac{1}{L}H)y^{(i)} + \frac{1}{L}J\hat{x}_0$ ▷ Gradient update
  $u^{(i+1)} \coloneqq \pi_{\mathbb{K}}(t^{(i)})$ ▷ Projection
  $y^{(i+1)} \coloneqq (1+\beta)t^{(i+1)} - \beta t^{(i)}$ ▷ Acceleration
end for

2.5.1 Fast Gradient Method

The main first-order method we examine in this work is the Fast Gradient Method (FGM), which was originally proposed by Nesterov to solve constrained convex programs, and was adapted in [151] to solve the condensed primal QP (2.7) with only input constraints. The FGM uses an accelerated gradient descent method to find the minimum of the cost (2.7a) while using a projection operator to satisfy the inequality constraints (2.7b). This algorithm has proven popular in embedded implementations, with some of the fastest reported FPGA implementations of linear MPC solvers utilizing it to solve the condensed problem.

The computational steps for the FGM can be seen in Algorithm 2.1, and consist of three main parts: gradient computation and update, projection into the feasible set, and acceleration. The two steps that can theoretically be the most computationally intensive are the gradient computation and the projection operation. In all implementations, the gradient step consists of a single matrix-vector multiplication of the previous accelerated iterate with the Hessian (which is usually full), followed by a vector addition operation. The projection step on the other hand can vary between implementations depending on the structure of the feasible set. In the majority of embedded implementations, the FGM is used with problems that have only upper and lower bounds, which then turns the projection step into a simple saturation operation and makes the main computational bottleneck become the gradient computation.

The FGM also has two other properties that make it amenable to implementation on embedded platforms

- a computable Upper Iteration Bound (UIB), and
- rules for sizing the fixed-point representation of all vectors while guaranteeing stability.

Upper iteration bound

Having a UIB means that the algorithm is guaranteed to converge in a fixed number of iterations, which provides two key benefits

- simple termination criteria (no residual computation), and
- constant algorithmic solve time no matter the initial state.

These benefits work together to allow for better implementations on FPGAs because it leads to fewer computations and a fixed solver latency. The UIB for the FGM was derived in [151], and is defined as

$$\text{UIB} \coloneqq \max\left\{0, \min\left\{a, b\right\}\right\}, \tag{2.13}$$
$$a \coloneqq \left\lceil \frac{\ln \epsilon - \ln \Delta}{\ln\left(1 - \sqrt{\frac{1}{\kappa}}\right)} \right\rceil, \quad b \coloneqq \left\lceil 2\sqrt{\frac{\Delta}{\epsilon}} - 2, \right\rceil,$$

where ϵ is the desired tolerance for the primal solution, κ is the condition number of the Hessian H, and Δ is a constant determined by the constraint set. Note that this bound is not known to be tight, and it is unknown if there exists a problem instance for which it is tight.

The value of Δ can be found by either solving one of the optimization problems given in [151] or utilizing an upper-bound, while the value for ϵ should be chosen to ensure stability properties of the system. A horizon-independent bound on ϵ and Δ for warm-started FGM was calculated in [151] as

$$\epsilon \leq \frac{\mu}{2}\frac{\delta_{max}^2}{\|B\|^2}, \qquad \Delta \leq \lim_{N\to\infty} \kappa\epsilon, \tag{2.14}$$

where μ is the smallest eigenvalue of the Hessian H, and δ_{max} is the largest acceptable error in the system state due to the suboptimal input (i.e. $\|x_+ - x_+^*\| \leq \delta_{max}$, where x_+ is the next state under the suboptimal input and x_+^* is the next state under the optimal input).

Data-type sizing rules

The implementation of the FGM in fixed-point arithmetic was analyzed in [83], which proposed several requirements and design guidelines to allow for a stable implementation. The first important part was deriving [83, Prop. 1], which contains the maximum values the vectors/values in the FGM can attain during all computations, allowing proper sizing of the integer portion of the fixed-point data types. The second part was stating a necessary condition to have stability of the FGM iteration using fixed-point arithmetic in the presence of round-off errors, which is given in Requirement 2.5.1.

Requirement 2.5.1 ([83, §IV-D])**.** *For the Fast Gradient Method to be stable in fixed-point arithmetic, it is necessary (but not sufficient) for the fixed-point version of the Hessian H to have all its eigenvalues in the open interval $(0, 1)$.*

Algorithm 2.2 The Dual Gradient Projection Method [140]

Input: $\hat{x}_0$ ▷ Current state measurement
Output: $y^{(I_{max})}$ ▷ Optimal Lagrange multipliers

Let: $L_d \coloneqq \lambda_{max}(H_d)$
Let: $E \coloneqq -H^{-1}G'$
Let: $e \coloneqq -H^{-1}\Phi\hat{x}_0$
Let: $b \coloneqq F\hat{x}_0 + g$
Let: $y^{(0)} = 0$

for $i = 0$ to $I_{max} - 1$ **do**
  $z^{(i)} \coloneqq Ey^{(i)} + e$ ▷ Compute current primal solution
  $g^{(i)} \coloneqq Gz^{(i)} - b$ ▷ Compute the dual gradient
  $y^{(i+1)} \coloneqq \left[y^{(i)} + \frac{1}{L_d}g^{(i)}\right]_+$ ▷ Multiplier update and projection
end for

This requirement essentially means that the fixed-point Hessian must be Schur-stable, which is a necessary step in ensuring the gradient vanishes over time and the FGM converges to a solution.

2.5.2 Dual Gradient Projection

Another first-order algorithm to solve the condensed linear MPC problem (2.7) is the Dual Gradient Projection (DGP) algorithm described in [140]. DGP is a non-accelerated gradient method that operates in the dual space by computing the Lagrange multipliers for the condensed dual problem (2.8). There are two main steps in the algorithm, which can be seen in Algorithm 2.2. The first step is to compute the dual gradient using the current primal solution and the associated constraint violations. The next step is to update the Lagrange multipliers using that gradient, and then project the Lagrange multipliers onto the non-negative orthant (which is a simple saturation operation). Since the projection can be done using a saturation operation regardless of the constraint set (2.7b), the DGP is more suitable for embedded applications with complex constraint sets than the FGM is.

Work has also been done to create fixed-point arithmetic design rules for the DGP, with [140] presenting results on the required number of fractional and integer bits. Additional results in [140] presented the following UIB for the DGP algorithm

$$\text{UIB} \coloneqq \frac{L_d D^2 \alpha^2}{2(\epsilon_g - 2D\epsilon_\xi)\alpha - 2(\epsilon_g + L_V \epsilon_z^2)} - 1, \tag{2.15}$$

where $L_d \coloneqq \lambda_{max}(H_d)$ and $L_V \coloneqq \lambda_{max}(H)$. ϵ_g is the largest permissible constraint satisfaction error, ϵ_z is the desired tolerance of the dual solution, and ϵ_ξ is the error in the dual gradient computation. D is the Upper Dual Bound (UDB), which is defined as

$D \coloneqq \|d\|_2$ where $d_i \coloneqq \max\{y_i^*, 1\}$ and y^* is the optimal dual vector. D can be estimated by solving one of several optimization problems given in [138] or [131], and α is a user-defined parameter to tune the convergence rate that can be calculated using the formula in [140].

2.6 Example systems

To numerically examine the theoretical results in Part II of this thesis, we will use the following five example linear systems.

Schur-stable system

The first two example systems have the dynamics given by the Schur-stable discrete-time system with four states and two inputs given in [88] with the state update equation

$$x_{k+1} = \begin{bmatrix} 0.7 & -0.1 & 0.0 & 0.0 \\ 0.2 & -0.5 & 0.1 & 0.0 \\ 0.0 & 0.1 & 0.1 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.5 \end{bmatrix} x_k + \begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 1.0 \\ 0.1 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} u_k.$$

The input of the systems is constrained to be $|u_k| \leq 0.5$, with a prediction horizon of $N = 10$. The first system has cost matrices

$$Q = \mathrm{diag}(10, 20, 30, 40), \quad R = \mathrm{diag}(10, 20), \tag{2.16}$$

and the second system has cost matrices

$$Q = \mathrm{diag}(100, 200, 300, 400), \quad R = \mathrm{diag}(0.001, 0.002). \tag{2.17}$$

Inverted pendulum

The next example system is a linearized inverted pendulum described by the continuous-time dynamics

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{3g}{2l} & -b & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{3}{2l} \\ 0 \\ 1 \end{bmatrix} u,$$

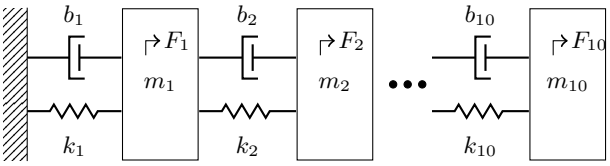


Figure 2.3: Configuration of the mass-spring-damper example system.

with $g = 9.8067$, $b = 1$, and $l = 0.21$. The system was discretized using a zero-order hold with a sampling time of 0.02 s, resulting in an unstable discrete-time system. The CLQR problem used the cost matrices $Q = \text{diag}(1000, 1, 100, 1), R = 10$, a prediction horizon of $N = 10$, and input constraints $|u| \leq 10$.

Distillation column

The next example system is a binary distillation column with 11 states and 3 inputs from the IFAC control benchmarks [44, Problem 90-01]. The system was discretized into a Schur-stable discrete-time system using a zero-order hold with a sampling time of 1.0 s (as done by the authors of the benchmark in [123]). The CLQR problem used a prediction horizon of $N = 100$ and the cost matrices

$$\begin{aligned} Q &= \text{diag}(10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110), \\ R &= \text{diag}(10, 20, 30), \end{aligned}$$

and the inputs were constrained to be

$$|u_1| \leq 2.5, \quad |u_2| \leq 2.5, \quad |u_3| \leq 0.30.$$

Mass-spring-damper

The final example system is a mass-spring-damper system from [97] where 10 masses are coupled together by a spring and a damper in parallel, with a force input on each mass. A schematic of the system can be seen in Figure 2.3, where for each mass i, m_i is the mass, b_i and k_i are the damping constant and spring constant, respectively, and F_i is the force input. The state vector x contains the displacement of each mass from its nominal position, and the input vector u contains the force inputs. The continuous-time system is discretized using a zero-order hold with a sampling time of $T_s = 0.1$ s. The continuous-time cost matrices are

$$Q_c = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix} \otimes I_{10}, \quad R_c = \text{diag}(100, 200, ..., \ 900, 1000),$$

which are then discretized as discussed in Section 2.1.1 in order to make the cost of the discrete-time problem equivalent to the cost of the continuous-time problem. This leads to dense Q and R matrices along with a cross-term matrix S in the discrete-time problem. The inputs and states of the system are constrained to be $|u_i| \leq 1$ and $|x_i| \leq 0.2$ respectively.

2.7 Conclusions

In this chapter, we provided an overview of the linear and nonlinear MPC problems that we will use in Parts II and III of this thesis, respectively. We also surveyed the existing implementations of linear MPC on FPGAs, and provided an overview of the key areas that designers focused on and the design choices they made. From this, we can see that the current trend in embedded implementations is towards the use of first-order methods such as the FGM, DGP or ADMM instead of second-order methods due to the smaller computational footprint of each algorithm iteration and reliance on only simple operations like matrix-vector multiplication.

A key advantage to using first-order methods like the FGM and DGP is that there exist upper iteration bounds, although not necessarily tight, for the solver that can guarantee the problem is solved to within a specified tolerance by that iteration. We will examine these iteration bounds more in Part II, where we will derive the pieces needed to create a horizon-independent bound in Chapter 4 and then examine how the bound changes when a preconditioner is applied in Chapter 6.

Chapter 3

An introduction to block Toeplitz matrices

Block Toeplitz matrices are block matrices that repeat the blocks from either the first block row or first block column down their respective diagonal, forming the matrix

$$\mathbf{T}_N = \begin{bmatrix} T_0 & T_{-1} & T_{-2} & T_{-3} & \cdots & T_{-N+2} & T_{-N+1} \\ T_1 & T_0 & T_{-1} & T_{-2} & \cdots & T_{-N+3} & T_{-N+2} \\ T_2 & T_1 & T_0 & T_{-1} & \cdots & T_{-N+4} & T_{-N+3} \\ T_3 & T_2 & T_1 & T_0 & \cdots & T_{-N+5} & T_{-N+4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{N-2} & T_{N-3} & T_{N-4} & T_{N-5} & \cdots & T_0 & T_{-1} \\ T_{N-1} & T_{N-2} & T_{N-3} & T_{N-4} & \cdots & T_1 & T_0 \end{bmatrix}, \tag{3.1}$$

where T_j is the block with n rows and m columns on the jth block diagonal of $\mathbf{T}_N$. Block Toeplitz matrices can be viewed as truncations of an infinite block matrix (denoted by bold face $\mathbf{T}$) by continuing the pattern down the diagonals and introducing new blocks as needed in the first block row/column. If the infinite block Toeplitz matrix stops after N blocks on the block rows/columns, the resulting finite-dimensional matrix is called a *truncated block Toeplitz matrix* with N blocks (denoted by $\mathbf{T}_N$).

The blocks T_j in (3.1) can be arbitrary matrices with no structure, or they can also possess a Toeplitz/block Toeplitz structure. If the blocks are also Toeplitz/block Toeplitz matrices, then the entire matrix $\mathbf{T}$ is known as a *block Toeplitz-Toeplitz block* (BTTB) matrix, and many of the properties/operations discussed in this chapter can be applied at the block level as well as to the full matrix.

Since a block Toeplitz matrix is fully defined by just the blocks in its first block row and block column, it can alternately be expressed as the block vector

$$\vec{\mathbf{T}}_N = \begin{bmatrix} T_{-N+1} & T_{-N+2} & \cdots & T_{-1} & T_0 & T_1 & \cdots & T_{N-2} & T_{N-1} \end{bmatrix}.$$

This vector representation can be used to efficiently store the matrix, making the storage used grow linearly with the number of block diagonals instead of quadratically.

Another type of matrix closely related to the block Toeplitz matrix is the block circulant matrix, where the blocks in the first column are repeated down their respective diagonal, and then looped to the upper diagonal when the lower diagonal ends. This gives the matrix

$$\mathbf{C}_N = \begin{bmatrix} C_0 & C_{-1} & C_{-2} & C_{-3} & \cdots & C_{2-N} & C_{1-N} \\ C_{1-N} & C_0 & C_{-1} & C_{-2} & \cdots & C_{3-N} & C_{2-N} \\ C_{2-N} & C_{1-N} & C_0 & C_{-1} & \cdots & C_{4-N} & C_{3-N} \\ C_{3-N} & C_{2-N} & C_{1-N} & C_0 & \cdots & C_{5-N} & C_{4-N} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{-2} & C_{-3} & C_{-4} & C_{-5} & \ddots & C_0 & C_{-1} \\ C_{-1} & C_{-2} & C_{-3} & C_{-4} & \cdots & C_{1-N} & C_0 \end{bmatrix}, \tag{3.2}$$

where C_{-j} is the $n \times m$ block in the first row of the jth column. Similar to the block Toeplitz matrix, a block circulant matrix can be expressed in block vector form using only its first row as

$$\vec{\mathbf{C}}_N = \begin{bmatrix} C_0 & C_{-1} & \cdots & C_{2-N} & C_{1-N} \end{bmatrix}.$$

Block Toeplitz and block circulant matrices appear in many applications in science and engineering, as we will show in Section 3.1. This chapter summarizes the relevant properties of block Toeplitz matrices we will use in the analysis of the LTI MPC problem in Part II.

3.1 Examples of block Toeplitz matrices

Before diving into the properties of block Toeplitz matrices, we will introduce five applications where block Toeplitz matrices can be found.

3.1.1 Queueing theory

The first application we show for a block Toeplitz matrix is in queueing theory and Markov chains. This example is based on [25], where there is a set of $k \geq 2$ servers, each with an infinite-length first-in, first-out queue, and each performing the same operation requiring the same amount of time (so all servers will start and end their operations at the same time points). The arrival of customers to the servers is modelled using a Poisson process with the rate λ, and when a new customer arrives they join the shortest queue.

The queue is modeled as a Markov chain with the relation $\pi = P\pi$, where π is the invariant probability vector, and P is the transition probability matrix with elements $p_{i,j}$ representing the probability that the overall number of customers in the queue changes from i to j. The P matrix then takes the form

$$P = \begin{bmatrix} \overbrace{a_0 & \dots & a_0}^{k} & 0 & 0 & 0 & \\ a_1 & \dots & a_1 & a_0 & 0 & 0 & \\ a_2 & \dots & a_2 & a_1 & a_0 & 0 & \\ a_3 & \dots & a_3 & a_2 & a_1 & a_0 & \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix},$$

where a_i is the probability that i customers arrive at a single time (defined as $a_i = \frac{\lambda^i}{i!}e^{-\lambda}$). By splitting the matrix into blocks of size $k-1$ by $k-1$, P can be viewed as the block matrix

$$P = \begin{bmatrix} B_1 & A_0 & 0 & 0 & 0 & \\ B_2 & A_1 & A_0 & 0 & 0 & \\ B_3 & A_2 & A_1 & A_0 & 0 & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{3.3}$$

The matrices with the structure of (3.3) are called M/G/1-type matrices, and have block Toeplitz structure everywhere except the first block column.

A common problem is to find the probability π_j of having j customers waiting to be served as the time goes to infinity. This can be done in various ways, but we briefly present the method from [25]. Finding the probability π can be done by solving the nonlinear equation

$$G = \sum_{i=0}^{\infty} G^i A_i, \tag{3.4}$$

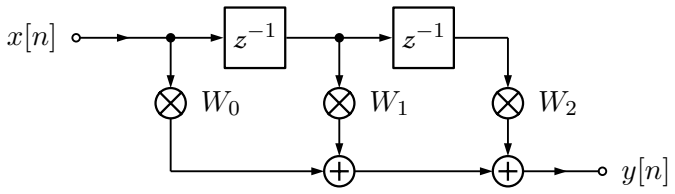


Figure 3.1: Block diagram of a 2nd order MIMO causal FIR filter. z^{-1} is a unit delay, which delays the signal for 1 time step.

for G and then using Ramaswami's formula as described in [25]. One possible way to solve (3.4) is to reformulate it into the system

$$\begin{bmatrix} G & G^2 & G^3 & \cdots \end{bmatrix} \begin{bmatrix} I - A_1 & -A_0 & 0 & 0 & \\ -A_2 & I - A_1 & A_0 & 0 & \\ -A_3 & -A_2 & I - A_1 & A_0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} = \begin{bmatrix} A_0 & 0 & 0 & \ldots \end{bmatrix}, \tag{3.5}$$

which then contains a block Toeplitz matrix. The system (3.5) can then be solved using methods that exploit the block Toeplitz structure of the coefficient matrix, such as cyclic reduction [24, 25].

3.1.2 Signal processing

A large application of block Toeplitz matrices is causal Multiple-Input Multiple-Output (MIMO) Finite Impulse Response (FIR) filters. An order k MIMO causal FIR filter takes a vector signal x and computes the filter response $y[n]$ at time n using the x vectors from times n through $n-k$. The structure of a 2nd order filter can be seen in Figure 3.1. The equation for a generalized order k filter is

$$y[n] = \sum_{j=0}^{k} W_j x[n-j], \tag{3.6}$$

where W_j are $n \times m$ matrices containing the coefficients for filter tap j.

The FIR filter equation (3.6) can be converted into a lower-triangular banded block Toeplitz matrix that represents the filter's response to the sequence of inputs

$$(x[0],\ x[1],\ x[2],\ \ldots)\,.$$

A filter of order k will have k bands below the diagonal. For example, the response of an order 2 filter can be computed using the matrix-vector product

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ \vdots \end{bmatrix} = \begin{bmatrix} W_0 & 0 & 0 & 0 & 0 & \cdots \\ W_1 & W_0 & 0 & 0 & 0 & \\ W_2 & W_1 & W_0 & 0 & 0 & \\ 0 & W_2 & W_1 & W_0 & 0 & \\ 0 & 0 & W_2 & W_1 & W_0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ \vdots \end{bmatrix}. \tag{3.7}$$

The block Toeplitz structure of the filter response matrix can then be exploited to perform analysis and design of the FIR filters, such as computing the mean square error [60].

3.1.3 Information theory

We next show an example from information theory — the vector Wide Sense Stationary (WSS) process. A vector random process $\vec{X}(t)$ is a vector WSS process with n elements if its mean and autocorrelation functions are time-invariant — i.e. the mean of $X(t)$ is the constant vector $\vec{\mu}$ and the autocorrelation function, representing the correlation of the signal to a delayed copy of itself, is only a function of the difference in time between the signals.

The auto-correlation matrix R_{XX} and the auto-covariance matrix K_{XX} for a sequence of samples from $\vec{X}(t)$ both are block Toeplitz structured with

$$R_{XX} \coloneqq \mathbb{E}(XX^*) = \begin{bmatrix} R_0 & R_{-1} & R_{-2} & R_{-3} & \cdots \\ R_1 & R_0 & R_{-1} & R_{-2} & \\ R_2 & R_1 & R_0 & R_{-1} & \\ R_3 & R_2 & R_1 & R_0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix},$$

$$K_{XX} \coloneqq \mathbb{E}((X-\vec{\mu})(X-\vec{\mu})^*) = \begin{bmatrix} K_0 & K_{-1} & K_{-2} & K_{-3} & \cdots \\ K_1 & K_0 & K_{-1} & K_{-2} & \\ K_2 & K_1 & K_0 & K_{-1} & \\ K_3 & K_2 & K_1 & K_0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix},$$

where R_j and K_j are the $n \times n$ matrices given by

$$R_j = \mathbb{E}(X_t X_{t+j}^*), \qquad K_j = \mathbb{E}(X_t X_{t+j}^*) - \vec{\mu}\vec{\mu}^*.$$

The WSS process can be combined with the model of the FIR filter from Section 3.1.2 to analyze properties of Gaussian communications channels such as their capacity [41, 62] or differential entropy and Minimum Mean Square Error [58], and can be used to design codes for communication over Gaussian channels [63, 182].

3.1.4 Control theory

The uses of block Toeplitz matrices in control theory are closely related to the MIMO filter from Section 3.1.2. A MIMO Linear Time-Invariant (LTI) system can be written as an infinite impulse response (IIR) filter (where the sum in (3.6) goes to infinity instead of k), which can be represented by a non-banded block Toeplitz matrix similar to (3.7). In the control case, we begin with a discrete-time LTI dynamical system

$$x^+ = Ax + Bu, \tag{3.8}$$

where x and u are the states and control inputs for the system, respectively. The state for system (3.8) at time n can be expressed as the sum of all the previous inputs experienced by the system, forming the summation

$$x[n] = \sum_{j=0}^{n} A^j B u[j].$$

This summation can then be transformed into the following block Toeplitz matrix representing the state evolution of the system over time

$$\begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ \vdots \end{bmatrix} = \begin{bmatrix} B & 0 & 0 & 0 & 0 & \cdots \\ AB & B & 0 & 0 & 0 & \\ A^2B & AB & B & 0 & 0 & \\ A^3B & A^2B & AB & B & 0 & \\ A^4B & A^3B & A^2B & AB & B & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \\ u[4] \\ \vdots \end{bmatrix}. \tag{3.9}$$

The matrix in (3.9) has found many uses in control analysis and design for linear systems, since it can be used as a *prediction matrix* to predict the system states given a specific input trajectory. For example, it can be used to perform stability analysis [169], identify transfer function zeros [158], and design optimal controllers [73]. Block Toeplitz matrices can also be found in the models used to learn the dynamics of a system from just the response information, such as the Autoregressive Moving-Average (ARMA) [58] and ARMA-Markov models [174].

3.1.5 2D boundary value problems

Another common application area for block Toeplitz matrices to appear in is 2D boundary value problems. As an example, we examine the 2D Poisson equation

$$-\frac{\mathrm{d}^2}{\mathrm{d}x^2}u(x,y) - \frac{\mathrm{d}^2}{\mathrm{d}y^2}u(x,y) = f(x,y), \quad (x,y) \in \Omega = [0,1] \times [0,1], \tag{3.10}$$

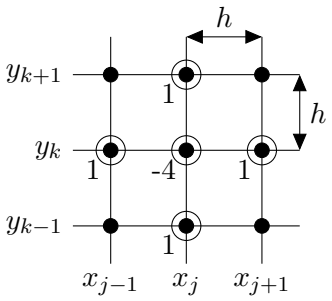


Figure 3.2: 5-point stencil for point (x_j, y_k) overlaid on the mesh for a 2D boundary value problem.

with Dirichlet boundary conditions from [35]. To solve (3.10), we overlay an $n \times n$ rectangular mesh with equispaced points $h = \frac{1}{n}$ apart onto Ω. Using this mesh, we compute the second centered difference at (x_j, y_k) in each direction to give the derivative estimates

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}u(x_j, y_k) \approx \frac{u_{j+1,k} - 2u_{j,k} + u_{j-1,k}}{h^2}, \quad \frac{\mathrm{d}^2}{\mathrm{d}y^2}u(x_j, y_k) \approx \frac{u_{j,k+1} - 2u_{j,k} + u_{j,k-1}}{h^2}, \tag{3.11}$$

with $u_{j,k} \coloneqq u(x_j, y_k)$. Plugging the derivative estimates (3.11) into (3.10) and simplifying then gives

$$-\frac{u_{j+1,k} + u_{j,k+1} - 4u_{j,k} + u_{j-1,k} + u_{j,k-1}}{h^2} = f(x_j, y_k), \quad \text{for } j,k = 1,\ldots,n-1. \tag{3.12}$$

This formulation is known as the 5-point stencil, and can be seen overlaid on top of the mesh in Figure 3.2.

To solve the discretized boundary value problem, we form a linear system using the 5-point stencil (3.12) and the boundary conditions

$$u_{0,k} = u_{n,k} = u_{j,0} = u_{n,0} = 0 \quad \text{for } j,k = 0,\ldots,n.$$

By defining the vector of unknowns as

$$\bar{u} \coloneqq \Big[u_{1,1},\ldots,u_{1,n},u_{2,1},\ldots,u_{n-2,n-1},u_{n-1,1},\ldots,u_{n-1,n-1}\Big]',$$

the linear system for the 5-point stencil can be written as $A\bar{u} = h^2\bar{f}$ with

$$A \coloneqq \begin{bmatrix} B & -I & 0 & 0 & \dots \\ -I & B & -I & 0 & \\ 0 & -I & B & -I & \\ 0 & 0 & -I & B & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}, \tag{3.13}$$

$\bar{f}$ the vector of right-hand sides from (3.12), I the identity matrix, and

$$B \coloneqq \begin{bmatrix} 4 & -1 & 0 & 0 & \dots \\ -1 & 4 & -1 & 0 & \\ 0 & -1 & 4 & -1 & \\ 0 & 0 & -1 & 4 & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

an $(n-1) \times (n-1)$ matrix. Note that A is a symmetric block tridiagonal and block Toeplitz matrix, composed of the blocks B and I, which are Toeplitz as well — so A is a BTTB matrix.

3.2 Block Toeplitz matrices and their matrix symbols

Now that we have shown how block Toeplitz matrices appear in various applications, we move on to discussing their properties.

We will be working in the sequence space l^2, which is the Hilbert space of square summable sequences — i.e. for a sequence of values $\{x_n\}_{n\in\mathbb{N}}$, $\sum_n |x_n|^2 < \infty$ holds. We define the set of complex numbers on the unit circle as

$$\mathbb{T} \coloneqq \{z \in \mathbb{C} : |z| = 1\}.$$

The functions we use are in the Banach space $\mathcal{L}^\infty_{2\pi}$ — which is the space of matrix-valued essentially bounded functions that are 2π-periodic (i.e. the domain of the functions is $\mathbb{T}$), and that have a finite Frobenius norm almost everywhere on that domain, i.e.

$$\mathcal{T}(\cdot) \in \mathcal{L}^\infty_{2\pi} \Leftrightarrow \inf\{y \in \mathbb{R} : \|\mathcal{T}(z)\|_F < y \text{ for almost every } z \in \mathbb{T}\} < +\infty.$$

We also define a restricted space of functions called $\tilde{\mathcal{C}}_{2\pi}$, which is the space of continuous functions inside $\mathcal{L}^\infty_{2\pi}$. Note that while we restrict our discussion to functions in $\mathcal{L}^\infty_{2\pi}$, many of the properties presented in this chapter also hold for 2π-periodic matrix symbols in $\mathcal{L}^2$ or $\mathcal{L}^1$ with some technical changes.

The representation of the 2π-continuous functions we use in this thesis will take a complex number on the unit circle as their argument (their domain is $\mathbb{T}$), represented by z. An alternate representation used in many works is to instead have the function take a real parameter θ and operate over the interval $\theta \in (-\pi, \pi)$ (or alternately $\theta \in (0, 2\pi)$), and have the term $e^{i\theta}$ instead of z.

One of the key ideas for block Toeplitz matrices is that matrices like (3.1) are finite dimensional truncations of the infinite dimensional matrix $\mathbf{T}$. This allows us to think of block Toeplitz matrices as if they were a bounded linear operator on the sequence space l^2.

3.2.1 The matrix symbol

To define the matrix symbol of a block Toeplitz matrix $\mathbf{T}$, we first form the sequence $\{T_n\}_{n\in\mathbb{Z}}$ using the blocks of $\mathbf{T}$. The matrix symbol $\mathcal{T}(\cdot)$ (which we denote using calligraphic script) for the block Toeplitz matrix $\mathbf{T}$ is then the function $\mathcal{T} \in \mathcal{L}^\infty_{2\pi}$ that has the sequence $\{T_n\}_{n\in\mathbb{Z}}$ as its Fourier coefficients. This means that the blocks T_n can be computed from the matrix symbol $\mathcal{T}(\cdot)$ using

$$T_n = \frac{1}{2\pi}\int_0^{2\pi} \mathcal{T}(e^{i\theta})e^{-in\theta}\,\mathrm{d}\theta, \qquad \forall n \in \mathbb{Z}. \tag{3.14}$$

The block Toeplitz matrix $\mathbf{T}$ then has a bounded operator on l^2 if and only if $\mathcal{T}$ and $\{T_n\}_{n\in\mathbb{Z}}$ are related by the Fourier transform (3.14).

The matrix symbol for the block Toeplitz matrix is also known as the *generating symbol* of the matrix, since relation (3.14) can be used to construct the block Toeplitz matrix from just the matrix symbol. To represent this relation, we introduce the *Toeplitz matrix operator* $\mathtt{T}(\mathcal{T}) \coloneqq \mathbf{T}$ (which we denote using typewriter script) to provide a mapping from the matrix symbol to its associated infinite-dimensional block Toeplitz matrix.

The matrix symbol can have a special meaning in some applications, and even represent another physical property. For example, the matrix symbol of the autocorrelation matrix R_{XX} in Section 3.1.3 is the power spectral density function for the WSS process $X(\cdot)$, so it will describe how the power in the random process is distributed over the frequency spectrum.

3.2.2 Forming the matrix symbol

Now that we have defined what the matrix symbol is, we show two ways to form the matrix symbol given an existing block Toeplitz matrix. Other types of functions in $\mathcal{L}^\infty_{2\pi}$ can be matrix symbols as well, but the two cases in this section usually are the easiest to identify the symbol for, since it is in general hard to identify functions from their Fourier coefficients.

Banded block Toeplitz matrices

In the case when the matrix $\mathbf{T}$ is banded (i.e. there is an N such that $T_j = 0_{n\times m}$ for all $|j| > N$), the matrix symbol can be formed as the trigonometric polynomial

$$\mathcal{T}(z) = \sum_{j=-N}^{N} T_j z^j, \qquad \forall z \in \mathbb{T}, \tag{3.15}$$

where the blocks of $\mathbf{T}$ are used as the coefficients.

Wiener class symbols

A function is in the Wiener class if and only if it is in $\tilde{\mathcal{C}}_{2\pi}$ and its Fourier coefficients are absolutely summable. Since the blocks of $\mathbf{T}$ are the Fourier coefficients of the matrix symbol, for the symbol to be in the Wiener class, the sequence of blocks $\{T_n\}_{n\in\mathbb{Z}}$ must be absolutely summable. This condition can be expressed as

$$\sum_{i=-\infty}^{\infty} |[T_i]_{r,s}| < \infty, \qquad 0 \le r \le n, 0 \le s \le m,$$

which means the summation over every element of the blocks T_n is finite for each element.

If the sequence of blocks $\{T_n\}_{n\in\mathbb{Z}}$ is absolutely summable, then the matrix symbol for $\mathbf{T}$ can be formed from the trigonometric polynomial using the blocks as the coefficients, i.e.

$$\mathcal{T}(z) = \sum_{j=-\infty}^{\infty} T_j z^j, \qquad \forall z \in \mathbb{T}.$$

3.2.3 Circulant matrices

Like block Toeplitz matrices, we can also associate a continuous and 2π-periodic function with a block circulant matrix as its matrix symbol. The relation between the truncated circulant matrix $\mathbf{C}_N$ with N blocks that are of size $n \times m$ and its matrix symbol $\mathcal{C} : \mathbb{T} \to \mathbb{C}^{n\times m}$ is

$$\mathbf{C}_N \coloneqq (V_N \otimes I_n)\tilde{C}(V_N \otimes I_m)^*, \tag{3.16}$$

with

$$\tilde{C} \coloneqq \operatorname{diag}\left(\mathcal{C}(0), \mathcal{C}\left(e^{-\frac{2\pi}{N}i}\right), \ldots, \mathcal{C}\left(e^{-\frac{2\pi(N-1)}{N}i}\right)\right)$$

a block diagonal matrix containing N equispaced samples of $\mathcal{C}(\cdot)$ around the unit circle and V_N the unitary Fourier matrix with elements

$$[V_N]_{j,k} \coloneqq \frac{1}{\sqrt{N}} e^{-\frac{2\pi(j-1)(k-1)}{N} i}, \qquad 1 \leq j, k \leq N.$$

Essentially, (3.16) uses the Fourier matrix to diagonalize the block circulant matrix. The resulting diagonal matrix is then composed of the matrices created by sampling the matrix symbol at N equally-spaced points on the unit circle.

3.3 Structure preserving operations

An important consideration when using structured matrices is how the structure changes as operations are performed on the matrix. For a block Toeplitz matrix there is an additional question: how does performing an operation on or between two block Toeplitz matrices affect the matrix symbol?

For simple operations, the block Toeplitz structure is preserved, and the matrix symbol is modified by performing the same operation on it. This can be seen in Lemma 3.3.1.

Lemma 3.3.1. *Let* $\mathbf{F}$ *and* $\mathbf{G}$ *be block Toeplitz matrices with the matrix symbols* $\mathcal{F}, \mathcal{G} \in \tilde{\mathcal{C}}_{2\pi} : \mathbb{T} \to \mathbb{C}^{m \times n}$*, respectively. Then*

1. $\mathtt{T}(\mathcal{F}^*) = (\mathtt{T}(\mathcal{F}))^*$ *(i.e.* $\mathbf{F}^*$ *is block Toeplitz with the matrix symbol* $\mathcal{F}^*$*),*
2. $\mathtt{T}(\alpha\mathcal{F} + \beta\mathcal{G}) = \alpha\mathtt{T}(\mathcal{F}) + \beta\mathtt{T}(\mathcal{G})$.

Proof. See [59, Lemma 4.2]. □

Another common operation that is performed between two or more matrices is matrix-matrix multiplication. Unfortunately, the multiplication of two block Toeplitz matrices is not in general block Toeplitz, as can be seen in Example 3.3.1.

Example 3.3.1. We can construct an example showing when the multiplication of two block Toeplitz matrices does not result in a block Toeplitz matrix by using a block size of $m = n = 1$ as follows

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 4 \\ 5 & 5 \end{bmatrix}.$$

□

There are some special cases when the product of block Toeplitz matrices will be block Toeplitz as well. One such case is that of lower and upper triangular block Toeplitz matrices, which each form a matrix subalgebra that is closed over matrix multiplication. This means that multiplying two lower triangular block Toeplitz matrices together will result in a lower triangular block Toeplitz matrix (and similarly for upper triangular block Toeplitz matrices).

Example 3.3.2. We start with two lower triangular block Toeplitz matrices with compatible block sizes

$$F \coloneqq \begin{bmatrix} F_0 & 0 & 0 & 0 & \\ F_1 & F_0 & 0 & 0 & \\ F_2 & F_1 & F_0 & 0 & \\ F_3 & F_2 & F_1 & F_0 & \\ \vdots & & & & \ddots \end{bmatrix}, \qquad G \coloneqq \begin{bmatrix} G_0 & 0 & 0 & 0 & \\ G_1 & G_0 & 0 & 0 & \\ G_2 & G_1 & G_0 & 0 & \\ G_3 & G_2 & G_1 & G_0 & \\ \vdots & & & & \ddots \end{bmatrix}.$$

We then form the product

$$\begin{aligned} FG &= \begin{bmatrix} F_0 & 0 & 0 & 0 & \\ F_1 & F_0 & 0 & 0 & \\ F_2 & F_1 & F_0 & 0 & \\ F_3 & F_2 & F_1 & F_0 & \\ \vdots & & & & \ddots \end{bmatrix} \begin{bmatrix} G_0 & 0 & 0 & 0 & \\ G_1 & G_0 & 0 & 0 & \\ G_2 & G_1 & G_0 & 0 & \\ G_3 & G_2 & G_1 & G_0 & \\ \vdots & & & & \ddots \end{bmatrix} \\ &= \begin{bmatrix} F_0G_0 & 0 & 0 & 0 & \\ F_1G_0 + F_0G_1 & F_0G_0 & 0 & 0 & \\ F_2G_0 + F_1G_1 + F_0G_2 & F_1G_0 + F_0G_1 & F_0G_0 & 0 & \\ F_3G_0 + F_2G_1 + F_1G_2 + F_0G_3 & F_2G_0 + F_1G_1 + F_0G_2 & F_1G_0 + F_0G_1 & F_0G_0 & \\ \vdots & & & & \ddots \end{bmatrix}. \end{aligned}$$

We can see from this that the structure of zeros in each row make the entries constant down each diagonal, and hence a block Toeplitz matrix. In fact, the entries in diagonal k of the product FG can be expressed as a finite summation of $k+1$ terms of the form

$$[FG]_k = \sum_{j=0}^{k} F_{k-j}G_j. \tag{3.17}$$

□

Example 3.3.2 shows that the truncated lower triangular block Toeplitz matrix keeps the block Toeplitz structure during multiplication, but that does not answer the question of what happens to the matrix symbol during that operation. To answer that we need Lemma 3.3.2.

Lemma 3.3.2. *Let* $\mathbf{F}$ *and* $\mathbf{G}$ *be lower triangular block Toeplitz matrices with matrix symbols* $\mathcal{F} \in \tilde{\mathcal{C}}_{2\pi} : \mathbb{T} \to \mathbb{C}^{k\times m}$ *and* $\mathcal{G} \in \tilde{\mathcal{C}}_{2\pi} : \mathbb{T} \to \mathbb{C}^{m\times n}$ *in the Wiener class, respectively, that are given by*

$$\mathcal{F} \coloneqq \sum_{l=0}^{\infty} F_l z^l, \qquad \mathcal{G} \coloneqq \sum_{h=0}^{\infty} G_h z^h, \qquad \forall z \in \mathbb{T}.$$

Then

$$\mathtt{T}(\mathcal{F}\mathcal{G}) = \mathtt{T}(\mathcal{F})\mathtt{T}(\mathcal{G}).$$

Proof. We begin by examining the Fourier transform of the product $\mathcal{F}\mathcal{G}$,

$$\frac{1}{2\pi}\int_0^{2\pi} \mathcal{F}(e^{\theta i})\mathcal{G}(e^{\theta i})e^{-k\theta i}\,\mathrm{d}\theta.$$

Plugging in the infinite series definition of $\mathcal{F}$ and $\mathcal{G}$, we transform the integral into

$$\frac{1}{2\pi}\int_0^{2\pi} \left(\sum_{l=0}^{\infty} F_l e^{l\theta i}\right)\left(\sum_{h=0}^{\infty} G_h e^{h\theta i}\right) e^{-k\theta i}\,\mathrm{d}\theta$$

We then extract the summations from the integral and combine the complex exponentials, giving

$$\sum_{l=0}^{\infty} F_l \left(\sum_{h=0}^{\infty} G_h \left(\frac{1}{2\pi}\int_0^{2\pi} e^{(l+h)\theta i} e^{-k\theta i}\,\mathrm{d}\theta\right)\right). \tag{3.18}$$

The integral term now can be viewed as generating the Fourier coefficients for the function $\mathcal{H}(z) = z^{l+h}$, which has the Fourier coefficients defined by the indicator function δ_k for when $k = l + h$,

$$H_k(\alpha) = \delta_k(\alpha) = \begin{cases} 1 & \text{If } k = \alpha, \\ 0 & \text{Otherwise,} \end{cases}$$

which then makes (3.18) become

$$\sum_{l=0}^{\infty} F_l \left(\sum_{h=0}^{\infty} G_h \delta_k(l+h)\right). \tag{3.19}$$

We can now use the nature of the indicator function to transform the summation bounds. First, we examine the outer summation over F_l and note that it is impossible for l to be greater than k since the internal summation always uses positive h and $l + h$ must sum to k — making the upper bound for the outer summation become k. Next we look at the inner summation over G_h and see that it will always generate a single term for every value of the

outer summation (since only one value of h will make $k = l + h$ true under these conditions), and that term will be at $k - l$, transforming (3.19) into

$$\sum_{l=0}^{k} F_l G_{k-l}. \tag{3.20}$$

Finally, we note that the summation (3.20) generates the same terms as summation (3.17) from Example 3.3.2, which describes the value of the blocks in the lower triangular block Toeplitz matrix-matrix multiplication. Therefore, we have shown that the matrix symbol formed by the product $\mathcal{FG}$ will generate the lower triangular block Toeplitz matrix given by the product $\mathbf{FG}$. □

Example 3.3.2 and Lemma 3.3.2 showed that the product of the multiplication of two lower triangular block Toeplitz matrices with blocks that are absolutely summable and matrix symbols in the Wiener class will be block Toeplitz with its matrix symbol simply the product of the two original symbols. Similar results to Example 3.3.2 and Lemma 3.3.2 can be derived for upper triangular block Toeplitz matrices as well.

We now look more generally at what happens to the matrix symbol during matrix-matrix multiplication of two block Toeplitz matrices. It turns out that the infinite-dimensional matrix generated by the product of two matrix symbols of compatible dimensions is known to be composed of two distinct terms:

1. the product of the infinite-dimensional block Toeplitz matrices generated by each individual symbol, and
2. the product of the infinite-dimensional block Hankel matrices generated by each symbol,

which we describe formally in Lemma 3.3.3.

Lemma 3.3.3. *Let $\mathcal{F} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{k\times m}$ and $\mathcal{G} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{m\times n}$ be the matrix symbols for the infinite block Toeplitz matrices $\mathbf{F}$ and $\mathbf{G}$, respectively. Then*

$$\mathtt{T}(\mathcal{FG}) = \mathtt{T}(\mathcal{F})\mathtt{T}(\mathcal{G}) + \mathtt{H}(\mathcal{F})\tilde{\mathtt{H}}(\mathcal{G}),$$

where $\mathtt{H}(\,\cdot\,)$ and $\tilde{\mathtt{H}}(\,\cdot\,)$ are operators that generate block Hankel matrices as follows

$$\mathtt{H}(\mathcal{F}) \coloneqq \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & \cdots \\ F_2 & F_3 & F_4 & \cdots & \cdots \\ F_3 & F_4 & \cdots & \cdots & \cdots \\ F_4 & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \qquad \tilde{\mathtt{H}}(\mathcal{G}) \coloneqq \begin{bmatrix} G_{-1} & G_{-2} & G_{-3} & G_{-4} & \cdots \\ G_{-2} & G_{-3} & G_{-4} & \cdots & \cdots \\ G_{-3} & G_{-4} & \cdots & \cdots & \cdots \\ G_{-4} & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Proof. Equation (6.2) in Section 6.1 of [29]. □

We can now apply Lemma 3.3.3 to different cases and make some observations about the multiplication of infinite-dimensional block Toeplitz matrices. First, note that it presents an alternative proof for Lemma 3.3.2, since if both $\mathcal{F}$ and $\mathcal{G}$ are lower triangular block Toeplitz matrices, then the Hankel matrix $\tilde{\mathtt{H}}(\mathcal{G})$ will be 0, giving $\mathtt{T}(\mathcal{FG}) = \mathtt{T}(\mathcal{F})\mathtt{T}(\mathcal{G})$. Second, we can derive the following result for multiplication involving both an upper and lower triangular matrix.

Corollary 3.3.1. *Let $\mathcal{F} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{k\times m}$, $\mathcal{G} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{m\times n}$ and $\mathcal{H} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{m\times m}$ be the matrix symbols for the infinite block Toeplitz matrices $\mathbf{F}$, $\mathbf{G}$, and $\mathbf{H}$, respectively. If $\mathbf{F}$ is an upper triangular block Toeplitz matrix and $\mathbf{G}$ is a lower triangular block Toeplitz matrix, then*

$$\mathtt{T}(\mathcal{FHG}) = \mathtt{T}(\mathcal{F})\mathtt{T}(\mathcal{H})\mathtt{T}(\mathcal{G}).$$

The final operation we examine is the inverse of a block Toeplitz matrix. In this case, it is known that for a Hermitian matrix the inverse will also be block Toeplitz, with the new matrix symbol the inverse of the original matrix symbol (as described in Lemma 3.3.4).

Lemma 3.3.4. *Let $\mathbf{F}$ be a Hermitian block Toeplitz matrix with the Hermitian matrix symbol $\mathcal{F} \in \tilde{\mathcal{C}}_{2\pi} : \mathbb{T} \to \mathbb{C}^{m\times m}$. If $0 \notin [\inf \mathcal{F}, \sup \mathcal{F}]$, then $\mathbf{F}^{-1}$ is block Toeplitz with the matrix symbol $(\mathcal{F}(\,\cdot\,))^{-1}$.*

Proof. See [59, Theorem 6.4] □

3.4 Spectral properties

One of the most used concepts in Toeplitz and block Toeplitz matrix theory is the relationship of the eigenvalue/singular value spectrum between the truncated block Toeplitz matrix and the matrix symbol. In this section, we will review some of the ideas and results in that relationship.

To do this, we first need to define notation that describes the minimum and maximum of the spectrum for both the matrix symbol $\mathcal{T}$ and the truncated/infinite matrix $\mathbf{T}_N/\mathbf{T}$. We let $\lambda_1 \leq \cdots \leq \lambda_k$ be the eigenvalues of a matrix in sorted order, with the set of all eigenvalues denoted by λ, and $0 \leq \sigma_k \leq \cdots \leq \sigma_1$ be the singular values of a matrix in sorted order, with the set of all singular values denoted by σ. We are then able to define the extremal values of the spectrum of the matrix symbol as follows.

Definition 3.4.1. *Let $\mathcal{T}(\cdot) \in \mathcal{L}_{2\pi}^{\infty}$ be a function that maps $\mathbb{T} \to \mathbb{C}^{m\times n}$. With $k = \min\{m, n\}$, the extreme singular values of $\mathcal{T}(\cdot)$ are*

$$\sigma_{min}(\mathcal{T}) \coloneqq \operatorname{ess\,sup}\{y \in \mathbb{R} \colon \sigma_k(\mathcal{T}(z)) > y \text{ for almost every } z \in \mathbb{T}\},$$
$$\sigma_{max}(\mathcal{T}) \coloneqq \operatorname{ess\,inf}\{y \in \mathbb{R} \colon \sigma_1(\mathcal{T}(z)) < y \text{ for almost every } z \in \mathbb{T}\}.$$

If $\mathcal{T}(\cdot)$ is a Hermitian matrix with $k = m = n$, then the extreme eigenvalues of $\mathcal{T}(\cdot)$ are

$$\lambda_{min}(\mathcal{T}) \coloneqq \operatorname{ess\,sup}\{y \in \mathbb{R} \colon \lambda_1(\mathcal{T}(z)) > y \text{ for almost every } z \in \mathbb{T}\},$$
$$\lambda_{max}(\mathcal{T}) \coloneqq \operatorname{ess\,inf}\{y \in \mathbb{R} \colon \lambda_k(\mathcal{T}(z)) < y \text{ for almost every } z \in \mathbb{T}\},$$

and the condition number of $\mathcal{T}(\cdot)$ is

$$\kappa(\mathcal{T}) \coloneqq \frac{\lambda_{max}(\mathcal{T})}{\lambda_{min}(\mathcal{T})}.$$

Definition 3.4.1 roughly means that the extremes of the spectrum for the matrix symbol are just the most extreme values found when computing the singular value/eigenvalue decomposition of the matrix symbol at all points on the unit circle excluding sets of measure zero.

For the infinite matrix $\mathbf{T}$, we will define its extremal eigenvalues in terms of its truncation as the size of the matrix grows towards infinity.

Definition 3.4.2. *Let $\mathbf{T}$ be Hermitian and let $\mathbf{T}_N \in \mathbb{R}^{k\times k}$ be its truncation after N blocks with k eigenvalues. If the limits exist, we then define the extremal eigenvalues of $\mathbf{T}$ as*

$$\lambda_{min}(\mathbf{T}) \coloneqq \lim_{N\to\infty} \lambda_1(\mathbf{T}_N), \qquad \lambda_{max}(\mathbf{T}) \coloneqq \lim_{N\to\infty} \lambda_k(\mathbf{T}_N).$$

A direct consequence of Definition 3.4.2 is that the truncated matrices will have their spectrum bounded by the spectrum of both the larger truncations and of the infinite matrix, as given in Lemma 3.4.1.

Lemma 3.4.1. *Let $\mathbf{T}$ be a Hermitian block Toeplitz matrix and let $\{\mathbf{T}_N\}_{N\in\mathbb{N}^+}$ be the set of its truncated matrices. Then*

$$\lambda_{min}(\mathbf{T}) \leq \lambda_{min}(\mathbf{T}_{N+1}) \leq \lambda_{min}(\mathbf{T}_N), \quad \lambda_{max}(\mathbf{T}_N) \leq \lambda_{max}(\mathbf{T}_{N+1}) \leq \lambda_{max}(\mathbf{T}) \qquad \forall N \in \mathbb{N}^+.$$

Proof. This follows from applying the Cauchy interlacing theorem for eigenvalues to the limit sequence in Definition 3.4.2. □

An additional consequence of Lemma 3.4.1 is that every truncation of a positive definite block Toeplitz matrix will also be positive definite.

3.4.1 Spectrum of a block circulant matrix

We now examine the relationship between the singular value and eigenvalue spectra of a block circulant matrix and its matrix symbol. In the matrix symbol (3.16), we can see the start of a Singular Value Decomposition (SVD). As a refresher, the SVD for the $n \times m$ matrix A with rank r is $A = U_l \Sigma U_r$ where $U_l \in \mathbb{C}^{n\times n}$ and $U_r \in \mathbb{C}^{m\times m}$ are unitary matrices (i.e. $U_r U_r^* = U_r^* U_r = I$) containing the left and right singular vectors, respectively, and $\Sigma \in \mathbb{R}^{n\times m}$ is a rectangular matrix containing the r singular values of A on its main diagonal and zeros in the other entries.

For a scalar circulant matrix ($n = m = 1$), the relation (3.16) is the SVD of the truncated circulant matrix, with the singular vectors given by the Fourier matrix and the singular values given by $\tilde{C}$. This means that the singular values for the scalar case can be computed by sampling the matrix symbol at N equispaced points on the unit circle.

In the rectangular block case, the matrix symbol gives matrices in $\mathbb{C}^{n\times m}$, so (3.16) no longer forms the SVD directly. Instead, we can replace $\tilde{C}$ with its SVD to create an SVD for the original block circulant matrix using the singular values of the matrix symbol. This means that we can exactly compute the spectrum of a truncated block circulant matrix $\mathbf{C}_N$ using the matrix symbol by computing the singular values of the matrix symbol at N equispaced locations on the unit circle. Put formally, for the matrix symbol $\mathcal{C}$ that gives matrices with rank r, the singular values for the truncated block circulant matrix $\mathbf{C}_N$ are given in Theorem 3.4.1.

Theorem 3.4.1. *Let $\mathbf{C}_N$ be a truncated block circulant matrix with N blocks of size $n \times m$. If $\mathcal{C} : \mathbb{T} \to \mathbb{C}^{n\times m}$ is the matrix symbol for $\mathbf{C}_N$ and r is the maximal rank of the matrices in the range of $\mathcal{C}(\cdot)$, then the singular values of $\mathbf{C}_N$ are*

$$\sigma(\mathbf{C}_N) = \bigcup_{j=0}^{N-1} \bigcup_{k=0}^{r} \sigma_k(\mathcal{C}(\omega_j)), \qquad \omega_j \coloneqq e^{-\frac{2\pi j}{N} i}. \tag{3.21}$$

Proof. Recall from Section 3.2.3 that a block circulant matrix can be partially-diagonalized using the Fourier matrix and samples from its matrix symbol into the form given in (3.16). We then introduce the Singular Value Decomposition (SVD) of the matrix symbol

$$\mathcal{C}(\omega) \coloneqq U_l(\omega)\Sigma(\omega)U_r^*(\omega), \tag{3.22}$$

where $U_l(\omega) \in \mathbb{C}^{n\times n}$ and $U_r(\omega) \in \mathbb{C}^{m\times m}$ are unitary matrices containing the left and right singular vectors, respectively, and $\Sigma(\omega) \in \mathbb{R}^{n\times m}$ is the rectangular matrix containing the r singular values of $\mathcal{C}(\omega)$. Substituting the SVD (3.22) into the matrix symbol (3.16) in place of $\tilde{C}$, we get

$$\mathbf{C}_N = (V_N \otimes I_n)\,\mathrm{diag}_{1\le j\le N}(U_l(\omega_j))\,\mathrm{diag}_{1\le j\le N}(\Sigma(\omega_j))\,\mathrm{diag}_{1\le j\le N}(U_r^*(\omega_j))(V_N \otimes I_m)^*, \quad (3.23)$$

where ω_j are the N equally-spaced samples on the unit circle

$$\omega_j \coloneqq e^{-\frac{2\pi j}{N}i}, \qquad 0 \le j \le N-1.$$

Note that the products

$$(V_N \otimes I_n)\,\mathrm{diag}_{1\le j\le N}(U_l(\omega_j)) \quad (3.24)$$

and

$$\mathrm{diag}_{1\le j\le N}(U_r^*(\omega_j))(V_N \otimes I_m)^* \quad (3.25)$$

produce unitary matrices, since the product of two unitary matrices is unitary. This means that we can interpret (3.23) as an SVD with the left singular vectors given by (3.24), the right singular vectors given by (3.25), and the singular values given by

$$\mathrm{diag}_{1\le j\le N}(\Sigma(\omega_j)).$$

We can then see that the singular values of the circulant matrix $\mathbf{C}_N$ are the singular values of the N samples of the matrix symbol that are used to form it, giving (3.21). □

3.4.2 Spectrum of a block Toeplitz matrix

Unlike a block circulant matrix, the eigenvalues and singular values of a block Toeplitz matrix cannot in general be known exactly outside of some special cases, such as block-tridiagonal matrices. Instead, most of the information available about the spectrum is in the form of asymptotic bounds on the eigenvalues and singular values.

Singular values

We begin by examining the singular values of a rectangular block Toeplitz matrix. After seeing that the eigenvalues of a circulant matrix can be related to the eigenvalues of its matrix symbol, it is natural to ask how we can relate the spectrum of the block Toeplitz matrices to the spectrum of their matrix symbols.

In the case of singular values, we cannot use the matrix symbol to fully bound the spectrum of the matrix in general because we cannot use the smallest singular value of the matrix symbol as the lower bound, as we show in Example 3.4.1.

Example 3.4.1. We can construct an example where $\sigma_{min}(\mathbf{T}_N) < \sigma_{min}(\mathcal{T})$ as follows. We let $n = m = 2$ and define the function

$$\mathcal{T}(z) = \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix}, \qquad z \in \mathbb{T},$$

as the matrix symbol for the block Toeplitz matrix of size $2N \times 2N$ with N blocks

$$\mathbf{T}_N = \begin{bmatrix} 0_{2\times2} & I_{2\times2} & 0_{2\times2} & & \\ & 0_{2\times2} & I_{2\times2} & \ddots & \\ & & 0_{2\times2} & \ddots & 0_{2\times2} \\ & & & \ddots & I_{2\times2} \\ & & & & 0_{2\times2} \end{bmatrix}.$$

When we analyze the singular values of $\mathcal{T}$, we can see that its singular values are a constant value of $\sqrt{2}$ for all $z \in \mathbb{T}$ (so $\sigma_{max}(\mathcal{T}) = \sigma_{min}(\mathcal{T}) = \sqrt{2}$). Inspection of $\mathbf{T}_N$ shows that $\mathbf{T}_N$ has rank $2N - 2$, so it will have 2 singular values at 0. Therefore, $\sigma_{min}(\mathbf{T}_N) < \sigma_{min}(\mathcal{T})$ for this matrix, and we cannot use the matrix symbol as a lower bound for the singular values. □

We can however use the matrix symbol to bound the *largest* singular value of the block Toeplitz matrix, as shown in Lemma 3.4.2.

Lemma 3.4.2. *If $\{\mathbf{T}_n\}_{n\in\mathbb{N}^+}$ is the set of truncated block Toeplitz matrices with the symbol $\mathcal{T} \in \mathcal{L}^\infty_{2\pi}$, then*

$$\sigma_{max}(\mathbf{T}_n) \leq \sigma_{max}(\mathcal{T}) \qquad \forall n \in \mathbb{N}^+.$$

Proof. See [168, Theorem 4.1]. □

Note that this bound also holds for the truncated matrices, so all sizes of the truncated matrix will have their largest singular value bounded by the largest singular value of the associated matrix symbol.

Eigenvalues

For analyzing the eigenvalues of a block Toeplitz matrix, we focus on Hermitian matrices and show that the eigenvalue spectrum of the block Toeplitz matrix (and its truncations) can be bounded using the largest and smallest eigenvalues of the associated matrix symbol, as in Lemma 3.4.3.

Lemma 3.4.3. *Let $\mathcal{T} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{n\times n}$ be Hermitian and the matrix symbol for the Hermitian block Toeplitz matrix $\mathbf{T}$ with truncations $\{\mathbf{T}_N\}_{N\in\mathbb{N}^+}$. Then,*

$$\lambda_{min}(\mathcal{T}) \leq \lambda_{min}(\mathbf{T}) \leq \lambda_{min}(\mathbf{T}_N) \leq \lambda_{max}(\mathbf{T}_N) \leq \lambda_{max}(\mathbf{T}) \leq \lambda_{max}(\mathcal{T}).$$

Proof. See [124, Section 2] and Lemma 3.4.1. □

Corollary 3.4.1. *Let $\mathcal{T} \in \mathcal{L}_{2\pi}^{\infty} : \mathbb{T} \to \mathbb{C}^{n\times n}$ be Hermitian and the matrix symbol for the Hermitian block Toeplitz matrix $\mathbf{T}$. If $0 \notin [\lambda_{min}(\mathcal{T}), \lambda_{max}(\mathcal{T})]$, then $\mathbf{T}$ and its truncations $\{\mathbf{T}_N\}_{N\in\mathbb{N}^+}$ are positive definite matrices.*

3.4.3 Szegö's limit theorem

Perhaps the most well known theoretical result in the study of the spectrum of Toeplitz/block Toeplitz matrices is the *Szegö limit theorem*. This theorem was first developed in 1915 by Szegö in his work [167] for the non-block case, and then subsequently extended to the block Toeplitz case by many authors, including [59, 125] for Hermitian matrices and [168] for rectangular matrices. The Szegö limit theorem for Hermitian block Toeplitz matrices is given in Theorem 3.4.2.

Theorem 3.4.2 (Szegö's limit theorem for Hermitian block Toeplitz matrices [125])**.** *Let $\mathcal{T} \in \mathcal{L}_{2\pi}^{\infty}$ be an $n\times n$ Hermitian function and the matrix symbol for the block Toeplitz matrix $\mathbf{T}_N$ with N blocks. Then*

$$\lim_{N\to\infty} \frac{1}{nN} \sum_{k=0}^{nN} g(\lambda_k(T_N)) = \frac{1}{2\pi} \int_0^{2\pi} \frac{1}{n} \sum_{k=0}^{n} g(\lambda_k(\mathcal{T}(\omega)))\, \mathrm{d}\omega \tag{3.26}$$

for all test functions $g(\,\cdot\,)$ continuous on the interval $[\lambda_{min}(\mathcal{T}), \lambda_{max}(\mathcal{T})]$.

Proof. [125, Theorem 3.3] □

Overall, the Szegö limit theorem as given in Theorem 3.4.2 is a beautiful piece of machinery for analyzing block Toeplitz matrices. Essentially, the Szegö limit theorem says that as the truncated block Toeplitz matrix approaches infinite size, the arithmetic mean of its

eigenvalues (or the function $g(\cdot)$ of them) will approach the arithmetic mean of the eigenvalues of the matrix symbol (or the function $g(\cdot)$ of them). Note that for the Hermitian case in Theorem 3.4.2, when $g(\lambda) \coloneqq \lambda$ this is equivalent to saying that the scaled traces of the truncated block Toeplitz matrix converge to the scaled integral over the trace of the matrix symbol (since the trace is the sum of the eigenvalues) as the matrix size grows to infinity.

There are a few technicalities in Theorem 3.4.2 though, namely it only applies to Hermitian block Toeplitz matrices and the function $g(\cdot)$ used on the eigenvalues must be continuous on the interval $[\lambda_{min}(\mathcal{T}), \lambda_{max}(\mathcal{T})]$. This interval can be roughly thought of as spanning between the smallest eigenvalue of $\mathcal{T}(\cdot)$ and the largest eigenvalue of $\mathcal{T}(\cdot)$ that exist anywhere on the unit circle while ignoring outliers on a set of zero measure.

The Szegö limit theorem forms the basis for many other relations between the truncated matrices and the matrix symbol through various choices of the function $g(\cdot)$, such as relating the matrix determinant. These relations can then be exploited in various fields to determine quantities in terms of the matrix symbol instead of the truncated matrix — such as computing the Minimum Mean Square Error (MMSE) or geometric MMSE of a linear predictor in information theory [57, 61].

3.5 Conclusions

In this chapter, we presented an overview of block Toeplitz matrices and some of their associated properties in preparation for their extensive use in Part II. We examined when operations involving block Toeplitz matrices will result in block Toeplitz matrices, showing that the basic operations of addition/transpose will always produce one while matrix multiplication only produces one in special cases. We then introduced the idea of the matrix symbol of a block Toeplitz matrix, and showed how the extrema of the symbol's spectrum can be used as bounds on the extrema of the spectrum of the truncated block Toeplitz matrices.

Part II

Toeplitz Operators in Linear MPC

Chapter 4

Spectral properties for the condensed problem[†]

The spectrum and condition number of the condensed Hessian from (2.7) is an important factor in the convergence and implementation of many first-order methods for solving the CLQR problem. In this chapter, we derive some spectral properties (i.e. bounds on the largest/smallest eigenvalue and the condition number) of the matrices used in the CLQR problem, with a focus on the primal and dual condensed Hessians. These spectral properties are horizon-independent and easily computable using the system transfer function and cost matrices. The results in this chapter will provide a grounding for the analysis and design methods we will present in Chapters 5, 6, and 7.

This chapter begins with the analysis of the prediction matrix Γ_c in Section 4.1. Those results are then used in Section 4.2 to analyze the condensed Hessian and derive its spectral properties. Next we examine the condensed constraint matrix in Section 4.3, and then examine the dual condensed Hessian in Section 4.4.

[†]Some of the material presented in this chapter is contained in the following work:
I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Horizon-independent preconditioner design for linear predictive control. *IEEE Transactions on Automatic Control*, 2022. doi: 10.1109/TAC.2022.3145657. (*in press*).

4.1 System prediction matrix

We start by analyzing the prediction matrix Γ_c from the numerically robust CLQR formulation (2.10) and note that its diagonals are constant blocks, which means that Γ_c is a block Toeplitz matrix (and is actually the same one that is given in the control theory example in Section 3.1.4). As we discussed in Chapter 3, finite-dimensional block Toeplitz matrices with blocks of size $m \times n$ can be viewed as the truncation of an infinite-dimensional block Toeplitz matrix by extending the block pattern, and these can then be related to a matrix-valued function in $\mathcal{L}^\infty_{2\pi}$, called the matrix symbol, that maps $\mathbb{T} \to \mathbb{C}^{m\times n}$.

As discussed in Section 3.2.1, the blocks on the diagonals of the matrix give the spectral coefficients of the matrix symbol. This means that one possible representation of the symbol is as a Fourier series with the coefficients given by the matrix blocks. For the prediction matrix Γ in the condensed CLQR formulation (2.7), the Fourier series that only uses the blocks in the horizon is given by

$$\sum_{i=0}^{N-1} A^i B z^{-i}, \quad \forall z \in \mathbb{T}.$$

As the horizon length increases, this series is only guaranteed to converge to a finite matrix symbol when the system $\mathcal{G}$ with state-space matrices A and B is Schur-stable.

To form a convergent Fourier series no matter the system, we use the numerically robust CLQR formulation from Section 2.3.2 to introduce a stabilizing linear state-feedback controller $u_k = -Kx_k + v_k$ to the prediction. For systems where the pair (A, B) is stabilizable, the prestabilizing controller leads to the prediction matrix Γ_c having a convergent Fourier series and the finite matrix symbol given in Lemma 4.1.1.

Lemma 4.1.1. *Let the pair (A, B) be stabilizable and $K \in \mathbb{R}^{m\times n}$ be a linear state-feedback control matrix used to form the prestabilized system* (2.9)*. The prediction matrix Γ_c then has the matrix symbol $\mathcal{P}_{\Gamma_c} \in \tilde{\mathcal{C}}_{2\pi}$ with*

$$\mathcal{P}_{\Gamma_c}(z) \coloneqq z(zI - (A - BK))^{-1}B = z\mathcal{G}_c(z), \quad \forall z \in \mathbb{T},$$

where $\mathcal{G}_c(\,\cdot\,)$ is the transfer function matrix for the discrete-time system $\mathcal{G}_c$.

Proof. The matrix symbol for the block Toeplitz matrix Γ_c is derived using the infinite-dimensional extension of Γ_c, called $\mathbf{\Gamma}_c$. The blocks that make up $\mathbf{\Gamma}_c$ can be extrapolated from (2.11) to

$$[\mathbf{\Gamma}_c]_i \coloneqq \begin{cases} 0 & \text{if } i < 0, \\ (A - BK)^i B & \text{if } i \geq 0, \end{cases} \tag{4.1}$$

where i is the number of the block diagonal of the matrix, with $i = 0$ being the main diagonal and positive i below the main diagonal. A possible way to form the matrix symbol of a block Toeplitz matrix is to define it as the trigonometric polynomial with the blocks of the matrix as coefficients [59, §4.3]. Doing that for $\mathbf{\Gamma}_c$ uses the blocks (4.1) as the coefficients to form the trigonometric polynomial

$$\sum_{i=0}^{\infty} z^{-i}(A - BK)^i B. \tag{4.2}$$

The constant B matrix can be factored from the summation, giving

$$\left(\sum_{i=0}^{\infty}(A - BK)^i z^{-i}\right) B.$$

Since K was designed to make $(A - BK)$ Schur-stable, the summation becomes a convergent Neumann series that converges to $z(zI - (A - BK))^{-1}$ [141, §3.4]. With the B matrix right-multiplying the summation, the result is the transfer function matrix for the time-shifted discrete-time system $z\mathcal{G}_c(z)$, giving the matrix symbol in the lemma. Finally, note that the coefficients in the sum (4.2) are absolutely summable, so $\mathcal{P}_{\Gamma_c}$ is in the Wiener class, meaning that $\mathcal{P}_{\Gamma_c} \in \mathcal{L}^\infty$ and is continuous and 2π-periodic, leading to $\mathcal{P}_{\Gamma_c} \in \tilde{\mathcal{C}}_{2\pi}$. □

It is tempting to only apply the stabilizing controller K after the horizon, like the CLQR stability theory in [114] does, but this will cause $\mathbf{\Gamma}_c$ to have $A^i B$ in the upper-left $N \times N$ block submatrix and $(A - BK)^i B$ in in the remaining part, breaking the block Toeplitz structure.

At this point, there is no restriction on the type of linear state-feedback controller used to prestabilize the system — any controller that results in a Schur-stable closed-loop system can be used. A convenient choice for K is the infinite-horizon LQR controller designed using the cost matrices in (2.2a) with P chosen to be the solution to the DARE (2.3) as described in Section 2.1.1.

If the system $\mathcal{G}$ is Schur-stable to begin with, then there is no need for a pre-stabilizing controller K and the matrices Γ and Γ_c can be the same. The matrix symbol for Γ_c can then be simplified as shown in Corollary 4.1.1.

Corollary 4.1.1. *If the system $\mathcal{G}$ is Schur-stable, then with $K = 0$ the prediction matrix Γ has a convergent Fourier series, producing the matrix symbol $\mathcal{P}_\Gamma \in \tilde{\mathcal{C}}_{2\pi}$ with*

$$\mathcal{P}_\Gamma(z) \coloneqq z(zI - A)^{-1}B = z\mathcal{G}(z), \qquad \forall z \in \mathbb{T},$$

where $\mathcal{G}(\,\cdot\,)$ is the transfer function matrix for the system $\mathcal{G}$.

One of the useful properties of block Toeplitz matrices is the relation between the spectrum of the matrix (and its finite-dimensional truncations) and the spectrum of its matrix symbol that we discussed in Section 3.4. While we showed in Example 3.4.1 that it is hard to bound the lower singular value of an arbitrary block Toeplitz matrix, we can actually find such a bound for the prediction matrix. That means we can find a bound for all the singular values of the prediction matrix, and also estimate their locations using the matrix symbol. Specifically, we can say that the distribution of the spectrum of the matrix symbol evaluated on $\mathbb{T}$ is the same as the distribution of the spectrum for $\mathbf{T}_n$ as $n \to \infty$, and the spectrum of $\mathbf{T}_n$ will always be contained in the spectrum of its symbol. This means that we can utilize $\mathcal{P}_{\Gamma_c}$ to find the distribution of the singular values of Γ_c and bound them.

Proposition 4.1.1. *Let $\mathcal{G}_c$ be the prestabilized Schur-stable system predicted over a horizon of length N, then the following are true:*

(a) $\sigma_{min}(\mathcal{G}_c) \leq \sigma(\mathbf{\Gamma}_c) \leq \|\mathcal{G}_c\|_{H_\infty}$

(b) $\lim_{N\to\infty} \kappa(\mathbf{\Gamma}_{c_N}) = \kappa(\mathcal{G}_c)$

(c) $\sigma(\mathbf{\Gamma}_{c_N}) \approx \bigcup_{\omega\in\Omega} \sigma(\mathcal{G}_c(e^{j\omega}))$
with $\Omega \coloneqq \left\{\omega : \omega = -\frac{\pi}{2} + \frac{2\pi}{N}i,\ i \in [0, 1, \ldots, N-1]\right\}$

Proof. $\mathbf{\Gamma}_c$ is lower-triangular, so the matrix $\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c$ is also block Toeplitz with the matrix symbol $\mathcal{G}_c^*\mathcal{G}_c \in \tilde{\mathcal{C}}_{2\pi}$ [59, Lemma 4.5] since $z^*z = 1$ for $z \in \mathbb{T}$.

(a) We can directly upper bound the singular values of $\mathbf{\Gamma}_c$ and its truncations using its matrix symbol as described in Lemma 3.4.2. Then note that the H_∞ norm of a system is its largest singular value, giving the upper bound.

To lower bound the singular values, we use the fact that the product $\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c$ is block Toeplitz with its symbol in $\tilde{\mathcal{C}}_{2\pi}$ (as mentioned at the start of the proof). This means that we can bound the eigenvalues of $\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c$ using Lemma 3.4.3 and say

$$\lambda_{min}(\mathcal{G}_c^*\mathcal{G}_c) \leq \lambda_{min}(\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c) \leq \lambda_{max}(\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c) \leq \lambda_{max}(\mathcal{G}_c^*\mathcal{G}_c). \tag{4.3}$$

Since we only require the lower bound, we focus on that portion of (4.3) and apply the knowledge that $\lambda(\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c) = \sigma(\mathbf{\Gamma}_c)^2$ to find

$$\sigma_{min}(\mathcal{G}_c)^2 \leq \sigma_{min}(\mathbf{\Gamma}_c)^2.$$

Since singular values are always positive, we can then take the square root of both sides and get the final inequality

$$\sigma_{min}(\mathcal{G}_c) \leq \sigma_{min}(\mathbf{\Gamma}_c).$$

(b) Taking the limit of the condition number as $N \to \infty$ gives:

$$\lim_{N\to\infty} \kappa(\mathbf{\Gamma}_{c_N}) = \lim_{N\to\infty} \frac{\sigma_{max}(\mathbf{\Gamma}_{c_N})}{\sigma_{min}(\mathbf{\Gamma}_{c_N})} = \frac{\sigma_{max}(\mathcal{G}_c)}{\sigma_{min}(\mathcal{G}_c)} = \kappa(\mathcal{G}_c).$$

(c) It is known that the singular values of the matrix $\mathbf{\Gamma}_c$ are related to the eigenvalues of $\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c$ through $\sigma(\mathbf{\Gamma}_c)^2 = \lambda(\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c)$. From [125], the eigenvalues of $\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c$ can be estimated as $N \to \infty$ by approximating the block Toeplitz matrix as a block circulant matrix with the same matrix symbol, and then finding the eigenvalues of the block circulant matrix. In this case, that means evaluating the eigenvalues of the matrix symbol $\mathcal{G}_c^*\mathcal{G}_c$ around the unit circle, i.e.

$$\lambda(\mathbf{\Gamma}_c^*\mathbf{\Gamma}_c) \approx \bigcup_{\omega\in\Omega} \lambda(\mathcal{G}_c(e^{j\omega})^*\mathcal{G}_c(e^{j\omega})). \tag{4.4}$$

Since the right-hand side of (4.4) evaluates to a matrix at every point ω, we can rewrite (4.4) as

$$(\sigma(\mathbf{\Gamma}_c))^2 \approx \bigcup_{\omega\in\Omega} \sigma(\mathcal{G}_c(e^{j\omega}))^2.$$

Taking the square root of both sides gives the final result.

□

To understand Proposition 4.1.1, we can examine Figure 4.1 — which is a plot showing the singular values of the prediction matrix for the first Schur-stable system in Section 2.6. In this plot, we can see the upper and lower bounds of the singular values computed using Proposition 4.1.1(a) shown as red lines. The actual singular values are shown as blue asterisks, and it can be seen that they are contained between the two red lines, showing the upper and lower bounds can contain the singular values. For the estimation of the singular values using Proposition 4.1.1(c), we can see that the actual singular values (the blue asterisks) and the estimated ones using Proposition 4.1.1(c) (the red squares) are distinctly separated for short horizon lengths, but they slowly move towards each other as the horizon length is increased.

4.2 Primal Hessian

4.2.1 Matrix symbol

The Hessian of the MPC problem formulation in (2.10) can be split into four distinct parts

$$H_c \coloneqq H_Q + H_P + H_K + H_R \tag{4.5}$$

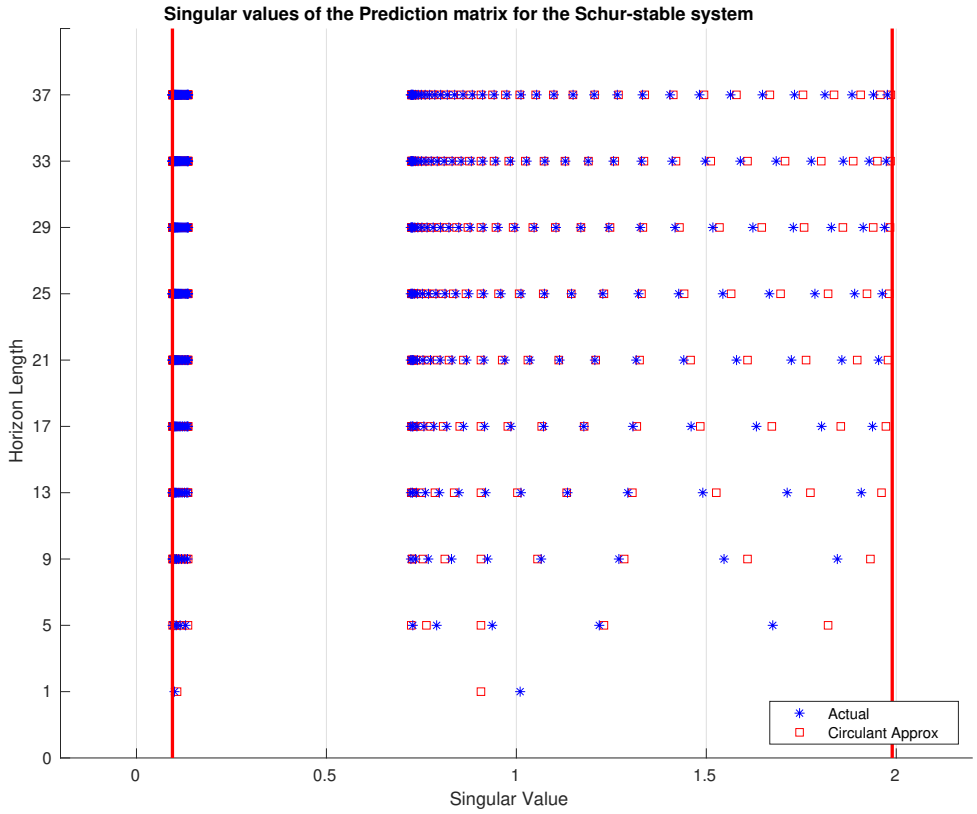


Figure 4.1: Singular values of the prediction matrix for the first Schur-stable system compared to the approximation from Proposition 4.1.1.

where H_Q, H_R, H_P and H_K are the parts that contain the matrices Q, R, P, and the K cross-terms, respectively. Slightly different analysis must be done depending on the choice of P, and in this work we focus on the two cases when $P = Q$ and P is the solution to the DARE (2.3) for the infinite-dimensional unconstrained LQR.

P is the same as Q

Choosing $P = Q$ for (2.2) allows the term H_P to be consolidated into H_Q, giving

$$\begin{aligned} H_c &= \Gamma_c' \bar{Q}_c \Gamma_c + \Gamma_c' \bar{K}' \bar{R} + \bar{R}\bar{K}\Gamma_c + \bar{R}, \\ \bar{Q}_c &= I_N \otimes (Q + K'RK). \end{aligned}$$

Analysis of the resulting infinite-dimensional Hessian $\mathbf{H}_c$, reveals that $\mathbf{H}_c$ is also block Toeplitz with the matrix symbol given in Lemma 4.2.1.

Lemma 4.2.1. *Let $P = Q$ and $\mathcal{P}_{\Gamma_c}$ be the matrix symbol for $\mathbf{\Gamma}_c$ from Lemma 4.1.1. The matrix $\mathbf{H}_c$ is then a block Toeplitz matrix with the matrix symbol $\mathcal{P}_{H_c} \in \tilde{\mathcal{C}}_{2\pi}$, where*

$$\mathcal{P}_{H_c}(z) \coloneqq \mathcal{P}^*_{\Gamma_c}(z)(Q + K'RK)\mathcal{P}_{\Gamma_c}(z) - \mathcal{P}^*_{\Gamma_c}(z)K'R - RK\mathcal{P}_{\Gamma_c}(z) + R, \quad \forall z \in \mathbb{T}.$$

Proof. Note that $\mathbf{H}_c$ is the sum of four matrices: $\mathbf{\Gamma}_c'\bar{\mathbf{Q}}_c\mathbf{\Gamma}_c$, $\mathbf{\Gamma}_c'\bar{\mathbf{K}}'\bar{\mathbf{R}}$, $\bar{\mathbf{R}}\bar{\mathbf{K}}\mathbf{\Gamma}_c$ and $\bar{\mathbf{R}}$. Using the assumption that $P = Q$, we can see that the new state weighting matrix $\bar{Q}_c$ is block diagonal with the same entry in each block, making $\bar{\mathbf{Q}}_c$ a block Toeplitz matrix with the symbol $\mathcal{P}_{\bar{Q}_c}(z) \coloneqq Q + K'RK$. Since $\mathbf{\Gamma}_c$ is a lower-triangular block matrix and $\mathbf{\Gamma}_c'$ is an upper-triangular block matrix, the product $\mathbf{\Gamma}_c'\bar{\mathbf{Q}}\mathbf{\Gamma}_c$ is block Toeplitz with matrix symbol $\mathcal{P}^*_{\Gamma_c}\mathcal{P}_{\bar{Q}_c}\mathcal{P}_{\Gamma_c}$ according to Corollary 3.3.1.

$\bar{\mathbf{R}}$ and $\bar{\mathbf{K}}$ are both block diagonal with the same entry in every block, making $\bar{\mathbf{R}}$ and $\bar{\mathbf{K}}$ both block Toeplitz matrices with symbols $\mathcal{P}_{\bar{R}}(z) \coloneqq R$ and $\mathcal{P}_{\bar{K}}(z) \coloneqq -K$, respectively. The multiplications $\mathbf{\Gamma}_c'\bar{\mathbf{K}}'\bar{\mathbf{R}}$ and $\bar{\mathbf{R}}\bar{\mathbf{K}}\mathbf{\Gamma}_c$ are then block Toeplitz, since the product $\bar{\mathbf{R}}\bar{\mathbf{K}}$ produces a block diagonal block Toeplitz matrix, which preserves the block Toeplitz structure of $\mathbf{\Gamma}_c$ during the multiplication. The matrix symbols for $\mathbf{\Gamma}_c'\bar{\mathbf{K}}'\bar{\mathbf{R}}$ and $\bar{\mathbf{R}}\bar{\mathbf{K}}\mathbf{\Gamma}_c$ are then $-\mathcal{P}^*_{\Gamma_c}(z)K'R$ and $-RK\mathcal{P}_{\Gamma_c}(z)$, respectively. Block Toeplitz structure is preserved over the addition of two or more block Toeplitz matrices, and the matrix symbol for the resulting sum is the sum of the original matrix symbols (Lemma 3.3.1). This means that the matrix $\mathbf{H}_c$ is block Toeplitz with the matrix symbol given in the lemma. □

Note that the structure of the matrix H_Q can be seen in (4.6), and that the truncation for finite horizons is not perfectly block Toeplitz. The reasoning behind for equating the actual H_Q and the infinite-horizon $\mathbf{H}_Q$ is that as the horizon grows, the discrepancy between H_Q

$$H_Q \coloneqq \begin{bmatrix} \sum_{i=0}^{N-1} B'(A_c^i)'Q_cA_c^iB & \sum_{i=0}^{N-2} B'(A_c^{i+1})'Q_cA_c^iB & \sum_{i=0}^{N-3} B'(A_c^{i+2})'Q_cA_c^iB & \cdots & B'(A_c^{N-1})'Q_cB \\ \sum_{i=0}^{N-2} B'(A_c^i)'Q_cA_c^{i+1}B & \sum_{i=0}^{N-2} B'(A_c^i)'Q_cA_c^iB & \sum_{i=0}^{N-3} B'(A_c^{i+1})'Q_cA_c^iB & \cdots & B'(A_c^{N-2})'Q_cB \\ \sum_{i=0}^{N-3} B'(A_c^i)'Q_cA_c^{i+2}B & \sum_{i=0}^{N-3} B'(A_c^i)'Q_cA_c^{i+1}B & \sum_{i=0}^{N-3} B'(A_c^i)'Q_cA_c^iB & \cdots & B'(A_c^{N-3})'Q_cB \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B'Q_cA_c^{N-1}B & B'Q_cA_c^{N-2}B & B'Q_cA_c^{N-3}B & \cdots & B'Q_cB \end{bmatrix} \tag{4.6}$$

$$H_P \coloneqq \begin{bmatrix} B'(A_c^N)'PA_c^NB & B'(A_c^N)'PA_c^{N-1}B & B'(A_c^N)'PA_c^{N-2}B & \cdots & B'(A_c^N)'PA_cB \\ B'(A_c^{N-1})'PA_c^NB & B'(A_c^{N-1})'PA_c^{N-1}B & B'(A_c^{N-1})'PA_c^{N-2}B & \cdots & B'(A_c^{N-1})'PA_cB \\ B'(A_c^{N-2})'PA_c^NB & B'(A_c^{N-2})'PA_c^{N-1}B & B'(A_c^{N-2})'PA_c^{N-2}B & \cdots & B'(A_c^{N-2})'PA_cB \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B'A_c'PA_c^NB & B'A_c'PA_c^{N-1}B & B'A_c'PA_c^{N-2}B & \cdots & B'A_c'PA_cB \end{bmatrix} \tag{4.7}$$

and the finite-dimensional truncation of $\mathbf{H}_Q$ will shrink since the new components in the summation will be very small. This means any discrepancy will be mostly limited to the final rows/columns — making the upper left submatrices of the two nearly identical. How quickly this happens in practice will depend on how quickly A^i decays to 0 as $i \to \infty$.

It is important to note that as shown in Section 3.3, the product of block Toeplitz matrices is not guaranteed to be block Toeplitz except in certain special cases, while the addition of multiple block Toeplitz matrices with compatible block sizes is always guaranteed to produce a block Toeplitz result. In this case, the lower-triangular structure of $\mathbf{\Gamma}_c$ and the block Toeplitz structure of $\bar{\mathbf{Q}}_c$ implies that the product $\mathbf{\Gamma}_c'\bar{\mathbf{Q}}\mathbf{\Gamma}_c$ is one of the special cases where the multiplication of the three block Toeplitz matrices of compatible block sizes is block Toeplitz. Choosing an arbitrary P with $P \neq Q$ will cause $\bar{\mathbf{Q}}_c$ to no longer be block Toeplitz, so the multiplication will not necessarily produce a block Toeplitz matrix and the Hessian may not be block Toeplitz.

P is the solution to the DARE

Choosing P as the solution to the DARE (2.3) causes the original $\bar{Q}_c$ from Section 2.3.2 to not be block Toeplitz, since $\bar{Q}_c$ will then have a different matrix in the lower-right corner than the rest of the main diagonal. This means that the analysis for the product $\Gamma_c'\bar{Q}_c\Gamma_c$ based on the multiplication of structured block Toeplitz matrices used in the proof of Lemma 4.2.1 no longer can be applied. However, the resulting $\mathbf{H}_c$ matrix is still block Toeplitz due to the fact that P will represent the cost of the controller applied after the horizon ends. We can then derive the matrix symbol for this case, which we show in Proposition 4.2.1 is the same as the case when $P = Q$.

Proposition 4.2.1. *If P is the solution to the DARE (2.3) and K is the infinite-horizon LQR controller for $\mathcal{G}$, then $\mathbf{H}_c$ is block Toeplitz and has the same matrix symbol as the case when $P = Q$ given in Lemma 4.2.1.*

Proof. Using the matrix splitting (4.5), the Hessian can be decomposed into three terms, with H_Q and H_P given by (4.6) and (4.7), respectively, and the remaining term given by $H_K + H_R$. We start by examining the first diagonal term of $H_Q + H_P$,

$$\sum_{i=0}^{N-1} B'(A_c^i)'Q_cA_c^iB + B'(A_c^N)'PA_c^NB. \tag{4.8}$$

Since P is the solution to the DARE (2.3), P is also the solution to the Lyapunov equation

$$(A - BK)'P(A - BK) + Q + K'RK = P \tag{4.9}$$

when K is the infinite-horizon LQR controller. This means that P can also be expressed as

$$P = \sum_{i=0}^{\infty}((A - BK)')^i(Q + K'RK)(A - BK)^i,$$

transforming (4.8) into

$$\sum_{i=0}^{N-1} B'(A_c^i)'Q_cA_c^iB + B'(A_c^N)'\left(\sum_{i=0}^{\infty}(A_c^i)'Q_cA_c^i\right)A_c^NB.$$

The $(A_c^N)'$ and A_c^N terms around the right summation can be consolidated into the summation, offsetting its starting point to be $i = N$ instead of 0. This means the right summation is simply the continuation of the left summation to infinity, allowing the two to be consolidated into

$$\sum_{i=0}^{\infty} B'(A_c^i)'Q_cA_c^iB. \tag{4.10}$$

The same analysis can be performed on the other terms on the main diagonal, which only differ by where the left summation ends and the right summation is offset to. Therefore, the main diagonal of the matrix sum $H_Q + H_P$ is composed of blocks with all the same terms. A similar analysis can be done on all diagonals above and below the main diagonal, showing that they are also composed of blocks with all the same terms down the diagonal.

Since all the diagonals are composed of the same blocks down their length, the matrix sum $H_Q + H_P$ is block Toeplitz, and the resulting finite-dimensional Hessian H_c is block Toeplitz as well, since $H_K + H_R$ is already known to be block Toeplitz from Lemma 4.2.1.

Now that we showed the actual Hessian is block Toeplitz, we can find the infinite-dimensional matrix $\mathbf{H}_c$ it is a truncation of and the associated matrix symbol. To construct the matrix symbol, we examine the elements in the matrix $H_Q + H_P$ and how they relate to the case when $P = Q$ that we have already worked with. Note that when $P = Q$, the individual elements of the matrix H_Q have a summation that terminates at the horizon length. Since the matrix symbol is based on the infinite-dimensional matrix, if the matrix H_Q is extrapolated to a horizon of infinity to form $\mathbf{H}_Q$, the summations in H_Q will all terminate at infinity. Therefore, the finite-dimensional sum $H_Q + H_P$ will have the same blocks as the infinite-dimensional $\mathbf{H}_Q$ when $P = Q$, so the Hessians for the cases when $P = Q$ and P is the solution to the DARE will both have the same matrix symbol. □

An interesting thing to note in this case is that the actual finite-dimensional truncation of the Hessian is block Toeplitz — which is different from the case when $P = Q$. This is a direct result of the fact that choosing P as the solution of the DARE is meant to capture the remaining cost from the end of the horizon to infinity, which we can see represented in the sums that comprise the components of the H_P matrix (4.7).

Simplification when $\mathcal{G}$ is Schur-stable

When $\mathcal{G}$ is Schur-stable and the results in Corollary 4.1.1 are used to simplify the matrix symbol of the prediction matrix, the results given in Lemma 4.2.1 and Proposition 4.2.1 can be simplified as well.

Corollary 4.2.1. *If the system $\mathcal{G}$ is Schur-stable, then with $K = 0$ and $P = Q$ or P the solution to the discrete Lyapunov equation* (2.4)*, the Hessian $\mathbf{H}$ has the matrix symbol $\mathcal{P}_H \in \tilde{\mathcal{C}}_{2\pi}$ with*

$$\mathcal{P}_H(z) \coloneqq \mathcal{P}_\Gamma^*(z) Q \mathcal{P}_\Gamma(z) + R, \quad \forall z \in \mathbb{T}.$$

where $\mathcal{G}(\cdot)$ is the transfer function matrix for the system $\mathcal{G}$.

Note that the matrix symbol in Corollary 4.2.1 has the same form as the matrix symbol in Lemma 4.2.1, however the choice of the matrix P in the CLQR problem differs.

In order for Proposition 4.2.1 to reduce to Corollary 4.2.1 in the Schur-stable case with $K = 0$, the terminal cost must be based on the solution to the discrete Lyapunov equation instead of the DARE, since using the DARE solution with no pre-stabilizing controller will cause the Lyapunov equation (4.9) used in the proof of Proposition 4.2.1 to be invalid.

4.2.2 Spectral bounds

One useful property of block Toeplitz matrices is that the eigenvalue spectrum for any finite-dimensional truncation of the infinite-dimensional block Toeplitz matrix is contained within the extremal eigenvalues of its matrix symbol. This means that the minimum and maximum eigenvalues of the matrix H_c can be bounded by analyzing the matrix symbol $\mathcal{P}_{H_c}$, since H_c is a finite-dimensional truncation of the infinite-dimensional matrix $\mathbf{H_c}$ to N blocks.

Theorem 4.2.1. *Let H_c be the condensed Hessian with P the solution to the DARE* (2.3) *for a prediction horizon of length N that is block Toeplitz with matrix symbol $\mathcal{P}_{H_c}$ given in Lemma 4.2.1, then the following hold:*

(a) $\lambda_{min}(\mathcal{P}_{H_c}) \leq \lambda(H_c) \leq \lambda_{max}(\mathcal{P}_{H_c})$

(b) $\lim_{N\to\infty} \kappa(H_c) = \kappa(\mathcal{P}_{H_c})$

Proof.

(a) The spectrum of a finite-dimensional truncation of a block Toeplitz matrix with a symbol in $\tilde{\mathcal{C}}_{2\pi}$ is bounded by the extremes of the spectrum of its symbol (Lemma 3.4.3).

(b) H_c is a Hermitian matrix, which means that it is also normal [71, §4.1]. Since it is both normal and positive semidefinite, the singular values are the same as the eigenvalues [70, §3.1], resulting in the condition number becoming $\kappa(H_c) = \frac{\lambda_n(H_c)}{\lambda_1(H_c)}$. Taking the limit of both sides in conjunction with the spectral bounds from part (a) gives

$$\lim_{N\to\infty} \kappa(H_c) = \kappa(\mathcal{P}_{H_c}).$$

□

Corollary 4.2.2. *If the system $\mathcal{G}$ is Schur-stable, then with $K = 0$ and P the solution to the discrete Lyapunov equation* (2.4)*, the results in Theorem 4.2.1 can be applied to H using the matrix symbol $\mathcal{P}_H$ instead of $\mathcal{P}_{H_c}$.*

Essentially, Theorem 4.2.1 say that the spectrum for the condensed Hessian in these cases will always be contained inside the interval defined by the maximum and minimum eigenvalues of the matrix symbol in Lemma 4.2.1/Corollary 4.2.1. Additionally, as $N \to \infty$ the extremal eigenvalues of the Hessian will converge asymptotically to the maximum and minimum eigenvalues of its symbol. This can be seen in Figure 4.2, where we have plotted the extremal eigenvalues of the Hessian matrix for the first Schur-stable system from Section 2.6 as the horizon length increases.

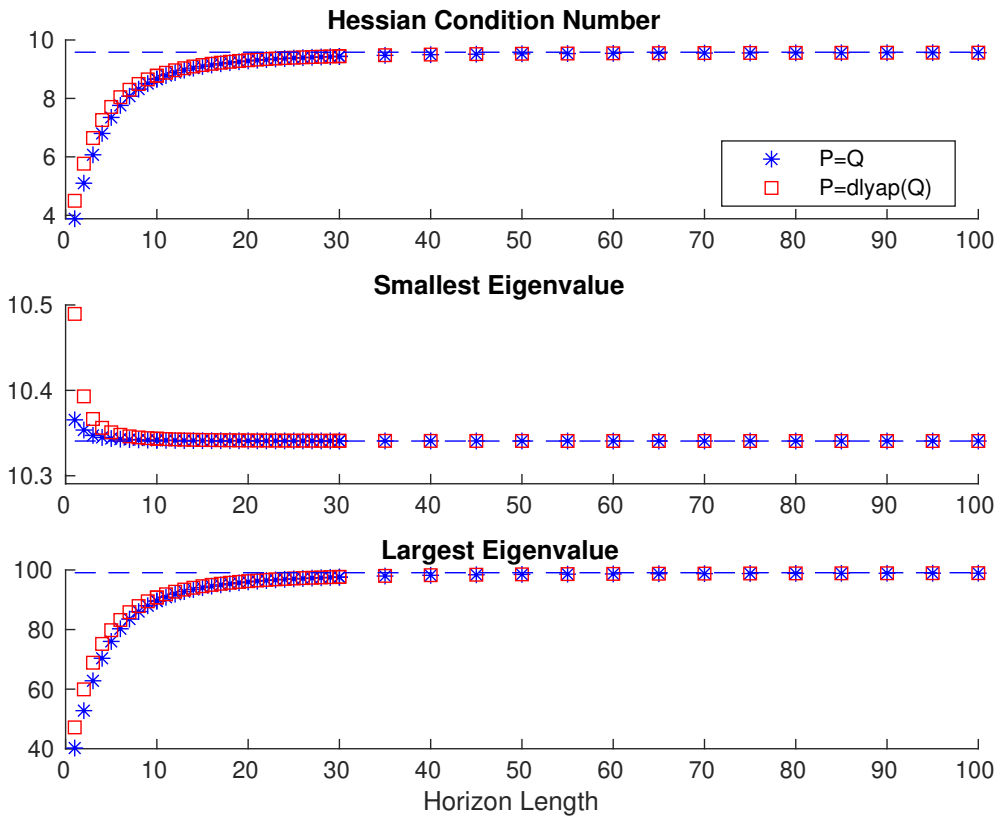


Figure 4.2: Spectral bounds for the condensed Hessian of the first Schur-stable example problem with either $P = Q$ or P the solution to the discrete Lyapunov equation.

We compare the eigenvalue spectrum of the Hessian with $P = Q$ and with P the solution to the discrete Lyapunov equation (since the system is Schur-stable) in Figure 4.3. In this figure, we can see that both cases behave similarly, with the major difference being the exact locations of the eigenvalues for shorter horizon lengths, suggesting that Theorem 4.2.1 may still apply to the case when $P = Q$ even though the truncated matrix is not block Toeplitz. As the horizon length increases, the eigenvalues appear to become dense on the ranges, showing that for long/infinite horizons the two matrices become the same.

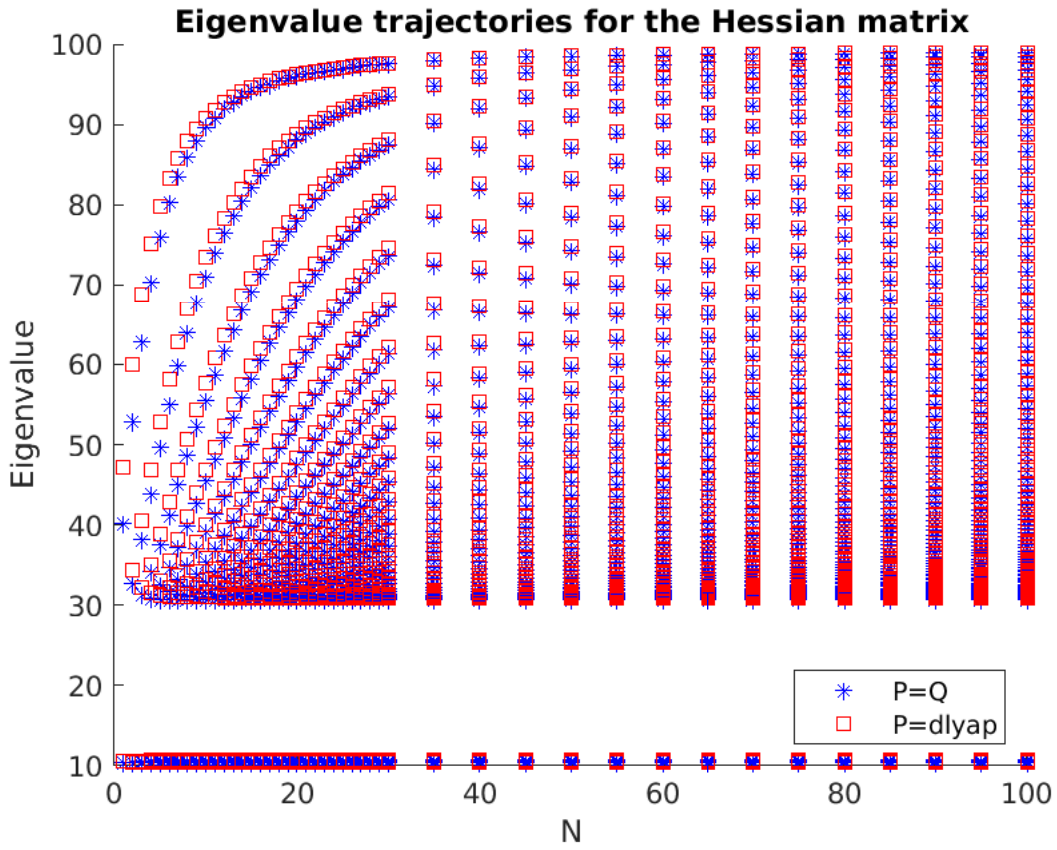


Figure 4.3: Eigenvalue spectrum for the condensed Hessian of the first Schur-stable example problem with either $P = Q$ or P the solution to the discrete Lyapunov equation.

4.2.3 Inclusion of a state-input cross term

We now examine what happens when the state-input cross-term weighting matrix S is included in problem (2.2), which can happen when the cost is discretized using the method in Section 2.1.1. To do this, we introduce a new term, H_S, to the splitting of the Hessian from (4.5) to create the new splitting

$$H_{cS} \coloneqq H_Q + H_P + H_K + H_S + H_R.$$

While the components H_Q, H_R, H_K and H_P are the same as in Section 4.2.1 and are block Toeplitz, the additional H_S term is not block Toeplitz, so the overall Hessian H_{cS} will not be block Toeplitz. This is due to the $\bar{S}$ matrix in Section 2.3.1 containing a 0 matrix instead of S in the lower-right corner, which when multiplied with Γ produces a row/column of zeros on the bottom/right of the matrix H_S.

To overcome the fact that H_{cS} is not block Toeplitz, we will instead split H_{cS} into two components, a nominal matrix H_n and a correction matrix H_e, such that

$$H_{cS} \coloneqq H_n - H_e.$$

We let the nominal matrix be

$$H_n \coloneqq H_Q + H_P + H_K + H_R + \bar{H}_S,$$

where $\bar{H}_S$ is H_S with an additional S weighting term on the final state, giving

$$\bar{H}_S \coloneqq (I_N \otimes S)'\Gamma + \Gamma'(I_N \otimes S).$$

Now $\bar{H}_S$ is block Toeplitz with the matrix symbol

$$\mathcal{P}_{\bar{H}_S}(z) \coloneqq S'\mathcal{P}_{\Gamma_c}(z) + \mathcal{P}^*_{\Gamma_c}(z)S \qquad \forall z \in \mathbb{T}.$$

This leads to H_n being block Toeplitz as well, with the matrix symbol $\mathcal{P}_{H_n} \in \tilde{\mathcal{C}}_{2\pi}$ given by

$$\mathcal{P}_{H_n}(z) = \mathcal{P}_{H_c}(z) + \mathcal{P}_{\bar{H}_S}(z) \qquad \forall z \in \mathbb{T}, \tag{4.11}$$

with $\mathcal{P}_{\Gamma_c}(\cdot)$ and $\mathcal{P}_{H_c}(\cdot)$ given by Lemmas 4.1.1 and 4.2.1, respectively.

We then correct for the additional weighting term introduced in H_n by subtracting a Hermitian correction matrix H_e, where

$$H_e \coloneqq S_c'\Gamma + \Gamma' S_c$$

with

$$S_c \coloneqq \begin{bmatrix} I_{N-1} \otimes 0 & 0 \\ 0 & S \end{bmatrix}.$$

To now understand how the correction term H_e affects the nominal matrix H_n, we first must analyze the spectrum of H_e.

Lemma 4.2.2. *Let $A_c \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and W_c be the state transition matrix, input matrix and the controllability Gramian, respectively, for the discrete-time system $\mathcal{G}_c$. If $S \neq 0$, then* $\operatorname{rank}(H_e) \leq 2m$*, and for $N \to \infty$ the $2m$ non-zero eigenvalues of H_e are the $2m$ eigenvalues of*

$$U \coloneqq \begin{bmatrix} B'S & I \\ S'W_cS & S'B \end{bmatrix}.$$

Proof. H_e can be decomposed as the outer product vu' with the matrices $v, u \in \mathbb{R}^{Nm\times 2m}$ defined as

$$v \coloneqq \begin{bmatrix} S'A_c^NB & S'A_c^{N-1}B & \cdots & S'A_cB & S'B \\ 0 & 0 & \cdots & 0 & I \end{bmatrix}',$$
$$u \coloneqq \begin{bmatrix} 0 & 0 & \cdots & 0 & I \\ S'A_c^NB & S'A_c^{N-1}B & \cdots & S'A_cB & S'B \end{bmatrix}'.$$

The rank of an outer product matrix can be no larger than the smallest rank of the component matrices, and we know that $\operatorname{rank}(u) \leq 2m$ and $\operatorname{rank}(v) \leq 2m$ — making $\operatorname{rank}(H_e) \leq 2m$. The non-zero eigenvalues of vu' (and consequently H_e) are the same as the eigenvalues of $u'v$ [71, Example 1.3.23], with

$$U \coloneqq u'v = \begin{bmatrix} B'S & I \\ \sum_{k=0}^{N} S'A^kBB'(A^k)'S & S'B \end{bmatrix}.$$

As $N \to \infty$, the summation in the lower-left element of $u'v$ converges to the controllability Gramian of the system $\mathcal{G}_c$ [36, §6.6], which makes the lower-left corner of the matrix U become $S'W_cS$. □

The results presented in Lemma 4.2.2 show that the correction matrix H_e has a finite and low rank independent of prediction horizon. Additionally, the limit points for the eigenvalues of H_e as $N \to \infty$ can be computed by finding the eigenvalues of the $2m \times 2m$ matrix U, which is independent of the horizon length. Knowledge of the eigenvalues of H_e then allows for the eigenvalues of the Hessian matrix H_{cS} to be bounded.

Theorem 4.2.2. *Let the system $\mathcal{G}_c$ be Schur-stable with the nominal part H_n of the Hessian H_{cS} having the matrix symbol from* (4.11) *and U from Lemma 4.2.2. Let $\gamma := \lambda_{max}(\mathcal{P}_{H_n})$, $\beta := \lambda_{min}(\mathcal{P}_{H_n})$, $\eta := \lambda_{max}(U)$, $\nu := \lambda_{min}(U)$. If $S \neq 0$, then the spectrum of the condensed Hessian H_{cS} has the following properties:*

(a) $\max\{0, \beta - \eta\} \leq \lambda(H_{cS}) \leq \gamma - \nu$

(b) $\lim\limits_{N\to\infty} \kappa(H_{cS}) \leq \begin{cases} \frac{\gamma-\nu}{\beta-\eta} & \text{if } \beta > \eta \\ \infty & \text{otherwise} \end{cases}$

Proof. Note that $H_{cS} = H_n - H_e$ can be viewed as the addition of the negation of H_e. Negating a matrix will negate all the eigenvalues, and consequently reverse their order. Since both H_n and H_e are Hermitian, the eigenvalues of H_{cS} can be bounded by [21, Fact 5.12.2]

$$\begin{aligned} \lambda_{min}(H_n) - \lambda_{max}(H_e) &\leq \lambda_{min}(H_{cS}), \\ \lambda_{max}(H_{cS}) &\leq \lambda_{max}(H_n) - \lambda_{min}(H_e), \end{aligned}$$

which gives the inequalities in part (a). No a priori bounds are provided for the value of η, so it is possible that $\beta - \eta < 0$. However, it is given that the Hessian is positive definite, so when $\beta - \eta < 0$ the lower bound is set to 0. The condition number in part (b) follows from applying the bounds in part (a) to the definition of the condition number. □

The results presented in Theorem 4.2.2 provide horizon-independent bounds for the extremal eigenvalues of the primal Hessian with S present. The results in Theorem 4.2.1 can provide the values for γ and β when $P = Q$ and for when P is the solution to the discrete-time Lyapunov equation. Horizon-independent values for η and ν can be computed using Lemma 4.2.2.

The results in Theorem 4.2.2 are conservative bounds on the spectrum. This can be seen in Figure 4.4, where the bound on λ_{min} is much lower than the actual computed eigenvalues. It is possible for the bound on the condition number to go to infinity if $\lambda_{max}(H_e) \geq \lambda_{min}(H_n)$, since then the lower bound on the eigenvalue will be 0 even though the actual Hessian H_{cS} remains positive definite.

An alternative method presented in [3] to handle a non-zero S matrix is to transform the problem into one with the system given by $(\tilde{A}, \tilde{B})$ and weight matrices $(\tilde{Q}, \tilde{R}, \tilde{S})$ with

$$\begin{aligned} \tilde{A} &:= A - BR^{-1}S', & \tilde{B} &= B, & \\ \tilde{Q} &:= Q - SR^{-1}S', & \tilde{R} &= R, & \tilde{S} &= 0. \end{aligned}$$

Note that the Schur-stability assumption must now hold for the transformed system $\tilde{A}$, which is not true in general.

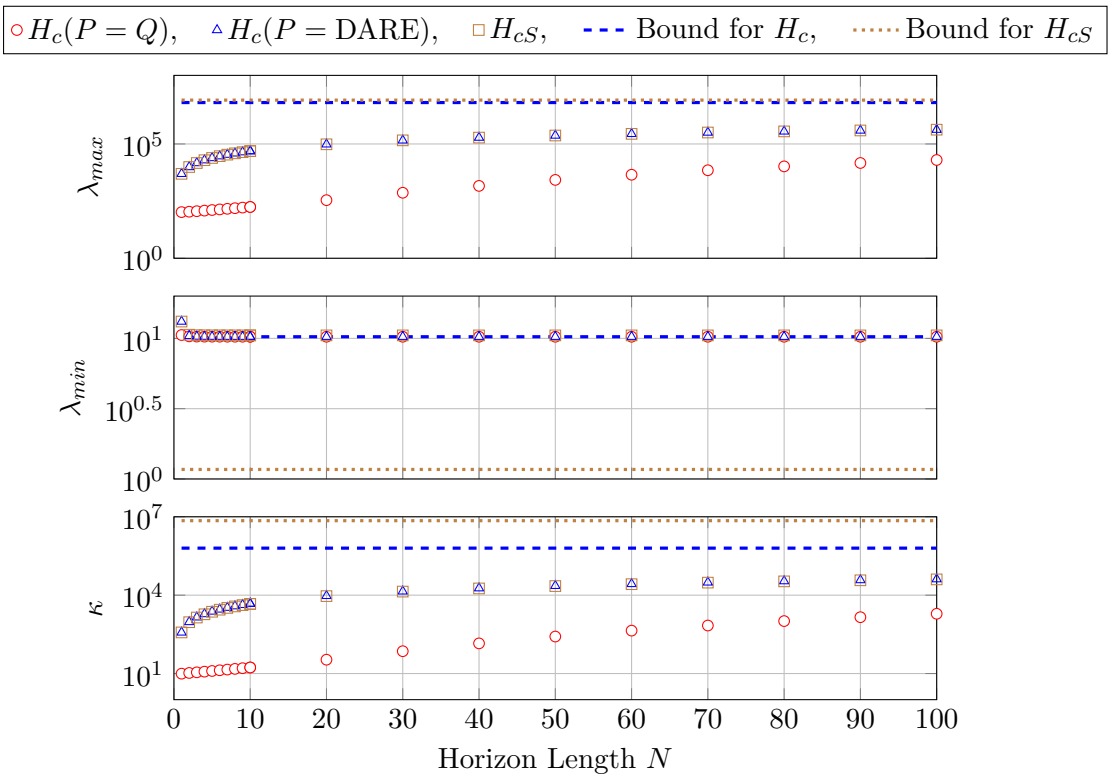


Figure 4.4: Spectral properties of the condensed primal Hessian for the mass-spring-damper system. The lines represent the bounds, and the markers represent the values of the condensed Hessian at that horizon.

4.3 Inequality constraint matrix

We now turn our focus to the constraints in (2.7) and show that the condensed constraint matrix is block Toeplitz. To begin, we must examine the shifted prediction matrix $\tilde{\Gamma}_c$ that is used in the numerically robust CLQR problem (2.10) to form the constraint matrix G_c. This shifted prediction matrix is formed by performing a shift on the original prediction matrix to move the contents of each diagonal onto a lower diagonal, which will keep the block Toeplitz structure of the prediction matrix but change the matrix symbol. The new symbol associated with $\tilde{\Gamma}_c$ is shown in Lemma 4.3.1.

Lemma 4.3.1. *Let the pair (A, B) be stabilizable and $K \in \mathbb{R}^{m\times n}$ be a linear state-feedback control matrix used to form the prestabilized system (2.9). The shifted prediction matrix $\tilde{\Gamma}_c$ then has the matrix symbol $\mathcal{P}_{\tilde{\Gamma}_c} \in \tilde{\mathcal{C}}_{2\pi}$ with*

$$\mathcal{P}_{\tilde{\Gamma}_c}(z) \coloneqq (zI - (A - BK))^{-1}B = \mathcal{G}_c(z), \quad \forall z \in \mathbb{T},$$

where $\mathcal{G}_c(\cdot)$ is the transfer function matrix for the discrete-time system $\mathcal{G}_c$.

Proof. We begin by expanding the multiplication

$$\tilde{\Gamma}_c = \begin{bmatrix} 0_{n\times Nm} \\ \begin{bmatrix} I_{n(N-1)} & 0 \end{bmatrix} \Gamma_c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ B & 0 & 0 & 0 & & 0 & 0 \\ A_cB & B & 0 & 0 & & 0 & 0 \\ A_c^2B & A_cB & B & 0 & & 0 & 0 \\ \vdots & & & & \ddots & \vdots & \vdots \\ A_c^{N-3}B & A_c^{N-4}B & A_c^{N-5}B & A_c^{N-6}B & \cdots & 0 & 0 \\ A_c^{N-2}B & A_c^{N-3}B & A_c^{N-4}B & A_c^{N-5}B & \cdots & B & 0 \end{bmatrix}.$$

From this, we can observe that the infinite-dimensional extension of $\tilde{\Gamma}_c$ (called $\tilde{\mathbf{\Gamma}}_c$) will have the blocks

$$\left[\tilde{\mathbf{\Gamma}}_c\right]_i \coloneqq \begin{cases} 0 & \text{if } i < 1, \\ (A - BK)^{i-1}B & \text{if } i \geq 1, \end{cases} \tag{4.12}$$

where i is the number of the block diagonal of the matrix, with $i = 0$ being the main diagonal and positive i below the main diagonal. Next, we use the trigonometric polynomial associated with $\tilde{\mathbf{\Gamma}}_c$ to form the matrix symbol $\mathcal{P}_{\tilde{\Gamma}_c}$. Using the blocks of the matrix given in (4.12) as the coefficients, the trigonometric polynomial is

$$\sum_{i=1}^{\infty} z^{-i}(A - BK)^{i-1}B.$$

By introducing the auxiliary variable $k = i - 1$, the starting point of the summation is shifted to 0, giving

$$\sum_{k=0}^{\infty} z^{-k-1}(A - BK)^{k}B.$$

By factoring the constants z^{-1} and B from the summation, the summation becomes

$$z^{-1}\left(\sum_{k=0}^{\infty} z^{-k}(A - BK)^{k}\right)B, \tag{4.13}$$

which is a Neumann series that converges to

$$z^{-1}\left(z(zI - (A - BK))^{-1}\right)B$$

when $A - BK$ is Schur-stable. Canceling the leading $z^{-1}z$ term and noting that the remaining $(zI - (A - BK))^{-1}B$ is the transfer function matrix of the discrete-time system $\mathcal{G}_c(z)$ then shows the matrix symbol for $\tilde{\mathbf{\Gamma}}_c$ is $\mathcal{P}_{\tilde{\Gamma}_c}(z) = \mathcal{G}_c(z)$. Finally, note that the coefficients in the sum (4.13) are absolutely summable, so $\mathcal{P}_{\tilde{\Gamma}_c}$ is in the Wiener class, meaning that $\mathcal{P}_{\tilde{\Gamma}_c} \in \mathcal{L}^{\infty}$ and is continuous and 2π-periodic, leading to $\mathcal{P}_{\tilde{\Gamma}_c} \in \tilde{\mathcal{C}}_{2\pi}$. □

We can now analyze the constraint matrix G_c for the numerically robust CLQR problem and show that G_c is a block Toeplitz matrix, with the matrix symbol given in Lemma 4.3.2.

Lemma 4.3.2. *Let $\mathcal{G}_c$ be a Schur-stable system with controller $K \in \mathbb{R}^{m\times n}$ and the shifted prediction matrix $\tilde{\Gamma}_c$. The condensed constraint matrix G_c is block Toeplitz with the matrix symbol $\mathcal{P}_{G_c} \in \tilde{\mathcal{C}}_{2\pi}$ with*

$$\mathcal{P}_{G_c}(z) \coloneqq (E^x - E^u K)\mathcal{P}_{\tilde{\Gamma}_c}(z) + E^u \qquad \forall z \in \mathbb{T},$$

where $\mathcal{P}_{\tilde{\Gamma}_c}(z)$ is the matrix symbol of $\tilde{\Gamma}_c$ given in Lemma 4.3.1.

Proof. We start with the fact that $G_c = (\bar{E}^x + \bar{E}^u \bar{K})\tilde{\Gamma}_c + \bar{E}^u$. Using the definitions of $\bar{E}^x$, $\bar{E}^u$ and $\bar{K}$ in Section 2.3.1, it is obvious they are block Toeplitz with symbols

$$\mathcal{P}_{\bar{E}^x}(z) \coloneqq E^x, \quad \mathcal{P}_{\bar{E}^u}(z) \coloneqq E^u, \quad \mathcal{P}_{\bar{K}}(z) \coloneqq -K \quad \forall z \in \mathbb{T},$$

respectively. The product $\bar{E}^u \bar{K}$ is then block Toeplitz since both $\bar{E}^u$ and $\bar{K}$ are block diagonal, with the matrix symbol of the product being $E^u K$. Since block Toeplitz structure is preserved over addition (Lemma 3.3.1), the sum $(\bar{E}^x + \bar{E}^u \bar{K})$ will be block Toeplitz and block diagonal (since both terms in the addition are both block Toeplitz and block diagonal). Then, since the product $(\bar{E}^x + \bar{E}^u \bar{K})\tilde{\Gamma}_c$ is the multiplication of a block diagonal block Toeplitz matrix with a block Toeplitz matrix, the resulting matrix will be block Toeplitz with the matrix symbol $(E^x - E^u K)\mathcal{P}_{\tilde{\Gamma}_c}$. Finally, since $\bar{E}^u$ is block Toeplitz, the final addition will result in a block Toeplitz matrix with the matrix symbol $\mathcal{P}_{G_c}$ given in the lemma. □

4.4 Dual Hessian

In this section, we derive some spectral properties of the dual condensed Hessian $H_d = GH_c^{-1}G'$ from (2.8) for two distinct cases:

1. H_c is arbitrary, and
2. H_c is block Toeplitz.

H_c is arbitrary

For problems where H_c is not block Toeplitz, the resulting dual condensed Hessian H_d will also not be block Toeplitz. This means that H_d will not have a matrix symbol or spectral bounds similar to those we found for H_c in Section 4.2.2. However, we can still place an upper bound on the spectrum of H_d using matrix norm inequalities.

Proposition 4.4.1. *Let G_c and H_c be the condensed constraint matrix and primal Hessian for the CLQR problem* (2.10)*, respectively. Then we can upper bound the largest eigenvalue of the dual condensed Hessian from* (2.8) *as*

$$\|H_d\|_2 \leq \frac{(\sigma_{max}(G_c))^2}{\lambda_{min}(H_c)}.$$

Proof. Combining the triangle inequality

$$\|H_d\|_2 \leq \|G_c\|_2 \|H_c^{-1}\|_2 \|G_c\|_2$$

with the fact that $\|H_c^{-1}\|_2 = \frac{1}{\lambda_{min}(H_c)}$ gives the result in the lemma. □

The result in Proposition 4.4.1 creates a loose upper bound for the spectrum of H_d using the maximum singular value of the constraint matrix G_c and the minimum eigenvalue of the primal condensed Hessian H_c. However, a non-zero lower bound for the spectrum of H_d does not in general exist, since H_d can be rank-deficient depending on the constraint set G_c.

Proposition 4.4.2. *Let $G_c \in \mathbb{R}^{Nl \times Nm}$ be the condensed constraint matrix for a horizon of length N, then* $\text{rank}(H_d) = \text{rank}(G_c)$.

Proof. Let H_c be the positive definite primal condensed Hessian from the CLQR problem (2.10). Recall that $H_d = G_c H_c^{-1} G_c'$. It is known from [21, Corollary 2.5.10] that for matrix multiplication of two matrices $A \in \mathbb{R}^{x \times y}$ and $B \in \mathbb{R}^{y \times z}$ with ranks a and b, respectively, the rank of the product AB is

$$a + b - y \leq \text{rank}(AB) \leq \min\{a, b\}. \tag{4.14}$$

We begin by examining the product $M \coloneqq G_c H_c^{-1}$, with $M \in \mathbb{R}^{N(j+l) \times Nm}$. Since H_c is positive definite, its inverse exists and is also full rank, meaning $\text{rank}(H_c^{-1}) = Nm$. Additionally, $\text{rank}(G_c) \leq Nm$ since one dimension of G_c is fixed at Nm. This means that both sides of (4.14) become $\text{rank}(G_c)$, making $\text{rank}(M) = \text{rank}(G_c)$. A similar process can be followed for the product MG_c', to get the final result. □

Proposition 4.4.2 shows that the rank of H_d is determined by the constraint set. If there are more constraints than inputs (i.e. $l > m$), then the dual Hessian will be rank deficient, and therefore be positive semidefinite.

H_c is Toeplitz

If the MPC problem (2.10) has a block Toeplitz Hessian (e.g. when P is chosen to be the solution to the DARE), then the eigenvalue distribution of H_d can be estimated from the eigenvalues of its matrix symbol. To do this, we first note that the dual condensed Hessian has the same non-zero eigenvalues as the matrix $H_{d1} \coloneqq H_c^{-1}G'G$.

Lemma 4.4.1. *Let H_c and G_c be the condensed Hessian and constraint matrices from the CLQR problem* (2.10)*. The non-zero eigenvalues of the dual condensed Hessian H_d in* (2.8) *are the same as the eigenvalues of $H_{d1} \coloneqq H_c^{-1}G_c'G_c$.*

Proof. This result follows directly from the fact that H_d can be viewed as the outer product uv' of the matrices $u \coloneqq G$ and $v \coloneqq H_c^{-1}G$. The non-zero eigenvalues of the outer product matrix uv' are the same as the eigenvalues of $v'u$ [71, Example 1.3.23] □

If H_c is limited to being a block Toeplitz matrix, then the spectrum of $\mathbf{H}_d$ can be bounded using a matrix symbol similar to the previous results for the primal condensed Hessian in Section 4.2.2.

Theorem 4.4.1. *Let the condensed primal Hessian H_c in* (2.10) *be block Toeplitz with the matrix symbol $\mathcal{P}_{H_c} \in \tilde{\mathcal{C}}_{2\pi}$, positive definite almost everywhere, then*

$$\lambda_{max}(\mathbf{H}_d) \leq \lambda_{max}(\mathcal{P}_{H_{d1}}) \leq \|\mathcal{P}_{H_{d1}}\|_{H_\infty}$$

where $\mathcal{P}_{H_{d1}}(z) \coloneqq (\mathcal{P}_{H_c}(z))^{-1}\mathcal{P}_{G_c}(z)^*\mathcal{P}_{G_c}(z)$ with $z \in \mathbb{T}$.

Proof. The product of the infinite block Toeplitz matrices $\mathbf{G}_c'\mathbf{G}_c$ is block Toeplitz based on Lemma 3.3.3 and the fact that $\mathbf{G}_c$ is lower triangular. Results in [125, Theorem 4.3] state that if $p, f \in \mathcal{L}_{2\pi}^{\infty}$ are the matrix symbols for the Toeplitz matrices $\mathbf{P}, \mathbf{F}$, respectively, then the eigenvalues of $\mathbf{P}^{-1}\mathbf{F}$ lie inside $[\lambda_{min}(p^{-1}f), \lambda_{max}(p^{-1}f)]$. This result, combined with Lemma 4.4.1 and $\lambda_{max} \leq \sigma_{max}$, gives the upper bound. □

Using the results in Theorem 4.4.1, the spectrum of $\mathbf{H}_d$ can be bounded using the matrix symbol of H_c provided that H_c is block Toeplitz. Unfortunately, the computation of λ_{max} for the symbol $\mathcal{P}_{H_{d1}}$ requires an exhaustive search over the unit circle to find the largest eigenvalue since H_{d1} is not symmetric. Instead, an upper bound on λ_{max} can be found using the H_∞ norm of $\mathcal{P}_{H_{d1}}$, which is a faster operation. Figure 4.5 shows $\|\mathcal{P}_{H_{d1}}\|_{H_\infty}$ and the asymptotic properties of λ_{max} for the first Schur-stable system in Section 2.6 with either only input constraints, only state constraints, or both input and state constraints. Note that the bound $\lambda_{max}(\mathbf{H}_d) \leq \lambda_{max}(\mathcal{P}_{H_{d1}})$ in Theorem 4.4.1 is tight, so there is a horizon above which

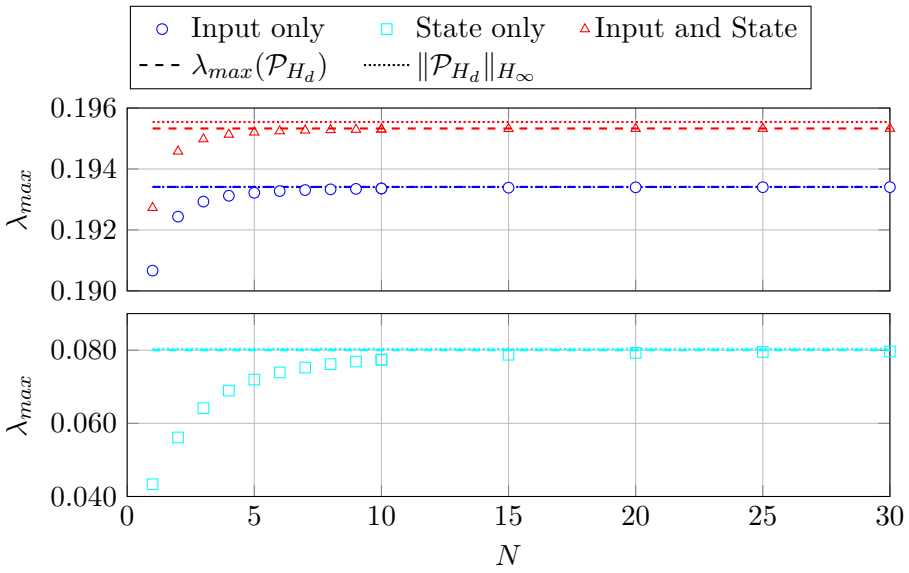


Figure 4.5: Maximum eigenvalue for the condensed dual Hessian of the first Schur-stable system. The lines represent the bounds computed using the results in Section 4.4, and the markers represent the values of the dual Hessian at that horizon.

equality occurs. Theorem 4.4.1 also provides better bounds than the 2-norm estimate from [139], which estimates that $\lambda_{max}(\mathcal{P}_{H_d})$ is less than 0.41, 1.65, and 2.06 for input, state and both input and state constraints, respectively.

4.5 Conclusions

In this chapter, we presented horizon-independent spectral bounds for many of the matrices contained in the CLQR problem, including the primal and dual condensed Hessians, the condensed prediction matrix and the condensed constraint matrix. These bounds provide several advantages in the implementation of first-order methods, including

- guaranteeing fixed-point hardware designs are compatible with any horizon length desired (removing the need to re-synthesize the designs if the prediction horizon were to be changed and allowing run-time variation of the horizon length e.g. for variable-horizon controllers), and
- providing horizon-independent upper iteration bounds for numerical algorithms.

We note that similar results to the ones in Sections 4.1 and 4.2 were also reported in [153] and [54, Sect. 11], but the analysis in this chapter differs in several ways. Specifically, the prior work requires that the discrete-time LTI system $\mathcal{G}$ has no eigenvalues on the unit circle, and that the system be transformed into a system with separable stable and unstable modes so

that they may be handled separately. Instead, the analysis presented in this chapter examined both the stable and unstable modes of $\mathcal{G}$ at the same time, and supports eigenvalues on the unit circle by using the numerically robust formulation described in Section 2.3.2. Additionally, the prior work assumes that $Q = C'C$ (where C is the output state-mapping matrix of the system), whereas the analysis in this chapter allows for an arbitrary positive-definite Q and a P matrix chosen as either Q or the solution to the DARE (2.3).

This chapter also highlighted the relationship between the transfer function and the spectrum of the condensed Hessian by showing that the extrema of the condensed Hessian's spectrum is bounded by the extrema of the spectrum for a complex-valued matrix symbol formed using the weighting matrices and the transfer function matrix of the predicted system. This relation will be further extended in Chapter 6 to show that the preconditioned primal condensed Hessian can also have its spectrum bounded in a similar manner.

Chapter 5

Computational complexity and system performance

In this chapter, we examine how changing the parameters in the CLQR cost function (specifically scaling the weighting matrices) affects the extremal eigenvalues and condition number of the condensed Hessian that we derived in Chapter 4 and the computational complexity of the FGM and DGP algorithms. The trade-off between the performance of the controller (which is dictated by the matrices in the cost function) and the computational complexity required is an important item for designers to consider, since performance requirements are usually given as bounds (e.g. "settling-time less than 1s" or "track this signal with less than 5% error") rather than an exact criterion, creating a space of possible controllers that can satisfy the requirements. These controllers may have different computational resource demands, opening up the opportunity to trade off the computation with the system performance.

Note that in this chapter we present the results assuming the Hessian for the condensed CLQR problem (2.7) with a Schur-stable system and P chosen as either Q or the solution of the discrete Lyapunov equation (2.4) is used.

5.1 Preliminaries

The bounds on the eigenvalues and condition number we will develop in this section utilize the matrix trace normalized against the matrix size. To allow for horizon-independent bounds, we show that these trace normalizations can be computed in a horizon-independent manner through Lemma 5.1.1.

Lemma 5.1.1. *Let H be the primal Hessian of size $mN \times mN$ from Section 2.3.1 with P the solution to the discrete Lyapunov equation (2.4) and S an arbitrary $n \times m$ matrix. Let the system $\mathcal{G}$ be Schur-stable with m inputs. Define the following two dynamical systems*

$$\mathcal{G}_Q(z) \coloneqq \left[\begin{array}{c|c} A & B \\ \hline Q^{1/2} & 0 \end{array}\right],$$
$$\mathcal{G}_{QR}(z) \coloneqq \left[\begin{array}{c|c} A & BR^{1/2} \\ \hline Q^{1/2} & 0 \end{array}\right],$$

and let $I_k \coloneqq \frac{1}{2\pi}\int_0^{2\pi} \mathcal{F}_k(e^{j\omega})\,\mathrm{d}\omega$ where

$$\begin{aligned}
\mathcal{F}_1(z) &\coloneqq \operatorname{Tr}\left(S'\mathcal{P}_\Gamma(z)\right), & \mathcal{F}_2(z) &\coloneqq \operatorname{Tr}\left(RS'\mathcal{P}_\Gamma(z)\right),\\
\mathcal{F}_3(z) &\coloneqq \operatorname{Tr}\left(S'\mathcal{P}_\Gamma(z)S'\mathcal{P}_\Gamma(z)\right), & \mathcal{F}_4(z) &\coloneqq \operatorname{Tr}\left(S'\mathcal{P}_\Gamma(z)\mathcal{P}_\Gamma^*(z)S\right),\\
\mathcal{F}_5(z) &\coloneqq \operatorname{Tr}\left(\mathcal{P}_\Gamma^*(z)Q\mathcal{P}_\Gamma(z)S'\mathcal{P}_\Gamma(z)\right), & \mathcal{F}_6(z) &\coloneqq \operatorname{Tr}\left(\mathcal{P}_\Gamma^*(z)Q\mathcal{P}_\Gamma(z)\mathcal{P}_\Gamma^*(z)Q\mathcal{P}_\Gamma(z)\right).
\end{aligned}$$

Then the quantities

$$a_N \coloneqq \frac{\operatorname{Tr}(H)}{mN}, \quad b_N \coloneqq \frac{\operatorname{Tr}(H^2)}{mN},$$

have limits as $N \to \infty$ of

$$\begin{aligned}
a_l &\coloneqq \lim_{N\to\infty} a_N = \frac{1}{m}\left(\|\mathcal{G}_Q\|_{H_2}^2 + 2I_1 + \|R^{1/2}\|_F^2\right),\\
b_l &\coloneqq \lim_{N\to\infty} b_N = \frac{1}{m}(I_6 + 4I_5 + 4I_2 + 2I_3 + 2I_4 + \|R\|_F^2 + 2\|\mathcal{G}_{QR}\|_{H_2}^2).
\end{aligned}$$

Proof. Since S is arbitrary, we use the matrix splitting (4.2.3) and substitute in the nominal and correction matrices to form

$$H = H_Q + H_P + H_R + H_n - H_e.$$

We begin by computing the limit

$$a_l \coloneqq \lim_{N\to\infty} \frac{1}{mN}\operatorname{Tr}(H). \tag{5.1}$$

Note that according to Lemma 4.2.2, H_e has a finite rank independent of its size, so the contribution of H_e to the trace of H will go to zero as the matrix size approaches infinity. This removes the H_e term from the expansion of (5.1), giving

$$a_l = \lim_{N\to\infty} \frac{1}{mN}\operatorname{Tr}(H_Q + H_P + H_R + H_n), \tag{5.2}$$

which contains only the block Toeplitz components of H. Using Szegö's limit theorem for block Toeplitz matrices of size $n \times n$ (Theorem 3.4.2), the limit as $n \to \infty$ of a function f of the eigenvalues of a block Toeplitz matrix can be transformed into a definite integral of f applied to the eigenvalues of the matrix symbol. Since the trace is the sum of the eigenvalues, (5.2) becomes

$$a_l = \frac{1}{2m\pi} \int_0^{2\pi} \operatorname{Tr}\left(\mathcal{P}_{H_n}(e^{j\omega})\right) \mathrm{d}\omega,$$

where $\mathcal{P}_{H_n}$ is given by (4.11). Expanding this and using the additive property of the matrix trace, we get

$$a_l = \frac{1}{2m\pi}\left(\int_0^{2\pi} \operatorname{Tr}\left(\mathcal{P}_\Gamma(e^{j\omega})^* Q \mathcal{P}_\Gamma(e^{j\omega})\right) + \operatorname{Tr}(R) + 2\operatorname{Tr}\left(S'\mathcal{P}_\Gamma(e^{j\omega})\right) \mathrm{d}\omega\right).$$

Using the definition of the H_2 norm and the Frobenius norm, a_l then becomes

$$a_l = \frac{1}{m}\left(\|\mathcal{G}_Q\|_{H_2}^2 + 2I_1 + \|R^{1/2}\|_F^2\right).$$

Computing b_l can be done in a similar manner to a_l. Note that

$$\operatorname{rank}(AB) \leq \min\{\operatorname{rank}(A), \operatorname{rank}(B)\},$$

meaning anything multiplied by H_e will have finite rank as the matrix size goes to infinity. This means those terms will go to zero in the limit, leaving only the block Topelitz component H_n

$$b_l = \lim_{N\to\infty} \frac{1}{mN} \operatorname{Tr}\left((H_n)^2\right). \tag{5.3}$$

Applying the Szegö limit theorem to (5.3) then gives

$$b_l = \frac{1}{2m\pi} \int_0^{2\pi} \operatorname{Tr}\left((\mathcal{P}_{H_n}(e^{j\omega}))^2\right) \mathrm{d}\omega$$

Expanding the integrand, and simplifying using properties of the trace and the definition of the H_2 and Frobenius norms produces the final result. □

Unlike the results in Section 4.2 where we had to separate the discussion of the condensed primal Hessian into two parts, with and without a state-input cross-term matrix S, the result given in Lemma 5.1.1 holds for both cases. This occurs because the normalized trace of a finite rank matrix (such as H_e) goes to zero as its size goes to infinity, which leaves only the Toeplitz component of the Hessian. Results similar to Lemma 5.1.1 could also be derived for other values of P, provided that H can be decomposed into a Toeplitz component plus a finite-rank correction term.

To analyze the effect of scaling the weight matrices, we can utilize the linearity of the trace to scale the various terms in Lemma 5.1.1, giving the following result.

Lemma 5.1.2. *Let $\hat{Q} \coloneqq \alpha_1 Q$, $\hat{R} \coloneqq \alpha_2 R$ and $\hat{S} \coloneqq \alpha_3 S$ be scaled versions of the weight matrices. Then if the scaled matrices are substituted into the terms used in Lemma 5.1.1, the terms scale as*

$$\begin{aligned} \hat{I}_1 &\coloneqq \alpha_3 I_1, & \hat{I}_2 &\coloneqq \alpha_2\alpha_3 I_2, & \hat{I}_3 &\coloneqq \alpha_3^2 I_3, \\ \hat{I}_4 &\coloneqq \alpha_3^2 I_4, & \hat{I}_5 &\coloneqq \alpha_1\alpha_3 I_5, & \hat{I}_6 &\coloneqq \alpha_1^2 I_6, \end{aligned}$$

$$\begin{aligned} \|\mathcal{G}_{\hat{Q}}\|_{H_2}^2 &\coloneqq \alpha_1 \|\mathcal{G}_Q\|_{H_2}^2, & \|\hat{R}\|_F^2 &\coloneqq \alpha_2^2 \|R\|_F^2, \\ \|\mathcal{G}_{\hat{Q}\hat{R}}\|_{H_2}^2 &\coloneqq \alpha_1\alpha_2 \|\mathcal{G}_{QR}\|_{H_2}^2, & \|\hat{R}^{1/2}\|_F^2 &\coloneqq \alpha_2 \|R^{1/2}\|_F^2. \end{aligned}$$

Proof. These results follow from the linearity of the trace and the integral operator. □

5.2 Extremal eigenvalues

In the complexity analysis of some algorithms such as the FGM and DGP, the extremal eigenvalues of the Hessian (both dual and primal) appear as a factor. This means that in order to understand how the computational complexity changes with the weight matrices, it is important to understand how the weighting matrices affect the extremal eigenvalues.

5.2.1 Primal Hessian

We begin by deriving bounds on the extremal eigenvalues for the primal Hessian.

Lemma 5.2.1. *Let H be the primal Hessian from* (2.7)*. Then*

$$0 < \lambda_{min}(H) \leq a_l \leq \lambda_{max}(H)$$

where a_l is defined in Lemma 5.1.1.

Proof. Bounds on the extremal eigenvalues were given in [164, Theorem 2.1] as

$$\begin{aligned} a_N - s_N p_N \leq \lambda_{min} \leq a_N - {}^{s_N}\!/_{p_N} \\ a_N - {}^{s_N}\!/_{p_N} \leq \lambda_{max} \leq a_N + s_N p_N \end{aligned}$$

where $s_N \coloneqq \sqrt{b_N - a_N^2}$ and $p_N \coloneqq \sqrt{mN-1}$ with a_N and b_N given in Lemma 5.1.1. The limit of these bounds as $N \to \infty$ gives a finite upper bound for λ_{min} and a finite lower bound for λ_{max}, both equal to a_l. The upper bound on λ_{max} is in general not finite, and the lower bound for λ_{min} is in general strictly greater-than 0 since H is positive definite. □

Equality can occur when $a_N^2 = b_N$ (i.e. $\mathrm{Tr}\,(H) = \|H\|_F^2$), since that makes $s = 0$. Since H is positive definite though, this can only occur when all eigenvalues of H are 1.

The bound in Lemma 5.2.1 essentially creates a dividing line between the extremal eigenvalues. The effect of the weight matrix scaling on this dividing line can then be estimated, as done in Theorem 5.2.1.

Theorem 5.2.1. *Let $\hat{H}$ be the primal Hessian with the scaled weight matrices $\hat{Q} \coloneqq \alpha_1 Q$, $\hat{R} \coloneqq \alpha_2 R$ and $\hat{S} \coloneqq \alpha_3 S$. Then, the bound on the extremal eigenvalues given in Lemma 5.2.1 grows linearly with α_1, α_2 and α_3.*

Proof. Combine the bounds in Lemma 5.2.1 with the scalings in Lemma 5.1.2. □

From Theorem 5.2.1 it can be seen that the bounds on the extremal eigenvalues grow linearly with the scaling of the weight matrices. This is demonstrated in Figures 5.1a and 5.1b for the case when $S = 0$, leaving only α_1 and α_2. When only one matrix is being scaled, there are two distinct regions in the bound: Q dominating and R dominating. Since this bound is both an upper bound for λ_{min} and a lower bound for λ_{max}, the transition between the two regions when α_1 is being increased will occur earlier for λ_{max} than λ_{min}. The region where R dominates when only Q is scaled is shown as a shaded region in Figures 5.1a and 5.1b.

5.2.2 Dual Hessian

For the dual Hessian, the largest eigenvalue can be bounded through the spectral norm as was done in Proposition 4.4.1. This bound is affected by both the spectrum of the constraint matrix G and the primal condensed Hessian H. When the weight matrices are scaled, only the primal condensed Hessian is affected, which means only the $\lambda_{min}(H)$ term in the denominator will change. Unfortunately, since the lower bound in Theorem 5.2.1 is zero, an upper bound on $\|H_d\|_2$ cannot be computed directly.

We can instead use the upper bound for λ_{min} given in Lemma 5.2.1 to examine how the bound from Proposition 4.4.1 changes with weight scaling. It is known from Theorem 5.2.1 that there is a linear relation between the magnitude of the cost matrices and the upper bound for λ_{min}. This implies that there will be an inverse relation between the cost scaling and λ_{max} of the dual Hessian, since as either α_1, α_2 or α_3 grow, the bound on $\lambda_{max}(H)$ grows as well.

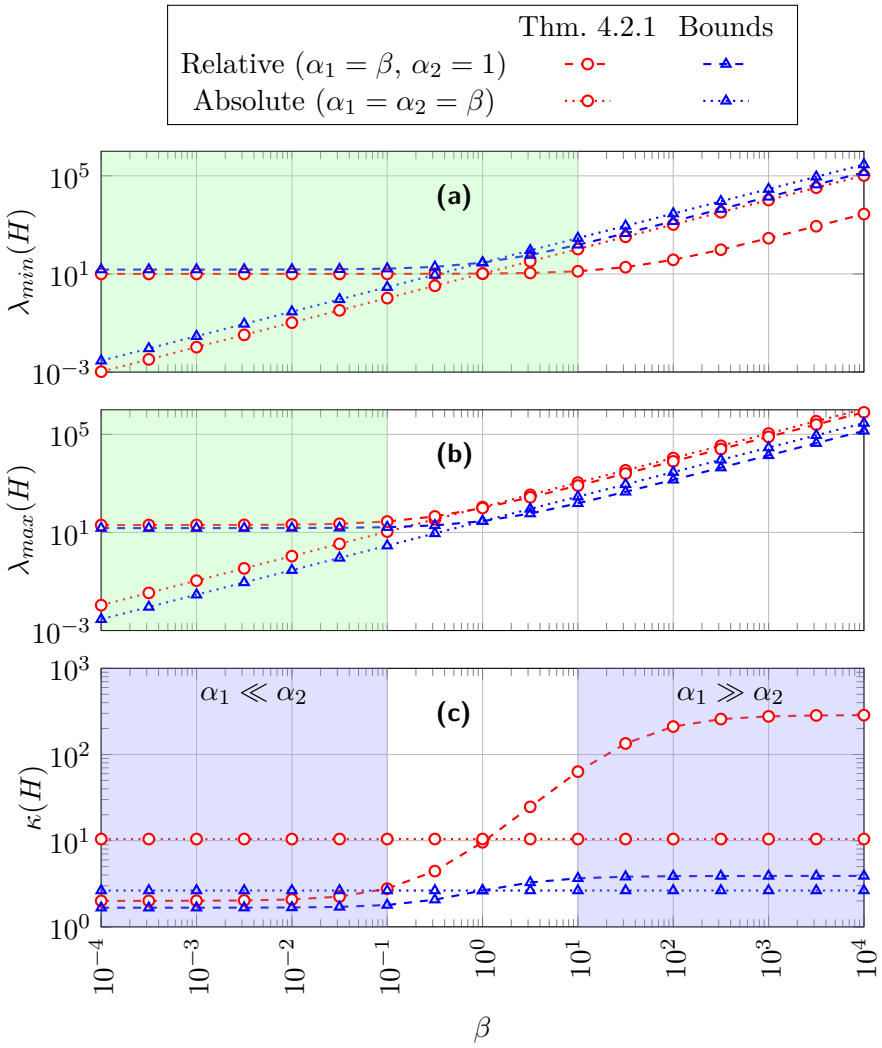


Figure 5.1: Horizon-independent extremal eigenvalue and condition number bounds from Lemma 5.2.1 and Theorem 5.3.1, respectively, for various scalings of Q and R matrices in the first Schur-stable system from Section 2.6.

5.3 Condition number

We now focus our analysis on the primal Hessian H, since the condition number of the dual Hessian H_d is in general unbounded due to H_d being only positive semidefinite and not positive definite.

5.3.1 Discrete-time weights

The results presented in Section 4.2 provide a means of calculating the condition number estimates for a given set of weighting matrices, but provide little intuition into how scaling the weighting matrices will affect the condition number. To examine the effect of scaling, we utilize a lower bound for the condition number of the Hessian H.

Lemma 5.3.1. *Let a_l and b_l be defined in Lemma 5.1.1. The primal Hessian H has a lower bound on the condition number given by*

$$1 + 2\frac{\sqrt{b_l - a_l^2}}{a_l} \leq \kappa(H)$$

Proof. We begin with the lower bound for the condition number of a matrix of dimension mN presented in [164, Corollary 2.3]:

$$1 + \frac{2s_N}{a_N - \frac{s_N}{\sqrt{mN-1}}} \leq \kappa_N(H) \tag{5.4}$$

with $s_N \coloneqq \sqrt{b_N - a_N^2}$, and a_N and b_N from Lemma 5.1.1.

To determine the asymptotic bound, we take the limit of (5.4) to find

$$\lim_{N\to\infty} 1 + \frac{2s_N}{a_N - \frac{s_N}{\sqrt{mN-1}}} = 1 + 2\frac{\sqrt{b_l - a_l^2}}{a_l}.$$

□

The lower bound presented in Lemma 5.3.1 is horizon-independent, and holds for any choice of S and either $P = Q$ or P the solution to the discrete Lyapunov equation (2.4). This lower bound represents the best possible condition number that can be obtained. Unfortunately, knowledge of the worst possible condition number (i.e. an upper bound) cannot be obtained in a horizon-independent manner since lower-bounds on the smallest eigenvalue of H go to 0 as the matrix size increases.

To more closely examine the effect of the matrix scaling, we examine the case when $S = 0$ and only Q and R are scaled. We assume that P is either Q or the solution to the discrete Lyapunov equation (2.4). For this case, the lower bound from Lemma 5.3.1 becomes the bound given in Theorem 5.3.1.

Theorem 5.3.1. *Let $\hat{H}$ be the Hessian with the scaled weight matrices $\hat{Q} \coloneqq \alpha_1 Q$, $\hat{R} \coloneqq \alpha_2 R$ and $S = 0$. Then given the dynamical systems $\mathcal{G}_Q$ and $\mathcal{G}_{QR}$ and the integrals in Lemma 5.1.1, a lower bound for the condition number of $\hat{H}$ is*

$$1 + 2\frac{\sqrt{\alpha_1^2\gamma_1 + 2\alpha_1\alpha_2\gamma_2 + \alpha_2^2\gamma_3}}{\alpha_1\|\mathcal{G}_Q\|_{H_2}^2 + \alpha_2\|R^{1/2}\|_F^2} \leq \kappa(\hat{H}),$$

with

$$\gamma_1 \coloneqq mI_6 - \|\mathcal{G}_Q\|_{H_2}^4, \quad \gamma_3 \coloneqq m\|R\|_F^2 - \|R^{1/2}\|_F^4,$$
$$\gamma_2 \coloneqq m\|\mathcal{G}_{QR}\|_{H_2}^2 - \|\mathcal{G}_Q\|_{H_2}^2\|R^{1/2}\|_F^2.$$

Proof. Starting with the lower bound from Lemma 5.3.1, we define $v_l \coloneqq b_l - a_l^2$ (i.e. the term under the square root in the numerator of the bound in Lemma 5.1.2). Substituting in the norm scalings from Lemma 5.1.2 into a_l gives

$$a_l = \alpha_1\|\mathcal{G}_Q\|_{H_2}^2 + \alpha_2\|R^{1/2}\|_F^2,$$

which is the final version of the denominator.

Substituting in the norm scalings from Lemma 5.1.2 into v_l and expanding the square gives

$$v_l = m\left(\alpha_1^2 I_6 + 2\alpha_1\alpha_2\|\mathcal{G}_{QR}\|_{H_2}^2 + \alpha_2^2\|R\|_F^2\right) - \alpha_1^2\|\mathcal{G}_Q\|_{H_2}^4 - \alpha_1\alpha_2\|\mathcal{G}_Q\|_{H_2}^2\|R^{1/2}\|_F^2 - \alpha_2^2\|R^{1/2}\|_F^4$$

Grouping the terms by the α_1 and α_2 coefficients leads to

$$v_l = \alpha_1^2\left(mI_6 - \|\mathcal{G}_Q\|_{H_2}^4\right) + \alpha_2^2\left(m\|R\|_F^2 - \|R^{1/2}\|_F^4\right) + 2\alpha_1\alpha_2\left(m\|\mathcal{G}_{QR}\|_{H_2}^2 - \|\mathcal{G}_Q\|_{H_2}^2\|R^{1/2}\|_F^2\right)$$

which gives the final version of the numerator. □

A numerical example for the results in Theorem 5.3.1 is presented for the first Schur-stable system from Section 2.6 in Figure 5.1c. Examining the behaviour of the bound, it can be seen that there exist three distinct regions (shown through shading in Figure 5.1c): when $\alpha_1 \ll \alpha_2$, when $\alpha_1 \gg \alpha_2$, and the transition region. The lower bounds for the regions when $\alpha_1 \ll \alpha_2$ and $\alpha_1 \gg \alpha_2$ can be estimated through Corollary 5.3.1.

Corollary 5.3.1. *The lower bound in Theorem 5.3.1 has asymptotic values*

$$\kappa(\hat{H}) \geq \begin{cases} 1 + 2\frac{\sqrt{mI_6 - \|\mathcal{G}_Q\|_{H_2}^4}}{\|\mathcal{G}_Q\|_{H_2}^2} & \text{if } \alpha_1 \gg \alpha_2 \\ 1 + 2\frac{\sqrt{m\|R\|_F^2 - \|R^{1/2}\|_F^4}}{\|R^{1/2}\|_F^2} & \text{if } \alpha_1 \ll \alpha_2 \end{cases}$$

When $\alpha_1 \ll \alpha_2$, the spectrum of R dominates the condition number. The transition region is caused by the fact that λ_{max} begins growing before λ_{min} when α_1 is scaled, causing their ratio to change. Then when λ_{min} also begins growing, the ratio becomes constant leading to the region where $\alpha_1 \gg \alpha_2$. In this region, the condition number is dominated by the singular value distribution of the dynamical system with an output mapping through the weighting matrix Q.

The bounds in these regions are related to the spread and mean of the spectrum of the matrices/system. The quantity in the numerators of Corollary 5.3.1 can be viewed as an upper bound on the spread of the spectrum (the largest distance between two eigenvalues) [126], while the denominator can be viewed as the mean of the spectrum. We can use this relation to see that for large ratios of R to Q, the condition number is dominated by the spread of the spectrum of R over its average. Alternatively, for large Q to R ratios the bound is dominated by the spread of the singular values of the physical system with an output compensator of $Q^{1/2}$ over the average of the singular values.

Another interesting phenomenon arises when both Q and R are scaled by the same amount.

Corollary 5.3.2. *If the relative scaling of the two weight matrices is held constant at $\alpha_1 = \eta\alpha_2$ for a constant $\eta > 0$, then the lower bound is constant with the value*

$$1 + 2\frac{\sqrt{\eta^2\gamma_1 + 2\eta\gamma_2 + \gamma_3}}{\eta\|\mathcal{G}_Q\|_{H_2}^2 + \|R^{1/2}\|_F^2} \leq \kappa(\hat{H}).$$

Essentially, if both Q and R are scaled equally (i.e. $\alpha_1 = \alpha_2$), then the condition number of the Hessian does not change. This is also true when $S \neq 0$ and is scaled equally with Q and R (i.e. $\alpha_1 = \alpha_2 = \alpha_3$), so it is only when the matrices are scaled separately that the condition number changes.

5.3.2 Discretization of continuous-time weight matrices

The results in Section 5.3 examine the effect of the scaling of the weights in the discrete-time problem (2.7). Alternatively, the weighting matrices Q, R and S can be computed from the continuous-time problem through the discretization given in (2.6).

When using this discretization method, scaling the weighting matrices in the continuous-time problem by $\hat{Q}^c := \alpha_1 Q^c$ and $\hat{R}^c := \alpha_2 R^c$ then leads to the following scalings for the resulting discrete-time matrices Q, R, and S

$$\hat{Q} = \alpha_1 Q, \quad \hat{S} = \alpha_1 S, \quad \hat{R} = \alpha_1 R + (\alpha_2 - \alpha_1)\tau R^c$$

Note that scaling α_1 affects all three weighting matrices, but α_2 only affects $\hat{R}$.

This scaling behaves very similarly to the scaling of the discrete-time matrices in Section 5.3. When $\alpha_1 = \alpha_2$, all matrices are scaled equally and the results from Corollary 5.3.2 say that the condition number will not change.

5.4 Computational complexity

Now we examine the change in computational complexity for the DGP and FGM algorithms when applied to the CLQR problem with scaled weight matrices. We specifically focus on the relative scaling case, which means that R is held constant and Q is scaled by $\alpha_1 \in [10^{-4}, 10^{6}]$.

5.4.1 Fast Gradient Method

When the upper bounds for Δ and ϵ in (2.14) are used in the UIB for the Fast Gradient Method, the UIB becomes dependent only on κ. Further simplification shows that $a \geq b > 0$ for $\kappa > 1$, which leads to

$$\text{UIB} = \lceil 2\sqrt{\kappa} - 2 \rceil. \tag{5.5}$$

The square-root dependence of (5.5) on κ means that the UIB will follow the same general trend as the condition number. Corollary 5.3.1 can then be used to investigate when the UIB for FGM will be small. For instance, Corollary 5.3.1 suggests that κ will be smaller if $\alpha_2 \gg \alpha_1$ (i.e. the inputs are more heavily weighted than the states). This will then lead to a smaller iteration bound for the FGM when the inputs are more heavily weighted, which is seen in Figure 5.2a on the first Schur-stable example from Section 2.6.

The performance of the CLQR controller using the FGM was examined by controlling the Schur-stable system without the state constraints and with a chosen suboptimality level of $\delta_{max} = 0.001$ and $N = 20$. The CLQR regulates the system starting from the initial condition of $x_0 = [0.1, 0.1, 0.1, 0.1]'$ to the origin. The performance was measured by taking the 2-norm of the state and input trajectories. To compare the performance at each scaling factor, the percent difference versus $\alpha_1 = 10^{-4}$ was computed, and is shown in Figures 5.2b and 5.2c.

For this problem, the overall change in the 2-norm of the state trajectories across the entire scaling range was less than 5%, while the input norm varied by approximately 200%. Additionally, the number of iterations required varied by 188% across the scaling range. An interesting feature of this is that the change in the norm of the input occurs at a different scaling than the change in the UIB. These results show that if the performance of the system states was the design criteria, an aggressive weighting will only produce a 5% decrease in the norm of the state trajectories, but will produce a 188% increase in the number of iterations required for the solver.

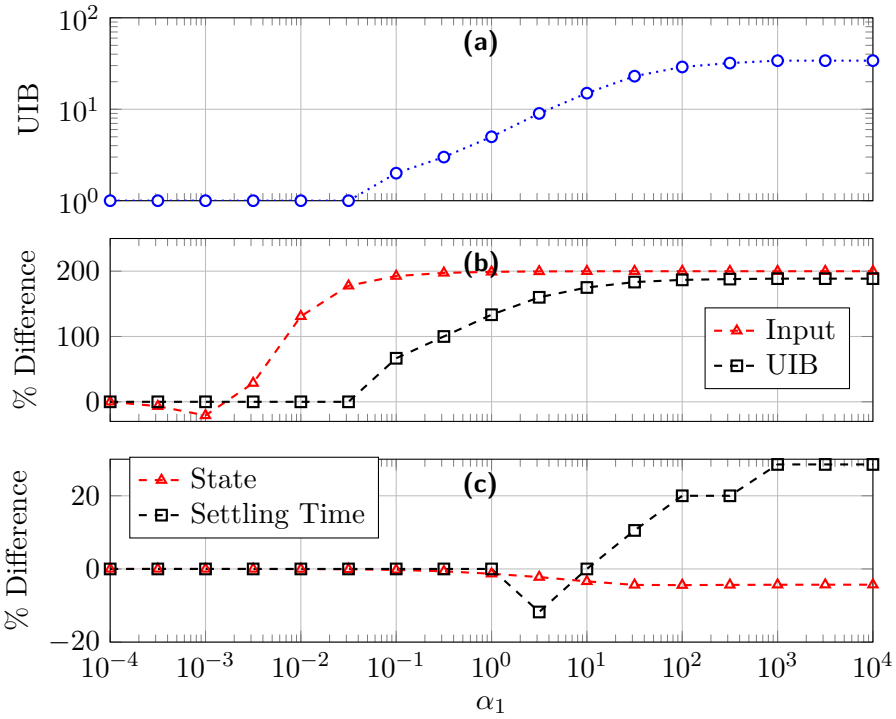


Figure 5.2: Effect of scaling the weight matrix Q by α_1 and holding R constant on the FGM when solving the first Schur-stable system from Section 2.6 with no state constraints.

5.4.2 Dual Gradient Projection

The UIB for the DGP given in (2.15) depends on the largest eigenvalue of both the primal and dual condensed Hessians, as well as the Upper Dual Bound (UDB) from Section 2.5.2. All three of these quantities depend on the horizon length chosen; but while a horizon-independent bound on the UDB is not known, the eigenvalues can be bounded using the results in Section 4.4. Examining (2.15), λ_{max} of the primal Hessian only affects the UIB when a suboptimal solution to the dual problem is requested (i.e. $\epsilon_z \neq 0$), while λ_{max} of the dual Hessian has a linear effect and the UDB has a quadratic effect on the UIB.

The numerical examples presented in Figures 5.3 and 5.4 use $N = 20$ with $\epsilon_z = \epsilon_\xi = 0$ (i.e. solve the dual exactly and with no computation error) while $\epsilon_g = 10^{-4}$ and $\epsilon_V = 10^{-2}$. The UDB for the Schur-stable system is shown in Figure 5.3b and is calculated by solving the Mixed Integer Linear Program (MILP) given in [131] using CPLEX with indicator constraints to implement the binary variables. The UDB for the mass-spring-damper system was upper bounded as D=20000 for all scaling factors by solving the same MILP, but terminating computation early and using the best objective value as the upper bound. As shown in Figure 5.3c, the tight bounds from Theorem 4.4.1 decrease the UIB by an order of magnitude compared with the estimate of λ_{max} given in [139].

The UIB for the DGP is affected by both the UDB and the maximal eigenvalues as the weight matrices are scaled. An interesting feature that appears when the exact UDB is used with the Schur-stable system is the interplay between D and $\lambda_{max}(H_d)$; specifically the slight peak in the UIB around $\alpha_1 = 100$ before it drops off again. The UDB though appears to have an asymptotic structure at the two extremes for α_1, while $\lambda_{max}(H_d)$ is decreasing as α_1 increases. A tradeoff in the closed-loop performance and computational complexity is also evident for the DGP, with the Schur-stable system showing a 200% increase in the UIB for a 200% decrease in the input norm and 5% increase in the state norm.

5.5 Conclusions

In this chapter, we explored the trade-offs between computational performance and the closed-loop performance of a system. We approached this through the lens of how the weight matrix selection (representing the desired system performance) affects the computational complexity bounds for the Fast Gradient Method and Dual Gradient Projection method.

We showed that the complexity bounds for FGM and DGP demonstrate distinct regions where the bound is influenced by the spectrum of the individual weighting matrices. With one region being influenced primarily by the weights applied to the inputs (the R matrix), and the other being influenced by the weights applied to the states (the Q matrix) and the singular values of the predicted system. Additionally, we showed that the complexity bounds of FGM and DGP behave differently when the cost function is scaled; the FGM bound increased as Q dominated while the DGP bound decreased as Q dominated. This suggests that including the algorithm as a design parameter that is optimized alongside the system performance could benefit the overall system design.

We demonstrated that the trade-off between computational resources and control performance is a worthwhile area to explore; with an example system showing that a 5% reduction in the state 2-norm requires a 188% increase in the computational complexity of the control algorithm.

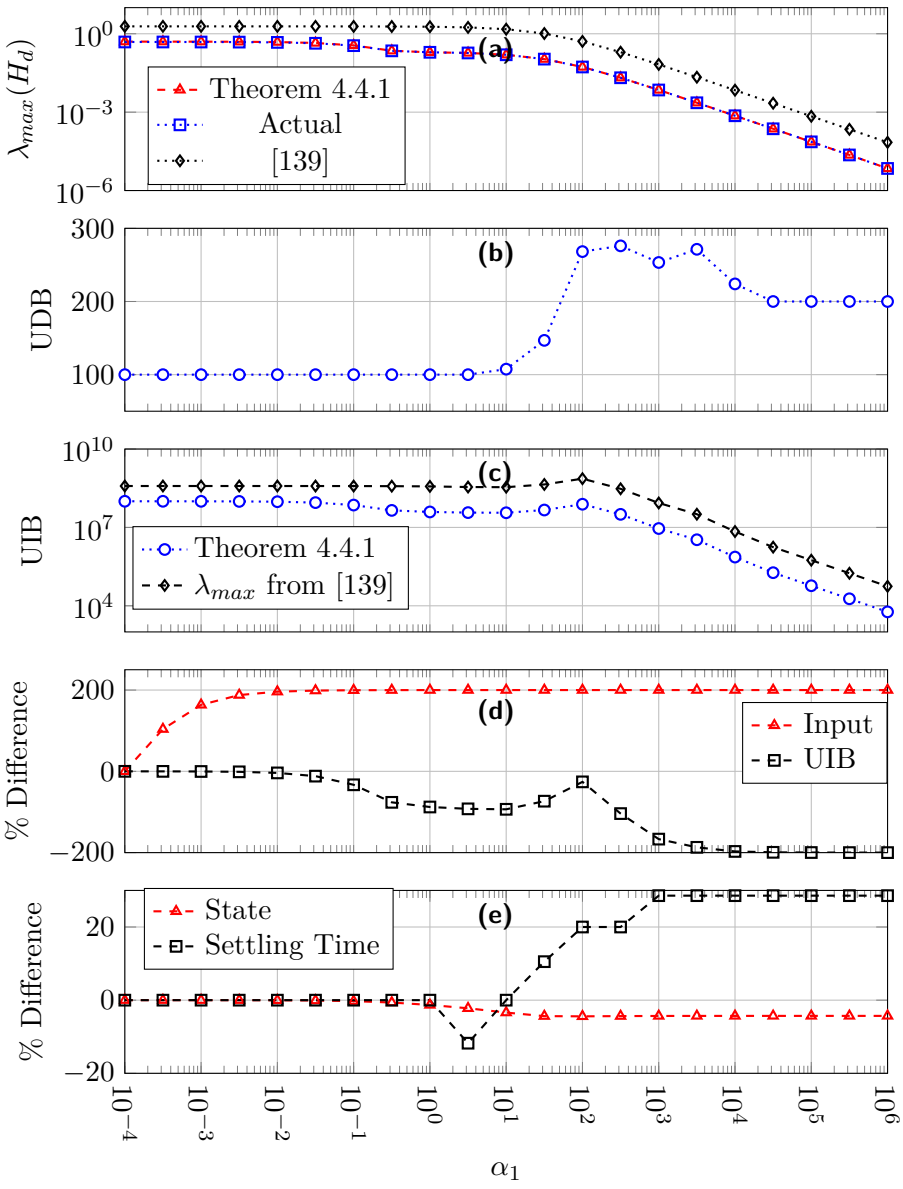


Figure 5.3: Effect of scaling the weight matrix Q by α_1 and holding R constant on the Dual Gradient Projection method when solving the first Schur-stable system with no state constraints.

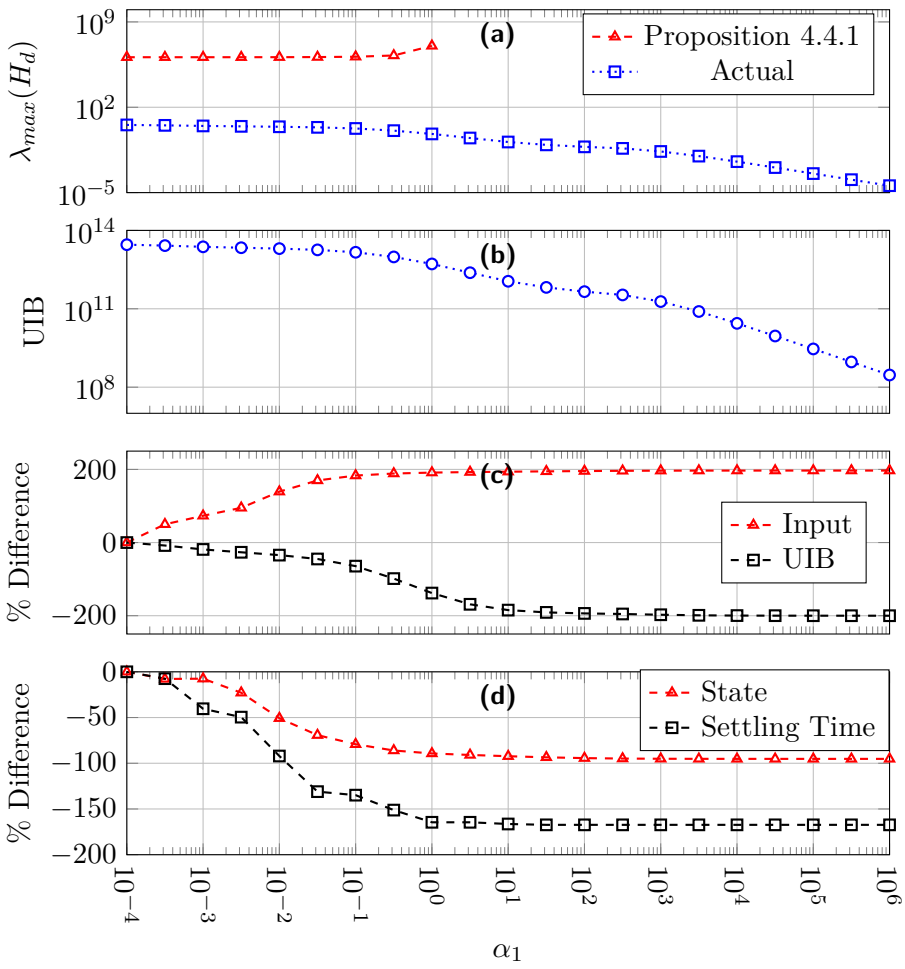


Figure 5.4: Effect of scaling the continuous-time weight matrix Q_c by α_1 and holding R_c constant on the Dual Gradient Projection method when solving the mass-spring-damper.

Chapter 6

Preconditioner design[†]

Decreasing the computational time of iterative optimization solvers is a key step in creating real-time-capable MPC algorithms. For many first-order optimization methods, the number of iterations required (and hence the time required to solve the problem) is directly related to the *conditioning* of the problem — measured by the condition number of the Hessian. One way to get a solution faster is then to improve the conditioning of the problem by reducing the condition number of the Hessian using a step called *preconditioning*. The preconditioning step can be thought of as transforming the variable space of the optimization problem from the original space to a "better" space for the optimization method.

In this chapter, we examine the preconditioning step using the lens of block Toeplitz theory. This includes extending the spectral analysis from Chapter 4 to include a symmetrically preconditioned Hessian, and deriving a novel preconditioner using circulant matrices that is faster and easier to compute.

6.1 Analysis of the preconditioned Hessian

For simplicity of discussion, we focus on the case when H_c is symmetrically preconditioned to form

$$H_L \coloneqq L_N^{-1} H_c (L_N^{-1})', \tag{6.1}$$

[†]Some of the material presented in this chapter is contained in the following work:
I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Horizon-independent preconditioner design for linear predictive control. *IEEE Transactions on Automatic Control*, 2022. doi: 10.1109/TAC.2022.3145657. (*in press*).

where $L_N \coloneqq I_N \otimes L$ is a block diagonal preconditioner with blocks $L \in \mathbb{R}^{m\times m}$. To simplify notation, we let $\bar{L} \coloneqq L^{-1}$ and $\bar{L}_N \coloneqq I_N \otimes \bar{L}$, making the preconditioner become

$$H_L = \bar{L}_N H_c \bar{L}_N'.$$

This case is most appropriate for first-order methods solving the CLQR problem, since symmetric preconditioning guarantees that the structure of the feasible set is preserved over the preconditioning operation and that the preconditioned Hessian is symmetric [151].

We can then show that the preconditioned Hessian H_L is block Toeplitz and derive its matrix symbol in Lemma 6.1.1.

Lemma 6.1.1. *Let H_c be a Hermitian block Toeplitz matrix with blocks of size $m \times m$ and the matrix symbol $\mathcal{P}_{H_c} \in \tilde{\mathcal{C}}_{2\pi} : \mathbb{T} \to \mathbb{C}^{m\times m}$. If $\bar{L}_N \coloneqq I_N \otimes \bar{L}$ with $\bar{L} \in \mathbb{R}^{m\times m}$, then*

$$H_L = \bar{L}_N H_c \bar{L}_N'$$

is block Toeplitz with the matrix symbol

$$\mathcal{P}_{H_L}(z) \coloneqq \bar{L}\mathcal{P}_{H_c}(z)\bar{L}', \qquad \forall z \in \mathbb{T}. \tag{6.2}$$

Proof. The matrix $\bar{L}_N$ is a block diagonal block Toeplitz matrix, and can be viewed as the truncation after N diagonals of the infinite block Toeplitz matrix $\bar{\mathbf{L}}$ with the matrix symbol $\mathcal{P}_{\bar{L}}(z) \coloneqq \bar{L},\ \forall z \in \mathbb{T}$.

We know from Corollary 3.3.1 that the result of the multiplication of the three matrices $\mathbf{EGF}$ will be block Toeplitz if $\mathbf{E}$ is an upper triangular block Toeplitz matrix, $\mathbf{G}$ is a block Toeplitz matrix, and $\mathbf{F}$ is a lower triangular block Toeplitz matrix. The product $\bar{\mathbf{L}}\mathbf{H}_c\bar{\mathbf{L}}'$ is then a block Toeplitz matrix due to Corollary 3.3.1 because $\mathbf{H}_c$ is known to be block Toeplitz and $\bar{\mathbf{L}}$ can be viewed as an upper triangular block Toeplitz matrix. We also know from Corollary 3.3.1 that the matrix symbol for the multiplication of the three matrices will just be the multiplication of the three symbols, meaning we get

$$\mathcal{P}_{H_L}(z) \coloneqq \bar{L}\mathcal{P}_{H_c}(z)\bar{L}', \qquad \forall z \in \mathbb{T}.$$

The final step is to verify the result of the multiplication of the truncated block Toeplitz matrices H_c and $\bar{L}_N$ is also block Toeplitz. Note that left-multiplying the block matrix H_c with the block diagonal matrix $\bar{L}_N$ to form $\bar{L}_N H_c$ will just multiply each block in the ith block column of H_c by the block on the diagonal of the ith block row of $\bar{L}_N$ (similar to how multiplication with a scalar diagonal matrix behaves). This means that since $\bar{L}_N$ and H_c are both block Toeplitz, every block in H_c would be multiplied by the same value, which

will preserve the block Toeplitz structure of H_c. A similar argument can be made for the right-multiplication of H_c by a block diagonal block Toeplitz matrix like $\bar{L}_N'$. This means that the result $\bar{L}_N H_c \bar{L}_N'$ will be block Toeplitz. □

Since we have shown that H_L is block Toeplitz and derived its matrix symbol, the spectral bounds in Section 4.2.2 can then be extended to the preconditioned matrix H_L by simply replacing $\mathcal{P}_{H_c}$ in Theorem 4.2.1(b) with $\mathcal{P}_{H_L}$ given by (6.2). A similar substitution can also be made when $\mathcal{G}$ is Schur-stable by using $\mathcal{P}_H$ in (6.2) with Corollary 4.2.2, since Lemma 6.1.1 will hold for both.

Remark 6.1.1. *This analysis requires L_N to be block Toeplitz, which may not be the case for many preconditioners unless such a constraint is added to their computation.*

Remark 6.1.2. *Results could also be derived for non-block diagonal preconditioners using [125, Thm 4.3] with $M^{-1}H_c$ where $M \coloneqq L_N L_N'$, but we do not discuss this extension here.*

6.2 Preconditioner design

There is a rich literature of preconditioners for Toeplitz and circulant matrices, with a focus on designing the preconditioners independent of the size of the matrix ([34] and references therein). These existing ideas can be applied to the block Toeplitz structure of the Hessian in the CLQR problem to design preconditioners to use when solving the QP.

One of the first circulant preconditioners was proposed by Gilbert Strang in [163]. This preconditioner was originally proposed for preconditioning iterative conjugate gradient methods, and is formed by simply copying the central diagonals of the Toeplitz matrix into the preconditioning matrix and wrapping them around to form a circulant matrix. Strang's preconditioner can be naturally extended to the block Toeplitz case by copying the individual blocks into the preconditioning matrix and wrapping them around to form a block circulant matrix.

In the case of the CLQR problem with a block diagonal preconditioning matrix, we can explicitly compute the block on the diagonal of the preconditioning matrix without forming the entire Hessian, as shown in Theorem 6.2.1.

Theorem 6.2.1. *Let H_c be the Hessian from* (2.10) *formed by choosing either:*

- *K as the infinite-horizon LQR controller for $\mathcal{G}$, with P the solution to the DARE* (2.3)*, or*
- *$K = 0$ with P the solution to the discrete-time Lyapunov equation* (2.4) *for a Schur-stable $\mathcal{G}$.*

The matrix H_c can then be symmetrically preconditioned as $L_N^{-1}H_c(L_N^{-1})'$, with $L_N \coloneqq I_N \otimes L$ and the blocks L given by the lower-triangular Cholesky decomposition of M with

$$M \coloneqq B'PB - RKB - B'K'R + R.$$

Proof. Based on the work in [163], a circulant preconditioning matrix W for the block Toeplitz matrix V will have entries that are obtained by copying the central diagonals of V and wrapping them around to form a circulant matrix since the main diagonal and its neighbors are usually strongly dominant. For example, the matrix

$$V = \begin{bmatrix} V_0 & V_1 & V_2 & V_3 & V_4 \\ V_{-1} & V_0 & V_1 & V_2 & V_3 \\ V_{-2} & V_{-1} & V_0 & V_1 & V_2 \\ V_{-3} & V_{-2} & V_{-1} & V_0 & V_1 \\ V_{-4} & V_{-3} & V_{-2} & V_{-1} & V_0 \end{bmatrix},$$

will have a block circulant preconditioning matrix

$$W = \begin{bmatrix} V_0 & V_1 & V_2 & V_{-2} & V_{-1} \\ V_{-1} & V_0 & V_1 & V_2 & V_{-2} \\ V_{-2} & V_{-1} & V_0 & V_1 & V_2 \\ V_2 & V_{-2} & V_{-1} & V_0 & V_1 \\ V_1 & V_2 & V_{-2} & V_{-1} & V_0 \end{bmatrix}.$$

Since we want our preconditioner to be block diagonal, we only need to focus on computing the appropriate diagonal block V_0 for the CQLR problem with Hessian H_c.

For the Hessian H_c, the $H_Q + H_P$ portion of the main diagonal block of the infinite dimensional block Toeplitz matrix is given in (4.10), which when the H_R and H_K components are added becomes

$$\sum_{i=0}^{\infty} B'(A_c^i)'Q_cA_c^iB - RKB - B'K'R + R.$$

Since A_c is Schur-stable and K is the LQR controller, this infinite sum converges to the solution of the DARE, making the diagonal block

$$V_0 = B'PB - RKB - B'K'R + R.$$

Since Strang's preconditioner is designed as a left preconditioner (i.e. $W^{-1}V$) and we want a symmetric preconditioner, we apply the lower-triangular Cholesky factorization to M, resulting in L. □

Remark 6.2.1. *The preconditioning matrix L in Theorem 6.2.1 can also be formed by using the matrix square-root of M instead of the Cholesky factorization, which could allow for L to have the same structure as M if the square root operation is structure preserving (e.g. if $\mathcal{G}$ is a circulant system, using the matrix square root can lead to L and H_L being circulant as well).*

Note that the matrices M and L will have dimension $m \times m$, and that the full preconditioning matrix L_N is formed by simply repeating L down the diagonal N times, so changing the horizon length means simply adding or removing blocks of L from the diagonal of L_N, meaning no new computations are required when the horizon length is changed. Also, all the matrices involved in computing M in Theorem 6.2.1 have dimensions on the order of m and n, and have no relation to the horizon length. This is in contrast to SDP-based preconditioner design techniques such as [151], which require the full Hessian to be placed inside the semidefinite optimization problem.

Since L_N is block Toeplitz, we can actually show that if $\mathcal{G}$ is single-input, the preconditioner will not affect the condition number of the Hessian.

Proposition 6.2.1. *If the system $\mathcal{G}$ is single-input (i.e. $m = 1$), then the block Toeplitz preconditioner L_N will not affect the condition number of H_c (i.e. $\kappa(H_L) = \kappa(H_c)$).*

Proof. A system with $m = 1$ will have $L \in \mathbb{R}$, making the preconditioner matrix simply $L_N = I_N \otimes L = LI_N$. For the symmetric preconditioning case, (6.1) becomes

$$\begin{aligned} H_L &= \left(\frac{1}{L} I_N\right) H_c \left(I_N \frac{1}{L}\right) \\ &= \frac{1}{L^2} H_c. \end{aligned}$$

Using the fact that $\lambda_i(\alpha H_c) = |\alpha| \lambda_i(H_c)$, we can then find that

$$\kappa(H_L) = \frac{\lambda_{max}\left(\frac{1}{L^2} H_c\right)}{\lambda_{min}\left(\frac{1}{L^2} H_c\right)} = \frac{|L^2|}{|L^2|} \frac{\lambda_{max}(H_c)}{\lambda_{min}(H_c)} = \kappa(H_c).$$

□

6.3 Numerical experiments

In this section, we present numerical examples showing the effect of the proposed preconditioner on the spectral properties of the condensed Hessian of the CLQR problem for three of the example systems given in Section 2.6.

6.3.1 Matrix conditioning

The first numerical results we present examine the Hessian conditioning for the example systems. The preconditioners were implemented using Julia 1.6.1, and the COSMO solver (version 0.8.1) [48] was used to compute the SDP preconditioner from [151]. It can be seen in Figures 6.1a and 6.2a that the limits presented in Theorem 4.2.1(b), are attained as the horizon length increases. Note though that the rate at which the condition number of the finite-dimensional Hessian approaches the bound is system-dependent, with the first Schur-stable example system converging within 0.01% of the bound by $N = 40$, while the inverted pendulum system requires $N = 225$ to converge to within 0.01%.

As described in Section 6.1, the condition number bound of Theorem 4.2.1(b) can be used to analyze the preconditioned Hessian, which is shown in Figures 6.1b and 6.2b. The bound was computed for the proposed preconditioner by finding L using Theorem 6.2.1, then substituting (6.2) into Theorem 4.2.1(b). Note that the bound computed using L from Theorem 6.2.1 does not hold for the SDP preconditioner from [151], since L will be different between the two preconditioners. Additionally, the SDP preconditioner does not guarantee that L_N will be block Toeplitz, so Theorem 4.2.1 cannot be used to compute horizon-independent limits when using the SDP preconditioner.

Comparing the proposed preconditioner against the existing SDP preconditioner shows that they produce nearly identical condition numbers, as can be seen in Figures 6.1b and 6.2b and Table 6.1. Proposition 6.2.1 can also be seen in Figure 6.2b, where the conditioning of the Hessian is not affected by the preconditioners, since the inverted pendulum is a single-input system.

As shown in Table 6.1, both the SDP preconditioner and our proposed preconditioner decrease the condition number by 66% for the Schur-stable system (2.16) and the already Schur-stable non-prestabilized distillation column, and decrease the condition number for the ill-conditioned Schur-stable system (2.17) by 97%. Applying the preconditioner to the prestabilized inverted pendulum system has no effect, leaving the condition number identical to the original, as predicted by Proposition 6.2.1 since this system has only one input.

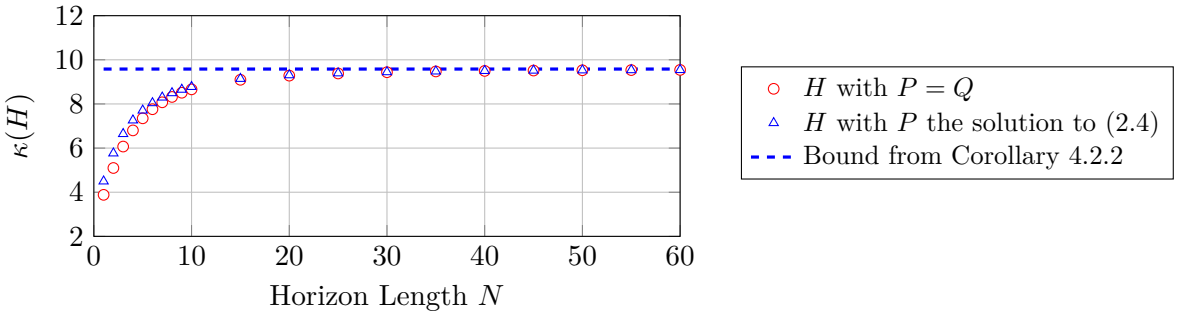
6.3.2 Performance of the Fast Gradient Method

To test the effect of the preconditioners on the actual performance of a first-order method solving the CLQR problem, the Fast Gradient Method (FGM) was implemented in Julia using the constant step-size scheme [83] and gradient map stopping criteria [150, Section 6.3.1] with a desired error of 10^{-5}. The inequality constraints were implemented through projection operations, with the non-prestabilized problems being simple projections onto a box and the prestabilized problems requiring a more complex projection algorithm. In these examples,

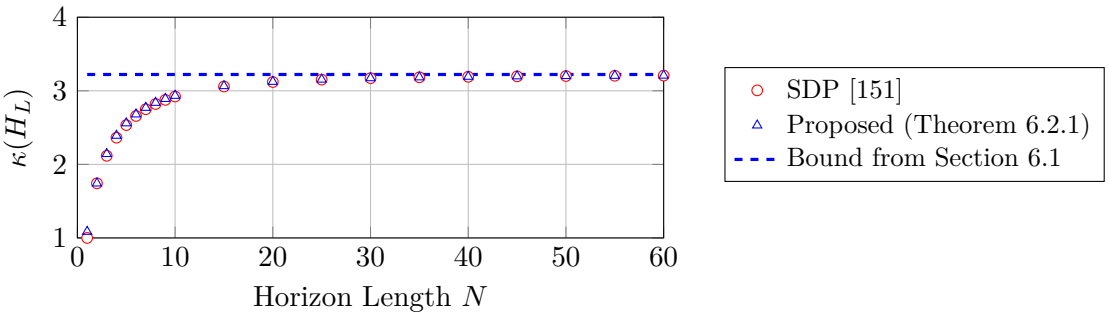
Table 6.1: Condition number of the preconditioned condensed Hessian.

System	Original	SDP [151]		Proposed (Thm 6.2.1)	
		$\kappa(H_L)$	% change	$\kappa(H_L)$	% change
Schur-stable (2.16)	8.776	2.922	-66.7%	2.933	-66.6%
Ill-conditioned Schur-stable (2.17)	254.66	7.415	-97.1%	7.500	-97.1%
Inverted pendulum (non-prestabilized)	42.512	42.468	-0.1%	(NC)[1]	(NC)[1]
Inverted pendulum (LQR prestabilized)	1.889	1.884	-0.21%	1.889	0%
Distillation column (non-prestabilized)	21.527	7.175	-66.67%	7.175	-66.67%
Distillation column (LQR prestabilized)	3.004	1.017	-66.14%	1.025	-65.86%

[1] Not computable (non-Schur-stable system)



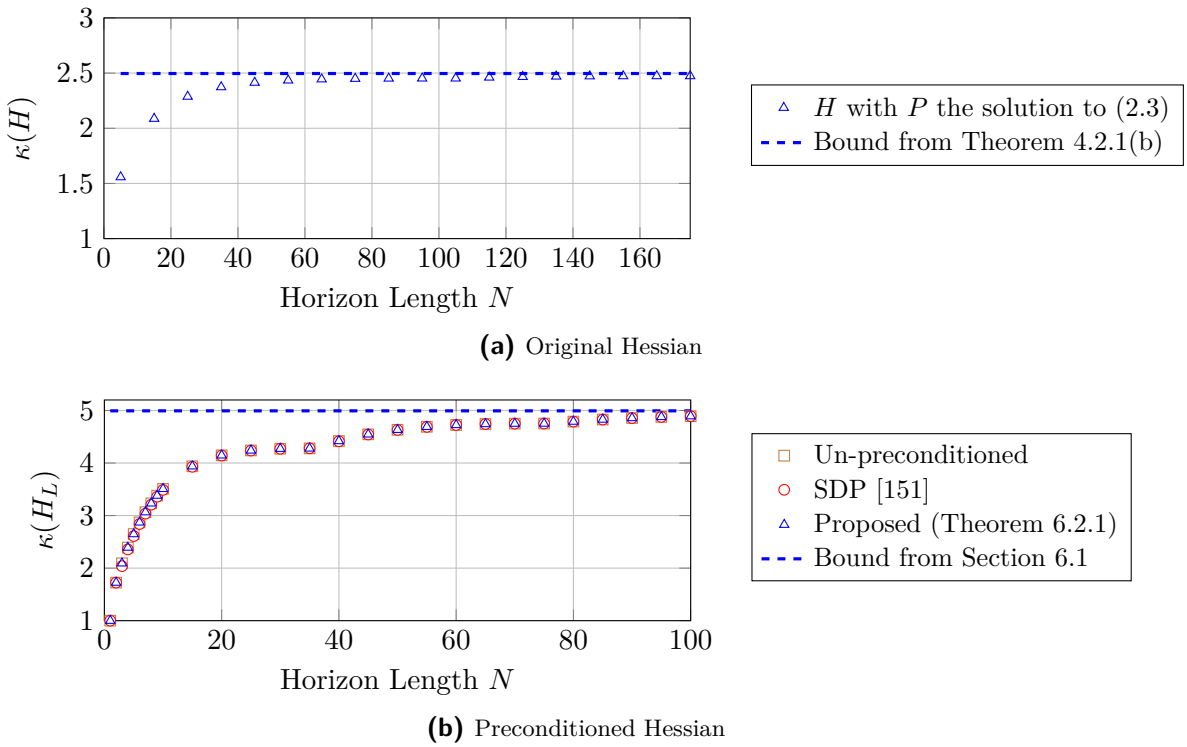
(a) Original Hessian



(b) Preconditioned Hessian

Figure 6.1: Condition number versus the horizon length of the condensed Hessian for the first Schur-stable system.

the projections for the prestabilized problems were computed by solving the projection QP directly; however, in an embedded application other techniques like an explicit QP [121] can be used instead.



(a) Original Hessian

(b) Preconditioned Hessian

Figure 6.2: Condition number versus the horizon length of the prestabilized condensed Hessian for the inverted pendulum system.

We present two types of iteration results in Table 6.2: the actual iterations taken by the FGM and the cold-start Upper Iteration Bound (UIB) given by [151]. The UIB will be the worst-case number of iterations needed to achieve convergence regardless of the number of active constraints in the solution or the initial solution vector in the solver, while the actual iterations is the number taken on a single example QP for the given CLQR problem.

When applying the proposed preconditioner to the Hessian, we see a 2.1x actual speedup and 2.6x UIB speedup for the Schur-stable system (2.16), a 3.6x actual speedup and 2.25x UIB speedup for the non-prestabilized distillation column, and a 4.5x actual speedup and 9.5x UIB speedup for the ill-conditioned Schur-stable system (2.17). When the SDP preconditioner is applied to the non-prestabilized inverted pendulum, it has no discernable effect on the actual iterations required for the FGM to converge, requiring 51 iterations in both cases.

A large decrease in iterations is caused by applying a prestabilizing controller to the unstable inverted pendulum though, which results in a 12.75x speedup compared to the non-prestabilized system. Applying a prestabilizing controller to the distillation column produces a speedup of 9.6x compared to the non-prestabilized system, a larger speedup than just preconditioning the non-prestabilized system.

Table 6.2: Iterations required for cold-start convergence of the preconditioned Fast Gradient Method to $\epsilon=10^{-5}$.

System	Original[1]	SDP [151]		Proposed (Thm 6.2.1)	
		Iterations[1]	Speedup	Iterations[1]	Speedup
Schur-stable (2.16)	42 / 19	16 / 9	2.62x / 2.11x	16 / 9	2.62x / 2.11x
Ill-conditioned Schur-stable (2.17)	294 / 114	32 / 25	9.19x / 4.56x	31 / 25	9.48x / 4.56x
Inverted pendulum (non-prestabilized)	143 / 51	129 / 51	1.10x / -	(NC)[2]	(NC)[2]
Inverted pendulum (LQR prestabilized)	18 / 3	17 / 3	1.05x / -	18 / 3	- / -
Distillation column (non-prestabilized)	97 / 48	43 / 25	2.25x / 1.92x	43 / 25	2.25x / 1.92x
Distillation column (LQR prestabilized)	22 / 5	3 / 3	7.33x / 1.66x	3 / 3	7.33x / 1.66x

[1] Cold-start upper iteration bound from [151] / actual
[2] Not computable (non-Schur-stable system)

Table 6.3: Time required for computing the preconditioners.

System	SDP [151] (ms)	Proposed (Thm. 6.2.1) (ms)	Speedup
Schur-stable (2.16)	197.4	0.213	926.7x
Ill-conditioned Schur-stable (2.17)	142.9	0.218	655.5x
Inverted pendulum (non-prestabilized)	18.69	(NC)[1]	(NC)[1]
Inverted pendulum (LQR prestabilized)	17.45	0.218	80.0x
Distillation column (non-prestabilized)	151,746	2.543	59,672x
Distillation column (LQR prestabilized)	81,929	2.545	32,192x

[1] Not computable (non-Schur-stable system)

The proposed preconditioner is also faster to compute than the SDP preconditioner for the example systems, with timing results for the computations on an Intel® Core™ i7-10710U at 1.10 GHz with 64 GB of RAM given in Table 6.3 showing a consistent speedup of at least an order of magnitude or more in the computation times. For example, the non-prestabilized distillation column example has a 59,672x speedup in the computation time between the proposed preconditioner and the SDP preconditioner, with the SDP preconditioner requiring 151.74 seconds to compute and the proposed preconditioner requiring only 2.5 milliseconds.

6.4 Conclusions

In this chapter, we presented a new preconditioner that is based on the block Toeplitz structure of the Hessian for the condensed CLQR problem when using either a prestabilizing LQR controller or a Schur-stable system with appropriate choice of the terminal weight matrix P. We showed that this preconditioner provides performance comparable to an existing SDP preconditioner on several numerical examples, leading to speedups of between 2.1x and 9.6x for the FGM. The proposed preconditioner is also faster to compute than the SDP preconditioner, especially for problems with a long-horizon, like the distillation column example, which showed a decrease in computation time of over 50,000x compared to the SDP preconditioner.

We also showed that the numerical prestabilization controller K has a direct effect on the spectrum of the condensed Hessian. This means that preconditioning could also be achieved through a careful choice of K instead of applying a separate preconditioner (at the expense of turning the constraint sets into more complex shapes). Future work could explore developing preconditioners using this idea, such as one based on the idea of loop-shaping the system to control the singular values of the predicted system and reduce the condition number of the Hessian.

The derivation and examples in this chapter focused on preconditioning the primal QP for the CLQR problem, however it is also common to use the dual form (2.8) with first-order methods such as Dual Gradient Projection or the Dual Fast Gradient Method. The preconditioner defined in Theorem 6.2.1 could be extended to handle the dual problem by using the diagonal blocks of the dual condensed Hessian in the preconditoner instead, however more work is needed to examine this. For instance, the theoretical bounds for the preconditioned condensed Hessian in Section 6.1 would need to be updated, since the cases in which the dual condensed Hessian possesses a block Toeplitz structure are different from those for the primal condensed Hessian, so there may be more technicalities in the resulting analysis.

Chapter 7

Fixed-point round-off error analysis[†]

As discussed in Chapter 2, many embedded implementations of MPC that utilize the FGM are done using fixed-point arithmetic to save on resources and reduce the computational complexity. A key part of designing these implementations is ensuring the data types used are able to guarantee the algorithmic stability of the FGM in fixed-point arithmetic. Requirement 2.5.1, which says the eigenvalues of the fixed-point Hessian (which we denote as $\hat{H}$) must be inside the unit circle, is a start, but it still leaves the question: how do we choose the fractional width to ensure this requirement is met? In this chapter, we answer that question by deriving a *rounding stability margin* for the Hessian and then use that margin to derive inequalities that can be used to compute the number of fractional bits needed.

7.1 Matrix pseudospectrum

In this analysis, we will utilize a property of a matrix called its *pseudospectrum*.

Definition 7.1.1 (Pseudospectrum [170, §2])**.** *Let $A \in \mathbb{C}^{n\times n}$, $p \in [1,\infty]$, and $\epsilon > 0$ be arbitrary. The ϵ-pseudospectrum $\lambda_\epsilon(A)$ of A is the set of $\tilde{\lambda} \in \mathbb{C}$ given by:*

1. $\left\|(\tilde{\lambda}I - A)^{-1}\right\|_p > \frac{1}{\epsilon}$, *(or $\lambda_{min}(\tilde{\lambda} - A) < \epsilon$ if $p = 2$), or*
2. $\tilde{\lambda} \in \lambda(A + E)$ *for some $E \in \mathbb{C}^{n\times n}$ with $\|E\|_p < \epsilon$.*

[†]Material presented in this chapter has been published in the following work:
I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Modeling round-off error in the fast gradient method for predictive control. In *58th IEEE Conference on Decision and Control (CDC)*, pages 4331–4336, Nice, France, 2019. IEEE. doi: 10.1109/CDC40024.2019.9029910.

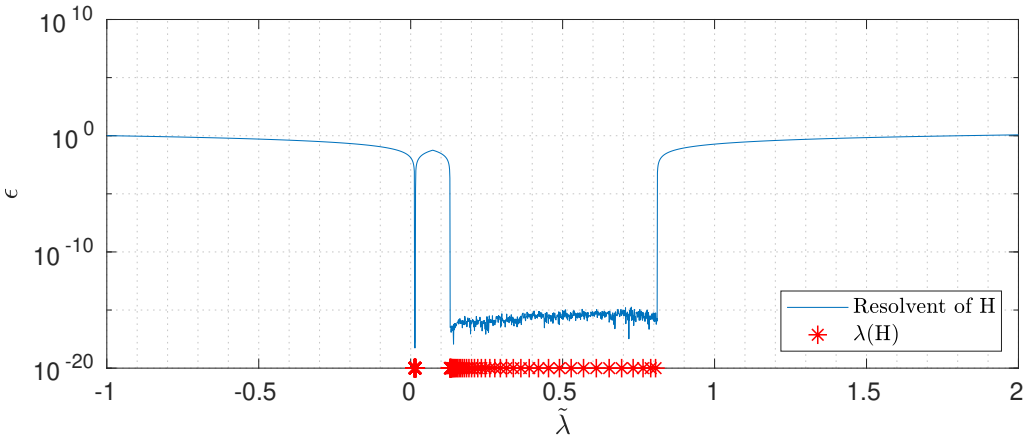


Figure 7.1: The inverse of the resolvent of H when $\tilde{\lambda}$ is constrained to the real line. The ϵ-pseudospectra of H are the level-sets taken at the value ϵ, and the eigenvalues of H are located at the red asterisks. ©2019 IEEE

Note that statements 1 and 2 in Definition 7.1.1 are equivalent. Statement 1 presents the pseudospectrum as being the subset of the complex plane where the norm of the resolvent of A is greater than ϵ^{-1}. Alternately, statement 2 presents the pseudospectrum as the subset of the complex plane containing the eigenvalues of A when A is perturbed by a matrix E with a given norm less than ϵ.

The pseudospectrum is a useful computational tool for describing the behavior of linear operators, especially those that are nonnormal (i.e. $AA' \neq A'A$). In this work though we focus on normal operators (i.e. $AA' = A'A$) since H is normal, which means that the ϵ-pseudospectrum can be interpreted as the set of real numbers that are within ϵ of an eigenvalue of A when using the 2-norm in Definition 7.1.1. An example pseudospectrum is shown in Figure 7.1, where the inverse-resolvent of the Hessian for the example system presented later in Section 7.3 is plotted along with the eigenvalues of the Hessian with a horizon length of 50. The ϵ-pseudospectra of H are then the level-sets taken at ϵ of the resolvent function, with the figure showing that the level sets for $\epsilon < 10^{-10}$ contain the eigenvalues of H in this example.

7.2 Choosing the fractional precision

When the Fast Gradient Method is implemented using a fixed-point data format, care must be taken to ensure that Requirement 2.5.1 is satisfied. The placement of the eigenvalues of $\hat{H}$ is determined by both the number of integer bits i (through overflow), and also by the number of fractional bits f (through loss of precision). We will focus only on the fractional length and assume the integer length is chosen using [83, Prop. 1] to prevent overflow.

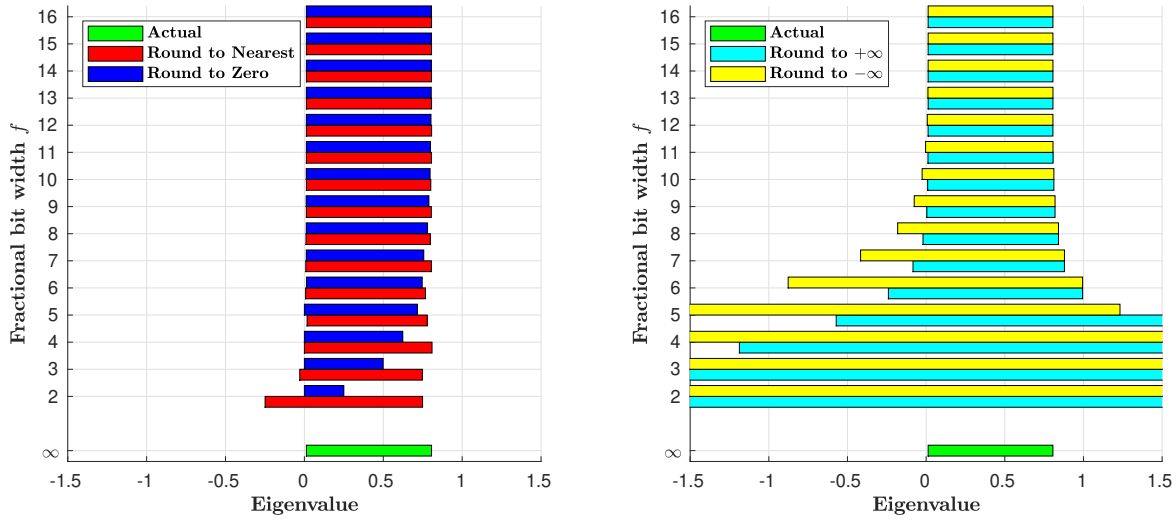


Figure 7.2: Interval containing the spectrum of $\hat{H}$ with different rounding modes.

The effect of the fractional length on the spectrum of the fixed-point Hessian $\hat{H}$ for the numerical example in Section 7.3 with a horizon of length 50 can be seen in Figures 7.2 and 7.3 for four of the rounding modes available in fixed-point arithmetic that are described in Table 2.2. In this example, using round to zero/nearest gives a fixed-point Hessian that is always Schur-stable but becomes indefinite (eigenvalues less than 0) for $f < 6$. If round to $\pm\infty$ is used, it is both unstable and indefinite in low precision (as seen in the intervals in Figure 7.2). This shows that the rounding choice makes a large impact on the error between $\hat{H}$ and H, and must be factored into any analysis to determine the required bit length.

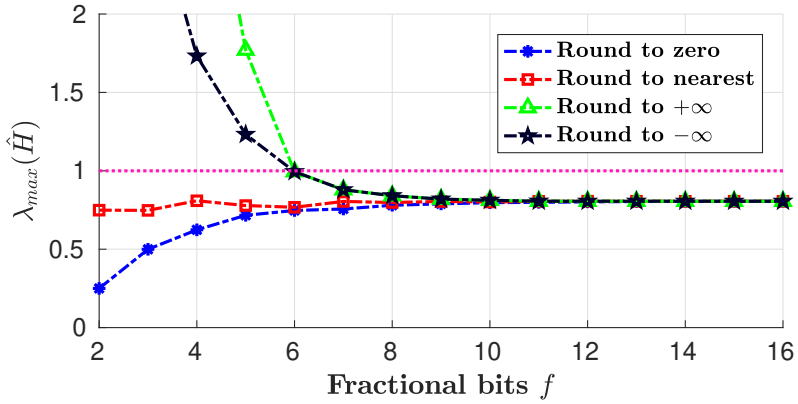
To analyze how the rounding affects the matrix, we model the round-off error as an additive matrix disturbance to H using

$$\hat{H} = H + E. \tag{7.1}$$

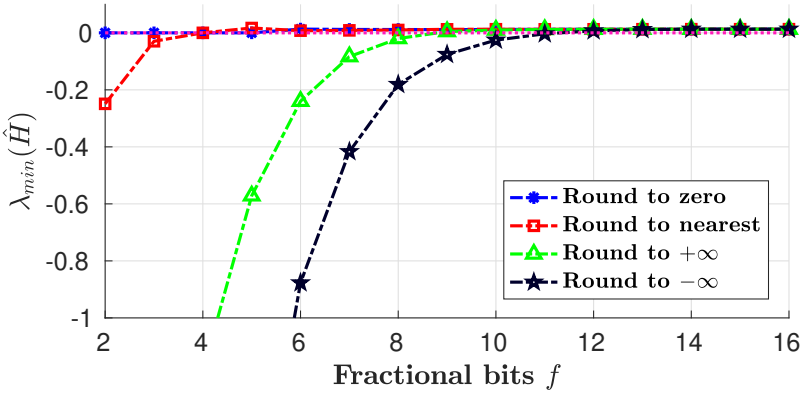
The exact values contained in the elements of the round-off matrix E in (7.1) will depend on the rounding mode chosen, but the magnitude of the values will always be less than the ϵ_f given in Table 2.2.

To quantify the effect that the round-off error has on the spectrum of $\hat{H}$, we introduce a metric called the *rounding stability margin*.

Definition 7.2.1 (Rounding stability margin)**.** *Let $\hat{H} = H + E$ with $\|E\|_2 = \beta$ and $\lambda(H) \in (0, 1)$. The rounding stability margin η is the smallest value of β that causes the eigenvalues of $\hat{H}$ to leave the interval $(0, 1)$.*



(a) Maximum Eigenvalue



(b) Minimum Eigenvalue

Figure 7.3: Extremal values of $\lambda(\hat{H})$ with $N = 50$. ©2019 IEEE

This margin represents the largest possible disturbance matrix that can be added to H before causing Requirement 2.5.1 to be violated. The margin can be calculated for symmetric matrices using the pseudospectrum of H as shown in Lemma 7.2.1.

Lemma 7.2.1. *Let $H = H'$ be a matrix with eigenvalues $\lambda(H) \in (0, 1)$. The rounding stability margin η of H is*

$$\eta(H) = \min\left\{\|(-H)^{-1}\|_2^{-1}, \|(I - H)^{-1}\|_2^{-1}\right\}.$$

Proof. We begin by noting that H is symmetric, meaning its spectrum is composed of only real eigenvalues and that its ϵ-pseudospectrum will be an interval on the real line. By statement 2 of Definition 7.1.1, the eigenvalues of H perturbed by E with $\|E\|_2 < \epsilon$ will leave the interval $(0, 1)$ and destabilize H when either 0 or 1 is contained inside $\lambda_\epsilon(A)$. The largest allowable value of ϵ can then be computed using statement 1 of Definition 7.1.1 by evaluating the resolvent of H at the points 0 and 1, and computing $1/\epsilon$ at each point. □

The result in Lemma 7.2.1 holds for any symmetric matrix H, so it can be used to find the rounding stability margin for the Hessian of the CLQR problem (2.7) with any system or weight matrices.

7.2.1 Generic rounding model

We now present a framework for computing the necessary number of fractional bits for the FGM under a generic round-off error model that encompasses all of the rounding methods described in Table 2.2. The basis for this model is that every element in H will experience an error of at most $\pm\epsilon_f$, so the worst-case perturbation matrix would then have entries of $\pm\epsilon_f$.

Definition 7.2.2 (Generic round-off error model)**.** *Let ϵ_f be the maximum round-off error created when a value is converted into a fixed-point representation with f fractional bits. Define $E_g \in \mathbb{R}^{k\times k}$ to be the worst-case component-wise round-off error matrix with $\pm\epsilon_f$ in every entry, i.e.*

$$E_g \coloneqq \begin{bmatrix} \pm\epsilon_f & \pm\epsilon_f & \dots \\ \pm\epsilon_f & \pm\epsilon_f & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{7.2}$$

While we call E_g the worst-case perturbation matrix, just choosing the extremes of each element in a matrix is not guaranteed to produce the worst-case perturbation of the eigenvalue spectrum. In this case though, we do not need to be worried about finding the exact worst-case perturbation matrices since we will be working with matrix norms, and the 1-norm and ∞-norm of E_g will always be greater than the norm of the actual worst-case perturbation matrix.

Since each element of (7.2) is the same, modulo the sign, the matrix 1-norm and ∞-norm can be computed exactly and then be used to upper-bound the 2-norm of E_g.

Lemma 7.2.2. *Let $E_g \in \mathbb{R}^{k\times k}$ be the round-off error matrix from Definition 7.2.2 and let ϵ_f be the maximum round-off error possible with the rounding method. It follows that*

$$\|E_g\|_2 \leq \|E_g\|_\infty = |\epsilon_f|k.$$

Proof. Recall that the matrix ∞-norm (or 1-norm) is the largest absolute row (or column) sum of the matrix. Since the 1-norm and ∞-norm take the absolute value of the entries before summing them, the sign of the rounding error is irrelevant. Since H is symmetric, this gives $\|E_g\|_1 = \|E_g\|_\infty = |\epsilon_f|k$. Then, note that $\|E_g\|_2 \leq \sqrt{\|E_g\|_1\|E_g\|_\infty}$ [21, Fact 9.8.23], which means that $\|E_g\|_2 \leq \|E_g\|_\infty$. □

To find the number of fractional bits needed to satisfy Requirement 2.5.1, we must find the number of bits needed to make the perturbation matrix less than the rounding stability margin (i.e. $\|E_g\|_2 < \eta$). This can be done in closed-form, as shown in Theorem 7.2.1.

Theorem 7.2.1. *Let $f \in \mathbb{N}^+$ be the number of fractional bits in the fixed-point number representation, and ϵ_f the maximum round-off error a number may experience through rounding in that representation. If H has a rounding stability margin of η, then the number of fractional bits sufficient to guarantee that $\lambda(\hat{H}) \in (0, 1)$ is*

$$f = \begin{cases} \lceil -\log_2\left(\frac{\eta}{mN}\right) \rceil - 1 & \textit{if using round to nearest,} \\ \lceil -\log_2\left(\frac{\eta}{mN}\right) \rceil & \textit{otherwise.} \end{cases}$$

Proof. Using statement 2 of Definition 7.1.1 and the concept of the rounding stability margin introduced in Definition 7.2.1, we can say that we need $\|E_g\|_2 \leq \eta$ to guarantee that $\lambda(\hat{H}) \in (0, 1)$. Using Lemma 7.2.2 and the fact that H is of dimension $mN \times mN$, we can transform that requirement into $|\epsilon_f| mN \leq \eta$. Turning the inequality into an equality and isolating ϵ_f then gives $|\epsilon_f| = \frac{\eta}{mN}$. Using the fact that $|\epsilon_f| \in \{2^{-f}, 2^{-(f+1)}\}$ depending on the rounding mode, we can substitute for ϵ_f and simplify to find f. □

The closed-form expressions for f given in Theorem 7.2.1 hold for any rounding mode, and any system/weight matrix combination. Note that all rounding modes have the same fractional length with the exception of round to nearest, where 1 fewer bit is required.

The value of f from Theorem 7.2.1 is dependent upon both the horizon length and the system input dimension, and is monotonically increasing in both, as shown for the horizon length in Figure 7.4. This increase in the bound is caused by the monotonic increase in $\|E_g\|_\infty$ when the number of fraction bits is held constant and the horizon length increases. This means that in general the fraction length computed for a specific horizon can be used with a shorter horizon (e.g. in a decreasing-horizon controller), but may not be sufficient for longer horizons.

The generic round-off error model in Definition 7.2.2 and the associated fractional widths from Theorem 7.2.1 can be used irrespective of the stability of the system being controlled (e.g. there is no assumption of Schur-stability needed). This means the results apply to both the normal condensed formulation (2.7) and the numerically robust condensed formulation (2.10).

7.2.2 Parametric rounding model

The generic model in Section 7.2.1 is conservative for some rounding modes when applied to the FGM with long horizons and Schur-stable systems, especially round to nearest and round to zero (as seen in Figure 7.4 when the actual bits needed for these two modes remains constant). To reduce the conservatism of the error estimation, we introduce a parametric

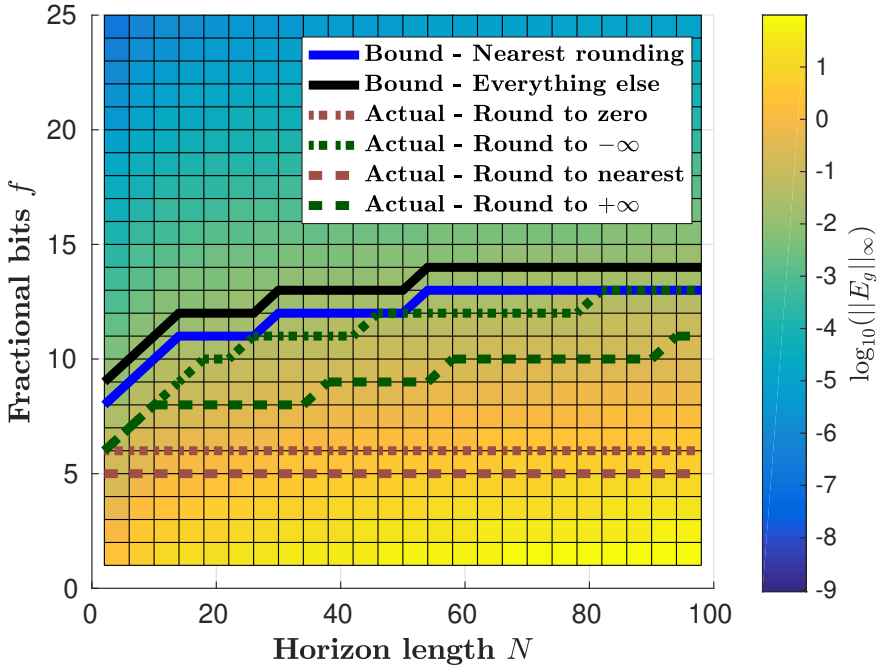


Figure 7.4: The minimum number of fractional bits required for a given horizon length when using the generic rounding model from Theorem 7.2.1. The background shows the log of the bound for $\|E_g\|_2$ computed using Lemma 7.2.2. ©2019 IEEE

model for the round-off error experienced by H that incorporates knowledge of both the decay of terms in H and its block Toeplitz structure. This model requires the predicted system to be Schur-stable, and for the FGM to use either the round to nearest or round to zero rounding mode.

Using the block Toeplitz structure of the condensed Hessian that we derived in Chapter 4, we can compute a horizon-independent value for the rounding stability margin η, as shown in Lemma 7.2.3.

Lemma 7.2.3. *Let H be the block Toepltiz Hessian from* (2.7) *with eigenvalues $\lambda(H) \in (0, 1)$, $\mathcal{G}$ be Schur-stable, $\mathcal{P}_H$ the matrix symbol of H, and P be the solution of the discrete Lyapunov equation* (2.4)*. Then the rounding stability margin η is*

$$\eta(H) \coloneqq \min\left\{ \left\| (-\mathcal{P}_H)^{-1} \right\|_{H_\infty}^{-1}, \left\| (I_m - \mathcal{P}_H)^{-1} \right\|_{H_\infty}^{-1} \right\}.$$

Proof. Since the resolvent in Definition 7.1.1 can be found using $\lambda_{min}(\tilde{\lambda} - H)$, we can use the results from Chapter 4 to replace H in Lemma 7.2.1 with its matrix symbol $\mathcal{P}_H$. □

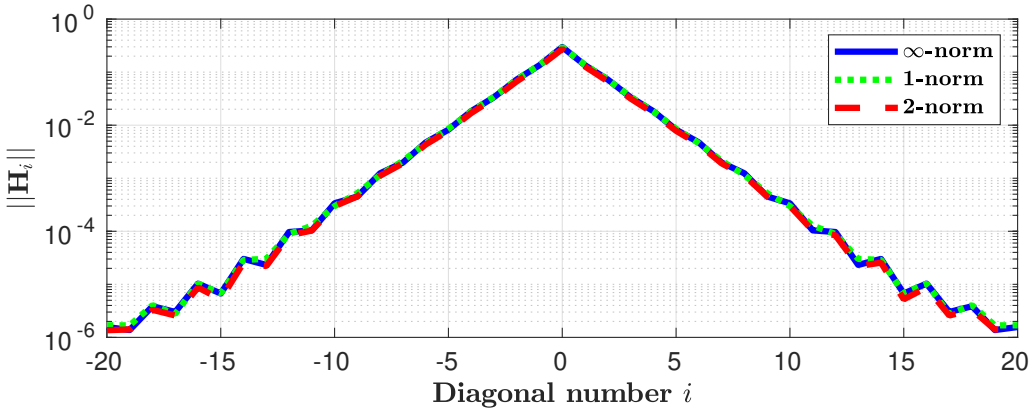


Figure 7.5: Value of $\|[\mathbf{H}]_i\|$ as the diagonal number grows. ©2019 IEEE

Note that we require H to have its eigenvalues inside the interval $(0, 1)$ for Lemma 7.2.3. This is not a restrictive assumption though, because if they are not in that interval, then the cost function (2.7a) can be multiplied by $\frac{1}{1.1\lambda_{max}(H)}$ to get the eigenvalues inside the interval. Doing this transformation will only change the value of the optimal cost and not change the optimal vector u that is found.

We can further exploit the block Toeplitz structure of H by noting that the diagonal blocks, $[\mathbf{H}]_i$ for diagonal $i \in \mathbb{Z}$, are given by

$$[\mathbf{H}]_i = \begin{cases} B'(A^i)'PB & \text{if } i > 0, \\ B'PB + R & \text{if } i = 0, \\ B'PA^{|i|}B & \text{if } i < 0, \end{cases} \tag{7.3}$$

with $[\mathbf{H}]_i = [\mathbf{H}]'_{-i}$ and P the solution of the discrete Lyapunov equation (2.4). We define diagonals as positive (or negative) if they are in the upper-triangular (or lower-triangular) region. If A corresponds to the state transition matrix of a Schur-stable system, then as the diagonal number i approaches $\pm\infty$, the block $[\mathbf{H}]_i$ tends to 0, as shown in Figure 7.5.

The idea behind the parametric model is to exploit this decay and switch from modeling the worst-case round-off error to instead modeling the actual round-off error after a certain diagonal number. We now present a formal definition for this model in Definition 7.2.3.

Definition 7.2.3 (Parametric round-off error model)**.** *Let $T_i \in \mathbb{R}^{l\times l}$ be the block on the i^{th} diagonal of the block Toeplitz matrix $\mathbf{T}$ with the property that $\lim_{i\to\infty}\|T_i\|_{max} = 0$. Let ϵ_f be the round-off error associated with the conversion to fixed-point representation using*

either round to nearest or round to zero, and k be the diagonal beyond which all blocks in the fixed-point representation $\hat{T}_i$ are 0, i.e.

$$k \coloneqq \min_i \left\{ i \in \mathbb{N}^+ \mid \|T_j\|_{max} < \epsilon_f, \forall |j| \geq i, j \in \mathbb{Z} \right\}.$$

Define the parametric round-off error matrix as

$$E_p \coloneqq E_G + E_T,$$

where E_G is a matrix of bandwidth $k-1$ with E_g as its blocks, i.e.

$$[E_G]_i \coloneqq \begin{cases} E_g & \text{if } |i| < k, \\ 0 & \text{otherwise,} \end{cases}$$

and E_T is composed of the diagonal components of $\mathbf{T}$ that are after diagonal k, i.e.

$$[E_T]_i \coloneqq \begin{cases} T_i & \text{if } |i| \geq k, \\ 0 & \text{otherwise.} \end{cases}$$

Note that in this model, the value of k is inclusive of the first diagonal block which becomes 0, and $k = 0$ is not possible, since this would imply that the entire matrix has been rounded to 0.

The matrix E_T in Definition 7.2.3 is block Toeplitz, so its spectrum can be bounded using the same system-theoretic techniques used in Chapter 4. This then allows for the computation of the round-off error for the Hessian of (2.7).

Theorem 7.2.2. *Let ϵ_f be the maximum round-off error when using either round to zero or round to nearest to convert H into fixed-point representation. Let η be the rounding stability margin of H from Lemma 7.2.3, and k be from Definition 7.2.3. If the parametric rounding model is used with a Schur-stable system $\mathcal{G}$ with P the solution to the discrete Lyapunov equation (2.4), then the fraction bit lengths sufficient for $\hat{H}$ to satisfy Requirement 2.5.1 also satisfies the inequality*

$$|\epsilon_f| m(2k-1) + 2\|\mathcal{P}_{\tilde{H}}(k, \cdot)\|_{H_\infty} < \eta,$$

where

$$\mathcal{P}_{\tilde{H}}(n, z) \coloneqq z\mathcal{G}_P(z) - B'P\mathcal{P}_n(z)B \quad \forall z \in \mathbb{T},$$

$$\mathcal{G}_P(z) \coloneqq \left[\begin{array}{c|c} A & B \\ \hline B'P & 0 \end{array}\right],$$

$$\mathcal{P}_n(z) \coloneqq \sum_{i=0}^{n-1} A^i z^{-i} \quad \forall z \in \mathbb{T}.$$

Proof. To guarantee that Requirement 2.5.1 holds, we need to find when $\|E_p\|_2 < \eta$. First apply the sub-additive property of the matrix norm to get the inequality

$$\|E_G\|_2 + \|E_T\|_2 \ < \ \eta. \tag{7.4}$$

The matrix E_G is banded, with non-zero blocks on diagonals $\{-(k-1), \ldots, 0, \ldots, k-1\}$. This gives the bound

$$\|E_G\|_2 \ \leq \ |\epsilon_f| m(2k-1), \tag{7.5}$$

since there are $2k-1$ diagonals containing blocks of E_g with dimension $m \times m$.

Since E_T is a Toeplitz matrix, we can construct the matrix symbol using the Fourier series of the components of E_T as

$$\mathcal{P}_{\bar{H}} = \sum_{i=k}^{\infty} B'PA^iBz^{-i} + \sum_{i=k}^{\infty} B'(A^i)'PBz^i. \tag{7.6}$$

Adding and subtracting the first k terms of the summations then allows (7.6) to simplify to

$$\mathcal{P}_{\bar{H}} = B'P\left((I - z^{-1}A)^{-1} - \mathcal{P}_k(z)\right)B + B'\left((I - zA')^{-1} - \mathcal{P}_k(z)^*\right)PB. \tag{7.7}$$

The first term in (7.7) then can be simplifed further to

$$z\mathcal{G}_P(z) - B'P\mathcal{P}_n(z)B. \tag{7.8}$$

Note that the two terms in (7.7) are the conjugate transpose of each other. This means that when the H_∞ norm of $\mathcal{P}_{\bar{H}}$ is taken, it will simply be twice the H_∞ norm of (7.8). We can then use the H_∞ norm of $\mathcal{P}_{\bar{H}}$ as a size-independent upper bound for $\|E_T\|_2$, giving the inequality

$$\|E_T\|_2 \leq 2\|\mathcal{P}_{\bar{H}}(k, \cdot)\|_{H_\infty}. \tag{7.9}$$

The inequalities (7.5) and (7.9) can then be substituted into the inequality (7.4) as upper bounds for the terms $\|E_G\|_2$ and $\|E_T\|_2$, respectively, producing the inequality contained in the lemma. □

To compute the fractional bit length from the inequality in Theorem 7.2.2, we do the following:

1. compute $\|H_i\|_2$ using (7.3) for various values of i, and then
2. iterate through each fraction length to determine if the inequality in Theorem 7.2.2 holds.

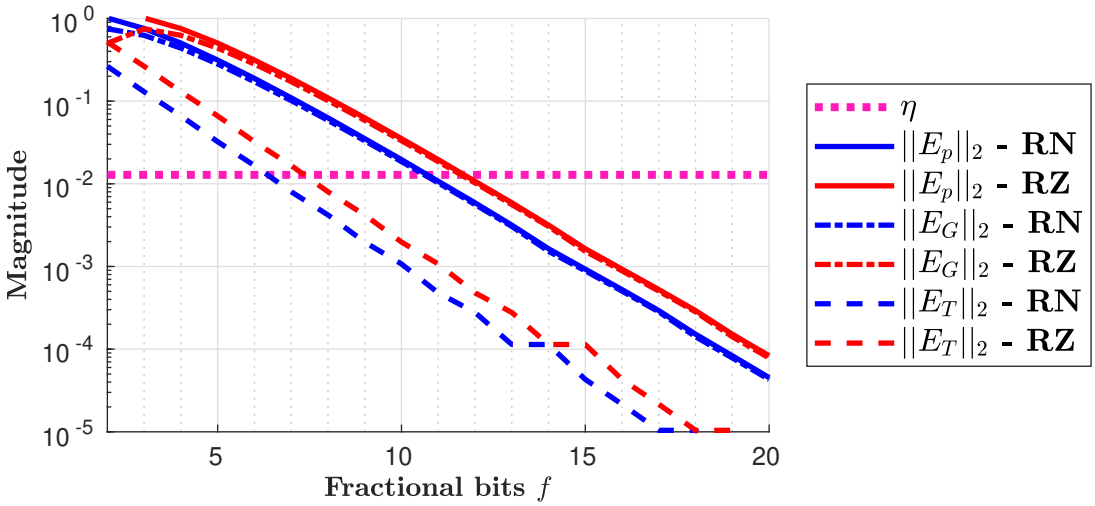


Figure 7.6: The matrix norms from Theorem 7.2.2, with round to nearest (RN), and round to zero (RZ). ©2019 IEEE

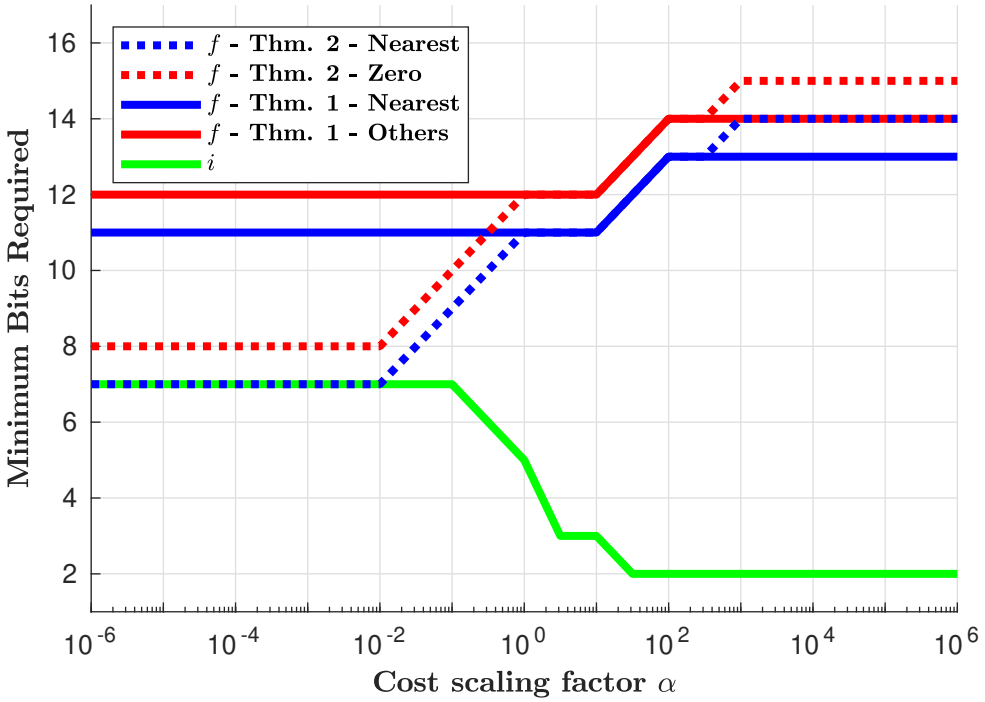
7.3 Numerical experiments

In this chapter, we use a slightly modified version of the Schur-stable example problem from Section 2.6, with the cost matrices $Q = \text{diag}(0.1, 0.2, 0.3, 0.4)$ and $R = \text{diag}(0.01, 0.02)$.

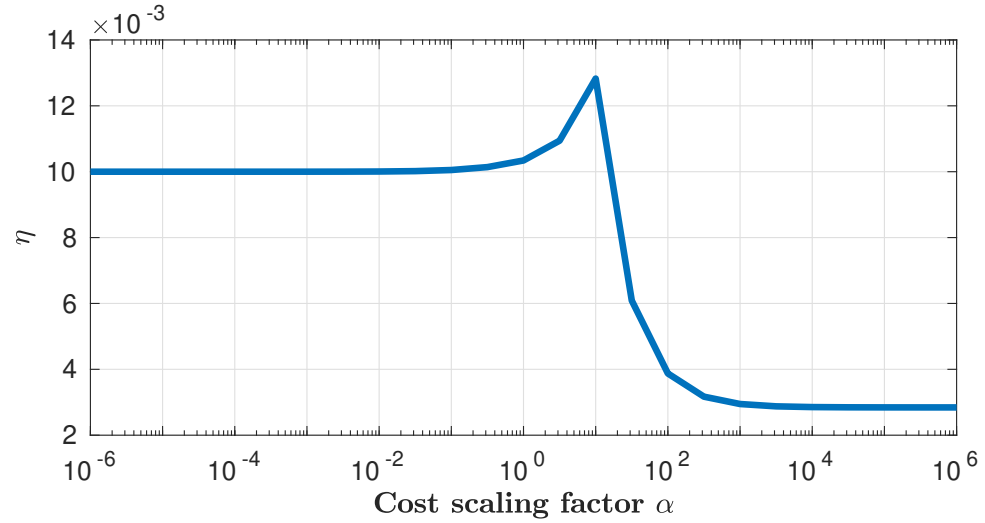
The result of computing the fractional bit length using Theorem 7.2.2 can be seen in Figure 7.6 for the example problem. In this example, E_G dominates E_T, so the worst-case round-off error from the non-zero banded component dominates the error caused by truncating the tail of H. The minimum number of fractional bits needed to satisfy Requirement 2.5.1 can be read from the intersection of $\|E_p\|_2$ with η in Figure 7.6. It is also of note that the horizon length was not used in any of the calculations in Theorem 7.2.2, meaning the computed bit length is valid for any horizon.

From Theorems 7.2.1 and 7.2.2, we can see a correlation between the problem data and the required data type size. To examine this, we performed experiments where Q was scaled by α and R was held constant, with the results reported in Figures 7.7 and 7.8. Note that for $\alpha > 10$ the eigenvalues of H leave the interval $(0, 1)$, so we introduce a scaling factor $c = \frac{1}{1.1\lambda_{max}(H)}$ to scale the matrices H and J in (2.7a) to bring the eigenvalues of H into the interval $(0, 1)$ before performing the analysis.

It can be seen from Figure 7.7a that for small values of α, the number of fraction bits needed is small, with the structure-exploitation in Theorem 7.2.2 producing a savings of nearly 40% compared to the generic model from Theorem 7.2.1. Additionally, once α becomes large and



(a) Fixed-point bits required



(b) Rounding stability margin

Figure 7.7: The effect of scaling the cost function on the rounding margin and fixed-point representation. Q was scaled by α while R was held constant with a horizon of $N = 20$. ©2019 IEEE

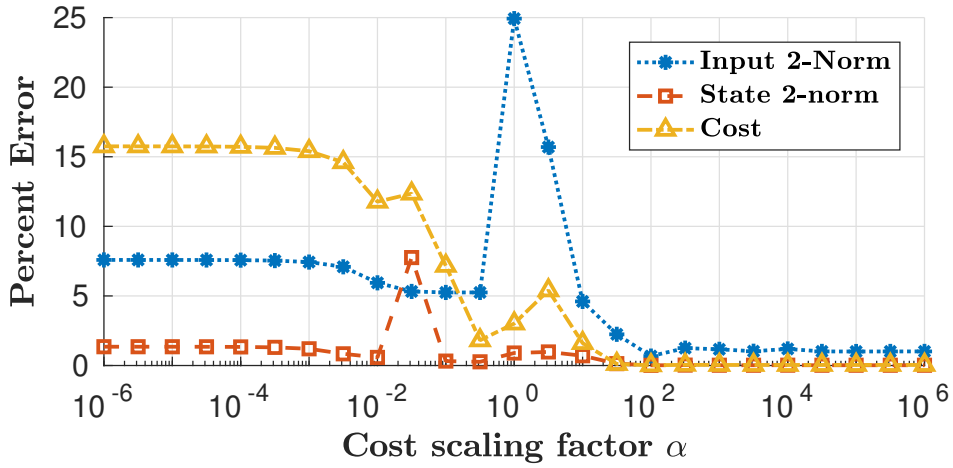


Figure 7.8: Percent error of fixed-point versus double-precision floating-point implementations of FGM using round to zero and the fractional lengths from Theorem 7.2.2. ©2019 IEEE

the scale factor c is needed, the number of integer bits decreases. This decrease in integer bits offsets the increase in fractional bits leaving the overall number of bits nearly the same as the number needed for small α.

Reducing the fractional length to the minimum needed can lead to a large decrease in the required resources, power requirements, and solution times for FPGA implementations compared with simply choosing either floating-point or a larger fixed-point data type to get stability. This can be seen in Table 7.1, where we present results for an FPGA implementation of the FGM using ProtoIP [95] targeting the Xilinx Zynq 7020 with a clock speed of 100MHz. An implementation with $f = 12$ uses 77% fewer memory blocks, 33% fewer Digital Signal Processing (DSP) computation blocks, and 25% less time when compared with an implementation for $f = 26$. The implementation with $f = 12$ also uses 85% fewer memory blocks and takes 45% less time when compared to a single-precision floating-point implementation. This lower precision can lead to convergence to a suboptimal solution though, with experiments showing that using the minimum data type can lead to as much as a 15% increase in the cost and degraded closed-loop performance, as shown in Figure 7.8.

7.4 Conclusions

In this chapter, we developed two methods to size the data types in the FGM to ensure the fixed-point Hessian has its eigenvalues in the unit circle. The first method provides a generic way that is capable of working with an arbitrary Hessian, but only provides horizon-dependent sizing rules. The second method was a structure-exploiting parametric model that requires a Schur-stable predicted system, but exploits that stability and the block Toeplitz structure

Table 7.1: Resource usage for FGM implemented using ProtoIP [95] on a Zynq 7020 at 100MHz with $N = 20, i = 5$. ©2019 IEEE

Fractional Length	**Logic Resources**				**Power** (mW)	**Solve Time** (μs)
	LUT	**FF**	**DSP**	**BRAM**		
f=12	947	768	4	2	20	532.17
f=16	1,136	912	4	2	25	612.17
f=21	887	1,033	8	8	43	701.77
f=26	993	1,237	12	9	48	701.77
float32[1]	2,161	1,545	5	14	51	982.17

[1] Single-precision floating-point representation

of the condensed Hessian to accurately estimate the round-off error caused by cutting off the matrix at horizon N. We then proposed the use of the parametric model for computing horizon-independent sizing rules for the round to nearest and round to zero rounding modes. The numerical examples showed that the sizing rules based on the parametric model can reduce the number of fractional bits needed by 30–45%, and reduce the hardware usage and solution time by up to 77% and 25%, respectively, for an implementation of the FGM on an FPGA.

These sizing rules also come at a cost though, with the data types created using these rules leading to a difference in the performance of the solver of up to 20% when compared to standard double precision arithmetic. This shows that future work is needed to explore the relationship between the data type size and the suboptimality/performance of the optimization solution, and the resulting closed-loop performance of the system.

An additional area that needs further work is the creation of stability guarantees in the presence of round-off errors for the optimization solver with the projection operation. The current stability analysis in [83] ignores the projection operation when examining the stability of the FGM. Future work could explore the application of robust control ideas, such as passivity analysis and Integral Quadratic Constraints, to the analysis of iterative solvers that can be modelled as dynamical systems. These analysis techniques could then provide stability guarantees for the algorithms under fixed-point arithmetic with the projection operation.

Part III

Derivative-free Optimization for NMPC

Chapter 8

Mesh Adaptive Direct Search algorithm[†]

Derivative-free Optimization (DFO) methods such as the Mesh Adaptive Direct Search (MADS) are a class of optimization solvers that only use information about the cost function value at strategically sampled points to locate the optimal solution. This means DFO solvers require no information about the derivatives or Hessian of the optimization problem in the solver computations, allowing the use of DFO methods with problems that have nonsmooth or blackbox functions. This makes DFO solvers popular in fields such as oil and gas well planning [75, 130], optimizing production processes [11, 45, 51, 52], and the tuning of algorithm parameters [5, 10, 13, 100], where the cost function is computed using long/complex/blackbox simulations.

This chapter provides an introduction to the MADS algorithm, which is an extension of the Generalized Pattern Search (GPS) derivative-free method, and was first proposed for continuous real variables in [6]. Over the years, MADS has been extended to work with other types of optimization variables (such as periodic [9], categorical [1], and granular and integer/binary [15]), model-based techniques and surrogate functions [39], and multi-objective optimization [14, 23].

[†]Material presented in this chapter has been published in the following work:
I. McInerney, L. Nita, Y. Nie, A. Oliveri, and E. C. Kerrigan. Towards a framework for nonlinear predictive control using derivative-free optimization. In *7th IFAC Conference on Nonlinear Model Predictive Control*, Bratislava, Slovakia, 2021. IFAC.

The presentation of the MADS algorithm in this chapter combines the ideas from several works into a single statement of the algorithm, and provides a discussion on two techniques for implementing constraints in MADS. The notation used in this chapter has been slightly modified from the original MADS papers to make it internally consistent and to allow for a clearer description of the NMPC formulation we will introduce in Chapter 9.

8.1 The MADS algorithm

The optimization problem that is solved using MADS is

$$\begin{aligned} &\min_{c} \mathcal{F}(c) \\ &\text{s.t. } (c, w) \in \Omega \coloneqq \{c \in \mathbb{R}^m, w \in \mathbb{R}^j : \omega_i(c, w) \leq 0, \forall i \in K\}, \end{aligned}$$

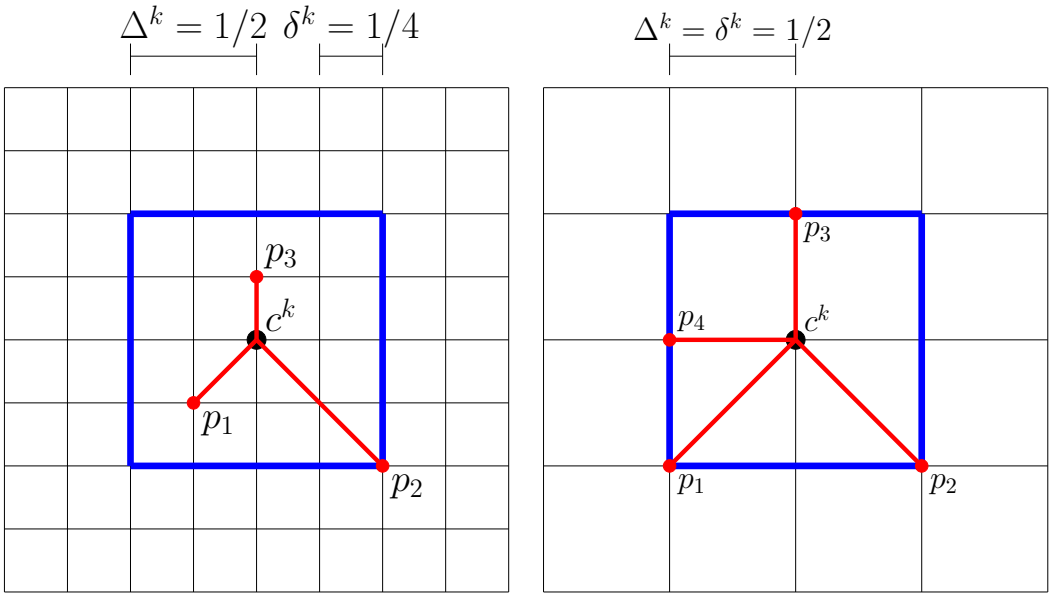
where $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function that computes the cost of the optimization problem. The vector $c \in \mathbb{R}^m$ contains the optimization variables that form the *search space* for the algorithm, and $w \in \mathbb{R}^j$ is a vector used to pass values from the blackbox computation of $\mathcal{F}$ to the constraints. The constraint set Ω is defined by the functions $\omega_i(\cdot)$ spanning the search space, and the variables from the cost function $\mathcal{F}$, with K the set of indices for the functions $\omega_i(\cdot)$ that define Ω.

Pattern search methods sample the search space in each iteration at a set of points called *poll points*. The poll points are located on a *mesh* of size δ^k inside a *frame* of size Δ^k around the current iterate c^k, as shown in Figure 8.1. The poll point with the lowest cost value is chosen as the next iterate and the process repeats.

The overall MADS algorithm with the constraints implemented using the progressive barrier method that will be introduced in Section 8.2.2 is given in Algorithm 8.1, and consists of three main parts: (i) a search phase, (ii) a poll phase, and (iii) a mesh/frame adaptation phase. In the search phase, a set of points $\mathbb{S}^k$ is generated, with no restriction on the method used to generate the points. The cost function is evaluated at each point $s \in \mathbb{S}^k$, and if any have a lower cost value than the current iterate c^k, the mesh and frame sizes are enlarged, and the search phase is repeated. If no better point is found, the poll phase is started.

The poll phase begins by generating a set of polling directions $\mathbb{D}^k$ that form a positive spanning set of the search space*. The set $\mathbb{D}^k$ is then used to create the set $\mathbb{P}^k$ of $m+1$ or more poll points centered at the current iterate c^k and contained inside the frame. The cost function is then evaluated at each poll point $p \in \mathbb{P}^k$. MADS can perform these evaluations in parallel

*A positive spanning set $\mathbb{D}$ of the set $\mathbb{X}$ is a set of n vectors $d_1, \ldots, d_n$ where the positive span of all the vectors is $\mathbb{X}$ — i.e. $\mathbb{X} = \left\{\sum_{i=1}^{n} k_i d_i : k_i \geq 0 \ \forall i \in \{1, \ldots, n\}\right\}$.



(a) MADS after 1 iteration **(b)** GPS after 1 iteration

Figure 8.1: The mesh (solid black lines spaced δ^k apart) and frame (region of size Δ^k inside the solid blue lines) for GPS and MADS after one successful iteration. ©2021 the authors of [119].

and in an opportunistic fashion, meaning it can evaluate multiple poll points at once and end the current iteration immediately after finding a point with a cost value lower than the current iterate.

The final step in the MADS algorithm is to adapt the mesh and frame based on the results of the poll phase. If a lower cost value has been found, then the mesh and frame get enlarged by the adjustment parameter τ to allow for a faster exploration of the search space. If no lower cost value is found, then the mesh and frame sizes are shrunk by τ to concentrate the search closer to the current iterate. MADS can be terminated once the mesh size reaches a pre-determined threshold, meaning that there is no better point within that distance of the current iterate.

The poll and mesh adaptation phases are the only two phases required in every iteration of the MADS algorithm to get convergence. The search phase is optional; however, its use can help MADS escape local minimums and speed up the convergence of the algorithm by allowing for more varied exploration of the search space and the exploration of regions in the search space further away from the current iterate faster than the frame can enlarge to include those regions.

Algorithm 8.1 The Mesh Adaptive Direct Search with progressive barrier constraints [8].

Let: $\prec_{\mathcal{F}}$ and $\prec_{\mathcal{H}}$ be as given in Definitions 8.2.1a and 8.2.1b, respectively
Require: $\Delta^0 \in (0, \infty)$ ▷ Initial frame size
Require: $\tau \in (0, 1)$ ▷ Mesh size adjustment parameter
Require: $\epsilon_{stop} \in [0, \infty)$ ▷ Stopping tolerance
Require: c_f^0 **and\or** c_i^0 ▷ Initial mesh centers

$k \leftarrow 0$
1: **while** $\Delta^k \geq \epsilon_{stop}$ **do**
2: $\mathbb{V}^k \leftarrow \emptyset$ ▷ Set of improving poll points
3: $\delta^k \leftarrow \min\{\Delta^k, (\Delta^k)^2\}$ ▷ Compute mesh size

4: **1) Search Phase:**
5: Generate the search points $\mathbb{S}^k$
6: **for all** $s \in \mathbb{S}^k$ **do**
7: Compute $\mathcal{F}$ and $\mathcal{H}$ at s
8: **if** $s \prec_{\mathcal{F}} c_f^k$ **then**
9: $c_f^{k+1} \leftarrow s$ ▷ Dominating
10: **goto** line 35
11: **else if** $s \prec_{\mathcal{H}} c_i^k$ **then**
12: $c_i^{k+1} \leftarrow s$, $\eta^{k+1} \leftarrow \mathcal{H}(s)$ ▷ Dominating
13: **goto** line 35
14: **end if**
15: **end for**

16: **2) Poll Phase:**
17: Generate a positive spanning set $\mathbb{D}^k$
18: $\mathbb{P}^k \leftarrow \{c_f^k + \delta^k d : d \in \mathbb{D}^k\} \cup \{c_i^k + \delta^k d : d \in \mathbb{D}^k\}$ ▷ Compute poll points
19: **for all** $p \in \mathbb{P}^k$ **do**
20: Compute $\mathcal{F}$ and $\mathcal{H}$ at p
21: **if** $p \prec_{\mathcal{F}} c_f^k$ **then**
22: $c_f^{k+1} \leftarrow p$ ▷ Dominating
23: **goto** line 35
24: **else if** $p \prec_{\mathcal{H}} c_i^k$ **then**
25: $c_i^{k+1} \leftarrow p$, $\eta^{k+1} \leftarrow \mathcal{H}(p)$ ▷ Dominating
26: **goto** line 35
27: **else if** $\mathcal{H}(p) < \eta^k$ **then**
28: $\mathbb{V}^k \leftarrow \mathbb{V}^k \cup \{p\}$ ▷ Improving
29: **end if**
30: **end for**

31: **if** Improving **then**
32: $v \leftarrow \text{argmax}\{\mathcal{H}(v) : \mathcal{H}(v) < \eta^k, v \in \mathbb{V}^k\}$
33: $c_i^{k+1} \leftarrow v$, $\eta^{k+1} \leftarrow \mathcal{H}(v)$
34: **end if**

35: **3) Mesh/Frame Update:**
36: **if** Dominating **then**
37: $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$ ▷ Expand frame size
38: **else if** Improving **then**
39: $\Delta^{k+1} \leftarrow \Delta^k$ ▷ Frame does not change
40: **else**
41: $\Delta^{k+1} \leftarrow \tau\Delta^k$ ▷ Shrink frame size
42: **end if**

43: $k \leftarrow k + 1$
44: **end while** ▷ Variables that are not set retain the same value in the next iteration.

8.2 Constraint handling

Constraints inside MADS can be handled in many ways, with two possible implementations being

- directly constrain the poll points and either reject any that violate the constraints or modify the poll points to satisfy the constraints, or
- compute a metric indicating the amount of constraint violation and use the amount of violation to change how the algorithm searches.

8.2.1 Extremal barrier constraints

The general idea behind the extremal barrier constraint method is to redefine the cost function to be an extremal barrier function that returns infinity when the constraints are violated and returns the cost function value otherwise, or put formally

$$\mathcal{F}_\Omega(c) \coloneqq \begin{cases} \mathcal{F}(c), & \forall (c, w) \in \Omega, \\ \infty, & \forall (c, w) \notin \Omega. \end{cases} \tag{8.1}$$

The barrier term (8.1) ensures that MADS optimizes over only the feasible points by forcing all infeasible points to have infinite cost, and thus to be effectively ignored by the algorithm.

If the constraint functions ω only require the poll point for their computation (i.e. only require c and not w), then the extremal barrier method lets MADS to assign an infinite cost to infeasible points without evaluating the cost function at them, reducing the number of function evaluations performed. If instead the constraint functions require both the poll point c and variables w, the cost function must still be evaluated at the poll point and passed through the barrier term (8.1) to determine the cost.

While easy to implement, the extremal barrier method has several drawbacks, including

- no information is provided to guide MADS from infeasible points to the feasible space, and
- a feasible starting point is required.

8.2.2 Progressive barrier constraints

Alternately, a method known as the progressive barrier technique was proposed in [7] that uses knowledge of the constraint violation to have both a feasible and infeasible iterate in the algorithm. In each iteration, the search phase tries to find a point that is better than either

the feasible or infeasible iterate through its exploration. In the poll phase, the n_p polling directions are used to generate a set of n_p poll points around both iterates (giving up to $2n_p$ total poll points to evaluate in each iteration). By tracking the best infeasible iterate, MADS can overcome the drawbacks of the extremal barrier method at the expense of doubling the required computations in each iteration.

We define the set $\Psi \supset \Omega$ as the relaxed constraint set, where some constraints are implemented using a progressive barrier form and others use an extremal barrier form. The function $\mathcal{H} : \mathbb{R}^{k+j} \to \mathbb{R}$ is then used to compute the constraint violation, and is defined as

$$\mathcal{H}(c,w) \coloneqq \begin{cases} 0, & \text{if } (c,w) \in \Omega, \\ \sum_{l \in K_\Psi} \left(\max\{\omega_l(c,w), 0\}\right)^2, & \text{if } (c,w) \in \Psi \setminus \Omega, \\ \infty, & \text{otherwise,} \end{cases}$$

where K_Ψ is the set of indices for the constraints $\omega(\cdot)$ that use the progressive barrier form.

Inside MADS, the progressive barrier relies on the notion of a dominating point, which is a point that is better than the others in a specific sense.

Definition 8.2.1a (Dominating Feasible Point)**.** *The feasible point $p \in \Omega$ dominates the point $y \in \Omega$, denoted $p \prec_{\mathcal{F}} y$, when $\mathcal{F}(p) < \mathcal{F}(y)$.*

Definition 8.2.1b (Dominating Infeasible Point)**.** *The infeasible point $p \in \Psi \setminus \Omega$ dominates the point $y \in \Psi \setminus \Omega$, denoted $p \prec_{\mathcal{H}} y$, when $\mathcal{F}(p) \leq \mathcal{F}(y)$ and $\mathcal{H}(p, w_p) \leq \mathcal{H}(y, w_y)$ (with at least one strict inequality).*

Definition 8.2.1a means that for feasible points, a point is dominating when its cost value is smaller than the other point, while for infeasible points Definition 8.2.1b means that the dominating point has a cost value and a constraint violation that are not bigger than the other point.

An iteration of the MADS algorithm is then one of three types: *dominating*, *improving* or *unsuccessful*.

Definition 8.2.2a (Dominating Iteration)**.** *A dominating iteration of MADS is when a point $y \in \Omega$ that dominates the feasible iterate c_f^k or a point $y \in \Psi \setminus \Omega$ that dominates the infeasible iterate c_i^k is found.*

Definition 8.2.2b (Improving Iteration)**.** *An improving iteration of MADS is when a non-dominating infeasible point $y \in \Psi \setminus \Omega$ with $0 < \mathcal{H}(y) < \eta$ is found.*

Definition 8.2.2c (Unsuccessful Iteration)**.** *An unsuccessful iteration of MADS is when neither a dominating nor an improving iteration occurs.*

Essentially, a dominating iteration for a feasible point is when MADS finds a new feasible point that has a lower cost value than the current feasible iterate. For an infeasible point, a dominating iteration occurs when MADS finds an infeasible point that has either a lower constraint violation with the same cost value or a lower cost value with the same constraint violation as the current infeasible iterate. In both cases, the iterates are updated, and the mesh is expanded to search in a larger region around the new iterates. In an improving iteration, no points with a lower cost were found, but an infeasible point with a constraint violation smaller than the penalty parameter η was found. In this case, the new point becomes the next infeasible iterate, the mesh size remains unchanged, and η is set to the constraint violation at the new infeasible iterate. All other iterations are considered unsuccessful, and the mesh will be shrunken to search in a region closer to the iterates in the next iteration.

The penalty parameter η is nonincreasing with the iteration number, starting at ∞ and decreasing to 0 as the algorithm runs. This means that at the beginning, MADS is prioritizing the search for the lowest cost value by allowing large constraint violations, but over time η decreases and forces the infeasible iterates to move towards the feasible space. The penalty barrier constraint method behaves similarly to a filter method in other optimization algorithms, such as SQP, but requires the penalty parameter to be only nonincreasing and not strictly decreasing.

8.3 Meshing

In MADS, the frame and mesh are controlled by different parameters (Δ^k and δ^k, respectively), with

$$\delta^k = \min\{\Delta^k, (\Delta^k)^2\} \tag{8.2}$$

normally used. The main difference between MADS and GPS is that in GPS $\delta^k = \Delta^k$ at all times. By allowing the mesh to shrink faster than the frame size in MADS, more possible polling points are created around the current iterate c^k. For example, Figure 8.1a shows the result of one iteration of MADS using update rule (8.2). In this case there are 24 possible polling points for MADS to choose in Figure 8.1a, versus 8 for GPS in Figure 8.1b.

8.3.1 Granular variables

An advantage to using the MADS algorithm is the native support for discrete/granular variables in the algorithm itself, which was first described in [15]. A granular variable (also called controlled decimal variables) is one that has a predetermined minimum spacing between the possible values. For instance, a variable that has granularity 0.01 will only take on values

0.01 apart, making the legal values 1.01, 1.02, 1.03, etc. Granular variables are a more general type, and can be used to represent integer variables (granularity of 1), and binary variables (granularity of 1 with a lower bound of 0 and upper bound of 1).

Granular variables in MADS are represented internally using two values $a \in \{1, 2, 5\}$ and $b \in \mathbb{Z}$. These two values are used to define the discrete set

$$\mathbb{G} \coloneqq \{a \times 10^b \colon a \in \{1, 2, 5\},\ b \in \mathbb{Z}\}.$$

Two operations are then defined to map a value from $\mathbb{G}$ to another value in $\mathbb{G}$:

$$\text{Increase}(a \times 10^b) \coloneqq \begin{cases} 2 \times 10^b & \text{if } a = 1, \\ 5 \times 10^b & \text{if } a = 2, \\ 1 \times 10^{b+1} & \text{if } a = 5, \end{cases}$$

$$\text{Decrease}(a \times 10^b) \coloneqq \begin{cases} 5 \times 10^{b-1} & \text{if } a = 1, \\ 1 \times 10^b & \text{if } a = 2, \\ 2 \times 10^b & \text{if } a = 5. \end{cases}$$

Granular variables have their mesh constrained to only take values from a slightly modified version of the discrete set $\mathbb{G}$. In order to handle different granularity levels for each variable, a granularity parameter δ_i^{min} for the ith variable is introduced. The poll parameter for the variable i is then constrained to be in the set

$$\Delta_i^k \in \mathbb{G}(\delta_i^{min}) \coloneqq \{a \times 10^b \times \delta_i^{min} \colon a \in \{1, 2, 5\},\ b \in \mathbb{Z}\}.$$

The update rules for the granular variables in the mesh/frame update phase are then modified so that the unsuccessful iteration update in Line 41 of Algorithm 8.1 becomes

$$\Delta_i^{k+1} = \delta_i^{min} \times \max\left\{1, \text{Decrease}\left(\frac{\Delta_i^k}{\delta_i^{min}}\right)\right\}, \tag{8.3}$$

and the dominating iteration update rule in Line 37 of Algorithm 8.1 becomes

$$\Delta_i^{k+1} = \text{Increase}\left(\Delta_i^k\right). \tag{8.4}$$

The new unsuccessful update rule (8.3) will continue to shrink the mesh size for the granular variable up until it reaches the minimum granularity — at which point it will stop and maintain a constant mesh size. The new dominating update rule (8.4) will allow the mesh size to continually grow to as large a size as possible, while ensuring the point lies on the defined granular mesh.

The mesh size parameter δ_i is then obtained using

$$\delta_i^k = \delta_i^{min} \times \max\left\{1, 10^{b_i^k - |b_i^k - b_i^0|}\right\}.$$

This mesh size parameter is then used in the creation of the polling set $\mathbb{P}^k$ to project the polling direction d_i onto the granular set for variable i.

8.4 Convergence

The fact that the mesh size in MADS shrinks faster than the frame size means that, as the algorithm converges, the closure of the cone containing all the generated polling directions will be equal to $\mathbb{R}^m$ in the limit (i.e. $\mathcal{D} := \bigcup_{k=1}^{\infty} \mathbb{D}^k = \mathbb{R}^m$), and the set of all polling directions $\mathcal{D}$ is then said to be asymptotically dense. It was shown in [6] that this set of asymptotically dense polling directions allows for the algorithm to converge to a Clarke stationary point with non-negative Clarke derivatives (e.g. a local minimum for the nonsmooth function) when linear constraints are placed on the search space. In contrast, GPS loses the theoretical convergence guarantees when simple bound constraints are applied. Additionally, when a progressive barrier approach is used for handling nonlinear constraints, it was shown in [7] that MADS is still effective at finding the stationary point.

8.5 Conclusions

In this chapter, we presented an overview of the mesh adaptive direct search algorithm for derivative-free optimization. This algorithm is an extension of the classic pattern search method that decouples the mesh size from the frame size and allows for the mesh size to decrease faster than the frame size. This in turn allows for the algorithm to produce search directions surrounding the incumbent point that are dense on the unit sphere — guaranteeing the convergence of MADS to a stationary point.

MADS also has explicit handling of inequality constraints in the algorithm formulation using the progressive barrier method, instead of relying on the extremal barrier method or a penalty barrier approach in the cost function. This allows for better handling of the constraints, and the separation of feasible and infeasible iterates. Finally, MADS also provides support in the underlying algorithm for multiple types of variables, such as granular/binary/integer, categorical, and periodic variables.

Chapter 9

DFO-NMPC problem formulation[†]

In this chapter, we present a novel problem formulation for solving the nonlinear MPC problem (2.1) using the MADS algorithm. This formulation is able to enforce the constraints across the entire time horizon instead of only at certain points, and uses the progressive barrier method described in Section 8.2.2 to implement the constraints instead of a penalty barrier method.

9.1 Proposed formulation

The core of the proposed problem formulation is based on the pattern search method in [53], where the search space for MADS is the set of all input trajectories and a single shooting simulation of the continuous-time dynamics (2.1b) is done to compute the cost. Instead of the basic pattern search method, the proposed formulation will use the MADS algorithm, increasing the number and diversity of the directions polled and improving the convergence of the algorithm.

Each function evaluation in the proposed framework simulates the system over the time horizon with the chosen input trajectory, and then computes the overall cost value and constraint violation for that trajectory. To easily handle the path constraints and the Lagrange term of the cost, an augmentation scheme is used to introduce new states to the system

[†]Some material presented in this chapter has been published in the following work:
I. McInerney, L. Nita, Y. Nie, A. Oliveri, and E. C. Kerrigan. Towards a framework for nonlinear predictive control using derivative-free optimization. In *7th IFAC Conference on Nonlinear Model Predictive Control*, Bratislava, Slovakia, 2021. IFAC. ©2021 the authors.

dynamics to represent the Lagrange term and the violation of the path constraints. This allows the path constraints to be enforced across the entire horizon instead of only at specific points in the horizon.

9.1.1 Cost functional

The NMPC cost functional (2.1a) is composed of two distinct terms, the Mayer term representing the contribution to the cost of the terminal states/inputs and the Lagrange term representing the contribution to the cost of the full trajectory of the system. In the proposed DFO-NMPC formulation, the cost functional (2.1a) is split into its two terms, and the Lagrange term is handled by augmenting the system dynamics with a new state $\ell(t)$ that represents the value of the Lagrange term at time t. This new state is governed by the dynamics equation

$$\dot{\ell}(t) = L(x(t), u(t), t). \tag{9.1}$$

This state is then computed alongside the trajectory of the system using the same dynamics solver at the same time.

9.1.2 Path constraints

The path constraints (2.1c) are formulated as progressive barrier constraints, putting them inside the set Ψ described in Section 8.2.2. An L_1 measure of constraint violation is used for each constraint, meaning the value reported as the violation experienced by constraint i is

$$v_i = \int_{t_0}^{t_f} \max\{0, g_i(x(t), u(t), t)\} \, \mathrm{d}t. \tag{9.2}$$

To compute the integral (9.2), new states $v(t)$ are added to the system dynamics to represent the constraint violation over time. Those states are governed by the dynamics equation

$$\dot{v}(t) = g^+(x(t), u(t), t) \tag{9.3}$$

where g^+ is the vector function representing the element-wise computation of

$$\max\{0, g_i(x(t), u(t), t)\}.$$

The value of the violation states at the final time (i.e. $v(t_f)$) is then used as the constraint violation measure in a progressive barrier constraint.

Algorithm 9.1 Function evaluation for the DFO formulation of the NMPC problem

Let: c be the point in the search space being evaluated

1: Construct the input trajectory u from c

2: Simulate the augmented dynamics (9.4) using an appropriate solver for the dynamics equations

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} f(\dot{x}(t), x(t), u(t), t) \\ L(x(t), u(t), t) - \dot{\ell}(t) \\ g^{+}(x(t), u(t), t) - \dot{v}(t) \end{bmatrix} \tag{9.4}$$

3: Compute the violation of the boundary conditions

$$v_b = \sum_i \rho_{b_i} |h_i(x(t_0), u(t_0), t_0, x(t_f), u(t_f), t_f)| \tag{9.5}$$

4: Compute the overall constraint violation

$$\mathcal{H} \leftarrow v_b^2 + \sum_i \rho_i (v_i(t_f))^2 \tag{9.6}$$

5: Compute the cost function value

$$\mathcal{F} \leftarrow \Phi(x(t_f), u(t_f), t_f) + \ell(t_f) \tag{9.7}$$

9.1.3 Overall problem formulation

An implementation of the formulation described in this section is given in Algorithm 9.1. The first step is to construct the input trajectory described by the current poll point c. The representation of the input trajectory will depend on the problem, but this formulation can support many different representations, including

- piecewise-constant trajectories with the input values at each sample given by the elements of the poll point,
- an interpolated polynomial with the interpolation points and coefficients given by the elements of the poll point, or
- a feedback policy with the parameters of the policy given by the elements of the poll point.

After constructing the input trajectory, the augmented dynamics (9.4) are simulated over the horizon length using a suitable numerical solver for differential equations.

After the simulation completes, the violation of the boundary conditions is computed in (9.5) followed by the total constraint violation in (9.6), which uses the value of the augmented states v at the final time as the measure of the violation of the path constraints. This formulation also includes weighting parameters ρ to allow the designer to set the priority/influence of each

path and boundary constraint in the overall algorithm. Finally, the value of the NMPC cost function (2.1a) for the selected poll point is computed in (9.7), which computes the Mayer term and then adds the final value of the Lagrange state $\ell(t_f)$.

Note that this formulation can be used with derivative-free solvers other than MADS, provided $\mathcal{F}$ and $\mathcal{H}$ inside the function given in Algorithm 9.1 are mapped to the appropriate variables in the desired algorithm. The main advantage to using MADS with the proposed formulation is that $\mathcal{H}$ is mapped to the special handling of the progressive barrier constraints, allowing for the distinct feasible and infeasible algorithm iterates. If this formulation were to be used with other algorithms (e.g. PSO, evolutionary methods, etc.), the value returned by $\mathcal{H}$ must be incorporated in another way, such as through a penalty term in the cost.

9.2 Discussion and conclusions

The DFO-NMPC formulation that is proposed in this chapter has several advantages over existing formulations, namely:

1. the ability to handle more variable types such as granular, periodic and categorical,
2. explicit handling of the constraints to produce a known-feasible solution,
3. constraint enforcement across the entire horizon, and
4. more accurate cost computation across the entire horizon.

Of these advantages, 1 and 2 are due to the choice of the MADS algorithm as the DFO solver, while 3 and 4 are due to the proposed problem formulation.

9.2.1 Advantages of using MADS

The choice of the MADS algorithm allows the formulation to take advantage of the existing extensions to the algorithm in the literature that handle different kinds of variables in the solver, so the DFO-NMPC formulation no longer has to be modified directly to support those variable types. For instance, if the system model requires granular input variables (e.g. if the inputs are quantized), the proposed DFO-NMPC formulation can just use the MADS technique discussed in Section 8.3.1 that implements granularity in the actual algorithm. This is in contrast to other DFO methods such as PSO that need the search space to be the real numbers, forcing the discretization into the actual simulation/cost function and making the algorithm not aware of it.

An additional advantage of MADS over the other DFO methods is the ability to separate feasible and infeasible iterates using the progressive barrier constraints. As discussed in Section 2.2.1, nearly all existing NMPC formulations for DFO solvers rely on penalty

barrier methods to implement the path constraints, which can complicate the solution of the optimization problem because now the solution will depend on the penalty parameter. For instance, the use of a fixed penalty will lead to simply trading off the constraint violation against the cost and won't ensure the constraints are fully enforced. To enforce the constraints, other NMPC formulations for DFO solvers must then also implement an update rule for the penalty parameter to ensure that the penalty parameter goes to infinity to force the constraint violation to zero. Instead, MADS provides inherent separation of the feasible and infeasible solutions, and also provides a built-in update rule to improve the infeasible solution as the algorithm progresses.

9.2.2 Augmentation scheme

The dynamics augmentation scheme contained in the proposed formulation provides the final two advantages. Using the augmentation scheme, the proposed formulation is able to actually account for the Lagrange cost and constraint violation between sampling intervals. This allows for not only a more accurate approximation of the continuous-time values, but also is more portable to other input representations.

When used with the simulation-based solvers (like single shooting), the existing method of enforcing the constraints at specific time points and integrating the Lagrange cost using a Riemann sum is tied closely to the idea of a piecewise representation of the input trajectory. If instead another representation is used, like an interpolating polynomial or function coefficients, there will no longer be discrete time points along the horizon that provide easy places to enforce the constraints or use as the points in the Riemann sum. Any usage of the existing formulations would then require the creation of arbitrary time points along the horizon at which to enforce the constraints, and a quadrature scheme to compute the Lagrange cost term.

The proposed formulation will instead handle the Lagrange term and constraint enforcement the same way for piecewise input trajectories and other types (like interpolating polynomials). This simplifies the transcription of the problem and implementation of the solver. Additionally, the proposed formulation will place the Lagrange cost and constraint enforcement onto the same mesh as the dynamics, and make the mesh point locations be determined by the dynamics solver instead of a post-processing mesh refinement scheme.

It should be noted that the idea of this augmentation is not actually novel in the world of optimal control. For instance, the idea of augmenting the dynamics with the Lagrange term is included in some optimal control courses, such as [32, Section 3.2.3]. Augmenting the dynamics with constraints was originally proposed by [76], with further work and experimentation in [172, Chapter 4] and [173]. Since then, however, solution methods such

as multiple shooting and direct collocation have instead chosen to enforce the constraints at the shooting nodes and collocation points, respectively, instead of augmenting the dynamics to incorporate them.

While the augmentation scheme seems easy to use and aptly suited for DFO-NMPC, this scheme can also introduce inefficiencies to the solver. By introducing the dynamics equations (9.1) and (9.3), we have made the solvability and the stiffness of the augmented dynamics equation (9.4) be dependent on the Lagrange term and the constraints as well as the original dynamics equations. This means that when the cost or constraints vary over time or are defined by stiff equations, they can cause the dynamics solver to struggle and possibly require smaller step sizes, slowing down the simulation process. Additionally, simply enforcing the constraint satisfaction using a single number at the final time does not provide the underlying optimization solver with any information about the locations of the infeasibility, or which inputs are causing it. Instead, [172, Section 4.3] provides some extensions to the augmentation scheme that introduce additional constraints across the horizon. Adapting these extensions to work with the proposed DFO-NMPC formulation, and adapting the MADS algorithm to better incorporate the information from them, is an open area of research.

Chapter 10

DFO-NMPC for a robust Goddard rocket problem[†]

In this chapter, we demonstrate how to apply the DFO-NMPC formulation from Chapter 9 to an optimal control problem. We present results for several different variations of the problem, including

- with/without path constraints,
- with penalty barrier terms/with the progressive barrier method for constraint handling, and
- with/without granular control variables.

We also present a comparison between a MADS and a PSO implementation of the resulting DFO-NMPC problem.

10.1 Example problem

The example problem used in this chapter is a modified version of the Goddard rocket problem in [132].

[†]Some material presented in this chapter has been published in the following work:
I. McInerney, L. Nita, Y. Nie, A. Oliveri, and E. C. Kerrigan. Towards a framework for nonlinear predictive control using derivative-free optimization. In *7th IFAC Conference on Nonlinear Model Predictive Control*, Bratislava, Slovakia, 2021. IFAC.

Table 10.1: Physical parameters for the rocket dynamics (10.1).

(a) Fixed parameters

Parameter	α	β	ρ_0	d	g_0	T_{max}
Value	2.255×10^{-5}	4.256	$1.225\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}$	17.5 cm	$9.81\,{}^{\mathrm{m}}/_{\mathrm{s}^2}$	1000

(b) Bounds on the uncertain parameters

Parameter	C_{D_l}	C_{D_u}	I_{sp_l}	I_{sp_u}
Value	0.4	0.5	1800.0	2200.0

10.1.1 System dynamics

The rocket is governed by the dynamics equation

$$\dot{x}(t) = f(x(t), T(t)) = \begin{bmatrix} v_v(t) \\ \frac{T(t) - 0.5 C_D \rho(h(t)) \frac{\pi d^2}{4} (v_v(t)^2)}{m(t)} - g(h(t)) \\ -\frac{T(t)}{I_{sp}} \end{bmatrix}, \tag{10.1}$$

with the state vector

$$x(t) = \begin{bmatrix} h(t) & v_v(t) & m(t) \end{bmatrix}'$$

composed of the altitude $h(\cdot)$ in meters, vertical velocity $v_v(\cdot)$ in meters per second and mass $m(\cdot)$ of the rocket in kilograms, respectively. d is the rocket diameter, $g(h) \coloneqq g_0$ is the gravitational constant at altitude h, and

$$\rho(h) \coloneqq \rho_0 \left(1 - \alpha h\right)^{\beta}$$

is the air density at altitude h, with all the parameter values given in Table 10.1a. The parameters C_D and I_{sp} are the drag coefficient and specific impulse, respectively. The thrust $T(\cdot)$ is specified as a function of time that takes values from the set $T(t) \in [0, T_{max}]$. The rocket has bounded uncertainty on the parameters C_D and I_{sp}, where $C_D \in [C_{D_l}, C_{D_u}]$ and $I_{sp} \in [I_{sp_l}, I_{sp_u}]$ with the bounds on the uncertainty given in Table 10.1b.

It was shown in [132, Section 3.5] that the rocket system (10.1) with uncertain parameters C_D and I_{sp} is a monotone system with respect to the thrust input and the uncertain parameters. This means that if the uncertain parameters of the differential equation f are thought of as inputs to the system (with the input vector given by $u(t) \coloneqq \begin{bmatrix} T(t) & C_D & I_{sp} \end{bmatrix}'$), for two state vectors $x_{(1)}$ and $x_{(2)}$ and two input vectors $u_{(1)}$ and $u_{(2)}$, the system dynamics satisfy the property

$$f(x_{(1)}, u_{(1)}) - f(x_{(2)}, u_{(2)}) \in \mathcal{X} \qquad \forall x_{(1)} - x_{(2)} \in \mathcal{X}, u_{(1)} - u_{(2)} \in \mathcal{U},$$

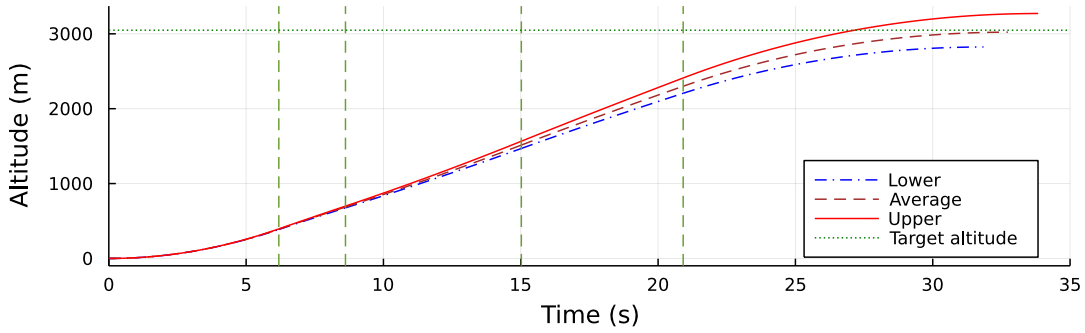


Figure 10.1: Robust tube for the robust Goddard rocket problem when the same input is applied to the upper and lower bound trajectories.

with $\mathcal{X} := \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\leq 0}$ and $\mathcal{U} := \mathbb{R}_{\geq 0} \times \mathbb{R}_{\leq 0} \times \mathbb{R}_{\leq 0}$ for all time. Essentially, the monotonicity property of the rocket system means that for two rockets with the exact same fixed parameters, if rocket 1 has the same or smaller specific impulse and drag parameter than rocket 2 and rocket 1 always has the same or more thrust applied than rocket 2 does, then the altitude and velocity in the state trajectory $x_{(1)}$ of rocket 1 will always be the same as or greater than the altitude and velocity in the state trajectory $x_{(2)}$ of rocket 2, and the mass in $x_{(1)}$ will always be the same as or smaller than the mass in $x_{(2)}$ when the trajectories start at the same point $x_{(1)}(t_0) = x_{(2)}(t_0)$ or when $x_{(1)}(t_0)$ has a larger altitude and velocity and smaller mass than $x_{(2)}(t_0)$.

The monotonicity of the rocket system, coupled with bounds on the uncertain parameters, initial states and the inputs to the system, allows for the creation of a *robust tube* that all system trajectories will lie inside, shown in Figure 10.1. For this rocket problem, the robust tube is bounded by two realizations of the rocket system with the following properties

upper bound system $f_u(\cdot)$ — system $f(\cdot)$ with parameters C_{D_l}, I_{sp_l} and final time t_{f_u},
lower bound system $f_l(\cdot)$ — system $f(\cdot)$ with parameters C_{D_u}, I_{sp_u} and final time t_{f_l},

where $t_{f_l} \leq t_{f_u}$. The state and input trajectories for the upper bound system are $x_u(\cdot)$ and $u_u(\cdot)$, respectively, and the state and input trajectories for the lower bound system are $x_l(\cdot)$ and $u_l(\cdot)$, respectively.

10.1.2 Optimal control problem

The objective of the optimal control problem is to determine the thrust profile of the rocket such that the apogee of the rocket is at the target altitude of 10,000 feet (3,048 meters). The input trajectory is defined as a piecewise constant function, with both the thrust settings T_i and the switching times σ_i as optimization variables. The input trajectory has 5 phases, with

the first phase forcing the thrust to be T_{max} and the last phase forcing the thrust to be 0, giving the piecewise-constant thrust function

$$T(t) \coloneqq \begin{cases} T_{max} & \text{if } 0 \leq t < \sigma_1, \\ T_2 & \text{if } \sigma_1 \leq t < \sigma_2, \\ T_3 & \text{if } \sigma_2 \leq t < \sigma_3, \\ T_4 & \text{if } \sigma_3 \leq t < \sigma_4, \\ 0.0 & \text{if } \sigma_4 \leq t. \end{cases} \tag{10.2}$$

The uncertainty in the dynamics model is handled by implementing a robust optimal control problem that exploits the monotone nature of the dynamics to only optimize two trajectories: the upper bound system and the lower bound system. By applying the same thrust input and the same initial state to the upper and lower bound systems, the monotonicity of the rocket system guarantees that any trajectory of the system with parameters that fall into the disturbance set will lie within the upper and lower bound systems (i.e. the robust tube).

This robust optimal control problem is expressed using a min-max formulation, where the overall objective of the OCP is to minimize the largest of the differences between the apogee of the two bound trajectories and the target altitude. The optimization uses the two bound trajectories instead of optimizing the average trajectory (which is formed using the average of the bounds for the uncertain parameters) because the average trajectory is not actually the center of the robust tube. This can be seen in Figure 10.1, where the apogee of the average trajectory is 3022.29 m and the center of the robust tube is at 3047.99 m. This means that optimizing to make the apogee of the average trajectory be the target altitude would introduce a bias in the optimal solution that would lead to some disturbance realizations overshooting the target altitude by a larger amount than if the robust tube were used.

The rocket will only be controlled until its apogee is reached, making the comparison between the attained altitude and target altitude occur at the final time points t_{f_l} and t_{f_u} for the lower and upper bound trajectories, respectively. The complete OCP for the robust rocket control problem is then,

$$\min_{T,\sigma} \max_{x_l,x_u} \{|h_u(t_{f_u}) - 3048|, |3048 - h_l(t_{f_l})|\} \tag{10.3a}$$

$$\text{s.t. } \dot{x}_l(t) = f_l(x_l(t), T(t)),\ \dot{x}_u(t) = f_u(x_u(t), T(t)) \tag{10.3b}$$

$$x_l(0) = x_u(0) = \begin{bmatrix} 0 & 0 & 33.5 \end{bmatrix}' \tag{10.3c}$$

$$v_{v_l}(t_{f_l}) = v_{v_u}(t_{f_u}) = 0 \tag{10.3d}$$

$$v_{v_l}(t) \leq 150 \qquad \forall t \in [0, t_{f_l}] \tag{10.3e}$$

$$v_{v_u}(t) \leq 150 \qquad \forall t \in [0, t_{f_u}] \tag{10.3f}$$

$$m_l(t_{f_l}) \geq 26,\ m_u(t_{f_u}) \geq 26 \tag{10.3g}$$

$$0 \leq \sigma_1 \leq \sigma_2 \leq \sigma_3 \leq \sigma_4 \leq \min\{t_{f_l}, t_{f_u}\} \tag{10.3h}$$

$$T_i \in [0, T_{max}] \qquad \forall i \in \{2, 3, 4\} \tag{10.3i}$$

where $T(\cdot)$ is the thrust function defined in (10.2). Both the upper and lower bound trajectories use the same thrust function, but each have their own final times given by t_{f_u} and t_{f_l} for the upper and lower bound trajectories, respectively. As an example path constraint, the rocket's velocity is constrained to be less than 150 $^{\text{m}}/_{\text{s}}$ to represent a modeling limitation where the bounds on the uncertainty for the drag coefficient are only known below 150 $^{\text{m}}/_{\text{s}}$.

Note that the objective function (10.3a) is non-differentiable, so this problem is well suited for a DFO solver since using conventional gradient-based methods would only be possible after a reformulation, which could increase the dimensions and complexity of the problem.

10.1.3 Transcription to DFO-NMPC

To convert the Goddard rocket problem from Section 10.1.2 into the DFO-NMPC framework from Chapter 9, the OCP (10.3) was transcribed into a MADS optimization problem as described in this section.

Search space

The first step in transcribing the rocket problem to a MADS optimization problem is to define the search space for the MADS solver. Since the initial phase is forced to start at time $t = 0$ and to have a thrust of T_{max}, and the final phase is forced to have a thrust of 0%, the thrust values for those phases will not be included in the search space, and instead will be hard-coded in the simulation as $T_1 = T_{max}$, and $T_5 = 0.0$.

Since the rocket problem is a free final time problem, an additional consideration is how the final times t_{f_l} and t_{f_u} should be represented in the optimization problem. In traditional methods, such as direct collocation or multiple shooting with a derivative-based solver, the final time would be added as a variable in the optimization solver by forcing a collocation point/shooting node at the final time. This collocation point/shooting node would then provide the mesh point where the final boundary conditions are enforced.

In this transcription method, the final times will not be included as variables in the optimization search space, and will instead be determined by the dynamical simulation. The OCP (10.3) includes a final state equality constraint (10.3d) on the velocity, which when coupled with the choice of a specific input trajectory, will force there to be a single unique final time for each simulation. This means that if the final time were to be included in the

search space, the problem would end up being over-defined. The search space of the over-defined optimization problem would then be composed of partially-disconnected feasible sets, where the variables representing the final times will have a single feasible value they can take that is dependent on all the other optimization variables.

Overall, this makes the search space for the MADS problem

$$c \coloneqq \begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{bmatrix}' = \begin{bmatrix} \sigma_1 & T_2 & \sigma_2 & T_3 & \sigma_3 & T_4 & \sigma_4 \end{bmatrix}'.$$

Constraint handling

To enforce the path constraints in the OCP (10.3), the state augmentation scheme described in Section 9.1.2 is used with the progressive barrier formulation of MADS.

The constraints enforcing the ordering of the switch times and the bounds on the thrust are strictly input constraints and do not require any state information to be computed. Therefore, there is no need to augment the dynamics with new states to compute their violation, so they are included as normal progressive barrier constraints, with the ω functions shown in (10.5a)–(10.5d) and (10.5e)–(10.5j) for the switch time constraints (10.3h) and thrust constraints (10.3i), respectively.

The boundary constraints on the final mass (10.3g) and the path constraints on the vertical velocity (10.3e) and (10.3f) depend on state information, and therefore information from the dynamics simulation performed in the cost function is needed to compute the constraint satisfaction. Since the velocity path constraints must be enforced across the entire horizon, the augmentation scheme is used to introduce two new states g_l and g_u to track the constraint violation of the velocity constraints for the lower and upper bound trajectories, respectively. Those two states have the dynamics

$$\begin{aligned} \dot{g}_l(t) &= \max\{0, v_{v_l}(t) - 150\}, \\ \dot{g}_u(t) &= \max\{0, v_{v_u}(t) - 150\}. \end{aligned}$$

The value of these states at the final times t_{f_l} and t_{f_u} will be used as the constraint violation in the progressive barrier constraints (10.5m) and (10.5n), so the values $g_u(t_{f_u})$ and $g_l(t_{f_l})$ are included in the w vector of simulation variables computed by the function $\mathcal{F}$.

Since the mass constraint is a terminal inequality constraint, there is no need to augment the dynamics to track the constraint violation over the entire horizon. Instead, only the value of the mass states at the final times need to be included in the w vector for use in the

progressive barrier constraints (10.5k) and (10.5l). Put together, the w vector computed by the cost function $\mathcal{F}$ and then used when computing the progressive barrier constraints is

$$w \coloneqq \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}' = \begin{bmatrix} g_l(t_{f_l}) & g_u(t_{f_u}) & m_l(t_{f_l}) & m_u(t_{f_u}) \end{bmatrix}'. \tag{10.4}$$

Overall, this leads to the following 14 progressive barrier constraint functions

$$\omega_1(c, w) = -c_1, \tag{10.5a}$$
$$\omega_2(c, w) = c_1 - c_3, \tag{10.5b}$$
$$\omega_3(c, w) = c_3 - c_5, \tag{10.5c}$$
$$\omega_4(c, w) = c_5 - c_7, \tag{10.5d}$$
$$\omega_5(c, w) = -c_2, \tag{10.5e}$$
$$\omega_6(c, w) = c_2 - T_{max}, \tag{10.5f}$$
$$\omega_7(c, w) = -c_4, \tag{10.5g}$$
$$\omega_8(c, w) = c_4 - T_{max}, \tag{10.5h}$$
$$\omega_9(c, w) = -c_6, \tag{10.5i}$$
$$\omega_{10}(c, w) = c_6 - T_{max}, \tag{10.5j}$$
$$\omega_{11}(c, w) = \max\{0, 26.0 - w_3\}, \tag{10.5k}$$
$$\omega_{12}(c, w) = \max\{0, 26.0 - w_4\}, \tag{10.5l}$$
$$\omega_{13}(c, w) = w_1, \tag{10.5m}$$
$$\omega_{14}(c, w) = w_2, \tag{10.5n}$$

with 10 requiring only information about the current poll point and the other 4 requiring values computed during the dynamics simulation.

Cost function and dynamics simulation

The OCP cost function (10.3a) only contains a Mayer term to penalize the altitude difference at the final time, and does not contain a Lagrange term. Since no Lagrange term is included, there is no need to augment the dynamics with a cost integral state ℓ.

The function $\mathcal{F}$ that is used as the cost function for MADS is then composed of three steps

1. simulate the augmented dynamics for the lower and upper bound systems,
2. compute the value of the Mayer term, and then
3. form the vector w (10.4) of the simulation variables for the constraints.

The value of the Mayer term is then returned to MADS as the cost value at the current poll point, and the vector w is returned to MADS to be used in the evaluation of the progressive barrier constraints (10.5) at the current poll point.

In the dynamics simulation, two dynamics solvers are used to compute the state trajectory of both the upper and lower bound trajectories of the rocket. One solver will compute the state trajectory and constraint violation of the lower bound system using the augmented dynamics

$$\begin{bmatrix} \dot{x}_l(t) \\ \dot{g}_l(t) \end{bmatrix} = \begin{bmatrix} f_l(x_l(t), T(t)) \\ \max\{0, v_{v_l}(t) - 150\} \end{bmatrix}, \tag{10.6}$$

while the other will compute the state trajectory and constraint violation of the upper bound system using the augmented dynamics

$$\begin{bmatrix} \dot{x}_u(t) \\ \dot{g}_u(t) \end{bmatrix} = \begin{bmatrix} f_u(x_u(t), T(t)) \\ \max\{0, v_{v_u}(t) - 150\} \end{bmatrix}. \tag{10.7}$$

These solvers will each terminate when their respective state vectors satisfy the boundary condition (10.3d) (i.e. the rocket has a vertical velocity of 0).

10.2 Experimental configuration

The numerical results presented in this chapter were computed using Julia 1.6.1 on a computer with an Intel® Core™ i7-10710U CPU at 1.10 GHz with 64 GB of RAM.

The augmented dynamics systems (10.6) and (10.7) were solved using the trapezoidal method solver from version 6.19.0 of the DifferentialEquations.jl Julia package [147] with a minimum time step size of 10×10^{-10}.

The PSO solver inside version 1.4.1 of the Optim.jl Julia package [127] was used to compute all the numerical results for the particle swarm version of the DFO-NMPC framework. The PSO solver was run using the default settings and the default stopping criteria.

The MADS solver inside the DirectSearch.jl* Julia package was used to compute all the MADS numerical results. The MADS solver was run using the default settings of the unit sphere polling method with an anisotropic mesh [15] with no search phase and with sequential evaluation of the poll points (i.e. no parallelism). The mesh size convergence criteria was left at its default setting of a minimum mesh size of 1.11×10^{-16}, and the maximum number of iterations and function evaluations were set to 10,000 and 50,000, respectively (instead of their defaults of 1,000 and 5,000, respectively).

*Available at https://github.com/ImperialCollegeLondon/DirectSearch.jl

Note that in the following numerical experiments, the thrust levels for the phases in the input trajectory are reported as being percentages of the maximum thrust T_{max}.

10.3 Solution without velocity path constraints

The first set of numerical results is for the OCP (10.3) solved without the velocity path constraints (10.3e) and (10.3f) being enforced, but with the remaining constraints enforced using the progressive barrier method with barrier functions (10.5a)–(10.5l). The results for this example can be seen in Figure 10.2, with a summary of the results and solver status in Table 10.2.

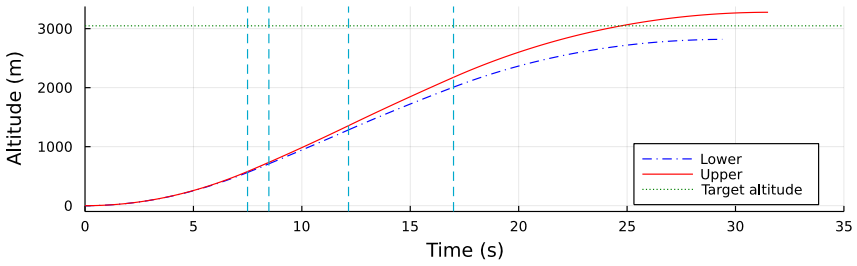
Overall, MADS was able to solve the DFO-NMPC problem successfully in 54 iterations and only evaluated the cost function using Algorithm 9.1 855 times (running 1710 dynamical simulations). MADS was able to converge to a solution that had a cost of 229.52, and terminated automatically once the mesh precision stopping criteria was satisfied. The MADS solver ran for 1.69 s, with the majority of this time spent performing the cost function computation and the dynamical simulations and only 0.0033 s spent in the internal MADS computations.

As can be seen in the altitude plot in Figure 10.2a and the results in Table 10.2a, the upper and lower bound trajectories produce a robust tube with the center exactly at the target altitude of 3048 m. However, examining the velocity of the rocket in Figure 10.2b shows that the thrust trajectory required to produce this robust tube will lead to the rocket attaining a velocity of more than 150 $^{\text{m}}/_{\text{s}}$ during the flight, with the profile for the amount of the cumulative constraint violation above 150 $^{\text{m}}/_{\text{s}}$ shown in Figure 10.2e.

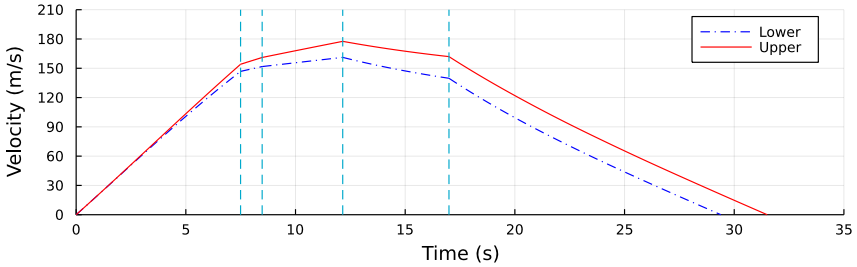
The optimal thrust trajectory found can be seen in Figure 10.2d and Table 10.2b. In the optimal trajectory, the first and last phases with the preset thrust values are visible, while the middle phase has three distinct regions of differing length and thrust values.

10.4 Solution with velocity path constraints

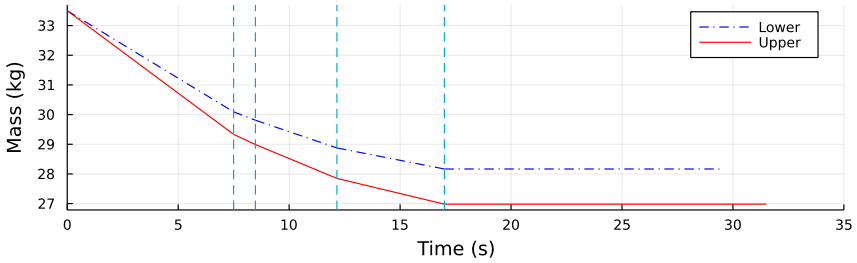
The next set of numerical results show what happens when the path constraints (10.3e) and (10.3f) on the velocity are included in the optimization problem one of two ways: either through penalty barrier terms in the cost function or using the progressive barrier method.



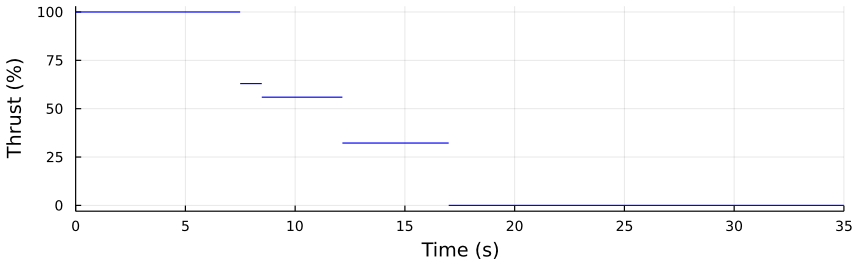
(a) Altitude profile



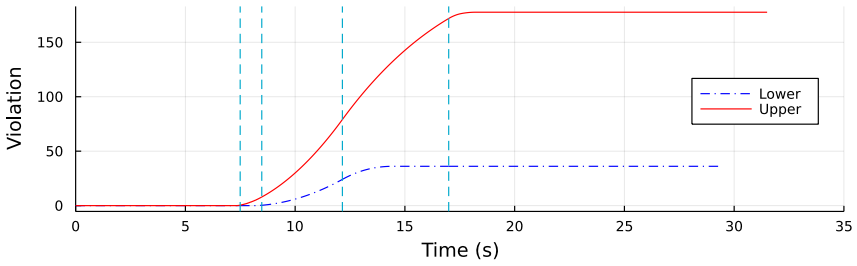
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.2: Open-loop solution to (10.3) with no velocity path constraints (switching times are at the vertical dashed lines).

Table 10.2: Results for MADS DFO-NMPC with no velocity path constraints.

(a) Results

$h_u(t_{f_u})$	3277.52 m
$h_l(t_{f_l})$	2818.47 m
Tube center	3048.00 m
Feasible cost	229.52
Infeasible cost	1182.20
Solver iterations	54
Function evaluations	855
Termination method	Mesh precision
Total solver runtime	1.69 s
Function eval. time	1.58 s
Internal MADS time	0.0033 s

(b) Thrust trajectory

Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	7.4957
2	7.4957	62.977	8.4795
3	8.4795	55.948	12.1525
4	12.1525	32.235	16.9956
5 (lower)	16.9956	0.000	29.3905
5 (upper)	16.9956	0.000	31.5019

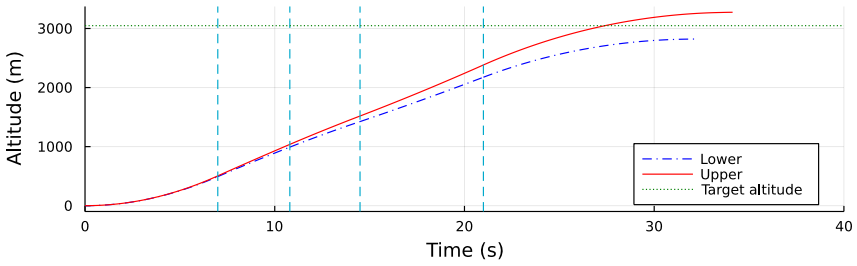
10.4.1 Penalty barrier implementation

As a baseline comparison, the path constraints (10.3e) and (10.3f) were transcribed into the DFO-NMPC problem by adding the penalty barrier term

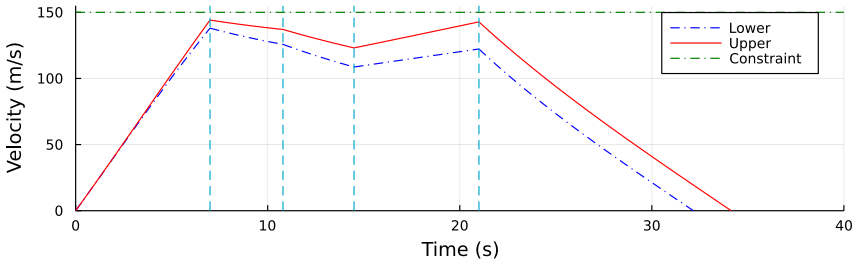
$$\phi = 100(g_l(t_{f_l}))^2 + 100(g_u(t_{f_u}))^2$$

to the cost function (10.3a). The remaining constraints were implemented using the progressive barrier method with the barrier functions (10.5a)–(10.5l).

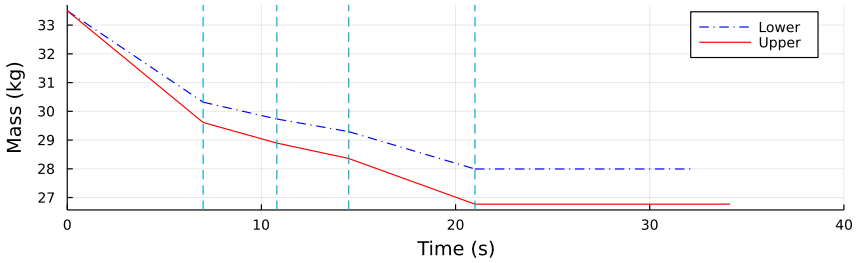
The results for MADS solving the resulting DFO-NMPC problem with the penalty barrier constraints can be seen in Figure 10.3, with the input trajectory values and solver results in Table 10.3. MADS was able to solve this problem with a cost value of 226.59 (slightly lower than the cost with no velocity path constraints), and in 58 iterations using 925 evaluations of Algorithm 9.1 to compute the cost function. As can be seen in the altitude plot in Figure 10.3a and the results in Table 10.3a, the penalty method is able to get the center of the robust tube to within 0.01 m of the target altitude of 3048 m. The effectiveness of the penalty barrier constraints can be seen in Figure 10.3b, where the velocity states for both the upper bound and lower bound systems are always less than the 150 m⁄s limit, leading to zero cumulative constraint violation in Figure 10.3e.



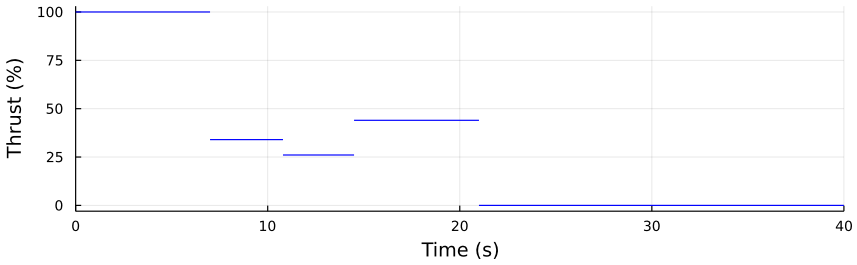
(a) Altitude profile



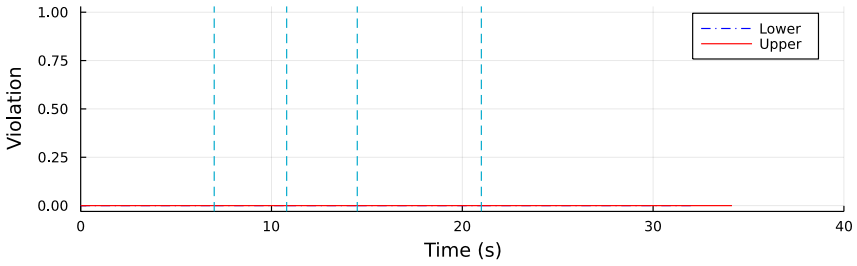
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.3: Open-loop solution to (10.3) with velocity path constraints using penalty barriers in the cost (switching times are at the vertical dashed lines).

Table 10.3: Results for MADS DFO-NMPC with velocity path constraints using penalty barriers in the cost.

(a) Results

$h_u(t_{f_u})$	3274.59 m
$h_l(t_{f_l})$	2821.40 m
Tube center	3047.99 m
Feasible cost	226.59
Infeasible cost	2.172e8
Solver iterations	58
Function evaluations	925
Termination method	Mesh precision
Total solver runtime	1.515 s
Function eval. time	1.427 s
Internal MADS time	0.0034 s

(b) Thrust trajectory

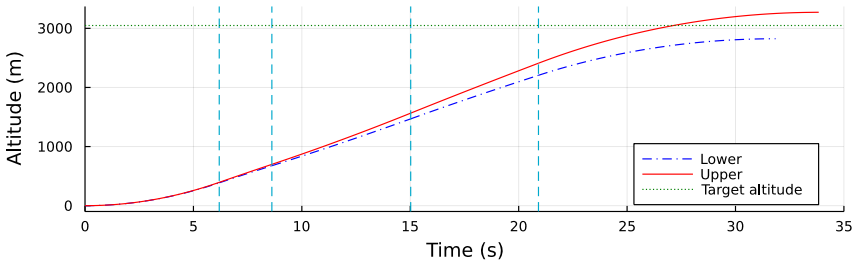
Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	6.9976
2	6.9976	34.003	10.7953
3	10.7953	26.051	14.4954
4	14.4954	43.994	20.9983
5 (lower)	20.9983	0.000	32.1613
5 (upper)	20.9983	0.000	34.1331

10.4.2 Progressive barrier implementation

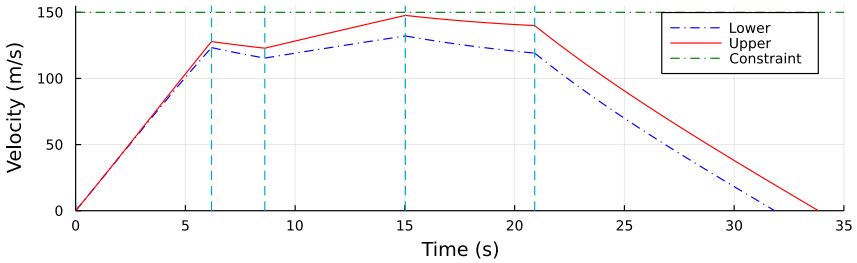
The results of using the progressive barrier method to implement the velocity path constraints (10.3e) and (10.3f) in the DFO-NMPC formulation of the rocket problem can be seen in Figure 10.4 and the results and input trajectory can be seen in Table 10.4. All other constraints were implemented using the progressive barrier method as well, meaning all the barrier terms in (10.5) were used in the MADS optimization problem.

MADS was able to solve the DFO-NMPC problem with a cost of 223.80 (slightly lower than the cost using the penalty barrier term), and in 72 iterations using 1149 evaluations of Algorithm 9.1 to compute the cost function before satisfying the mesh precision termination criteria. The altitude plot in Figure 10.4a and the results in Table 10.4a show that the progressive barrier method was able to find a solution such that the upper and lower bound systems form a tube with its center only 0.01 m away from the target altitude of 3048 m. The progressive barrier term also ensured that the velocity path constraints were satisfied for the optimal solution, as can be seen in Figure 10.4b where the velocity is always less than 150 $^{m}/_{s}$ at all times and in Figure 10.4e where there is no cumulative constraint violation.

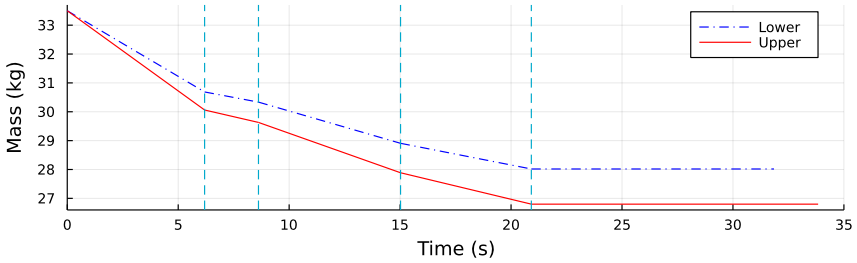
While finding the optimal solution using the progressive barrier method required more iterations of MADS than the penalty barrier approach, the penalty barrier method required slightly more time to find the optimal solution, needing 1.515 s instead of the 1.455 s the progressive barrier required. The progressive barrier method did slightly increase the computational time needed for the internal MADS computations, with the internal MADS computations taking 0.004 s instead of the 0.0034 s for the penalty barrier method.



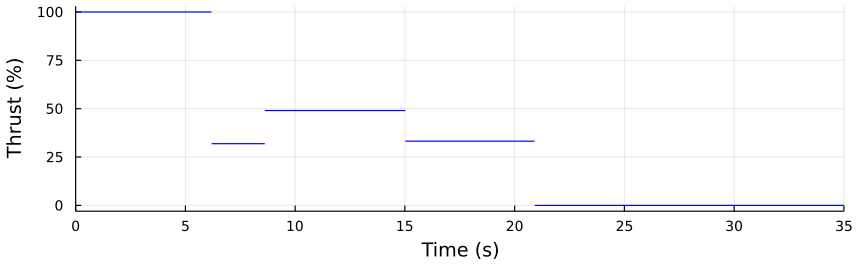
(a) Altitude profile



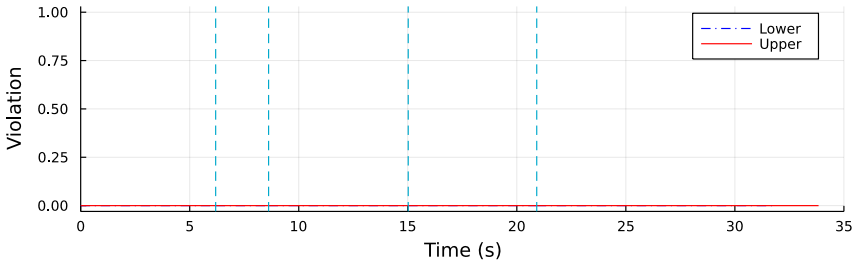
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.4: Open-loop solution to (10.3) with velocity path constraints using the progressive barrier method (switching times are at the vertical dashed lines).

Table 10.4: Results for MADS DFO-NMPC with velocity path constraints using the progressive barrier method.

(a) Results

$h_u(t_{f_u})$	3271.80 m
$h_l(t_{f_l})$	2824.19 m
Tube center	3047.99 m
Feasible cost	223.80
Infeasible cost	3031.49
Solver iterations	72
Function evaluations	1149
Termination method	Mesh precision
Total solver runtime	1.455 s
Function eval. time	1.353 s
Internal MADS time	0.004 s

(b) Thrust trajectory

Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	6.1901
2	6.1901	31.900	8.6174
3	8.6174	49.032	15.0201
4	15.0201	33.213	20.9136
5 (lower)	20.9136	0.000	31.8484
5 (upper)	20.9136	0.000	33.8382

10.4.3 Comparison against particle swarm optimization

To compare the proposed MADS DFO-NMPC implementation against an existing DFO method, the OCP (10.3) was solved using a Particle Swarm Optimization (PSO) solver. The transcription used to convert the OCP (10.3) into the optimization problem for the PSO solver was similar to the one described in Section 10.1.3, with several key differences in the handling of the constraints.

The Optim.jl [127] PSO solver used for the comparison can only natively handle bound constraints on the optimization variables. Therefore, the constraints in the OCP (10.3) that were originally implemented using the progressive barrier method in MADS had to be modified to instead be a combination of bound constraints, extremal barrier constraints and penalty barrier terms.

The upper/lower bounds on the thrust values from (10.5e)–(10.5j) were passed to the solver as explicit bounds on the optimization variables representing the thrust levels. Explicit bound constraints were also added to ensure all the switching times were in the interval $[0, t_{max}]$, with $t_{max} = 40\,\text{s}$ chosen as a suitable value based on the previous simulation results.

The constraints (10.5a)–(10.5d) that enforce the ordering of the switching times were transformed into extremal barrier constraints in the cost function of the PSO problem, causing the cost to return infinity if any of those constraints were violated. The velocity path constraints (10.3e) and (10.3f) were enforced using the dynamics augmentation approach to compute the overall constraint violation across the entire prediction horizon. The overall violation of the velocity path constraints was then used inside the penalty barrier term

$$\phi = 100(g_l(t_{f_l}))^2 + 100(g_u(t_{f_u}))^2 + 100(\max\{0, 26.0 - m_u(t_{f_u})\})^2 + 100(\max\{0, 26.0 - m_l(t_{f_l})\})^2,$$

Table 10.5: Results for DFO-NMPC solved with PSO and with velocity path constraints.

(a) Results

$h_u(t_{f_u})$	3270.00 m
$h_l(t_{f_l})$	2825.99 m
Tube center	3047.99 m
Feasible cost	222.0
Solver iterations	1000
Function evaluations	8007
Termination method	Max iters
Total solver runtime	8.708 s

(b) Thrust trajectory

Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	0.0
2	0.0	99.697	5.1043
3	5.1043	60.324	9.8586
4	9.8586	40.578	19.8546
5 (lower)	19.8546	0.000	31.5245
5 (upper)	19.8546	0.000	33.5295

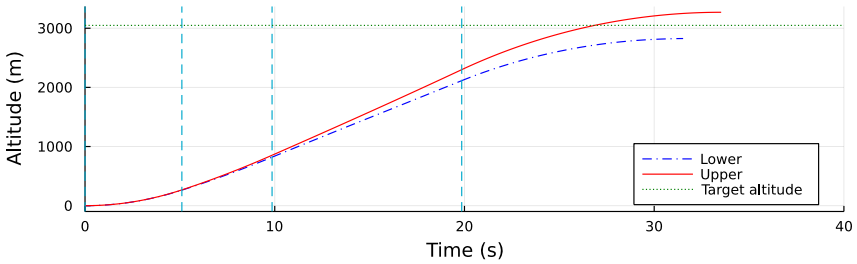
alongside two terms imposing the constraints (10.5k)–(10.5l) on the final mass of the rocket.

The solution found by the PSO solver can be seen in Figure 10.5, with the solver results and the input trajectory in Table 10.5. The resulting thrust trajectory causes the rocket to follow the altitude profile plotted in Figure 10.5a, with the upper and lower bound systems forming a robust tube centered 0.01 m below the target altitude of 3048 m. A key difference between the PSO solution and the solution from MADS with the progressive barrier method is in the constraint satisfaction of the velocity path constraints. As shown in Figure 10.5e, the PSO solution introduces a small amount of constraint violation in the velocity constraint for the upper bound system while the progressive barrier method does not.

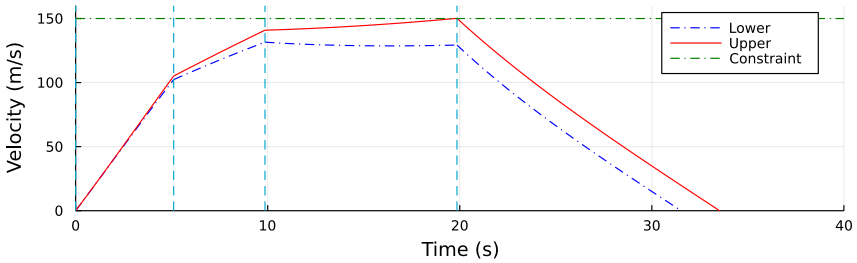
Another difference between the PSO implementation and the MADS implementation is in the behavior of the solver when finding the solution. To mimic the approach a design engineer might use, both of these solvers were run using near-default settings/values, with no tuning of constants/parameters attempted. For MADS using the progressive barrier method, Table 10.4a shows that these settings resulted in only 1149 function evaluations and MADS terminating when the mesh size was below the stopping threshold. In comparison, the PSO method shown in Table 10.5a used 8007 function evaluations and only terminated due to reaching the maximum number of permitted iterations of the solver (1000 iterations in the default settings). This meant that the PSO solver ran for 8.708 s while the MADS solver ran for only 1.455 s.

10.5 Quantized input variables

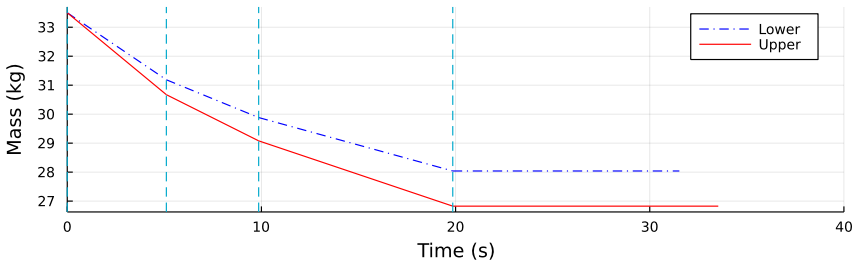
The next example examines when the inputs to the rocket are quantized and forced to only take certain values. As can be seen in the computed input trajectories for the previous examples, the DFO-NMPC method will return thrust values and switching times that have several decimal places of information. This is not always realizable in the real-world system



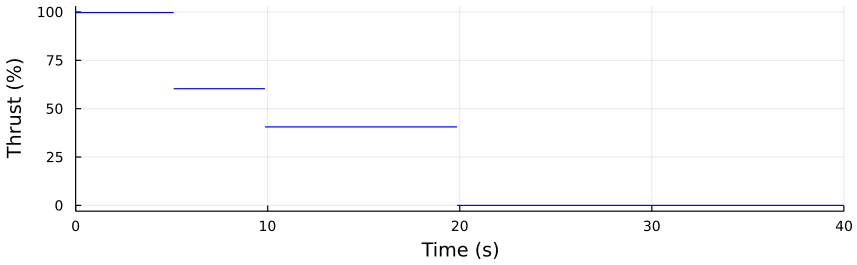
(a) Altitude profile



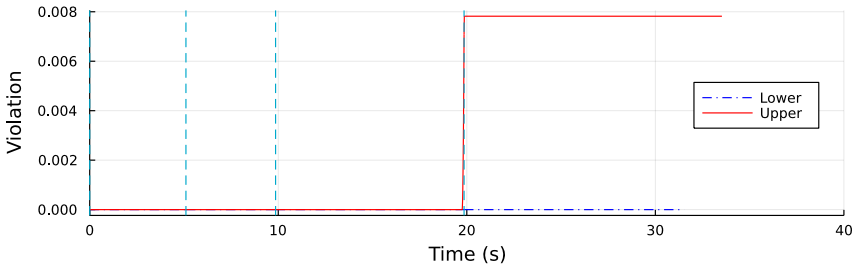
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.5: Open-loop solution to (10.3) using PSO with velocity path constraints as penalty barriers (switching times are at the vertical dashed lines).

though, because control inputs will only be applied at certain intervals and physical actuators have a fixed number of possible positions. This means there will be an inherent quantization stage between when the control is computed using all the real numbers and when it is applied to the actual system using only the allowed timings and positions.

For the rocket problem examined here, we will introduce quantization on both the time steps and the thrust input. The time steps are limited to only 1 second increments, representing a real-world system that either will only apply the controller every second or a limitation of the actuator that means the actuator position can only be changed every second. The thrust levels are limited to be in the set $T_i \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$, representing an actuator that only has 5 possible positions. Using the terminology for granular variables that was introduced in Section 8.3.1, this means $\delta_T^{min} := 0.25$ (since the thrust is represented as a value between 0 and 1) and $\delta_\sigma^{min} := 1.0$.

10.5.1 Quantized version of continuous results

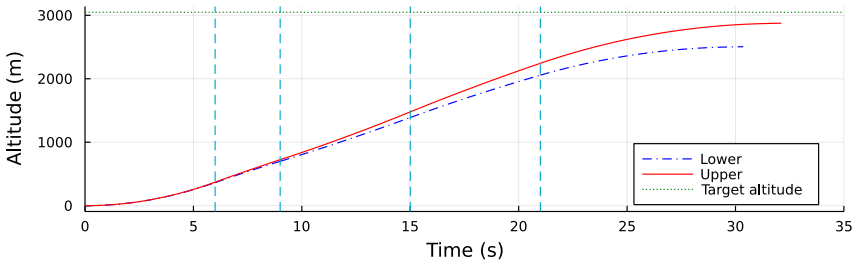
To understand the effect just the quantization of the input trajectory has on the trajectory of the rocket, the input trajectory computed using the unquantized MADS with progressive barrier constraints in Section 10.4.2 and given in Table 10.4b was passed through the quantization functions

$$\hat{\sigma}_i = \delta_\sigma^{min} \operatorname{Round}\left(\frac{\sigma_i}{\delta_\sigma^{min}}\right), \tag{10.8a}$$

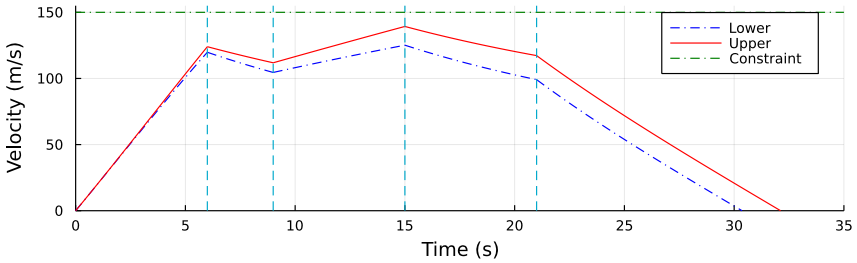
$$\hat{T}_i = \delta_T^{min} \operatorname{Round}\left(\frac{T_i}{\delta_T^{min}}\right), \tag{10.8b}$$

to produce the quantized trajectory given in Table 10.6b. The quantization operation has caused a percent decrease in the thrust applied in phases 2 and 4 by 21% and 24%, respectively, causing the velocity profile in Figure 10.6b to be lower than the original profile in Figure 10.4b.

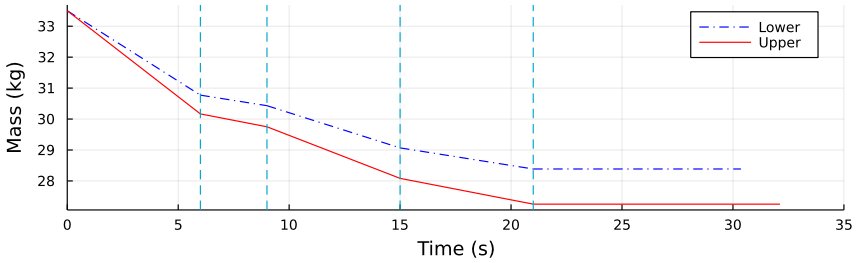
This reduction in velocity has then caused a reduction in the apogee altitude of each bound system. The upper bound system now only attains 2875.28 m whereas before it attained 3271.80 m, so the robust tube will no longer contain the target altitude of 3048 m. A corresponding decrease also occurred with the lower bound system, meaning the center of the robust tube is now at 2690.10 m. This means the trajectory that would be implemented will cause the rocket to undershoot the target altitude by over 300 m when it was originally computed to have the center of the robust tube nearly perfectly at the target altitude.



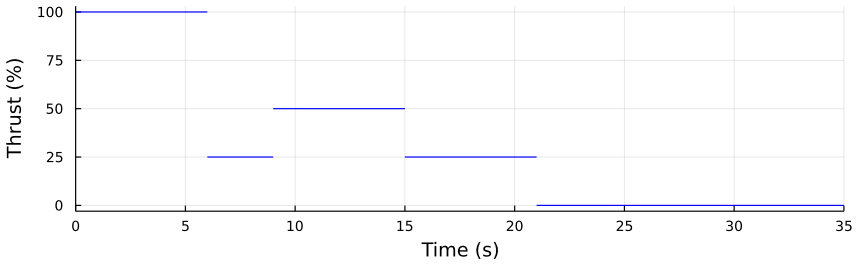
(a) Altitude profile



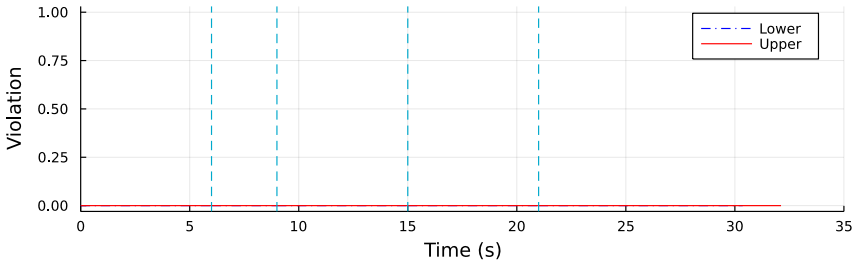
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.6: Open-loop solution to (10.3) using the quantized input trajectory in Table 10.6b (switching times are at the vertical dashed lines).

Table 10.6: Results using a quantized version of the input trajectory in Table 10.4b.

(a) Results

$h_u(t_{f_u})$	2875.28 m
$h_l(t_{f_l})$	2504.93 m
Tube center	2690.10 m

(b) Thrust trajectory

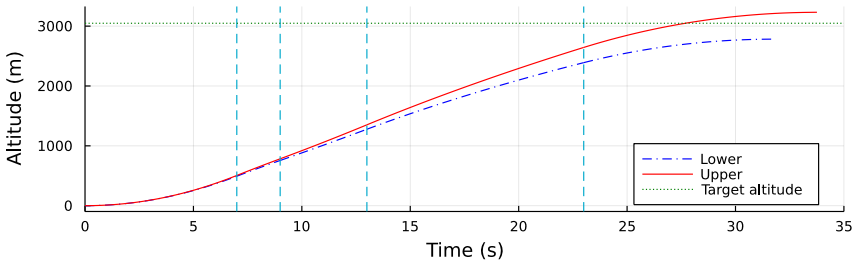
Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	6.0
2	6.0	25.0	9.0
3	9.0	50.0	15.0
4	15.0	25.0	21.0
5 (lower)	21.0	0.0	30.36
5 (upper)	21.0	0.0	32.11

10.5.2 Using granular MADS

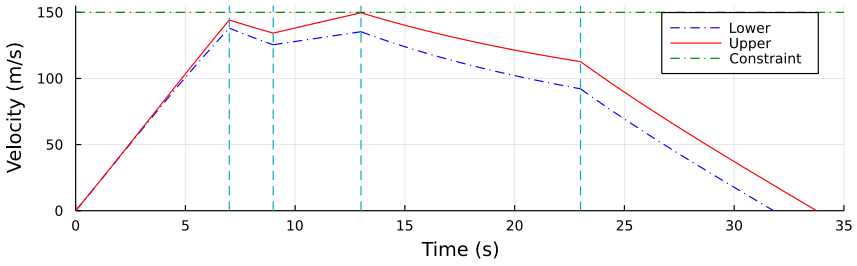
To handle the granularity of the switching times and thrust values that was introduced in this section, the granular MADS solver described in Section 8.3.1 was used with the original OCP (10.3) transcribed into a DFO-NMPC problem using the same transcription method as described in Section 10.1.3. Exchanging the regular MADS solver with progressive barrier constraints for the granular MADS solver with progressive barrier constraints was as simple as specifying the granularities on all the input variables in the problem data and rerunning the code — there were no modifications to the cost/constraint functions needed to make this change.

The granular MADS solver results and the computed input trajectory can be seen in Table 10.7, with the resulting system response shown Figure 10.7. Granular MADS was able to find a solution using 67 iterations (5 fewer than the non-granular case) and only used 1071 evaluations of Algorithm 9.1 to compute the cost function (78 fewer than the non-granular case) before converging to the solution. The cost of the optimal trajectory was 265.33, which is only an 18.5% increase in the cost value.

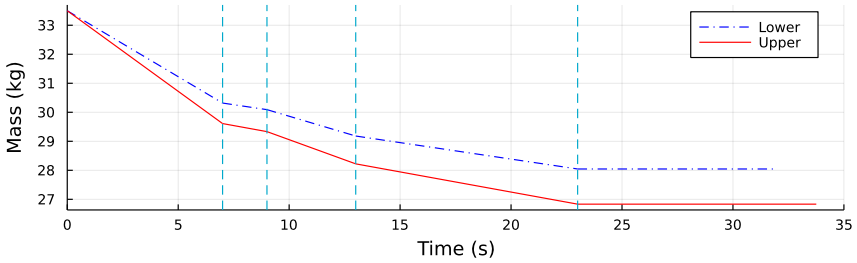
The granular solution computed by MADS has the upper bound system reaching an altitude of 3231.59 m, a 40 m decrease compared to the regular MADS solution. The lower bound system also has a smaller apogee altitude of 2782.66 m, leading to the center of the robust rube decreasing to 3007.13 m — a 40 m decrease from the non-granular problem. Compared to the quantized input trajectory created in Section 10.5.1, the solution using the granular MADS solver now has the target altitude in the computed tube, while the quantization of the trajectory from regular MADS shown in Table 10.6b did not.



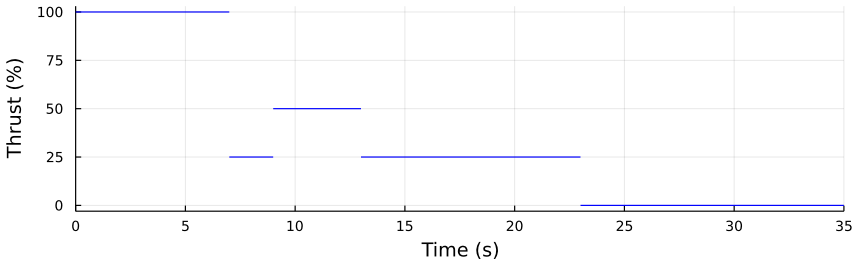
(a) Altitude profile



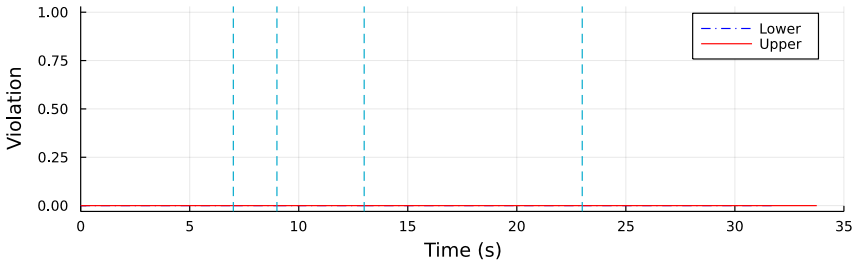
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.7: Open-loop solution to (10.3) with quantized input variables using granular MADS (switching times are at the vertical dashed lines).

Table 10.7: Results with quantized inputs solved using granular MADS.

(a) Results

$h_u(t_{f_u})$	3231.59 m
$h_l(t_{f_l})$	2782.66 m
Tube center	3007.13 m
Feasible cost	265.33
Infeasible cost	1310.85
Solver Iterations	67
Function evaluations	1071
Termination method	Mesh precision
Total solver runtime	1.207 s
Function eval. time	1.091 s
Internal MADS time	0.025 s

(b) Thrust trajectory

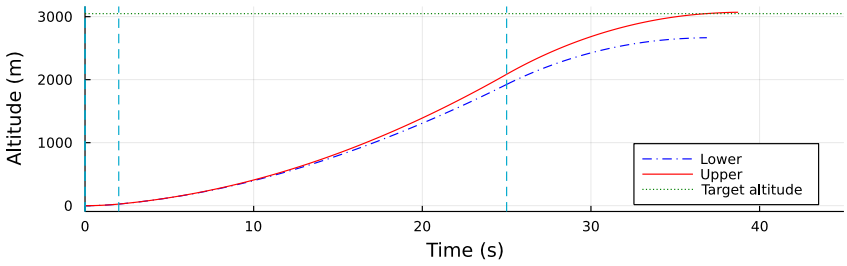
Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	7.0
2	7.0	25.0	9.0
3	9.0	50.0	13.0
4	13.0	25.0	23.0
5 (lower)	23.0	0.0	31.79
5 (upper)	23.0	0.0	33.75

10.5.3 Using quantized PSO

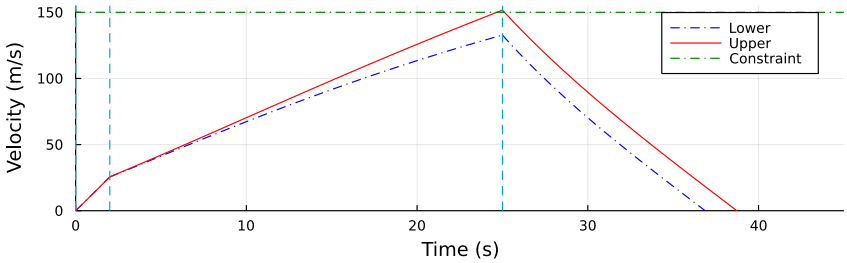
As a comparison, PSO was also used to solve the OCP with quantized inputs using a slightly modified form of the problem formulation described in Section 10.4.3. In this case, since there was no built-in support in the PSO solver for handling the granular variables, an additional step is needed at the beginning of the cost function to apply the quantization scheme given in (10.8) to the vector of variables passed to the cost function by the PSO solver. This quantized vector was then used in the computation of the constraints, dynamics simulation, and final cost value.

Overall, this quantization scheme appears to lead to the PSO solver becoming somewhat confused when searching for the optimal input trajectory, resulting in a solution with a cost of 396.02. The solver output and system response can be seen in Table 10.8 and Figure 10.8, respectively. The PSO solver terminated after 1000 iterations, the maximum number allowed, and in the process performed 8007 evaluations of the objective function using Algorithm 9.1. One point to notice about the input trajectory is that the solver decided to remove the first two thrust phases (so phase 3 starts at time $t = 0.0$ s) and only allow for 2 intermediate thrust levels in the solution.

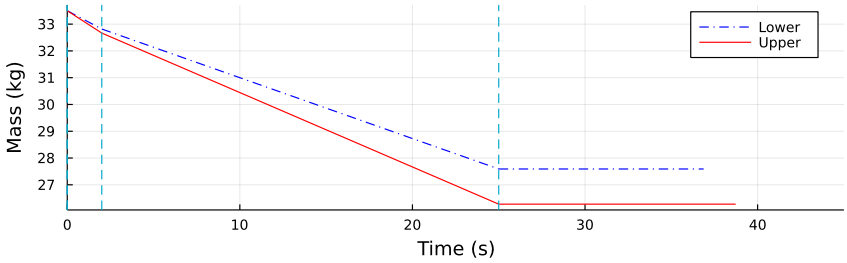
The end result is that the upper bound system only attains the altitude of 3069.63 m at its apogee — barely above the target altitude of 3048 m. In the process, the solver also introduced a minor violation of the velocity path constraint, as can be seen in Figure 10.8e. The end result is that the center of the robust tube is now at 2867.57 m, which is now 180 m below the target altitude.



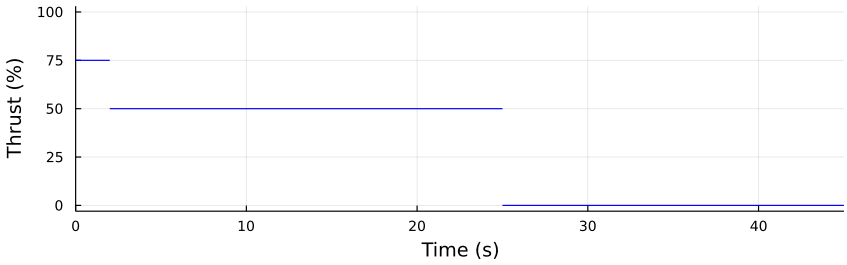
(a) Altitude profile



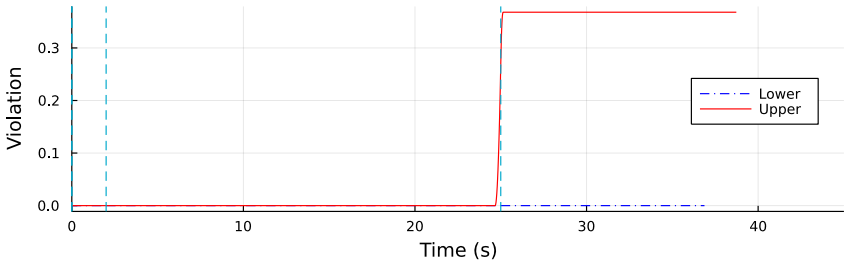
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.8: Open-loop solution to (10.3) with quantized input variables solved using PSO (switching times are at the vertical dashed lines).

Table 10.8: Results with quantized inputs solved using PSO.

(a) Results

$h_u(t_{f_u})$	3069.63 m
$h_l(t_{f_l})$	2665.51 m
Tube center	2867.57 m
Feasible cost	396.02
Solver iterations	1000
Function evaluations	8007
Termination method	Max iters
Total solver runtime	13.195 s

(b) Thrust trajectory

Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	0.0
2	0.0	100.0	0.0
3	0.0	75.0	2.0
4	2.0	50.0	25.0
5 (lower)	25.0	0.0	36.87
5 (upper)	25.0	0.0	38.73

10.6 Overdetermined problem formulation

While describing the problem transcription in Section 10.1.3, we noted that the final times for the upper and lower bound systems were actually dependent upon the switching times and input trajectory, and so should not be included in the search space of the DFO solver. The example in this section shows what happens if the final times are included in the DFO solver search space, which could happen when simply using a transcription method based on the ideas from direct collocation or multiple shooting, where the final times must be provided as variables to the solver to ensure there is a collocation point/shooting node at the final times to enforce the terminal constraints.

To include the final times in the MADS optimization problem, the problem transcription described in Section 10.1.3 was modified to have the final times t_{f_l} and t_{f_u} in the vector of optimization variables used by MADS, giving

$$c = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 \end{bmatrix}' = \begin{bmatrix} \sigma_1 & T_1 & \sigma_2 & T_2 & \sigma_3 & T_3 & \sigma_4 & t_{f_l} & t_{f_u} \end{bmatrix}'.$$

The dynamics solver then simulates the dynamics for the lower bound and upper bound systems until the time given by c_8 and c_9, respectively. Since the dynamics simulation no longer terminates when the velocity state is zero, new constraints were added to enforce the final velocity constraint (10.3d). The new constraints on the final velocity require the value of the velocity states at the final time point to be included in the w vector passed to the progressive barrier constraints, giving

$$w = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{bmatrix}' = \begin{bmatrix} g_l(t_{f_l}) & g_u(t_{f_u}) & m_l(t_{f_l}) & m_u(t_{f_u}) & v_{v_l}(t_{f_l}) & v_{v_u}(t_{f_u}) \end{bmatrix}'.$$

Five new progressive barrier constraints were then added to the problem formulation, with

$$\omega_{15}(c, w) = c_7 - c_8,$$

$$\omega_{16}(c, w) = c_7 - c_9,$$

constraining the final times to be greater than the last switching time,

$$\omega_{17}(c, w) = c_8 - c_9,$$

constraining the final time for the upper bound system to be larger than the final time for the lower bound system, and

$$\omega_{18}(c, w) = |w_5|,$$
$$\omega_{19}(c, w) = |w_6|,$$

constraining the final velocity of the rocket to be zero.

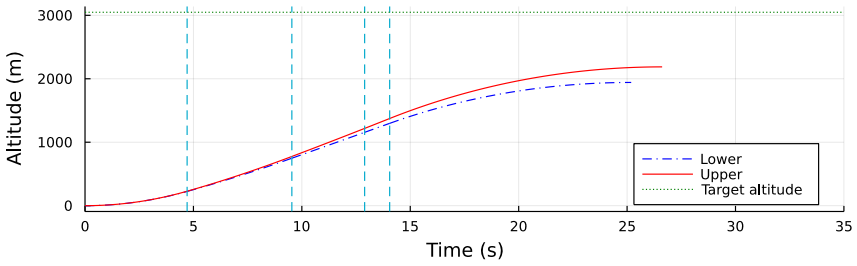
The results from the formulation that included the final times in the optimization can be seen in Table 10.9 and Figure 10.9. The first point to notice is that the MADS solver required 24.656 s to solve the problem, using 1334 iterations and 26531 evaluations of Algorithm 9.1 before terminating when the mesh size decreased to below the termination criteria. Compared with the progressive barrier MADS in Section 10.4.2 that only required 1.455 s and 72 iterations, including the final times in the optimization variables made MADS use 1200 more iterations, 25000 more function evaluations, and take 23 s of additional runtime.

The solution that MADS located has the lower bound system reaching an altitude of 1941.95 m at apogee, and the upper bound system reaching an altitude of 2187.48 m at apogee. This means the center of the robust tube is only at 2064.71 m, nearly 1000 m below the target altitude of 3048 m.

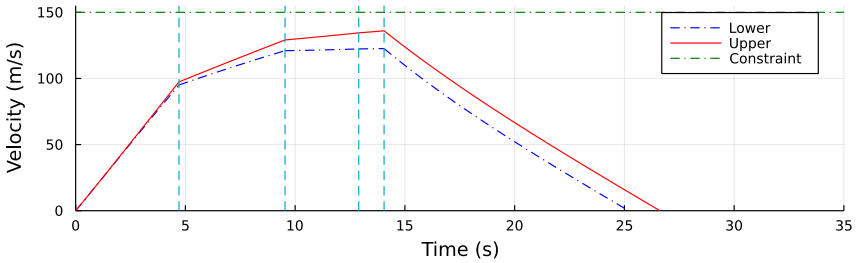
Overall, this solution shows that MADS was unable to locate any other point in the search space in its polling states that were better able to satisfy the boundary constraints and make the rocket have an apogee near the target altitude. This is in contrast to the results shown in Section 10.4.2, where not including the final times in the optimization vector and instead handling the terminal velocity constraints in the simulation allowed MADS to find a solution that placed the center of the robust tube within 0.01 m of the target altitude.

10.7 Conclusions

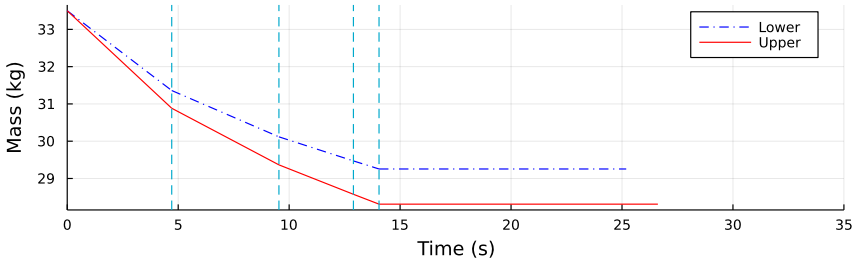
In this chapter, we demonstrated the DFO-NMPC formulation from Chapter 9 on a robust version of the Goddard rocket optimal control problem. We presented results for several different variations of both the DFO-NMPC problem and the transcription method described in Section 10.1.3, including



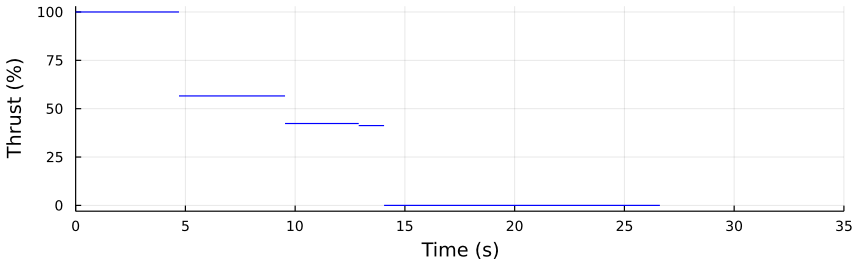
(a) Altitude profile



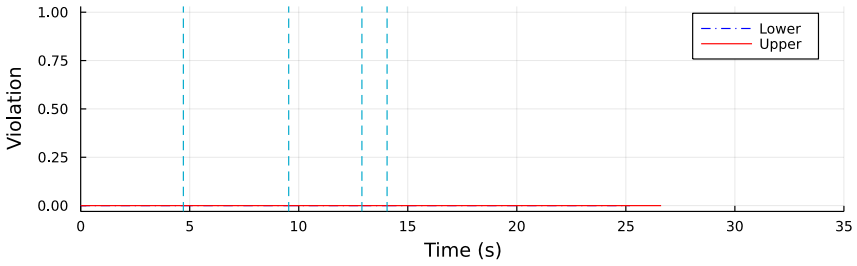
(b) Velocity profile



(c) Mass profile



(d) Input trajectory



(e) Cumulative violation of (10.3e) and (10.3f)

Figure 10.9: Open-loop solution to (10.3) including t_{f_l} and t_{f_u} in the MADS optimization problem (switching times are at the vertical dashed lines).

Table 10.9: Results for MADS DFO-NMPC with t_{f_l} and t_{f_u} included in the optimization problem.

(a) Results

$h_u(t_{f_u})$	2187.48 m
$h_l(t_{f_l})$	1941.95 m
Tube center	2064.71 m
Feasible cost	1106.05
Infeasible cost	1106.09
Solver iterations	1334
Function evaluations	26531
Termination method	Mesh precision
Total solver runtime	24.656 s
Function eval. time	23.952 s
Internal MADS time	0.089 s

(b) Thrust trajectory

Phase	**Start** (s)	**Thrust** (%)	**Finish** (s)
1	0.0	100.0	4.7086
2	4.7086	56.592	9.5393
3	9.5393	42.330	12.8963
4	12.8963	41.258	14.0507
5 (lower)	14.0507	0.000	25.1896
5 (upper)	14.0507	0.000	26.6167

- no path constraints in the OCP,
- handling the path constraints using penalty barrier terms in the cost,
- handling the path constraints using the progressive barrier method,
- using granular MADS to handle the quantized system inputs, and
- including the final times in the search space.

Additionally, the MADS implementation was compared against a DFO-NMPC implementation using PSO with penalty barrier/extremal barrier constraints.

These results show that the proposed DFO-NMPC formulation using MADS is able to solve a robust version of the Goddard rocket problem, and that the progressive barrier method for handling constraints in MADS can be applied to DFO-NMPC. The result of the DFO-NMPC MADS solver was the same as the one found by the PSO solver, with the resulting robust tubes having their centers within 0.01 m of the target altitude. The key difference between the two solvers was the iterations/function evaluations required — with the MADS solver requiring only 72 iterations (1149 function evaluations) to reach the mesh size termination threshold and terminate with the optimal solution compared to the 1000 iterations (8007 function evaluations) used by the PSO solver before reaching the maximum number of iterations allowed.

The key observation in this example is that the proposed formulation using MADS is more generalizable to other problem types than formulations using PSO, as shown in the example where quantization is introduced on the input and time variables for the Goddard rocket problem. To use the PSO framework with quantized inputs, the cost function needed to be modified to include an additional step to actually perform the quantization of the input variables before evaluating the constraints and cost, since the underlying PSO solver has no method of handling quantized variables natively. This led to the PSO solver using 1000

iterations (8007 function evaluations) to find a robust tube that barely included the target altitude and had its center 180 m below the target altitude. The proposed DFO-NMPC framework using MADS required no modifications to the cost function or constraints to implement the input/time quantization, and only required informing the underlying MADS algorithm about the quantization level for each variable. The solution the granular MADS solver found then had the tube center 40 m below the target altitude and only required 67 iterations and 1071 function evaluations (7 more iterations than the non-granular MADS solver) before reaching the mesh size termination criteria.

The numerical examples also show that the MADS solver is computationally lightweight, with the majority of the runtime in the DFO-NMPC examples spent inside the cost function evaluation/dynamics simulation instead of the MADS solver itself. Introducing granular variables in the DFO-NMPC problem did increase the time required for the internal MADS computations, going from 0.004 s for MADS using the progressive barrier method to 0.025 s for MADS using both granular variables and the progressive barrier method. This increase in the time required for the internal MADS computations had virtually no impact on the overall runtime though, since the 0.025 s spent in the MADS solver for the quantized input example is still orders of magnitude smaller than the total runtime of 1.207 s. This shows that granular MADS can provide a computationally lightweight solver for problems with quantized/integer variables, eliminating the need to use more complex and computationally expensive mixed integer solvers based on techniques such as branch and bound or cutting planes.

The final observation that we can make about the DFO-NMPC framework is that the designer should not approach the problem transcription for DFO methods the same as they would the transcription for derivative-based methods such as direct collocation or multiple-shooting. Using the problem transcription methods commonly associated with derivative-based methods with a derivative-free solver instead can lead to problems that are overdetermined and that can confuse the DFO solver. Instead, the designer should modify the transcription to suit the different DFO-NMPC problem formulation and the new degrees of freedom in transcription that are provided. For instance, the example rocket problem showed that for problems with terminal constraints and a free end time, it may be better to not include the end time in the optimization search space and instead use the dynamics simulation with a termination condition based on the terminal constraints.

Part IV

Conclusions

Chapter 11

Conclusions and future work

In this thesis, we examined two aspects of numerical methods for MPC. First, we showed in Part II how block Toeplitz operators appear in the condensed linear MPC problem and how they can be used to analyze/design extensions for the optimization solvers for the condensed linear MPC problem. Next, in Part III we developed a way to transcribe the nonlinear MPC problem (2.1) into an optimization problem suitable for the Mesh Adaptive Direct Search derivative-free optimization solver, and then demonstrated this transcription method on a robust rocket control problem.

11.1 Toeplitz operators in linear MPC

Examining how Toeplitz operators appear in control theory is not a new field, with prior work already identifying the relation between Toeplitz operators and both H_∞ optimal control and some parts of linear MPC. In this work, we extended the earlier results and showed that block Toeplitz operators are related to many of the matrices in the condensed linear MPC problem, and that many properties of these operators can be exploited to create bounds on the condition number/eigenvalues for the matrices in linear MPC, design preconditioners, and model the round-off error in embedded implementations.

Analysis and computational complexity

The largest amount of theoretical work in this thesis is contained in Chapter 4, which examines how the block Toeplitz structure of the prediction matrix, primal condensed Hessian, condensed constraint matrix, and the dual condensed Hessian in linear MPC can

be exploited to form a relation between the spectrum of the individual matrices and the singular value spectrum of the predicted system. This relation says that the extrema of the eigenvalue/singular value spectrum of the matrix can be bounded by the spectrum of a function containing the transfer function matrix of the predicted system, provided it is Schur stable (which can be guaranteed by using the numerically robust CLQR formulation (2.10)). These relations also hold for any size of the matrices, allowing for the computed bounds to hold for any horizon length and for the computation of an upper bound on the condition number of the primal condensed Hessian that will hold for any horizon length.

In Chapter 5, we then used the spectral bounds on the primal condensed Hessian to examine how the computational complexity can change depending on the cost function of the linear MPC problem. This examination was done in two parts: (i) deriving theory for how the eigenvalues/condition number scales as the weight matrices are scaled, and (ii) examining how that scaling affects the upper iteration bounds of the FGM and DGP algorithms. One of the key observations that we made using these results is that the eigenvalue spectrum/condition number of the condensed Hessian has two distinct components, and that when the cost function is modified, one of these components will then dominate the other and have the largest effect on the spectrum of the condensed Hessian. For example, when the Q matrix is scaled to be larger than the R matrix, the spectrum of the condensed Hessian is then affected by a combination of the Q matrix and the singular values of the predicted system and not the R matrix. Alternately, if the R matrix is larger, the singular values of the predicted system and the Q matrix will not have a large/noticeable effect on the condensed Hessian.

This leads to interesting behavior in the UIBs for the FGM and DGP, where we showed there are two distinct regions for the bound that follow the regions from the spectrum of the condensed Hessian. The computational requirements of the optimization solver can vary drastically between the two regions — with an example system showing that it is possible to tune a controller to have the same system performance, but with a 188% lower computational cost, by choosing the right weighting matrices.

This shows that it is not only important to consider the desired system performance during the controller design, but also that designers should consider the computational cost of the resulting controller. By adding the computational complexity into the controller design phase, it opens up the opportunity to trade-off computational performance with the system performance, ensuring that the controller can not only be implemented in real-time, but also that it can meet the low energy and resource usage requirements that newer cyber-physical systems are imposing.

Design methods

In the last two chapters of Part II, we transitioned from the analysis of the matrices and solvers for the condensed linear MPC problem to instead focus on how the block Toeplitz operators can be used to help design the implementation of the solvers themselves.

The first part of this was the new preconditioner proposed in Chapter 6. This preconditioner uses the block Toeplitz structure of the condensed Hessian to design a block circulant preconditioner that has performance comparable to an existing SDP-based preconditioner. In the numerical examples, using this preconditioner gives a speedup of between 2.1x and 9.6x for the FGM — equivalent to the speedup of the SDP preconditioner. However, the proposed preconditioner is able to also speedup the computation of the preconditioner itself, showing between an 80x and 59,672x speedup in the preconditioner computation compared to the existing SDP preconditioner for the example problems. The spectral analysis that leads to the bounds on the condition number can also be applied to the preconditioned Hessian, allowing for horizon-independent bounds for the eigenvalues/condition number of the preconditioned Hessian to be computed.

The numerical examples for the preconditioner in Section 6.3 also highlight that the numerically robust CLQR formulation (2.10) provides some preconditioning effect when used — sometimes providing a speedup of up to 9.6x when only applying the prestabilizing controller and no preconditioner. This observation, when coupled with the relation between the predicted system and the spectrum of the Hessian explored in Chapter 4, points to new opportunities for preconditioner design via the prestabilization controller. Meaning that instead of designing preconditioners based on linear algebra techniques that are applied after the controller is designed, the preconditioner would be designed as part of the controller design process using control techniques, such as H_∞ or loop shaping controllers for the prestabilization.

The final area explored in this part was how the block Toeplitz structure of the condensed Hessian can be used to guide the design of the implementation of the FGM on FPGAs. To do this, two different models for the round-off error that the condensed Hessian experiences when it is converted to fixed-point were derived, with one of those models exploiting the block Toeplitz structure to bound the round-off error in a horizon-independent manner. These models then allowed for the development of rules that will give the required size of the fixed-point variables used in the FGM so that the conversion of the condensed Hessian into fixed-point will satisfy the requirements for a stable FGM iteration. Using these sizing rules then allows for guarantees on the stability of the algorithm using smaller data types, potentially reducing the computing hardware needed by up to 75% and reducing the solution time by up to 25%.

Using these smaller data types in the optimization algorithms comes at a cost to the performance of the closed loop system though, with an example system showing up to 15–20% error in the closed-loop performance compared to the double precision implementation of the algorithm. Future work should try to quantify the relation between this closed-loop error and the data type of the implementation in order to better understand the effect the data type choice has on the closed-loop performance. Additional work should also be conducted to further prove the stability of the FGM with the projection operation, and other related algorithms, under finite-precision arithmetic using fixed-point arithmetic. One possible avenue to explore for this is to model the optimization solvers as dynamical systems and then use robust control theory to analyze the stability and performance of the solvers in the presence of round-off errors.

11.2 DFO-NMPC

The final part of this thesis presented a framework for implementing nonlinear MPC using derivative-free optimization solvers. This framework was built on top of the Mesh Adaptive Direct Search solver, but can also be modified to work with other DFO solvers such as PSO and evolutionary methods. The novelty in the framework comes from four key parts:

- the ability to handle more variable types such as granular, periodic and categorical,
- explicit handling of the constraints to produce a known-feasible solution,
- constraint enforcement across the entire horizon, and
- more accurate cost computation across the entire horizon.

The first two parts come from the use of MADS as the base algorithm for the framework, since it allows for the use of the progressive barrier constraint method and has native support for other non-continuous variables directly in the solver. This advantage is shown in the robust rocket control example in Chapter 10, where the proposed MADS framework with quantized inputs was able to find an input trajectory that gave a robust tube with a center closer to the target altitude than an equivalent PSO method.

The final two novel parts in the framework are not unique to MADS, and can even be implemented on top of other derivative-free solvers. The introduction of the new states to track the Lagrange cost term and the constraint violation along the horizon allows for the cost and constraints to be more accurately computed across the entire horizon, and decouples their computation/enforcement from the chosen input representation. This decoupling is important for the generalizability of the framework, since other frameworks currently assume a piecewise-constant input trajectory, and so they just enforce the constraints at the points where the input trajectory changes and compute the cost using a Riemann sum using those same points.

The proposed framework is able to handle other input representations, such as interpolated polynomials or the fitting of function parameters, without the need to change/modify the constraint enforcement or cost computation.

This framework is not fully completed though, and there are many open questions left to answer. As a start, we have proposed a single-shooting solver for the dynamics simulation, however this is known to be unsuitable for stiff or unstable systems, and is not routinely used in the mainstream optimal control solvers. To extend this framework to a wider range of systems (including ones that may be unstable), it is important to see if this framework can be extended to use the ideas of multiple-shooting to split the simulation horizon and handle the instability. Alternately, it could be interesting to explore integrating new machine learning techniques to attempt to learn the stable manifold of the system and then constrain the optimization solver to search only on that manifold.

Another key aspect that needs to be examined is the suitability of MADS and this framework for the use as a closed-loop controller in real-time, instead of simply as an open-loop planning controller. This opens up many new questions, including how to effectively warm start the MADS algorithm, how to effectively perform the simulations under timing constraints, and can MADS terminate early and provide a control scheme similar to the real-time iteration used by many derivative-based NMPC solvers in use today.

A known limitation of the proposed framework is that the computational performance of the MADS solver can become very poor when the optimization problem has a lot of optimization variables. This performance degradation can limit the application of the proposed framework to larger optimal control problems, such as those involving systems with many inputs or ones where the input representation requires a large number of optimization variables. Several ideas to modify MADS to perform better on larger problems have been proposed, such as using a parallel space decomposition scheme to break the large problem into several subproblems [12] or to mainly operate on a subset of the optimization variables that are identified as being statistically significant [2]. Future work should try applying these existing modifications to the proposed DFO-NMPC framework, and also examine if the MADS algorithm can be modified to specifically handle any structure that appears in the DFO-NMPC optimization problem.

Finally, it is also important to continue applying this framework to different classes/types of optimal control problems and systems. The example presented in Chapter 10 is a fairly small example with only a nonsmooth cost function and without any complicated dynamics or constraints. Applying the framework to more examples that have nonsmooth dynamics (such as contact problems), complicated constraints (such as in path planning problems), or problems with data-driven models should be done to explore and demonstrate the generalizability of the framework, and potentially suggest new areas of improvement for the framework.

References

[1] M. A. Abramson, C. Audet, J. W. Chrissis, and J. G. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47, 2009. ISSN 18624472. doi: 10.1007/s11590-008-0089-2.

[2] L. Adjengue, C. Audet, and I. Ben Yahia. A variance-based method to rank input variables of the Mesh Adaptive Direct Search algorithm. *Optimization Letters*, 8(5): 1599–1610, 2014. ISSN 18624480. doi: 10.1007/s11590-013-0688-4.

[3] B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall International, 1982. ISBN 0-13-638651.

[4] A. Araujo Barrientos. *Implementation of a High Performance Embedded MPC on FPGA using High-Level Synthesis*. Master thesis, Technische Universität Ilmenau, 2017. URL http://tesis.pucp.edu.pe/repositorio/handle/123456789/8833.

[5] C. Audet. Tuning Runge-Kutta parameters on a family of ordinary differential equations. *International Journal of Mathematical Modelling and Numerical Optimisation*, 8(3), Jan. 2018. doi: 10.1504/IJMMNO.2018.088992.

[6] C. Audet and J. E. Dennis. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006. ISSN 1052-6234.

[7] C. Audet and J. E. Dennis Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009.

[8] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer, Cham, Switzerland, 2017. ISBN 978-3-319-68912-8.

[9] C. Audet and S. Le Digabel. The mesh adaptive direct search algorithm for periodic variables. *Pacific Journal of Optimization*, 8(1):103–119, 2012. ISSN 13489151. doi: 10.1101/pdb.prot069708.

[10] C. Audet and D. Orban. Finding Optimal Algorithmic Parameters Using Derivative-Free Optimization. *SIAM Journal on Optimization*, 17(3):642–664, 2006. ISSN 1052-6234. doi: 10.1137/040620886.

[11] C. Audet, V. Béchard, and J. Chaouki. Spent potliner treatment process optimization using a MADS algorithm. *Optimization and Engineering*, 9(2):143–160, June 2008. ISSN 1573-2924. doi: 10.1007/s11081-007-9030-2.

[12] C. Audet, J. E. Dennis, and S. Le Digabel. Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm. *SIAM Journal on Optimization*, 19(3):1150–1170, 2008. doi: 10.1137/070707518.

[13] C. Audet, K. Dang, and D. Orban. Algorithmic Parameter Optimization of the DFO Method with the OPAL Framework. In K. Naono, K. Teranishi, J. Cavazos, and R. Suda, editors, *Software Automatic Tuning: From Concepts to State-of-the-Art Results*, pages 255–274. Springer, New York, NY, Aug. 2010. doi: 10.1007/978-1-4419-6935-4_15.

[14] C. Audet, G. Savard, and W. Zghal. A mesh adaptive direct search algorithm for multiobjective optimization. *European Journal of Operational Research*, 204(3):545–556, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.11.010.

[15] C. Audet, S. Le Digabel, and C. Tribes. The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. *SIAM Journal on Optimization*, 29(2):1164–1189, Jan. 2019. ISSN 1052-6234. doi: 10.1137/18M1175872.

[16] D. Axehill. Controlling the level of sparsity in MPC. *Systems and Control Letters*, 76: 1–7, 2015. doi: 10.1016/j.sysconle.2014.12.002.

[17] H. Ayala, R. Sampaio, D. M. Munoz, C. Llanos, L. Coelho, and R. Jacobi. Nonlinear model predictive control hardware implementation with custom-precision floating point operations. In *24th Mediterranean Conference on Control and Automation (MED)*, pages 135–140, Athens, Greece, 2016. IEEE. doi: 10.1109/MED.2016.7535908.

[18] K. Basterretxea and K. Benkrid. Embedded high-speed Model Predictive Controller on a FPGA. In *Proceedings of the 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 327–335, San Diego, CA, 2011. doi: 10.1109/AHS.2011.5963955.

[19] I. Baturone, M. C. Martínez-Rodríguez, P. Brox, A. Gersnoviez, and S. Sánchez-Solano. Digital Implementation of Hierarchical Piecewise-Affine Controllers. In *2011 IEEE International Symposium on Industrial Electronics*, pages 1497–1502, Gdansk, Poland, 2011. IEEE. doi: 10.1109/ISIE.2011.5984382.

[20] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002. doi: 10.1016/S0005-1098(01)00174-1.

[21] D. S. Bernstein. *Matrix Mathematics*. Princeton University Press, Princeton, second edition, 2009.

[22] J. T. Betts. Survey of Numerical Methods for Trajectory Optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998. doi: 10.2514/2.4231.

[23] J. Bigeon, S. Le Digabel, and L. Salomon. DMulti-MADS: Mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization. *Computational Optimization and Applications*, 79(2):301–338, June 2021. ISSN 1573-2894. doi: 10.1007/s10589-021-00272-9.

[24] D. Bini and B. Meini. On the Solution of a Nonlinear Matrix Equation Arising in Queueing Problems. *SIAM Journal on Matrix Analysis and Applications*, 17(4):906–926, Oct. 1996. ISSN 0895-4798, 1095-7162. doi: 10.1137/S0895479895284804.

[25] D. A. Bini and B. Meini. Exploiting the Toeplitz Structure in Certain Queueing Problems. *Calcolo*, 33(3-4):289–305, 1996. doi: 10.1007/BF02576006.

[26] E. Bini and G. M. Buttazzo. The Optimal Sampling Pattern for Linear Control Systems. *IEEE Transactions on Automatic Control*, 59(1):78–90, 2014. doi: 10.1109/TAC.2013.2279913.

[27] M. A. Boéchat, J. Liu, H. Peyrl, A. Zanarini, and T. Besselmann. An Architecture for Solving Quadratic Programs with the Fast Gradient Method on a Field Programmable Gate Array. In *2013 21st Mediterranean Conference on Control and Automation (MED)*, pages 1557–1562, Platanias-Chania, Crete, Greece, 2013. doi: 10.1109/MED.2013.6608929.

[28] D. Boland and G. A. Constantinides. Optimising Memory Bandwidth Use for Matrix-Vector Multiplication in Iterative Methods. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*, pages 169–181, Bangkok, Thailand, 2010. doi: 10.1007/978-3-642-12133-3_17.

[29] A. Bottcher and B. Silbermann. *Introduction to Large Truncated Toeplitz Matrices*. Springer-Verlag, New York, 1991. ISBN 978-1-4612-1426-7.

[30] P. Brox, J. Castro-Ramírez, M. C. Martínez-Rodríguez, E. Tena, C. J. Jiménez, I. Baturone, and A. J. Acosta. A programmable and configurable ASIC to generate piecewise-affine functions defined over general partitions. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(12):3182–3194, 2013.

[31] P. Brox, M. C. Martínez-Rodríguez, E. Tena-Sánchez, I. Baturone, and A. J. Acosta. Application specific integrated circuit solution for multi-input multi-output piecewise-affine functions. *International Journal of Circuit Theory and Applications*, 44(1):4–20, 2016.

[32] B. Chachuat, editor. *Nonlinear and Dynamic Optimization: From Theory to Practice*. EPFL, Lausanne, CH, 2007.

[33] K. J. Chan, J. A. Paulson, and A. Mesbah. Deep learning-based approximate nonlinear model predictive control with offset-free tracking for embedded applications. In *Proceedings of the 2021 American Control Conference (ACC)*, New Orleans, LA, USA, May 2021. IEEE.

[34] R. H. Chan and X.-Q. Jin. *An Introduction to Iterative Toeplitz Solvers*. SIAM, 2007.

[35] R. H. Chan and M. K. Ng. Conjugate Gradient Methods for Toeplitz Systems. *SIAM Review*, 38(3):427–482, 1996.

[36] C.-T. Chen. *Linear System Theory and Design*. Oxford University Press, New York, NY, US, third edition, 1999. ISBN 0-19-511777-8.

[37] H. Chen, F. Xu, and Y. Xi. Field programmable gate array/system on a programmable chip-based implementation of model predictive controller. *IET Control Theory and Applications*, 6(8), 2012. doi: 10.1049/iet-cta.2010.0443.

[38] F. Comaschi, B. A. Genuit, A. Oliveri, W. M. H. Heemels, and M. Storace. FPGA implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(12):2920–2933, 2012.

[39] A. R. Conn and S. Le Digabel. Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. *Optimization Methods and Software*, 28 (1):139–158, 2013. ISSN 10556788. doi: 10.1080/10556788.2011.623162.

[40] G. A. Constantinides. Tutorial Paper: Parallel Architectures for Model Predictive Control. In *Proceedings of the 2009 European Control Conference (ECC)*, pages 138–143, Budapest, Hungary, 2009. doi: 10.23919/ECC.2009.7074393.

[41] P. M. Crespo and J. Gutierrez-Gutierrez. On the application of asymptotically equivalent sequences of matrices to the derivation of the continuous time Gaussian channel capacity. *AEU - International Journal of Electronics and Communications*, 60 (8):573–581, Sept. 2006. ISSN 14348411. doi: 10.1016/j.aeue.2005.11.010.

[42] J. S. Dæhlen, G. O. Eikrem, and T. A. Johansen. Nonlinear model predictive control using trust-region derivative-free optimization. *Journal of Process Control*, 24(7):1106–1120, 2014. ISSN 09591524.

[43] T. V. Dang, K. V. Ling, and J. M. Maciejowski. Embedded ADMM-based QP solver for MPC with polytopic constraints. In *2015 European Control Conference (ECC)*, pages 3446–3451, Linz, Austria, 2015. doi: 10.1109/ECC.2015.7331067.

[44] E. J. Davison. Benchmark Problems for Control System Design: Report of the IFAC Theory Committee. Technical report, IFAC, 1990.

[45] D. de Berg, T. Savage, P. Petsagkourakis, D. Zhang, N. Shah, and E. A. del Rio-Chanona. Data-driven optimization for process systems engineering applications. *Chemical Engineering Science*, IN press:117135, Sept. 2021. ISSN 00092509. doi: 10.1016/j.ces.2021.117135.

[46] J. Drgoňa, J. Arroyo, I. Cupeiro Figueroa, D. Blum, K. Arendt, D. Kim, E. P. Ollé, J. Oravec, M. Wetter, D. L. Vrabie, and L. Helsen. All you need to know about model predictive control for buildings. *Annual Reviews in Control*, 50:190–232, Jan. 2020. ISSN 1367-5788. doi: 10.1016/j.arcontrol.2020.09.001.

[47] J. L. Garriga and M. Soroush. Model Predictive Control Tuning Methods: A Review. *Industrial and Engineering Chemistry Research*, 49(8):3505–3515, 2010. doi: 10.1021/ie900323c.

[48] M. Garstka, M. Cannon, and P. Goulart. COSMO: A conic operator splitting method for large convex problems. In *2019 18th European Control Conference (ECC)*, pages 1951–1956, Naples, Italy, June 2019. IEEE. doi: 10.23919/ECC.2019.8796161.

[49] S. Gerkšič and B. Pregelj. Finite-word-length FPGA implementation of model predictive control for ITER resistive wall mode control. *Fusion Engineering and Design*, 169: 112480, Aug. 2021. ISSN 0920-3796. doi: 10.1016/j.fusengdes.2021.112480.

[50] A. Gersnoviez, M. Brox, and I. Baturone. High-speed and low-cost implementation of explicit model predictive controllers. *IEEE Transactions on Control Systems Technology*, 27(2):647–662, 2019.

[51] A. E. Gheribi, C. Audet, S. Le Digabel, E. Bélisle, C. W. Bale, and A. D. Pelton. Calculating optimal conditions for alloy and process design using thermodynamic and property databases, the FactSage software and the Mesh Adaptive Direct Search algorithm. *Calphad*, 36:135–143, Mar. 2012. ISSN 0364-5916. doi: 10.1016/j.calphad.2011.06.003.

[52] A. E. Gheribi, S. Le Digabel, C. Audet, and P. Chartrand. Identifying optimal conditions for magnesium based alloy design using the Mesh Adaptive Direct Search algorithm. *Thermochimica Acta*, 559:107–110, May 2013. ISSN 0040-6031. doi: 10.1016/j.tca.2013.02.004.

[53] J. D. Gibson. A direct search approach to optimization for nonlinear model predictive control. *Optimal Control Applications and Methods*, 36(2):139–157, Mar. 2015.

[54] G. C. Goodwin, M. M. Seron, and J. A. De Dona. *Constrained Control and Estimation - An Optimisation Based Approach*. Springer, London, UK, 2005. ISBN 1-85233-548-3.

[55] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, PA, 1997.

[56] A. Guiggiani, P. Patrinos, and A. Bemporad. Fixed-Point Implementation of a Proximal Newton Method for Embedded Model Predictive Control. *IFAC Proceedings Volume*, 47(3):2921–2926, 2014. ISSN 14746670. doi: 10.3182/20140824-6-ZA-1003.00992.

[57] J. Gutiérrez-Gutiérrez and P. M. Crespo. Asymptotically equivalent sequences of matrices and Hermitian block Toeplitz matrices with continuous symbols: Applications to MIMO systems. *IEEE Transactions on Information Theory*, 54(12):5671–5680, 2008. ISSN 00189448. doi: 10.1109/TIT.2008.2006401.

[58] J. Gutiérrez-Gutiérrez and P. M. Crespo. Asymptotically equivalent sequences of matrices and multivariate ARMA processes. *IEEE Transactions on Information Theory*, 57(8):5444–5454, 2011. ISSN 00189448. doi: 10.1109/TIT.2011.2159042.

[59] J. Gutiérrez-Gutiérrez and P. M. Crespo. Block toeplitz matrices: Asymptotic results and applications. *Foundations and Trends in Communications and Information Theory*, 8(3):179–257, 2012. doi: 10.1561/0100000066.

[60] J. Gutiérrez-Gutiérrez, P. M. Crespo, and A. Böttcher. Functions of banded Hermitian block Toeplitz matrices in signal processing. *Linear Algebra and its Applications*, 422 (2-3):788–807, Apr. 2007. ISSN 00243795. doi: 10.1016/j.laa.2006.12.008.

[61] J. Gutiérrez-Gutiérrez, I. Iglesias, and A. Podhorski. Geometric MMSE for one-sided and two-sided vector linear predictors: From the finite-length case to the infinite-length case. *Signal Processing*, 91(9):2237–2245, Sept. 2011. ISSN 01651684. doi: 10.1016/j.sigpro.2011.04.001.

[62] J. Gutiérrez-Gutiérrez, P. M. Crespo, M. Zárraga-Rodríguez, and B. O. Hogstad. Asymptotically Equivalent Sequences of Matrices and Capacity of a Discrete-Time Gaussian MIMO Channel With Memory. *IEEE Transactions on Information Theory*, 63(9):6000–6003, Sept. 2017. ISSN 1557-9654. doi: 10.1109/TIT.2017.2715044.

[63] J. Gutiérrez-Gutiérrez, M. Zárraga-Rodríguez, X. Insausti, and B. Hogstad. On the Complexity Reduction of Coding WSS Vector Processes by Using a Sequence of Block Circulant Matrices. *Entropy*, 19(3):95, Mar. 2017. ISSN 1099-4300. doi: 10.3390/e19030095.

[64] K. Harder, M. Buchholz, J. Niemeyer, J. Remele, and K. Graichen. A Real-Time nonlinear MPC Scheme with Emission Constraints for Heavy-Duty Diesel Engines. In *2017 American Control Conference (ACC)*, pages 240–245, Seattle, WA, 2017. IEEE. doi: 10.23919/ACC.2017.7962960.

[65] E. N. Hartley and J. M. Maciejowski. Predictive control for spacecraft rendezvous in an elliptical orbit using an FPGA. In *2013 European Control Conference (ECC)*, pages 1359–1364, Zurich, 2013. IEEE. doi: 10.23919/ECC.2013.6669196.

[66] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides. Predictive control of a Boeing 747 aircraft using an FPGA. *IFAC Proceedings Volume*, 45(17):80–85, 2012. ISSN 14746670. doi: 10.3182/20120823-5-NL-3013.00016.

[67] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides. Predictive Control Using an FPGA With Application to Aircraft Control. *IEEE Transactions on Control Systems Technology*, 22(3):1006–1017, 2014. doi: 10.1109/TCST.2013.2271791.

[68] T. Hausberger, A. Kugi, A. Deutschmann, A. Eder, and W. Kemmetmüller. A Nonlinear MPC Strategy for AC/DC-Converters tailored to the Implementation on FPGAs. *IFAC-PapersOnLine*, 52(16):376–381, 2019. ISSN 24058963. doi: 10.1016/j.ifacol.2019.11.809.

[69] T. Hausberger, A. Kugi, A. Eder, and W. Kemmetmüller. High-speed nonlinear model predictive control of an interleaved switching DC/DC-converter. *Control Engineering Practice*, 103:104576, Oct. 2020. ISSN 09670661. doi: 10.1016/j.conengprac.2020.104576.

[70] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1994. ISBN 978-0-521-46713-1.

[71] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, second edition, 2013. ISBN 978-0-521-54823-6.

[72] M. Huang, D. Liao-mcpherson, S. Kim, K. Butts, and I. Kolmanovsky. Toward Real-Time Automotive Model Predictive Control: A Perspective from a Diesel Air Path Control Development. In *2018 American Control Conference (ACC)*, pages 2425–2430, Milwaukee, WI, US, 2018. IEEE. ISBN 978-1-5386-5427-9.

[73] Z. Hurák and A. Böttcher. MIMO l1-optimal control via block Toeplitz operators. In *Proceedings of the 16th International Symposium on Mathematical Theory of Networks and Systems (MTNS'04)*, Leuven, Belgium, 2004.

[74] P. Hyatt and M. D. Killpack. Real-Time Nonlinear Model Predictive Control of Robots Using a Graphics Processing Unit. *IEEE Robotics and Automation Letters*, 5(2):1468–1475, Apr. 2020. ISSN 2377-3766. doi: 10.1109/LRA.2020.2965393.

[75] O. J. Isebor, L. J. Durlofsky, and D. Echeverría Ciaurri. A derivative-free methodology with local and global search for the constrained joint optimization of well locations and controls. *Computational Geosciences*, 18(3-4):463–482, Aug. 2014. ISSN 1420-0597, 1573-1499. doi: 10.1007/s10596-013-9383-x.

[76] D. Jacobson and M. Lele. A transformation technique for optimal control problems with a state variable inequality constraint. *IEEE Transactions on Automatic Control*, 14(5):457–464, Oct. 1969. ISSN 1558-2523. doi: 10.1109/TAC.1969.1099283.

[77] M. Jeong, S. Fuchs, and J. Biela. When FPGAs Meet Regionless Explicit MPC: An Implementation of Long-horizon Linear MPC for Power Electronic Systems. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 3085–3092, Singapore, Oct. 2020. IEEE. doi: 10.1109/IECON43393.2020.9254277.

[78] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. FPGA Implementation of an Interior Point Solver for Linear Model Predictive Control. In *2010 International Conference on Field-Programmable Technology (FPT)*, pages 316–319, Beijing, China, 2010. doi: 10.1109/FPT.2010.5681439.

[79] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. Towards a fixed point QP solver for predictive control. In *51st IEEE Conference on Decision and Control (CDC)*, pages 675–680, Maui, HI, 2012. doi: 10.1109/CDC.2012.6427015.

[80] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica*, 48(5):999–1002, 2012. doi: 10.1016/j.automatica.2012.03.010.

[81] J. L. Jerez, K. V. Ling, G. A. Constantinides, and E. C. Kerrigan. Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry. *IET Control Theory and Applications*, 6(8):1029 – 1041, 2012. doi: 10.1049/iet-cta.2010.0441.

[82] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded Predictive Control on an FPGA using the Fast Gradient Method. In *2013 European Control Conference (ECC)*, pages 3614–3620, Zurich, 2013. doi: 10.23919/ECC.2013.6669598.

[83] 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*, 59(12):3238–3251, 2014. doi: 10.1109/TAC.2014.2351991.

[84] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. A Low Complexity Scaling Method for the Lanczos Kernel in Fixed-Point Arithmetic. *IEEE Transactions on Computers*, 64(2):303–315, 2015. doi: 10.1109/TC.2013.162.

[85] T. A. Johansen. Toward Dependable Embedded Model Predictive Control. *IEEE Systems Journal*, 11(2):1208–1219, 2017. doi: 10.1109/JSYST.2014.2368129.

[86] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tondel. Hardware Architecture Design for Explicit Model Predictive Control. In *2006 American Control Conference (ACC)*, pages 1924–1929, Minneapolis, MN, USA, 2006. IEEE. doi: 10.1109/ACC.2006.1656501.

[87] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tondel. Hardware Synthesis of Explicit Model Predictive Controllers. *IEEE Transactions on Control Systems Technology*, 15(1):191–197, 2007. doi: 10.1109/TCST.2006.883206.

[88] C. N. Jones and M. Morari. The double description method for the approximation of explicit MPC control laws. In *47th IEEE Conference on Decision and Control (CDC)*, pages 4724–4730, Cancun, Mexico, 2008. IEEE. doi: 10.1109/CDC.2008.4738794.

[89] A. Joos and W. Fichter. Parallel Implementation of Constrained Nonlinear Model Predictive Controller for an FPGA-Based Onboard Flight Computer. In F. Holzapfel and S. Theil, editors, *Advances in Aerospace Guidance, Navigation and Control*, pages 273–286. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-19817-5.

[90] A. Joos, P. Heritier, C. Huber, and W. Fichter. Method for Parallel FPGA Implementation of Nonlinear Model Predictive Control. *IFAC Proceedings Volume*, 45(1):73–78, 2012. ISSN 14746670. doi: 10.3182/20120213-3-IN-4034.00016.

[91] B. Kapernick, S. Süß, E. Schubert, and K. Graichen. A synthesis strategy for nonlinear model predictive controller on FPGA. In *2014 UKACC International Conference on Control*, pages 662–667, Loughborough, UK, 2014. IEEE. doi: 10.1109/CONTROL.2014.6915218.

[92] P. Karamanakos, E. Liegmann, T. Geyer, and R. Kennel. Model Predictive Control of Power Electronic Systems: Methods, Results, and Challenges. *IEEE Open Journal of Industry Applications*, 1:95–114, 2020. ISSN 2644-1241. doi: 10.1109/OJIA.2020.3020184.

[93] M. Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, 2017. ISSN 0036-1445. doi: 10.1137/16M1062569.

[94] E. C. Kerrigan, Y. Nie, O. Faqir, C. H. Kennedy, S. A. Niederer, J. A. Solis-Lemus, P. Vincent, and S. E. Williams. Direct Transcription for Dynamic Optimization: A Tutorial with a Case Study on Dual-Patient Ventilation During the COVID-19

Pandemic. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 2597–2614, Jeju Island, Republic of Korea, Dec. 2020. IEEE. doi: 10.1109/CDC42340.2020.9304378.

[95] B. Khusainov, E. C. Kerrigan, A. Suardi, and G. A. Constantinides. Nonlinear predictive control on a heterogeneous computing platform. *IFAC-PapersOnLine*, 50(1), 2017. ISSN 24058963. doi: 10.1016/j.ifacol.2017.08.1413.

[96] B. Khusainov, E. Kerrigan, A. Suardi, and G. Constantinides. Nonlinear predictive control on a heterogeneous computing platform. *Control Engineering Practice*, 78 (September):105–115, 2018. doi: 10.1016/j.conengprac.2018.06.016.

[97] B. Khusainov, E. C. Kerrigan, and G. A. Constantinides. Automatic Software and Computing Hardware Codesign for Predictive Control. *IEEE Transactions on Control Systems Technology*, 27(5):2295–2304, 2019. doi: 10.1109/TCST.2018.2855666.

[98] G. Knagge, A. Wills, A. Mills, and B. Ninness. ASIC and FPGA implementation strategies for Model Predictive Control. In *Proceedings of the 2009 European Control Conference (ECC)*, pages 144–149, Budapest, Hungary, 2009. IEEE. doi: 10.23919/ECC.2009.7074394.

[99] S. Kouro, M. A. Perez, J. Rodriguez, A. M. Llor, and H. A. Young. Model Predictive Control: MPC's Role in the Evolution of Power Electronics. *IEEE Industrial Electronics Magazine*, 9(4):8–21, 2015. ISSN 19324529.

[100] D. Lakhmiri, S. Le Digabel, and C. Tribes. HyperNOMAD: Hyperparameter Optimization of Deep Neural Networks Using Mesh Adaptive Direct Search. *ACM Transactions on Mathematical Software*, 47(3):27:1–27:27, 2021. doi: 10.1145/3450975.

[101] M. S. K. Lau, S. P. Yue, K. V. Ling, and J. M. Maciejowski. A Comparison of Interior Point and Active Set Methods for FPGA Implementation of Model Predictive Control. In *Proceedings of the 2009 European Control Conference (ECC)*, pages 3–8, Budapest, Hungary, 2009. doi: 10.23919/ECC.2009.7074396.

[102] H. Li, I. McInerney, J. Davis, and G. A. Constantinides. Digit stability inference for iterative methods using redundant number representation. *IEEE Transactions on Computers*, 70(7):1074–1080, 2021. doi: 10.1109/TC.2020.3003529.

[103] K. V. Ling, S. P. Yue, and J. M. Maciejowski. A FPGA Implementation of Model Predictive Control. In *2006 American Control Conference (ACC)*, pages 1930–1935, Minneapolis, MN, USA, 2006. doi: 10.1109/ACC.2006.1656502.

[104] K. V. Ling, B. F. Wu, and J. Maciejowski. Embedded Model Predictive Control (MPC) using a FPGA. *IFAC Proceedings Volume*, 41(2):15250–15255, 2008. ISSN 14746670. doi: 10.3182/20080706-5-KR-1001.2303.

[105] J. Liu, H. Peyrl, A. Burg, and G. A. Constantinides. FPGA implementation of an interior point method for high-speed model predictive control. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, Montreal, QC, Canada, 2014. doi: 10.1109/FPL.2014.6927473.

[106] S. Longo, E. C. Kerrigan, and G. A. Constantinides. Constrained LQR for low-precision data representation. *Automatica*, 50(1):162–168, 2014. doi: 10.1016/j.automatica.2013.09.035.

[107] A. R. Lopes, A. Shahzad, G. A. Constantinides, and E. C. Kerrigan. More Flops or More Precision? Accuracy Parameterizable Linear Equation Solvers for Model Predictive Control. In *Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '09)*, pages 209–216, Napa, CA, 2009. doi: 10.1109/FCCM.2009.19.

[108] S. Lucia, M. Kögel, P. Zometa, D. E. Quevedo, and R. Findeisen. Predictive control, embedded cyberphysical systems and systems of systems - a perspective. *Annual Reviews in Control*, 41:193–207, 2016. doi: 10.1016/j.arcontrol.2016.04.002.

[109] S. Lucia, D. Navarro, O. Lucia, P. Zometa, and R. Findeisen. Optimized FPGA Implementation of Model Predictive Control for Embedded Systems Using High Level Synthesis Tool. *IEEE Transactions on Industrial Informatics*, 14(1):137–145, 2018. doi: 10.1109/TII.2017.2719940.

[110] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and Ó. Lucía. Deep Learning-Based Model Predictive Control for Resonant Power Converters. *IEEE Transactions on Industrial Informatics*, 17(1):409–420, Jan. 2021. ISSN 1941-0050. doi: 10.1109/TII.2020.2969729.

[111] A. K. Madsen and D. G. Perera. Efficient embedded architectures for fast-charge model predictive controller for battery cell management in electric vehicles. *EURASIP Journal on Embedded Systems*, 2018(2):1–36, 2018. ISSN 1687-3963. doi: 10.1186/s13639-018-0084-3.

[112] M. C. Martínez-Rodríguez, I. Baturone, and P. Brox. Circuit Implementation of Piecewise-Affine Functions Based on Lattice Representation. In *2011 20th European Conference on Circuit Theory and Design (ECCTD 2011)*, pages 644–647, Linkoping, Sweden, 2011. IEEE. doi: 10.1109/ECCTD.2011.6043625.

[113] M. C. Martínez-Rodríguez, P. Brox, and I. Baturone. Digital VLSI implementation of piecewise-affine controllers based on lattice approach. *IEEE Transactions on Control Systems Technology*, 23(3):842–854, 2014.

[114] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000. doi: 10.1016/S0005-1098(99)00214-9.

[115] I. McInerney and E. C. Kerrigan. Automated project-based assessment in a predictive control course. In *2018 UKACC 12th International Conference on Control (CONTROL)*, page 443, Sheffield, UK, Aug. 2018. IEEE. doi: 10.1109/CONTROL.2018.8516728.

[116] I. McInerney, G. A. Constantinides, and E. C. Kerrigan. A survey of the implementation of linear model predictive control on FPGAs. In *6th IFAC Conference on Nonlinear Model Predictive Control*, pages 381–387, Madison, Wisconsin, US, 2018. IFAC. doi: 10.1016/j.ifacol.2018.11.063.

[117] I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Bounding computational complexity under cost function scaling in predictive control. *arXiv:1902.02221*, 2019.

[118] I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Modeling round-off error in the fast gradient method for predictive control. In *58th IEEE Conference on Decision and Control (CDC)*, pages 4331–4336, Nice, France, 2019. IEEE. doi: 10.1109/CDC40024.2019.9029910.

[119] I. McInerney, L. Nita, Y. Nie, A. Oliveri, and E. C. Kerrigan. Towards a framework for nonlinear predictive control using derivative-free optimization. In *7th IFAC Conference on Nonlinear Model Predictive Control*, Bratislava, Slovakia, 2021. IFAC.

[120] I. McInerney, E. C. Kerrigan, and G. A. Constantinides. Horizon-independent preconditioner design for linear predictive control. *IEEE Transactions on Automatic Control*, 2022. doi: 10.1109/TAC.2022.3145657. (*in press*).

[121] S. Merkli, J. Jerez, A. Domahidi, R. S. Smith, and M. Morari. Circuit Generation for Efficient Projection onto Polyhedral Sets in First-Order Methods. In *2015 European Control Conference (ECC)*, pages 3440–3445, Linz, Austria, 2015. IEEE. ISBN 978-3-9524269-3-7. doi: 10.1109/ECC.2015.7331066.

[122] A. Mills, A. G. Wills, S. R. Weller, and B. Ninness. Implementation of linear model predictive control using a field-programmable gate array. *IET Control Theory and Applications*, 6(8):1042–1054, 2012. doi: 10.1049/iet-cta.2010.0739.

[123] R. Milman and E. J. Davison. Fast computation of the quadratic programming subproblem in model predictive control. In *Proceedings of the 2003 American Control Conference*, pages 4723–4729, Denver, CO, USA, 2003. IEEE. doi: 10.1109/ACC.2003.1242469.

[124] M. Miranda and P. Tilli. Block Toeplitz Matrices and Preconditioning. *Calcolo*, 33 (1-2):79–86, 1996. doi: 10.1007/BF02575709.

[125] M. Miranda and P. Tilli. Asymptotic spectra of Hermitian block Toeplitz matrices and preconditioning results. *SIAM Journal on Matrix Analysis and Applications*, 21(3): 867–881, 2000.

[126] L. Mirsky. The Spread of a Matrix. *Mathematika*, 3(2):127–130, 1956. doi: 10.1112/S0025579300001790.

[127] P. K. Mogensen and A. N. Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, 2018. doi: 10.21105/joss.00615.

[128] M. Mönnigmann and M. Kastsian. Fast explicit MPC with multiway trees. *IFAC Proceedings Volume*, 44(1):1356–1361, 2011. ISSN 14746670. doi: 10.3182/20110828-6-IT-1002.00686.

[129] M. W. Mueller and R. D'Andrea. A model predictive controller for quadrocopter state interception. In *2013 European Control Conference (ECC)*, Zurich, Switzerland, 2013. IEEE.

[130] Y. Nasir, W. Yu, and K. Sepehrnoori. Hybrid derivative-free technique and effective machine learning surrogate for nonlinear constrained well placement and production optimization. *Journal of Petroleum Science and Engineering*, 186:106726, Mar. 2020. ISSN 09204105. doi: 10.1016/j.petrol.2019.106726.

[131] I. Necoara, F. Stoican, D. Clipici, A. Patrascu, and M. Hovd. A linear MPC algorithm for embedded systems with computational complexity guarantees. In *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 363–368, Sinaia, Romania, 2014. IEEE. doi: 10.1109/ICSTCC.2014.6982443.

[132] L. Nita. *Robust Rocket Throttle Control*. MEng thesis, Imperial College London, London, UK, 2020.

[133] A. Oliveri and M. Storace. Hardware-in-the-loop simulations of circuit architectures for the computation of exact and approximate explicit MPC control functions. In *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pages 380–383. IEEE, 2012.

[134] A. Oliveri, A. Oliveri, T. Poggi, and M. Storace. Circuit implementation of piecewise-affine functions based on a binary search tree. In *2009 European Conference on Circuit Theory and Design*, pages 145–148, Antalya, Turkey, 2009. IEEE. doi: 10.1109/ECCTD.2009.5274957.

[135] A. Oliveri, M. Lodi, and M. Storace. Design and circuit implementation of approximate switched MPC. In *2013 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4. IEEE, 2013.

[136] A. Oliveri, C. Gianoglio, E. Ragusa, and M. Storace. Low-complexity digital architecture for solving the point location problem in explicit model predictive control. *Journal of the Franklin Institute*, 352(6):2249–2258, 2015.

[137] A. Oliveri, M. Lodi, and M. Storace. High-speed explicit nonlinear model predictive control. In *2017 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, Catania, Italy, 2017. IEEE. doi: 10.1109/ECCTD.2017.8093239.

[138] P. Patrinos and A. Bemporad. An Accelerated Dual Gradient-Projection Algorithm for Embedded Linear Model Predictive Control. *IEEE Transactions on Automatic Control*, 59(1):18–33, 2014. doi: 10.1109/TAC.2013.2275667.

[139] P. Patrinos, A. Guiggiani, and A. Bemporad. Fixed-point dual gradient projection for embedded model predictive control. In *2013 European Control Conference (ECC)*, Zurich, Switzerland, 2013. IEEE. doi: 10.23919/ECC.2013.6669412.

[140] P. Patrinos, A. Guiggiani, and A. Bemporad. A dual gradient-projection algorithm for model predictive control in fixed-point arithmetic. *Automatica*, 55:226–235, 2015. doi: 10.1016/j.automatica.2015.03.002.

[141] K. B. Peterson and M. S. Pederson. The Matrix Cookbook, 2012.

[142] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M. A. Boéchat. Parallel implementations of the fast gradient method for high-speed MPC. *Control Engineering Practice*, 33:22–34, 2014. doi: 10.1016/j.conengprac.2014.08.010.

[143] H. Peyrl, H. J. Ferreau, and D. Kouzoupis. A Hybrid Hardware Implementation for Nonlinear Model Predictive Control. *IFAC-PapersOnLine*, 48(23):87–93, 2015. ISSN 24058963. doi: 10.1016/j.ifacol.2015.11.266.

[144] T. Poggi, F. Comaschi, and M. Storace. Digital circuit realization of piecewise-affine functions with nonuniform resolution: Theory and FPGA implementation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(2):131–135, 2010.

[145] T. Poggi, S. Trimboli, A. Bemporad, and M. Storace. Explicit hybrid model predictive control: discontinuous piecewise-affine approximation and FPGA implementation. *IFAC Proceedings Volumes*, 44(1):1350–1355, 2011.

[146] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[147] C. Rackauckas and Q. Nie. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5(1):15, May 2017. ISSN 2049-9647. doi: 10.5334/jors.151.

[148] A. Raha, A. Chakrabarty, V. Raghunathan, and G. T. Buzzard. Embedding Approximate Nonlinear Model Predictive Control at Ultrahigh Speed and Extremely Low Power. *IEEE Transactions on Control Systems Technology*, 28(3):1092–1099, 2020. ISSN 1063-6536. doi: 10.1109/TCST.2019.2898835.

[149] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications*, 99(3): 723–757, 1998. doi: 10.1023/A:1021711402723.

[150] S. Richter. *Computational complexity certification of gradient methods for real-time model predictive control*. Doctoral thesis, ETH Zurich, 2012.

[151] S. Richter, C. N. Jones, and M. Morari. Computational complexity certification for real-time MPC with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2012. doi: 10.1109/TAC.2011.2176389.

[152] J.-M. Rodriguez-Bernuz, I. McInerney, A. Junyent-Ferré, and E. C. Kerrigan. Design of a linear time-varying model predictive control energy regulator for grid-tied VSCs. *IEEE Transactions on Energy Conversion*, 36(2):1425–1434, 2021. doi: 10.1109/TEC.2021.3060319.

[153] O. J. Rojas and G. C. Goodwin. On the asymptotic properties of the hessian in discrete-time linear quadratic control. In *2004 American Control Conference (ACC)*, pages 239–244, Boston, MA, 2004. IEEE.

[154] J. A. Rossiter, B. Kouvaritakis, and M. J. Rice. A Numerically Robust State-Space Approach to Stable-Predictive Control Strategies. *Automatica*, 34(1):65–73, 1998. doi: 10.1016/S0005-1098(97)00171-4.

[155] 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*, 26(15):3292–3310, 2016. doi: 10.1002/rnc.3507.

[156] J. R. Sabo and A. A. Adegbege. A Primal-Dual Architecture for Embedded Implementation of Linear Model Predictive Control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1827–1832, Miami Beach, FL, USA, 2018. IEEE. doi: 10.1109/CDC.2018.8619294.

[157] A. Sadrieh and P. A. Bahri. Application of Graphic Processing Unit in Model Predictive Control. In *21st European Symposium on Computer-Aided Process Engineering (ESCAPE-21)*, volume 29, pages 492–496, Porto Carras Resort, Chalkidiki, Greece, 2011. Elsevier B.V.

[158] S. Sanjeevini and D. S. Bernstein. Counting Zeros Using Observability and Block Toeplitz Matrices. *IEEE Transactions on Automatic Control*, 66(3):1301–1305, Mar. 2021. ISSN 1558-2523. doi: 10.1109/TAC.2020.2989269.

[159] C. E. Santos, L. d. S. Coelho, R. C. Sampaio, R. Jacobi, H. Ayala, and C. H. Llanos. A SVM Optimization Tool and FPGA System Architecture Applied to NMPC. In *30th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 96–102, Fortaleza, Brazil, 2017. IEEE. doi: 10.1145/3109984.3110007.

[160] G. Shah and S. Engell. Tuning MPC for desired closed-loop performance for MIMO systems. In *2011 American Control Conference (ACC)*, pages 4404–4409, San Francisco, CA, 2011. IEEE. doi: 10.1109/ACC.2011.5991581.

[161] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones. Software and Hardware Code Generation for Predictive Control Using Splitting Methods. *IFAC-PapersOnLine*, 50(1):14386–14391, 2017. ISSN 24058963. doi: 10.1016/j.ifacol.2017.08.2025.

[162] M. Storace and T. Poggi. Digital architectures realizing piecewise-linear multivariate functions: two FPGA implementations. *International Journal of Circuit Theory and Applications*, 39(1):1–15, 2011.

[163] G. Strang. A proposal for Toeplitz matrix calculations. *Studies in Applied Mathematics*, 74(2):171–176, 1986. ISSN 1467-9590. doi: 10.1002/sapm1986742171.

[164] G. P. H. Styan and H. Wolkowicz. Bounds for Eigenvalues Using Traces. *Linear Algebra and its Applications*, 29:471–506, 1983. doi: 10.1016/0024-3795(83)90170-2.

[165] A. Suardi, E. C. Kerrigan, and G. A. Constantinides. Fast FPGA prototyping toolbox for embedded optimization. In *2015 European Control Conference (ECC)*, pages 2589–2594, Linz, Austria, 2015. IEEE. doi: 10.1109/ECC.2015.7330928.

[166] E. E. Swartzlander and A. G. Alexopoulos. The Sign/Logarithm Number System. *IEEE Transactions on Computers*, C-24(12):1238–1242, 1975. doi: 10.1145/1023833.1023845.

[167] G. Szegö. Ein Grenzwertsatz über die Toeplitzschen Determinanten einer reellen positiven Funktion. *Mathematische Annalen*, 76:490–503, 1915. doi: 10.1007/BF01458220.

[168] P. Tilli. Singular values and eigenvalues of non-Hermitian block Toeplitz matrices. *Linear Algebra and its Applications*, 272(1-3):59–89, 1998. doi: 10.1016/S0024-3795(97)00308-X.

[169] Q. N. Tran, L. Ozkan, J. Ludlage, and A. C. P. M. Backx. Asymptotic behaviour of Toeplitz matrix in multi-input multi-output model predictive control. In *2013 European Control Conference (ECC)*, pages 3784–3789, Zurich, Switzerland, 2013. IEEE.

[170] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton University Press, Princeton, NJ, USA, 2005. ISBN 9780691119465.

[171] D. F. Valencia, R. Tarvirdilu-Asl, C. Garcia, J. Rodriguez, and A. Emadi. A Review of Predictive Control Techniques for Switched Reluctance Machine Drives. Part I: Fundamentals and Current Control. *IEEE Transactions on Energy Conversion*, 36 (2):1313–1322, June 2021. ISSN 1558-0059. doi: 10.1109/TEC.2020.3047983.

[172] V. Vassiliadis. *Computational Solution of Dynamic Optimization Problems with General Differential-Algebraic Constraints*. PhD Thesis, Imperial College, University of London, London, UK, 1993.

[173] V. S. Vassiliadis, R. W. H. Sargent, and C. C. Pantelides. Solution of a Class of Multistage Dynamic Optimization Problems. 2. Problems with Path Constraints. *Industrial & Engineering Chemistry Research*, 33(9):2123–2133, Sept. 1994. ISSN 0888-5885, 1520-5045. doi: 10.1021/ie00033a015.

[174] R. Venugopal and D. Bernstein. Adaptive disturbance rejection using ARMARKOV/Toeplitz models. *IEEE Transactions on Control Systems Technology*, 8 (2):257–269, Mar. 2000. ISSN 1558-0865. doi: 10.1109/87.826797.

[175] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare. A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5):1006–1017, 2009. doi: 10.1109/TCST.2008.2004503.

[176] Y. Wang, M. Soltani, and D. M. A. Hussain. Attitude stabilization of Marine Satellite Tracking Antenna using Model Predictive Control. *IFAC Journal of Systems and Control*, 17:100173, Sept. 2021. ISSN 2468-6018. doi: 10.1016/j.ifacsc.2021.100173.

[177] A. Wills, A. Mills, and B. Ninness. FPGA implementation of an interior-point solution for linear model predictive control. *IFAC Proceedings Volume*, 44(1):14527–14532, 2011. ISSN 14746670. doi: 10.3182/20110828-6-IT-1002.02857.

[178] A. G. Wills, G. Knagge, and B. Ninness. Fast Linear Model Predictive Control Via Custom Integrated Circuit Architecture. *IEEE Transactions on Control Systems Technology*, 20(1):59–71, 2012. doi: 10.1109/TCST.2010.2096224.

[179] F. Xu, H. Chen, X. Gong, and Q. Mei. Fast Nonlinear Model Predictive Control on FPGA Using Particle Swarm Optimization. *IEEE Transactions on Industrial Electronics*, 63(1):310–321, 2016. doi: 10.1109/TIE.2015.2464171.

[180] Y. Xu, D. Li, Y. Xi, J. Lan, and T. Jiang. An Improved Predictive Controller on the FPGA by Hardware Matrix Inversion. *IEEE Transactions on Industrial Electronics*, 65 (9):7395–7405, 2018. ISSN 02780046. doi: 10.1109/TIE.2018.2798563.

[181] N. Yang, D. Li, J. Zhang, and Y. Xi. Model predictive controller design and implementation on FPGA with application to motor servo system. *Control Engineering Practice*, 20(11):1229–1235, 2012. doi: 10.1016/j.conengprac.2012.06.012.

[182] M. Zárraga-Rodríguez, J. Gutiérrez-Gutiérrez, and X. Insausti. A Low-Complexity and Asymptotically Optimal Coding Strategy for Gaussian Vector Sources. *Entropy*, 21(10): 965, Oct. 2019. doi: 10.3390/e21100965.

[183] L. Zhang, Z. Zhou, Q. Chen, R. Long, and S. Quan. Model Predictive Control for Electrochemical Impedance Spectroscopy Measurement of Fuel Cells Based on Neural Network Optimization. *IEEE Transactions on Transportation Electrification*, 5(2):524–534, 2019. doi: 10.1109/TTE.2019.2909687.

[184] P. Zhang, J. Zambreno, and P. H. Jones. An Embedded Scalable Linear Model Predictive Hardware-based Controller using ADMM. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Seattle, WA, 2017. doi: 10.1109/ASAP.2017.7995276.

[185] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice Hall, 1996.