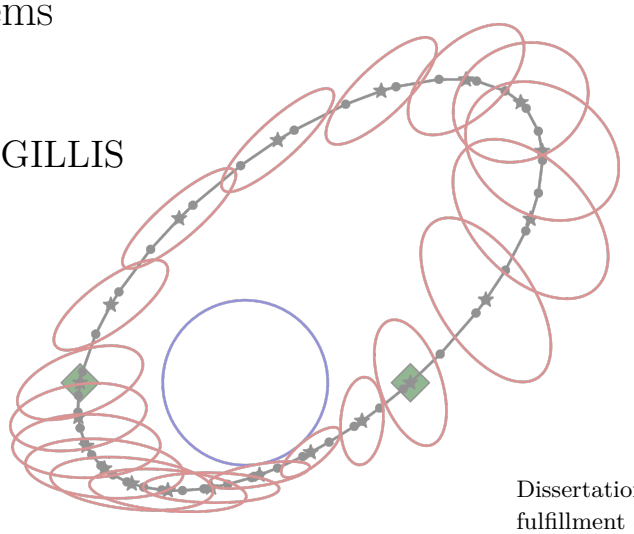
KATHOLIEKE UNIVERSITEIT
LEUVEN

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

Practical methods for approximate robust periodic optimal control of nonlinear mechanical systems

Joris GILLIS



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

March 2015

Practical methods for approximate robust periodic optimal control of nonlinear mechanical systems

Joris GILLIS

Jury:
Prof. dr. ir. P. Van Houtte, chair
Prof. dr. M. Diehl, supervisor
Prof. dr. ir. E. Van den Bulck, co-supervisor
Prof. dr. ir. J. Swevers, co-supervisor
Prof. dr. ir. J. De Schutter
Prof. dr. ir. D. Roose
Prof. dr. ir. F. Logist

Prof. dr. ir. P. Van Dooren
(UCL)

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

March 2015

Preface

This text tells a story.
A story written in the language of mathematics.
A story of a journey, driven by passion for engineering and programming.
A journey embarked on in the pursuit of knowledge and scientific contribution.
A long journey, with failure and successes, dead-ends and eureka-moments.
A journey guided by wise men and colleagues, supported by friends and family.

This is the story of my PhD.

Acknowledgements

Als ingenieursstudent die zich hoofdzakelijk bezighield met het gulzig opslorpen van kennis, ontbrak het me hoegenaamd niet aan motivatie om mijn tanden te zetten in een uitdagend project.

Uiteindelijk was het Prof. Jan Swevers die me overtuigde om een doctoraat te overwegen. Toen moest de keuze uit een waaier aan mogelijke doctoraatsprojecten echter nog beginnen. Het feit dat zulke keuze eerder een luxe-probleem was, maakte de keuze er niet makkelijker op.

Met Prof. Eric Van den Bulck ging ik aan de slag om een vormgesloten onderwater-energieopwekkingsmodule met gyroscopische transmissie te ontwerpen. Een grondige ommezwaai volgde na twee jaar toen ik me toespitste op meer methodisch onderzoek over robuuste optimale controle met Prof. Moritz Diehl.

Deze drie proffen zijn promotor van dit werk en ik wens hen uitdrukkelijk te bedanken voor hun begrip voor mijn omzwervingen en eigenzinnigheid. Verder specifieke dank aan Jan voor praktische goede raad, Eric voor inzichtelijke discussies, en Moritz voor het aanreiken van uitstekende ideeën.

Ook de andere leden van mijn doctoraatscomissie, Dirk Roose, Joris De Schutter, Paul Van Dooren, Filip Logist en voorzitter Paul Van Houtte ben ik dankbaar voor de samenwerking en de grondige en constructieve feedback op de gepresenteerde thesis.

Een goede werksfeer is een must om goed te functioneren. Van de collega's tijdens de jaren op TME wens ik Vladimir, Maarten S., Tom, Lieven, Jeroen, Wouter, Tijs, Daniel, Roel, Ruben, Joris, Dieter en Kathleen te bedanken om het mooie weer te maken. Ook de middag-uitstapjes naar Viavia met Maarten V., Bart, Kristof, Lieboud en Jeroen maakten het werkleven aangenaam. Verder was het een fijne ervaring om samen met Anouk en Geert de cursus fluidummechanica van William te verzorgen.

Special collegial thanks goes to Asim Onder for many a great night of partying, Stefan Antonov for outdoors activities and Jay Goit and Clara Verhelst for infectious enthousiasm.

later on, I switched to the group of Moritz. I very much enjoyed the warm welcome and cooperation with Adeleh, Atilla, Andrew, Boris, Carlo, Jacqueline, Janick, Joachim, Julia, Mario, Milan, Quoc, Reinhart, Rien, Sebastien and Slava.

In special I wish to thank Joel Andersson, the architect of the CasADi optimal control tool which soon caught my deep interest, and Greg Horn for awesome conversation and goofing around.

Verder wens ik enkele mensen te bedanken die nieuwe werelden openden tijdens het doctoraat: Elisabeth en Stephanie voor een restyling, en Olivier Tilleuil en Koen Huylebroeck voor kennismaking met de ondernemerswereld.

Ook zou het moeilijk geweest zijn het doctoraatsproject succesvol af te ronden zonder de steun van familie en schoonfamilie, vrienden van de universiteit (Andy, Eline, Joram, Kurt & Liesl, Niels, Ninah) en vrienden van het middelbaar (Ellen, Joke, Jonathan, Mats, Lien, Lies, Steven, Sven).

Motivatie is een cruciaal element dat ons als mens drijft in ieder aspect van ons leven. Met genoeg motivatie en toewijding kan je de wereld aan, daar ben ik van overtuigd.

Het is zonder twijfel de verdienste van mijn ouders Renaat Gillis en Lieve Nauwelaerts om motivatie mee te geven. "Doe wat je graag doet" was het eeuwige moto, en daar heb ik me steeds aan gehouden.

Ten slotte, en niet in het minste, wil ik mijn verloofde Hayke Verdeyen bedanken voor al het begrip en praktische steun op moeilijke momenten, het delen van de vreugde tijdens de doorbraken en het aansporen om niet al te fel van de

planning af te wijken.

Joris Gillis
Leuven, Februari 2015

This work was made possible thanks to generous funding from the FWO Vlaanderen, from the European Union via EMBOCON, HIGHWIND and Eurostars SMART, as well as from the KU Leuven Research Council via OPTEC.

Abstract

This thesis investigates efficient formulations and methods to solve robust periodic optimal control problems. We assume that systems are essentially nonlinear over the broad range of state space spanned by a limit cycle, requiring nonlinear programming techniques, yet behave as time varying linear systems along that limit cycle.

The existing Lyapunov framework is taken as a given. This framework introduces the periodic Lyapunov differential equations into the optimal control problem. Using an interpretation of these continuous equations as covariance propagation, the framework allows one to robustify path constraints in a first-order approximation with respect to Gaussian disturbances.

The framework was improved in this thesis in terms of formulation, discretisation accuracy, computational complexity, and structure exploitation as to allow larger scale applications.

The main formulation proposed in this work is based on a separate treatment of Lyapunov states such that the discrete periodic Lyapunov equations (DPLE) arise, which come with a guarantee on preservation of positive definiteness.

The computational complexity of the resulting nonlinear program is reduced by eliminating the Lyapunov states and using a dedicated DPLE solver based on the periodic Schur decomposition.

This solver is implemented as infinitely-differentiable component in the modular open-source optimal control framework CasADi: it is embeddable into a symbolic computational graph.

This formulation is found to be highly beneficial for large scale applications with limited time-horizon.

In addition to open-loop trajectory/control design, time-variant linear feedback control design is demonstrated with the method.

Applications include an automated process to select the safest modes of operation of a carousel device for launching an airborne wind-energy system, and a benchmark application for time-optimal periodic quadcopter flight.

Beknopte samenvatting

Het onderzoek in deze thesis spitst zich toe op efficiënte formuleringen en methodes om robuuste periodische optimale controle problemen op te lossen.

In deze thesis gaan we uit van systemen die zich niet-linear gedragen overheen het volle bereik van toestandsruimte dat de limietcyclus doorkuist, zodat het gebruik van niet-lineaire programmma's noodzakelijk is, maar die zich toch gedragen als tijdsvariabele lineare systemen langsheen die limitiecyclus.

Het vertrekpunt van de thesis is het Lyapunov framework. Dit framework introduceert de periodische Lyapunov differentiaalvergelijkingen binnen een optimaal controle probleem. Via interpretatie van deze vergelijkingen als propagatievergelijkingen voor covariantie, laat het framework toe om in een eertse-orde benadering systeembeperkingen robuust te maken voor Gaussische stoorinvloeden.

Dit framework werd in deze thesis verbeterd op vlak van formuleringen, nauwkeurigheid van discretisatie, complexiteit, en uitbuiten van structuur zodat toepassingen op grotere schaal mogelijk worden.

De hoofdformulering van deze thesis is gebaseerd op een aparte behandeling van de Lyapunov toestanden zodanig dat de discrete periodische Lyapunov vergelijkingen (DPLE) naar voren komen, die garanderen dat de Lyapunov variabelen positief definiet blijven.

De complexiteit van het resulterend niet-linear programma wordt gereduceerd door de Lyapunov toestanden te elimineren en gebruik te maken van een DPLE solver op basis van de periodische Schur decompositie.

Deze solver werd geïmplememententeerd als oneindig-afleidbare component in het modulaire open-source optimale controle framework CasADi: ze kan geëmbed worden in een symbolische computationele graaf.

De formulering werd aangetoond om voordelig te zijn voor toepassinngen met grote toestandsruimte en beperkte tijdshorizon.

Naast open-loop traject en controle ontwerp, wordt ook de mogelijkheid van tijdsvariabele lineaire feedback controle ontwerp gedemonstreerd met het uitgebreide Lyapunov framework.

Onderzochte toepassingen zijn de automatische selectie van veilige modi om een carousel ter lancering van airborne wind-energy systemen te opereren, en een tijdsoptimale periodische controle van een quadcopter.

Abbreviations

AD	Algorithmic differentiation
AWE	Airborne wind energy
BFGS	Broyden–Fletcher–Goldfarb–Shanno (approximate Newton)
CAS	Computer algebra system
CLE	Continuous algebraic Lyapunov equation
DAE	Differential algebraic equations
DLE	Discrete Lyapunov equation
DPLE	Discrete periodic Lyapunov equations
EKF	Extended Kalman filter
FMI	Functional mockup interface (a model exchange standard)
KKT	Karush-Kuhn-Tucker conditions (first order optimality conditions)
LICQ	Linear independence constraint qualification
LMI	Linear matrix inequalities
LQR	Linear quadratic regulator
LRDPLE	Low-rank discrete periodic Lyapunov equations
LTI	Linear time-invariant
MHE	Moving horizon estimation
MPC	Model predictive control
MX	Matrix-valued expression

NED	North-east-down coordinates
NLP	Nonlinear program
NMPC	Nonlinear model predictive control
OCP	Optimal control problem
ODE	Ordinary differential equations
PDE	Partial differential equations
PDPLD	Positive definiteness preserving Lyapunov discretisation (a proposed discretisation)
PLDE	Periodic Lyapunov differential equations
PSD	Positive semidefinite
QP	Quadratic program
RC	Radio controlled
RK	Runge-Kutta (an integration scheme)
SDP	Semi-definite program
SQP	Sequential quadratic programming
SX	Scalar-valued expression
UKF	Unscented Kalman filter

Contents

Chapter 1

Introduction

1.1 Motivation

Our planet is flooded by energy from the sun. Ample streams of flowing air and water are at humanity's disposal to solve the currently faced energy problem.

Lacking is only the courage of policy makers to think big, and the ability of scientists and engineers to bring down costs of harvesting devices.

Optimal control is a technique that uses a simulated virtual reality to both design and control such systems in a way that is optimal.

Also in more classic industries, such as the process industry and manufacturing, optimal control is starting to make an impact.

A characteristic of mathematical optimisation is that it tends towards suggesting extreme settings. For optimal control, this translates into active path constraints and control bounds.

Exogenous disturbances and uncertainties in modelling make it inappropriate to operate in these nominally optimal regimes in spite of their benefits.

Much economic gain can be found in quantifying the minimal margins necessary for safe operation. This is the domain of robust optimal control. On top of the open-loop control trajectory that optimal control provides, a feedback controller can be used to reject disturbances that act on the system in an online setting.

We focus on periodic robust optimal control, and the main application is a new generation of wind energy harvesters that sweep around in the air as kites. For

such systems, it is turbulences in the flowing air mass that form the greatest source of uncertainty.

On the one hand, we seek to jointly generate periodic state trajectories and open-loop controls that optimise a combination of performance and stability, while respecting robustified path constraints. Of particular interest is the possibility to identify and optimise for open-loop control. When applicable, such type of control eliminates the need for (costly) sensors and state estimators.

On the other hand, for unstable systems, we wish to provide an answer to the reservation of industrial practioners about applying difficult-to-grasp state-of-the-art optimal control such as model predictive control (MPC) in a closed-loop setting. The robust periodic optimal control framework will be used to design simple time-variant controllers that can be used online in a gain-scheduling manner while reserving the advanced optimal control stage to an offline computation.

In essence, the aim of this thesis is to develop efficient numerical techniques to solve robust periodic optimal control problems.

1.2 Overview of existing practice

Stochastic programming is an extension of mathematical programming (optimisation) to include uncertain parameters in the *problem formulation*. The uncertainty is modelled as a stochastic variable w with a *known* bounded or unbounded distribution, in contrast to robust optimisation which assumes no distributions. Stochastic programming is unrelated to stochastic optimisation, which is about non-deterministic numerical methods to solve optimisation problems. The most basic type of stochastic programming with decision variables x involves expected values of a function $g(x, w)$ in the objective or constraints. This contrasts to having $g(x, \bar{w})$ with $\bar{w}$ the expected value of the stochastic variable in that it involves propagation of uncertainty through some model. By extension, different statistical properties (e.g. moments) – obtained analytically or by sampling techniques – may enter the problem formulation. A particular form of these so-called *here-and-now* stochastic programs is probabilistic or *chance constraint programming* (Charnes 1959; Miller 1965), with major research effort directed towards convexity analysis (Prèkopa 1970). The origin of stochastic programming lies in operations research and in particular in linear programming for multi-stage decision making (Dantzig 1955). In these formulations, only the first stage has a *here-and-now* quality, with the remaining stages having a degree of freedom to react to a concrete

realisation of the stochastic variables. We refer the interested reader to an overview of the entire field in Diwekar 2008.

Linear Quadratic Gaussian control is a classic combination of a linear-quadratic Gaussian estimator (Kalman 1960) with a linear quadratic regulator, aimed at controlling a linear system subject to Gaussian disturbances. It is also widely applied to linear time-varying systems arising from linearisation of nonlinear systems (Athans 1971) in spite of theoretical shortcomings, i.e. having no guarantees on stability margins (Doyle 1978). The design methodology requires solving the Riccati equation.

H-∞ control and robust convex optimisation Named after the norm of the transfer function that is minimised, H-∞ control is a classic approach to designing a linear robust controller (Dullerud 1999). More generally, robust min-max formulations for linear systems involving ellipsoidal or polytopic sets of bounded uncertainty can be treated as convex optimisation problems. Advances in both formulations and in convex problem solving back-ends make this a very active area of research; see e.g. Boyd 1994. For a unifying overview of these methods, we refer to Pipeleers 2009.

Robust nonlinear control General nonlinear robust control is a hard problem that can only be handled approximately or sub-optimally. The idea is to describe how uncertainty sets propagate through nonlinear system dynamics, and to use these sets in order to robustify constraints in a min-max formulation. The exact problems are computationally intractable. Propagation by Taylor expansion, proposed by Diehl 2006; Nagy 2004, offers a way to approximate the solutions of these problems relatively cheaply. More expensive algorithms have been proposed to yield conservative estimates Floudas 2007; Houska 2011a; Stein 2003, with one particular idea implemented as a special purpose integrator in Houska 2013. For a general discussion of the min-max formulations, we refer to Houska 2011.

The Lyapunov framework The research presented here builds on the framework of periodic Lyapunov differential equations (PLDE) in the context of periodic optimal control problem (OCP) formulations (Bolzern 1988; Houska 2007, 2011b). It falls in the class of robust nonlinear control, with a first-order Taylor expansion of ellipsoidal sets leading to an explicit expression for maximal violation of constraints, such that the min-max formulation reduces to a tractable nonlinear problem.

Given a nonlinear system model acting in state space $x \in \mathbb{R}^n$,

$$\dot{x} = f(x, u) \tag{1.1}$$

and the periodicity condition

$$x(0) = x(T), \tag{1.2}$$

both of which appear as constraints in the OCP, the PLDE framework augments the state space with a matrix differential equation for $P \in \mathbb{R}^{n \times n}$:

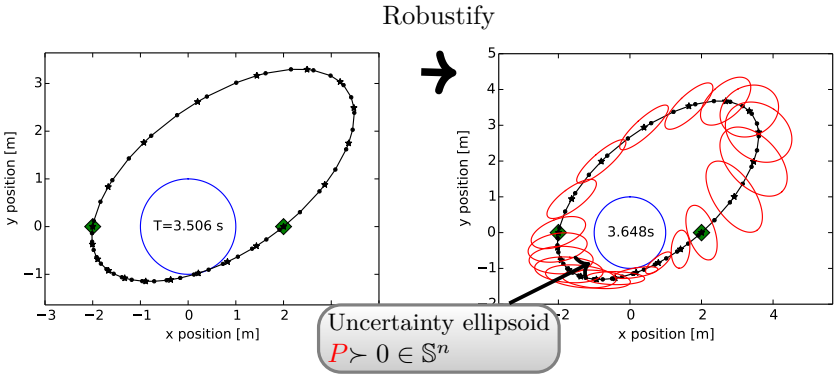
$$\dot{P} = AP + PA^\top + Q, \quad \text{where } A = \frac{\partial f}{\partial x}. \tag{1.3}$$

The main lemma in the PLDE framework amounts to the following. If a periodic solution $(x(t), P(t))$ is found with $P(t)$ everywhere positive definite, the periodic trajectory $x(t)$ of the system is asymptotically stable (Bolzern 1988; Kalman 1963).

The matrix $P(t)$ has two attractive properties (Houska 2007):

- The trace of P is a measure for stability that – in contrast to other metrics such as Floquet multipliers in Mombaur 2005 – is naturally smooth and can hence be used in an objective function and solved with powerful derivative-based optimization techniques.
- When white Gaussian disturbances enter the system, P takes an interpretation as covariance/uncertainty ellipsoid. This interpretation allows one to robustify constraints in the OCP formulation: to produce a margin that reduces the probability of violation to a desired quantity. Such formulation can be categorised as chance constraint programming and the constraints are in general not convex.

The focus in this thesis is on the latter property. The PLDE framework will be used to robustify the path constraints in periodic optimal control problems. The figure below is a preview of Chapter 8 results, in which a system is chasing past a circular obstacle. The Lyapunov framework constructs a safety-margin to back off from the obstacle in the presence of disturbances acting on the system.



An obvious down-side to the Lyapunov framework is that it uses local linearisation. As such, results for presented applications do hold only approximately. Nonlinearity can be treated conservatively as extra disturbances, but obtaining estimates for the approximation error is a hard problem. For an elaborate discussion of approximation errors, we refer to Houska 2011.

For the scope of this thesis, we assume that systems are essentially nonlinear over the broad range of state space spanned by a limit cycle, requiring nonlinear programming techniques, yet behave as time varying linear systems along that limit cycle. An example of such system is a mechanical robot with joint actuators. We are interested in optimal trajectories for which the joint angles span a large range. An example of nonlinearity that is not treatable in the used framework is hysteresis, since it affects the small-signal behaviour of the system.

1.3 Aim of the thesis

There is a certain elegance to the combined simplicity and power of the Lyapunov framework as a mathematical tool for approximate robust optimisation. Despite the promising successes demonstrated with the framework for low-dimensional systems, it has remained elusive to apply it to everyday engineering problems.

First, the original Lyapunov formulation is expensive to compute. With memory requirements scaling as the state dimension to the power of five, and computation time scaling to the power of six, it is clear that tackling large problems is problematic, even when keeping in mind that computational resources are getting larger and cheaper every year.

Second, the periodic optimal control problems that we are interested in are challenging enough to solve in the nominal case already. With essential

nonlinearities that destroy convexity, it is critical to construct a good initial guess. Adding the standard Lyapunov framework may well degrade the quality of convergence up to the point of not finding any solution at all.

One essential goal of this thesis is to identify formulations and numerical methods that overcome the above drawbacks, such that successful application to larger-dimensional (20 states) engineering-type problems lies within reach. Another goal is to explore the offline design of time-variant linear controllers using the Lyapunov framework.

The approach taken in the research is both pragmatic and generic. Where possible, decisions are made for methods that allow for off-the-shelf and well-proven libraries to be used without however restricting the scope of problem classes that can be handled. In particular, standard integrators are preferred over symplectic ones, and generic nonlinear optimisation problem solvers preferred over more restrictive convex ones. For an embedded Lyapunov solver, a standard control library was used as back-end. The open-source optimisation framework CasADi lends itself well to set up and combine standard modules to formulate optimal control problems that do not fit in standard form. Collateral benefits of this research include the improvement of CasADi to handle large-scale problems.

The style of this thesis text is didactic, conceived as a self-contained guidebook aimed at getting an apt student up to speed with solving the large-scale applications found at the end. Indeed, the improvements to the classic Lyapunov framework are such that an exciting range of new applications becomes treatable, and the ideas and computational tools delivered in this thesis may be exploited to investigate them in future application-oriented research.

1.4 Overview of the thesis and contributions

Part I: Formulations and Methods The first part of this thesis explores formulations and methods to solve robust optimal control problems efficiently.

2 **Chapter 2. Nonlinear optimal control** *Introductory chapter that sets the stage with formulating and solving optimal control problems (OCP).* The chapter starts by defining the scope of dynamic systems we wish to study (differential algebraic equations), along with techniques to integrate them, and notions of stability for steady-state and limit cycles. Next, derivative-based nonlinear optimisation techniques are briefly introduced together with the notion of algorithmic differentiation (AD) to compute gradients efficiently. Finally, optimisation and dynamic systems are combined to form OCPs, and discretisation-based direct methods are discussed as classic solution techniques. Links to state-of-the art software packages are provided where possible.

3 **Chapter 3. Systems with uncertainties and the Lyapunov Framework** *Introductory chapter to the Lyapunov framework for robust OCPs in continuous time.* In a first part, the notion of state-covariance for systems linearised along a limit cycle is introduced together with rules for its propagation in continuous and discrete time, and with a numerical example. The propagation rules lead to the Lyapunov differential equations (LDE), which are treated in the second part. Well known results from Bolzern 1988; Houska 2007, 2011b are cited to introduce the Lyapunov framework as an extension to nominal periodic OCPs to provide a mechanism for approximate robustification of path constraints, as well as a mechanism for stability optimisation. Central to the formulation are the periodic Lyapunov differential equations (PLDE). For systems with invariants, we discuss an existing treatment involving projection of constraints onto a compliant subspace (Sternberg 2012a) and propose a new formulation to propagate covariance on a reduced space. This leads to discrete periodic Lyapunov equations (DPLE) of lower dimension and hence lower computational demands. Invariant propagation is demonstrated on a simple pendulum example.

Chapter 4. Positive definiteness preserving Lyapunov discretisation *Publication-based formulations chapter on a smart discretisation for robust OCPs.* The chapter starts with highlighting a shortcoming in the classic state-augmentation approach to solving the robust OCPs of the Lyapunov framework: non-preservation of positive-definiteness after discretisation. Its contribution is that it proposes an alternative Lyapunov discretisation (PDPLD) scheme that does preserve the positive-definiteness property. An efficient formulation of PDPLD with a direct collocation method is analysed by means of a simple 4-state concrete example. This formulation still retains Lyapunov matrix entries as decision variables. This chapter is based on Gillis 2013.

4

Chapter 5. Embedded Lyapunov solvers for optimal control *Methods chapter on identifying efficient Lyapunov solvers ready for embedding.* This chapter proposes to *eliminate* Lyapunov matrix entries as decision variables from a discretised robust OCP, allowing for dedicated discrete periodic Lyapunov equations (DPLE) solvers to be exploited. The scope of optimal control problems guides the search of algorithms in the domain of linear algebra. An overview of such algorithms is provided, together with transformations between different classes of Lyapunov problems. The first main contribution is to make these solvers able to be used in a derivative based optimisation framework such as CasADi. Rules for algorithmic differentiation are derived such that the algorithms can be embedded in CasADi and derivatives of arbitrary order be computed. Two classes of algorithms are treated in-depth: periodic Schur based solvers and Smith type solvers. The second main contribution of this chapter is to compare resource usage for various methods to solve the robust OCP formulations in the Lyapunov framework. In a first part, the embedded solvers are compared in a standalone fashion. The periodic Schur method is shown to have run-time complexity $O(n^3N)$ instead of the classic $O(n^6N)$ for n-dimensional systems with horizon length N. In a second part, the performance of the embedded solvers in the context of a robust OCP is compared to the classic state-augmentation approach and the PDPLD approach from Chapter 4. Here, the periodic Schur solver leads to $O(n^3N^2)$ instead of the classic $O(n^6N)$; an improvement for large-scale systems only.

5

Part II: Software and applications The second part of this thesis explores software and applications.

6 **Chapter 6. CasADi** *Software chapter on the CasADi optimisation.* This chapter introduces the open-source software CasADi, which was co-developed with Joel Andersson during the course of the PhD research. The chapter explains how the modular and extensible design of the tool allows for fast development of new formulations, methods and algorithms for large scale optimal control problems by providing sparse-matrix valued expression graphs with algorithmic differentiation capability. A special syntax for faster prototyping formulations is introduced, followed by implementation details of the Lyapunov solvers and an example of their usage in the tool. The chapter also contributes a new method to speed up sparsity calculations for Jacobians from $O(N)$ to $O(\log(N))$ (Gillis 2014a). This chapter is all about extending and refining an existing software package.

7 **Chapter 7. Steady-state analysis and control of towed aeroplanes** *Application chapter based on Gillis 2014b publication.* This chapter elaborates on the application of airborne wind energy. After a summary of the modelling, the chapter proceeds with steady-state analysis of rotational start-up of an experimental device (carousel). The bulk result comes in a section on joint design of stochastically safe setpoints and controllers, for which the Lyapunov framework was used in a steady-state setting. It proposes a method for identifying the safest way to extend the tether on the carousel device in an open-loop fashion, as well as closed-loop. An extensive comparison between formulations is provided in a concluding section.

8 **Chapter 8. Robust periodic control of quadcopter flight** *Applications chapter.* This chapter combines all techniques discussed in previous chapters: a time-variant controller is designed concurrently with trajectory and feed-forward inputs using an embedded Lyapunov solver for a system with invariants. The application is a time-optimal periodic flight of a quadcopter model with 17 states and invariants. After problem statement and a modelling section follows a homotopy of related OCP formulations that leads to an initial guess for the final robust OCP.

Highlights of this thesis are conveniently summarised in a poster contribution to DYSCO day of spring 2014, printed on the following page.

OPTEC K.U.Leuven Optimization in Engineering Center

fwo Opening new horizons CasADi

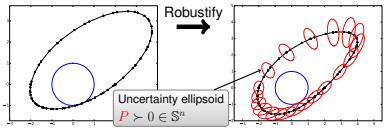
Efficient numerical methods for robust periodic optimal control with Lyapunov equations

Joris Gillis, Moritz Diehl
Department of Electrical Engineering (ESAT/STADIUS), KU Leuven

Abstract

This work compares various numerical methods to robustify periodic optimal control problems using the paradigm of Lyapunov differential equations. In this paradigm, estimates for state-covariance are obtained by solving the periodic Lyapunov equations for the a system linearised along a to-be-optimized trajectory, and are added to objective or constraints of the original optimal control problem. For non-trivial dynamical systems, method details were found to be critical to obtain algorithms with reasonable time complexity. An application for time-optimal quadcopter flight is worked out numerically with the optimal-control tool CasADi, which was extended by the author to solve discrete periodic Lyapunov equations using the SLICOT library.

Problem statement



Goal: Robustify the path constraints in a periodic optimal control problem (OCP), using the method of Lyapunov differential equations[2]

$$\begin{aligned}
\underset{x(t),u(t),P(t)}{\text{minimize}} \quad & J(x(\bullet),u(\bullet)) && \text{Objective}\\
\text{s.t. } \dot{x}(t) &= f(x(t),u(t)) && \text{System dynamics}\\
x(0) &= x(T) && \text{Periodic state}\\
0 &\leq \underbrace{h(x(t))}_{\text{tuning knob}\rightarrow} - \gamma \sqrt{\underbrace{\frac{\partial h}{\partial x} P(t) \frac{\partial h^T}{\partial x}}_{\text{Variance of } h(x)}} && \text{Scalar path constraint}\\
\dot{P}(t) &= \underbrace{A(t)P(t) + P(t)A^T(t)}_{\text{sink}} + \underbrace{Q(t)}_{\text{source}} && \text{Covariance propagation}\\
P(0) &= P(T) && \text{Periodic covariance}
\end{aligned}$$

Classic method: Augment state $[x; \text{vec}(P)]$ and feed to your favourite OCP solver (multiple shooting, direct collocation, ...)

- pendulum example ($n = 2 \ldots 4$ states)
- real applications ($n = 10 \ldots 20$)

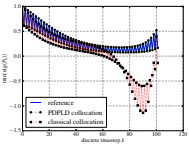
Smarter discretization[1]

Problem: Property of positive-definiteness not preserved during integration, e.g. forward Euler:

$$\begin{aligned}
P_{k+1} &= P_k + \Delta t(A_k P_k + P_k A_k^T)\\
&= (\mathbf{1} + A_k \Delta t) P_k (\mathbf{1} + A_k \Delta t)^T - (\Delta t)^2 A_k P_k A_k^T
\end{aligned}$$

Solution: Work directly in discrete time, using integrator sensitivities $\frac{\partial I}{\partial x}(x_k, u_k) \equiv \bar{A}_k$ (automatic differentiation – AD):

$$P_{k+1} = \bar{A}_k P_k \bar{A}_k^T$$



Better complexity $O(n^6) \rightarrow O(n^3)$

Observation: The (discrete) robustified OCP has n^2N extra decision variables ($P_\bullet$) and n^2N extra (linear) constraints:

$$P_{(k+1) \mod N} = \bar{A}_k P_k \bar{A}_k^T + \bar{Q}_k, \quad k = 0 \ldots (N-1)$$

→ Non-linear problem (NLP) with dense $n^2 - \text{by} - n^2$ blocks in constraint Jacobian, $O(n^6)$ runtime

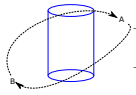
Improvement: Eliminate $P_\bullet$ and its constraints from the NLP:

$$P_\bullet = \texttt{LyapSolver}\left(\bar{A}_\bullet(x_\bullet, u_\bullet), \bar{Q}_\bullet(x_\bullet, u_\bullet)\right)$$

→ Implemented periodic Schur decomposition solver[3], using SLICOT, $O(n^3)$ runtime

→ Embedded in a CasADi expression graph, implemented forward and adjoint mode AD

Quadcopter application



→ Fly periodically $A \rightarrow B$ around obstacle as fast as possible

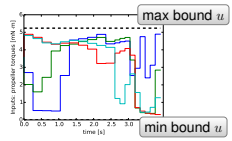
→ Nonlinear model ($n = 17$) with quaternions for orientation

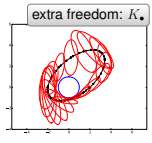
→ Linear feedback controller $K_\bullet$ to stabilize the system

Numerical: $N = 20$, 3rd-order Radau, IPOPT with BFGS

→ 1444 variables, 1548 constraints, 183739 nonzeros in Jacobian

→ 41 iterations to convergence, 2.4s for a Jacobian evaluation





References

[1] J. Gillis and M. Diehl, *A Positive Definiteness Preserving Discretization Method for nonlinear Lyapunov Differential Equations*, in Proceedings of the 52nd IEEE Conference on Decision and Control, 2013.

[2] B. Houska, *Robustness and Stability Optimization of Open-Loop Controlled Power Generating Kites*, Master's thesis, University of Heidelberg, 2007.

[3] A. Varga, *Periodic Lyapunov equations: some applications and new algorithms*, International Journal of Control, 67 (1997), pp. 69–88.

1.5 Mathematical notation and identities used in the thesis

Scalar arithmetic A thick dot ($\bullet$) is used to denote a generic symbol or number, which can be scalar, vector or matrix depending on the context. Using this notation, we can define operations to round a number:

$\lfloor \bullet \rfloor$ round down to nearest integer, $\lceil \bullet \rceil$ round up to nearest integer.

For the Kronecker delta function, we use subscripts to denote the arguments:

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0. & i \neq j \end{cases}$$

The modulo or remainder after division will be denoted as $a \bmod b$.

For compactness of notation, we will use $\tilde{N} \equiv N - 1$ in optimal control problem formulations.

Complex numbers are denoted with j as the imaginary number, e.g. $1 + 3j$.

Matrices and vectors When discussing numerical algorithms, matrices and vector symbols are not shown visually distinct from scalars; their dimensions follow from the context.

Square brackets are used when listing all entries:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \in \mathbb{R}^{2\times 3},$$

$$b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \in \mathbb{R}^2.$$

Common matrices are the identity matrix of dimension n,

$$I_n,$$

and a matrix filled with zeros of dimension m-by-n:

$$0_{m\times n}$$

The transposition of a matrix A is written as $A^\top$.

The flattening or vectorising operation reorders the entries of a matrix into a vector or column matrix:

$$\mathrm{vec}\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\right) = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix}.$$

A matrix can have properties like positive-semidefiniteness,

$$A \succeq 0,$$

or the property of symmetry:

$$A \in \mathbb{S}^n.$$

Apart from the ubiquitous matrix-matrix product, denoted by the absence of an operation symbol,

$$C = AB,$$

we consider the Kronecker product which creates a np-by-mq shaped matrix by tiling copies $B \in \mathbb{R}^{p \times q}$ multiplied with a scaling factor originating from the elements of $A \in \mathbb{R}^{n \times m}$:

$$C = A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix}$$

The Kronecker product is useful to reorder matrix-matrix products:

$$\mathrm{vec}\left(APB\right) = \left[A \otimes B^\top\right] \mathrm{vec}\left(P\right).$$

The Kronecker product is distributive with respect to transposition:

$$\left(A \otimes B\right)^\top = A^\top \otimes B^\top.$$

Matrix or vector norms are denoted as:

$$||\bullet||,$$

where a 2-norm is assumed in the absence of subscript.

One last operation on matrices is the trace, which delivers the sum of diagonal entries:

$$\mathrm{tr}A = \sum_i a_{ii}.$$

Functions Consider a continuously differentiable vector-valued function $g(x) : \mathbb{R}^n \to \mathbb{R}^m$.

The Jacobian of g serves to write a linearisation or first order Taylor approximation of such function:

$$g(x) \approx g(0) + \frac{\partial g}{\partial x}(0)x,$$

with the columns of the Jacobian enumerating the input arguments, and the rows enumerating the outputs:

$$\frac{\partial g}{\partial x}(x) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1}(x) & \frac{\partial g_1}{\partial x_2}(x) \\ \frac{\partial g_2}{\partial x_1}(x) & \frac{\partial g_2}{\partial x_2}(x) \end{bmatrix}.$$

When using the nabla notation, there is an extra transposition involved:

$$\frac{\partial g}{\partial x}(x) \in \mathbb{R}^{m \times n} = [\nabla_x g(x)]^\top$$

Trajectories For optimal control problems, we use a special notation to denote trajectories, i.e. time-series of states or controls, where the horizon is given by the context:

$x(\bullet)$ continuous state trajectory, $x_\bullet$ discrete state trajectory.

Part I

Formulations and Methods

Chapter 2

Nonlinear optimal control

Optimal control is a framework in which a time-series of *inputs* to a *dynamic system* is obtained through *optimisation* of a performance metric for that system whilst observing bounds on system properties. The mathematical formulation used in this framework is termed optimal control problem (OCP).

This chapter serves to give a birds-eye view of nonlinear optimal control. It sets the stage by highlighting concepts and notation that will be used throughout this thesis text.

Section 2.1 discusses dynamic systems in continuous time: how they are described, how their evolution is computed, and how they behave. Next, non-linear optimisation is discussed in Section 2.2. Section 2.3 concludes by combining these two concepts to formulate and solve OCPs.

The next chapter extends the OCP formulation by introducing uncertainties.

2.1 Dynamic systems

From the early days of childhood, a human brain tries to make sense of its changing environment. The ability to distinguish cause from effect, and to predict how some configuration of the environment evolves over time are critical skills for our complex society.

Learning to walk, understanding social relationships or managing hedge funds, are all activities that require building up and analysing mental models of systems.

In this thesis, we focus on continuous-time systems for which the mental models are well-proven and tested: mathematical models. In particular, lumped-element models are considered which arise in the fields of mechanical, electrical or chemical engineering, and admit a description as differential equation.

2.1.1 Mathematical modelling

Models are abstractions that describe the behaviour of natural or artificial systems in a formal way. The natural sciences have produced a rich compendium of physical laws, all formalised in the language of mathematics, which serves the modeller to create first-principles (a.k.a. white-box) models on the proverbial back-of-the-envelope as exquisitely displayed in Jones 1982. By performing experimental observations of the system in action – a resource-intensive undertaking– estimates for model parameters can be improved with parameter estimation, leading to grey-box models.

In industrial practice, it is not unusual for a model to be developed and improved by various people over several years, leading to a complex heterogeneous patchwork of possibly legacy, closed-source, badly documented or non-portable computer codes. Still, these complex models are valuable assets because they encode know-how and experience, and are successfully used for simulation and development of simple control techniques.

While recent standardisation efforts like the model exchange standard FMI and the equation-based object-oriented modelling standard Modelica alleviate some of the maintainability concerns, *simulation models* are generally not usable for advanced control techniques that promise better closed-loop performance. For these techniques, we need low-dimensionality (say up to 100 states) and differentiability; look-up tables, switch statements, embedded numerical processes, etc. are problematic. Rather than being a matter of plug-and-play, some modelling effort and model insight is deemed a necessary condition for the successful application of optimal control. Of course, existing simulation models are still valuable to validate *optimisation-ready* models. In the course of this text, optimisation-ready models will be assumed available or created from first-principles.

Our interest goes to dynamic systems, and hence to models that predict their behaviour, using a strictly Newtonian notion of time. In discrete time, a model can be written as a mapping from a joint mathematical structure of states and controls at one point in time to the state at the next point in time. This mapping can be written as a recurrence equation:

$$x_{k+1} = f(x_k, u_k). \quad k \in \mathbb{N}. \tag{2.1}$$

In continuous time, the models we consider are described by explicit ordinary differential equations (ODE):

$$\frac{dx}{dt}(t) = f(x(t), u(t)).$$

2

In control engineering practice, the mathematical structure of choice for states and controls is simply a vector space, usually over the field of the real numbers: $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ and f a vector-valued function with range $\mathbb{R}^n$. Mundane as though this choice may seem to the abstract mathematician, the choice has proven to be very powerful and convenient, not least because it is very intuitive and maps well onto the architecture of today's computers – except that the real domain is approximated by a finely-sampled grid of rational numbers using the floating point method – and onto matrix-algebra oriented computational software such as `Matlab`. On the other hand, many engineers may not even realise that a choice is involved at all and that some practical problems they experience may stem from the structure of their system failing to be diffeomorphic to a vector space (Lewis 2007). For example, rigid-body orientation is better represented by the $SO(3)$ Lie group, eliminating singularities of Euler angles. For more details, we refer to the textbooks of Hamermesh 1989 and Holm 2008. The next best thing to using the appropriate mathematical structure is to impose algebraic relations between some elements of $\mathbb{R}^n$, while inflating the dimension of the state space: a non-minimal coordinates approach.

Such type of modelling leads to differential algebraic equations (DAE), which in semi-explicit form are written as:

$$\frac{dx}{dt}(t) = f(x(t), u(t), z(t)),$$

$$0 = g(x(t), u(t), z(t)),$$

with $z \in \mathbb{R}^l$ the algebraic variables or states i.e. variables that have no derivative in the dynamics function and which can be eliminated, and g a vector-valued function with range $\mathbb{R}^l$.

Some continuous-time systems, notably mechanical systems, are usually written in an implicit form:

$$M(x(t), u(t))\dot{x} = f(x(t), u(t)),$$

with M a sparse invertible matrix. Explicitly inverting this matrix symbolically, even if computationally tractable, may not be numerically stable and will destroy its sparsity.

There are dedicated integrators available to work with exploit this particular structure, but if a specific integrator offers an interface that can handle only the semi-explicit form, one might resort to casting this system in that form:

$$\begin{cases} \dot{x} & = z \\ 0 & = M(x,u)z - f(x,u). \end{cases}$$

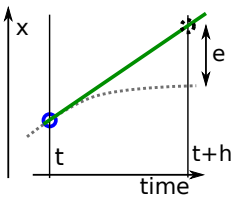
2.1.2 Time-integration

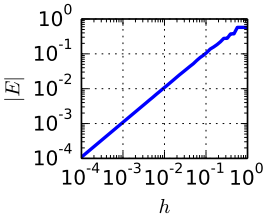
Once a suitable deterministic mathematical model is composed for a dynamic system, and the system state at the present time is given as well as the control inputs over time, it is possible to predict its future or retrace the past trajectory by integration (under conditions that typically hold in engineering practice).

A few of such such *initial-value problems* have analytic solutions, yet we deal exclusively with numerical integration here. For convenience of notation, $f(x(t),u(t))$ is written as $f(t,x(t))$ in this section because the presence of the control $u(t)$ does not change the presented methods.

Forward-Euler integration simply predicts the state after an elapsed time h by linear extrapolation of the rate of change at the present time:

$$\tilde{x}(t+h) = x(t) + f(t,x)h$$





Clearly, the *local* prediction error $e = \tilde{x}(t+h) - x(t+h)$, as seen in the middle figure, shrinks when the integration interval h is reduced. Indeed, from a Taylor argument follows $e \sim O(h^2)$. To accurately integrate the system over a horizon T, the horizon must be subdivided in a large number of integration intervals of length $\frac{T}{h}$. The figure on the right shows a typical relation between the *global* integration error $E = \tilde{x}(T) - x(T)$ over such horizon and the size of the intervals used, in logarithmic scale. The slope reveals that forward-Euler is of order 1 in the global error.

To obtain machine accuracy ($\sim 1 \times 10^{-16}$) for the global integration error with an Euler scheme, the amount of required function evaluations $f(t, x)$ may be unreasonably high (10^{16} in the graphic example above), depending on the time-scales present in the dynamics in comparison with the integration horizon.

Integration schemes of higher order allow one to reach high-accuracy with less integration intervals, to the expense of an increased per-interval computation time. We consider the family of Runge-Kutta (RK) schemes, in which the non-linear right-hand side f is sampled multiple times during one integration interval:

$$\tilde{x}(t+h) = x(t) + \sum_{i=1}^{s} b_i k_i$$

$$k_i = hf(t + c_i h, x(t) + \sum_{i=j}^{s} a_{ij} k_j) \qquad i = 1, 2, \ldots, s$$

The coefficients in this scheme can conveniently be arranged in a so-called Butcher tableau:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline & b_1 & b_2 & \ldots & b_s \end{array} = \begin{array}{c|c} c & A \\ \hline & b^\top \end{array} \qquad \text{with} \quad \begin{array}{c} 0 \leq c \leq 1 \\ \sum_{i=1}^{s} b_i = 1 \end{array}$$

When $a_{ij} = 0$ for $j \geq i$ – no blue entries in the tableau – the scheme is *explicit*: one can simply obtain all k_i by recursion starting from $k_1 = hf(t + c_1 h, x(t))$. For improved *stability* of the integration method, in particular for *stiff* systems, *implicit* schemes are often used. In that case, the equations defining $k_1, \ldots, k_s$ constitute a system of non-linear algebraic equations, and is solved by root-finding as discussed in Section 2.2. Another advantage of an implicit scheme is that index-1 DAEs are trivial to treat: the algebraic part of the DAE is simply added to the system of equations of the scheme.

The combination of dimension s and numerical values for the tableau elements determines the order of the scheme. A popular choice is the fourth-order explicit Runge-Kutta scheme, since it is easy to implement and corresponds to the largest order explicit scheme for which the minimal required dimension s matches the order (Hairer 1993). Its Butcher tableau reads:

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

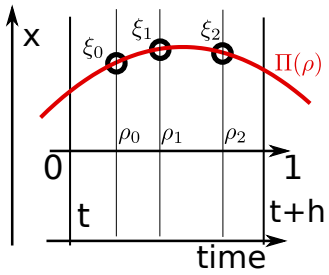
An important class of implicit RK schemes are the *collocation* methods. A collocation method of degree d introduces helper variables $\xi_0, \xi_1, \ldots, \xi_d \in \mathbb{R}^n$ at times $h\rho_0, h\rho_1, \ldots, h\rho_d$ from the start of the integration interval to parametrise a polynomial on the interval.

For convenience, let us group these helper variables into a single vector:

$$z = \begin{pmatrix} \xi_0 \\ \ldots \\ \xi_d \end{pmatrix}$$

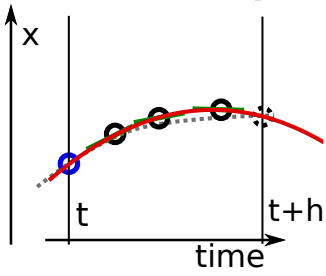
The polynomial $\Pi(\rho; z) : \mathbb{R} \times \mathbb{R}^{(d+1)n} \to \mathbb{R}^n, \rho \mapsto \Pi(\rho; z)$ interpolates the points (ρ_j, ξ_j) in an exact way:

$$\xi_j \equiv \Pi(\rho_j; z). \qquad j = 0 \ldots d$$



To perform integration with this polynomial, we require first that the polynomial evaluated at the start of the integration interval should match the initial value for that interval, and second that derivatives of the interpolating polynomial should match the state dynamics at each collocation point:

$$\begin{pmatrix} \Pi(0; z) - x(t) \\ \frac{\partial \Pi}{\partial \rho}(\rho_1; z) - hf\left(t + h\rho_1, \Pi(\rho_1; z)\right) \\ \ldots \\ \frac{\partial \Pi}{\partial \rho}(\rho_d; z) - hf\left(t + h\rho_d, \Pi(\rho_d; z)\right) \end{pmatrix} = 0.$$



This set of $n(d+1)$ nonlinear equations implicitly defines a solution z^* for the helper variables. Lastly, the state at the end of the integration interval can be obtained as:

$$\tilde{x}(t+h) = \Pi(1; z^*).$$

A collocation method is defined completely by the location of its points $(\rho_0, \ldots, \rho_d)$. Two popular classes exist: Radau (which has $\rho_d = 1$) and Legendre. Both have $\rho_0 = 0$.

By combining two methods of different order, an estimate for the integration error can be obtained. *Adaptive step-size* solvers use this information online to adapt h for each integration interval, such that a desired local accuracy can be guaranteed with a minimum of computation effort. The Sundials suite (Sundials 2009) is a mature package for efficient adaptive-step size integration, using *multistep* methods. In such method, rather than sampling the right-hand side f multiple times in each integration interval as in RK schemes, samples at the boundaries of preceding integration intervals are used to construct a higher order scheme.

For the purpose of this thesis, time-integration is considered a straightforward task; the used models have no mode changes inside the integration intervals (though it is still possible to treat modes on a higher level, i.e. multi-stage), and they exhibit no singularities. Widely different time-scales can be avoided by the use of DAE models or dealt with by the use of off-the-shelf adaptive step-size solvers.

2.1.3 Stability

Equilibrium Some autonomous systems $\dot{x} = f(x)$ have steady-state or equilibrium solutions, i.e. there exists a state $x^\star$ such that:

$$0 = f(x^\star).$$

The notion of stability deals with characterising how the system behaves in a small neighbourhood of state space around the equilibrium. The following is a summary of standard stability definitions that can be found in appropriate textbooks such as Teschl 2012. For a historical perspective, we refer to Leine 2010.

Definition An equilibrium of a system is *Lyapunov stable* when limiting the magnitude δ of an initial excursion can be used to confine all future states to an arbitrarily neighbourhood ϵ of the equilibrium:

$$\forall \epsilon > 0 \quad \exists\, \delta > 0 : [||x(0) - x^\star|| < \delta \Rightarrow ||x(t) - x^\star|| < \epsilon \, \forall t \geq 0] \,.$$

A stricter version of stability requires the magnitude of the perturbation to die out over time:

Definition An equilibrium of a system is *asymptotically stable* if and only if it is Lyapunov stable and there exists a magnitude of excursion that always leads to asymptotically reaching the equilibrium:

$$\exists\, \delta > 0 : \left[||x(0) - x^\star|| < \delta \Rightarrow \lim_{t \to \infty} ||x(t) - x^\star|| = 0 \right].$$

Lyapunov conceived the mathematical apparatus to study the stability of systems (Lyapunov 1907). Classically, the construction of a Lyapunov function $V(x)$ is used to prove stability:

Theorem 2.1.1. *An equilibrium of a system is Lyapunov stable in some neighbourhood if there exists a scalar function $V(x)$ which is positive everywhere and only zero at $x^\star$, and with $\dot{V}(x) = \frac{\partial V}{\partial x}\frac{dx}{dt}$ negative-or-zero everywhere and zero at $x^\star$.*

More strongly, asymptotic stability follows when $\dot{V}(x)$ is negative everywhere and only zero at $x^\star$.

Requiring $x^\star = 0$ without loss of generality, $V(x)$ can be taken as a quadratic in the neighbourhood of the equilibrium:

$$V(x) = x^\top P x. \qquad P \in \mathbb{S}^n$$

Locally, the Taylor expansion $f(x) = \frac{\partial f}{\partial x}(0)x$ approximates f around $x^\star = 0$, and we can approximate $\dot{V}(x)$:

$$\dot{V}(x) = \frac{\partial V}{\partial x}\frac{dx}{dt} = x^\top \left[\frac{\partial f}{\partial x}^\top P + P \frac{\partial f}{\partial x} \right] x.$$

The conditions for local Lyapunov stability are satisfied when:

$$P \succ 0 \qquad \text{positive definite} \tag{2.2}$$

$$\frac{\partial f}{\partial x}^\top (0) P + P \frac{\partial f}{\partial x}(0) \preceq 0. \qquad \text{negative semidefinite} \tag{2.3}$$

Conditions for local asymptotic stability are obtained from the above by making the second matrix inequality strict. A sufficient condition for this inequality is that all eigenvalues of $\frac{\partial f}{\partial x}$ lie strictly in the left half-plane. We refer to Willems 1972 for proofs and physical interpretation in terms of loss of energy or dissipativity of systems.

For linear systems, controller design with stability requirements is typically cast in the form of a semi-definite program (SDP) with linear matrix inequality (LMI) constraints (Boyd 1994). By contrast, in this thesis, nonlinear systems lead to general non-convex nonlinear programs (NLP).

Limit cycles An extension to equilibrium points is the notion of limit cycles, i.e. closed trajectories in state space with a period T:

$$x^\star(t+T) = x^\star(t)$$

Limit cycles can occur naturally in autonomous systems:

$$\dot{x} = f(x),$$

see example in Figure 2.1, or may occur when an system is forced using a periodic control input, in general:

$$\dot{x} = f(t,x) \quad \text{with } f(t,x) = f(t+T,x).$$

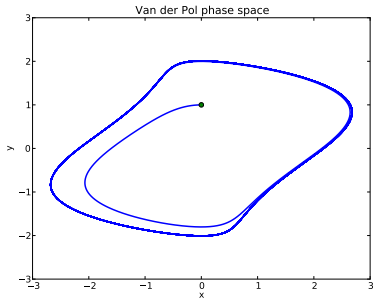
As before, stability is about behaviour of excursions. The figure illustrates a stable attractor: excursions in any direction evolve towards the limit-cycle, and the mapping of an individual excursion $\delta x(0)$ on top of $x^\star(0)$ tends to undergo a linear transformation in the limit of small excursions:

$$\delta x(t) \sim S(t)\delta x(0), \qquad \delta x(0) \to 0 \tag{2.4}$$

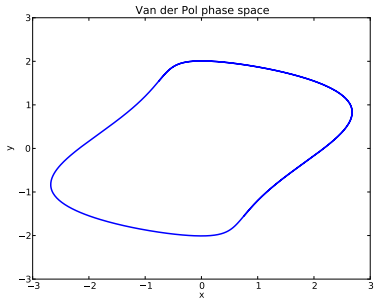
with $S(t) \equiv \frac{\partial x(t)}{\partial x(0)}$ the sensitivity matrix, computable as:

$$\dot{S}(t) = \frac{\partial f}{\partial x}(t, x(t))S(t). \qquad S(0) = I_n \tag{2.5}$$

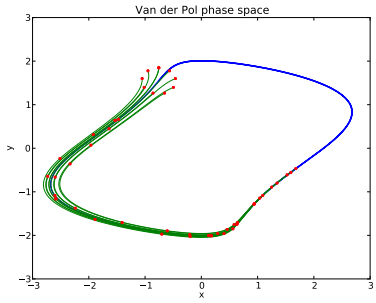
We refer to Appendix A.2 for a derivation.



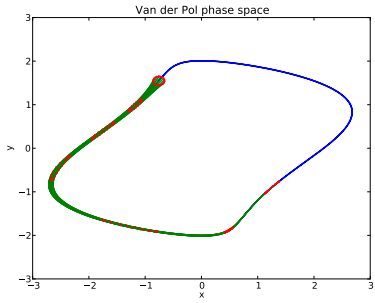
(a) Transient behaviour to reach the limit cycle.



(b) Limit cycle $x^\star(t)$ reached.



(c) Time evolution of *large* excursions from the limit cycle



(d) Time evolution of *small* excursions from the limit cycle

Figure 2.1: Stable attractor of the autonomous Van der Pol oscillator.

For the case of linear systems, $S(t)$ is better known as the *principal fundamental matrix solution*, and more specifically for linear time-invariant (LTI) systems, it is given by the matrix exponential: $S(t) = \exp(At)$.

Consider now how the excursion evolves over several periods:

$$\delta x(T) = S(T)\delta x(0)$$

$$\delta x(2T) = S(T)^2\delta x(0)$$

$$\vdots$$

It is clear that the eigenvalues of $S(T)$, also known as the *monodromy matrix* M, are critical in defining a notion of stability. For autonomous systems, these

eigenvalues – also known as *Floquet multipliers* – must all-but-one be inside the unit circle to have asymptotic stability of the limit cycle. The exception (an eigenvalue of $1 + j0$) comes from the direction tangent to the trajectory: such direction amounts to a time-delay and there is no process that could shorten or lengthen a time-delay when the system has reached the limit cycle in all other directions.

More formally, the stability of limit cycles is studied by means of a *Poincaré section*: a hyperplane cutting through the trajectory in state space and making an angle with it at the crossing point. The question of n-dimensional continuous-time limit cycle stability is transformed to the stability of the $(n-1)$-dimensional discrete-time system mapping from one crossing point with the hyperplane to the next crossing after a time T.

The interested reader can find more details in textbooks such as Thompson 2002 and literature (Ghaffari 2009).

2.2 Non-linear continuous optimisation

A human observer can usually spot the extremum (maximum or minimum) of a non-linear scalar function by a quick glance at a plot. This is possible if a) a computer has tabulated the function on a fine mesh and mapped to pixel positions, and b) the decision variable is of low dimension. Such brute-force technique is called *grid search* and could well be automated by a computer program, given the basic assumption of Lipschitz continuity.

The functions that arise in direct methods for optimal control are large-dimensional (thousands of decision variables) and expensive to compute (milliseconds – seconds), such that grid search is out of the question. The methods presented in this section can be found in classic textbooks (Nocedal 2006).

In general, the nonlinear programs (NLP) of interest can be written as:

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimise}} & f(x) & \text{Objective function: } \mathbb{R}^n \to \mathbb{R} \\ \text{subject to} & g(x) = 0, & \text{Equality constraints: } \mathbb{R}^n \to \mathbb{R}^m \\ & h(x) \geq 0, & \text{Inequality constraints: } \mathbb{R}^n \to \mathbb{R}^p \end{array} \tag{2.6}$$

where it should be clear from the context that x, f and g bear totally different meanings than in the previous section and where we make no assumptions on convexity.

We wish to exploit the inherent smoothness of these functions by using derivative-based methods, excluding evolutionary or stochastic algorithms.

On a practical often-overlooked note, it is important for the to-be-discussed optimisation schemes that decision variables, objective functions and constraints are all properly scaled, i.e. have numerical values on the order of 1, let's say 0.01 – 100. Omitting to scale may impact convergence behaviour severely to the point of no convergence at all. The main reasons behind this are a) the importance of conditioning of matrices inside the methods and b) algorithms rely on heuristics involving unscaled norms such as the Euclidean norm.

2.2.1 Newton-type techniques

The Newton-method for root-finding of a function $F(w) = 0$ is based on iterative linear extrapolation using the first order Taylor expansion of the function:

Input: Starting value w_0, $\epsilon > 0$
Output: $w^\star$ for which $||F(w^\star)|| \leq \epsilon$
$k \leftarrow 0$;
while $||F(w_k)|| > \epsilon$
do
  Solve the linear system: $F(w_k) + \frac{\partial F}{\partial w}(w_k)p_k = 0$ for p_k ;
  Perform a step: $w_{k+1} = w_k + p_k$;
  $k \leftarrow k + 1$;
end
$w^\star \leftarrow w_k$;

Under technical conditions, local quadratic convergence is achieved. Globalisation techniques (e.g. line-search) can help to extend the region of convergence.

The Newton-method can be applied to the first-order optimality conditions (Karush-Kuhn-Tucker or KKT conditions) of an equality constrained NLP, i.e.:

$$\frac{\partial f}{\partial x}^\top - \frac{\partial g}{\partial x}^\top \lambda = 0,$$
$$g(x) = 0,$$

with $\lambda \in \mathbb{R}^m$ the Lagrange multipliers. The solution of the linearised system,

$$\begin{bmatrix} \frac{\partial f}{\partial x}^\top(x_k) \\ g(x_k) \end{bmatrix} + \begin{bmatrix} H(x_k, \lambda_k) & \frac{\partial g}{\partial x}^\top(x_k) \\ \frac{\partial g}{\partial x}(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ -\lambda_{k+1} \end{bmatrix} = 0,$$

is used to produce a step $p_k = [\Delta x_k, \lambda_{k+1} - \lambda_k]^\top$ in the iteration variable space $w = [x, \lambda]^\top$. The derivation requires that the constraint Jacobian $\frac{\partial g}{\partial x}$ is of full row-rank (LICQ condition).

The symbol H is the Hessian of the Lagrangian:

$$H = \frac{\partial^2 f}{\partial x^2} - \sum_{i=1}^{m} \lambda_i \frac{\partial^2 g_i}{\partial x^2}.$$

At a minimiser, this Hessian has positive-or-zero curvature in all directions compatible with the equality constraints; the reduced Hessian is positive semidefinite: $Z^\top H Z \succeq 0$ for $\frac{\partial g}{\partial x} Z = 0$.

To avoid the computational effort of obtaining the exact Hessian, and to ensure positive definiteness in all iterates, H is often approximated in the limit by low-rank updates arising from first-order sensitivities such as the BFGS method (Broyden 1969; Fletcher 1970; Goldfarb 1970; Shanno 1970). This does have the effect of reducing the convergence rate from quadratic to super-linear.

Sequential quadratic programming The Newton-type scheme above can be interpreted as a sequential quadratic programming (SQP) method. With an extension for inequalities (Lagrange multipliers $\nu \in \mathbb{R}^p$), this method is given

by:

Input: Starting value x_0, λ_0, ν_0, $\epsilon_{\text{pr}}, \epsilon_{\text{du}} > 0$
Output: Local minimum $x^\star$
$k \leftarrow 0$;
while $\left\| \begin{bmatrix} h^-(x_k) \\ g(x_k) \end{bmatrix} \right\| > \epsilon_{pr}$ **or** $\left\| \frac{\partial f}{\partial x}(x_k) - \lambda_k^\top \frac{\partial g}{\partial x} - \nu_k^\top \frac{\partial h}{\partial x} \right\| > \epsilon_{du}$
do
  Solve a QP, obtaining p_k, λ, ν:

$$\begin{aligned} \underset{p_k \in \mathbb{R}^n}{\text{minimise}} \quad & \frac{\partial f}{\partial x}(x_k) p_k + \tfrac{1}{2} p_k^\top H(x_k, \lambda_k, \nu_k) p_k \\ \text{subject to} \quad & g(x_k) + \frac{\partial g}{\partial x}(x_k) p_k = 0 \\ & h(x_k) + \frac{\partial h}{\partial x}(x_k) p_k \geq 0 \end{aligned}$$

  Determine a step length $0 < \alpha \leq 1$;
  Perform a step:

$$x_{k+1} = x_k + \alpha p_k$$

$$\lambda_{k+1} = (1 - \alpha)\lambda_k + \alpha\lambda$$

$$\nu_{k+1} = (1 - \alpha)\nu_k + \alpha\nu$$

  $k \leftarrow k + 1$;
end
$x^\star \leftarrow x_k$;

with h^- the part of the inequality constraints that are violated.

Here, H is again the Lagrange Hessian, but now with extra multipliers:

$$H = \frac{\partial^2 f}{\partial x^2} - \sum_{i=1}^{m} \lambda_i \frac{\partial^2 g_i}{\partial x^2} - \sum_{i=1}^{p} \nu_i \frac{\partial^2 h_i}{\partial x^2},$$

and is commonly approximated by BFGS-like techniques.

The main challenge in building a performant SQP solver lies in creating globalisation strategies; i.e. constructing a good merit function for line-search to choose the step length.

Several large-scale SQP solvers exist as off-the-shelf libraries. The WORHP solver (Büskens 2012) supports exact Hessians as well as numerous BFGS-type approximations. The SNOPT solver (Gill 2005) supports the latter only. Neither is open-source but free academic licenses are available. In this thesis text we will

also make use of the open-source `SQPMethod` solver inside CasADi (Andersson 2012), which is a simple textbook-style SQP implementation.

Popular choices for large-scale sparse QP solvers are the open-source OOQP package (Gertz 2003), which solves only positive definite QPs and thus requires regularisation of an exact Hessian or the use of BFGS, and the commercial CPLEX package (IBM Corp. 2009), which can find a local solution in the indefinite case.

Interior point methods Very simply put, interior point methods work by replacing inequality constraints by smooth barrier functions:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimise}} & f(x) - \mu \sum_{i=1}^{p} \log(h_i(x)) \\ \text{subject to} & g(x) = 0. \end{array} \tag{2.7}$$

The steepness of the barrier, μ, is gradually adjusted while the method converges. This constitutes *homotopy*: a gradual adaption from a simple problem to a complex problem.

As before, a Newton-type method can be constructed from the KKT conditions of this problem.

A key difficulty with interior-point methods is their inability to *hotstart*: even if the solver is initialised at – or very close to – the optimal solution, an interior-point solver may walk away only to converge again (hopefully!) to the same point in the end.

On the other hand, these methods can be exceptionally robust in terms of handling bad initial guesses or ill-posed problems. For the purpose of off-line optimal control, they are well-suited. When used in a control setting in which similar problems are solved successively, SQP methods are a better match (Diehl 2009a).

IPOPT (Wächter 2006b) is a high-quality open-source implementation of an interior-point method.

Linear algebra considerations At the heart of the discussed Newton-type solvers is the factorisation of a large-scale sparse symmetric indefinite matrix:

$$K = \begin{bmatrix} B & A^\top \\ A & 0 \end{bmatrix},$$

where B is the exact or approximated Lagrange Hessian, augmented with an extra diagonal term in the case of an interior point method, and A is the Jacobian of equality and (active) inequality constraints.

The NLP solvers used in this thesis employ a direct $LDL^\top$ factorisation, an adaption to *Cholesky* factorisation ($LL^\top$) to treat indefiniteness (Gill 1981). A symbolic factorisation using the sparsity pattern is performed upfront to obtain the structure of the factors. It is crucial for scalability of the method that a permutation of K is found that results in minimal fill-in of L.

Finding the permutation that minimises the fill-in of Cholesky factors is NP-hard, but efficient heuristic ordering methods exist: two widely used methods are *approximate minimum degree* ordering (AMD, Amestoy 1996) and the algorithm of the METIS library (Karypis 1998).

Figure 2.2 shows a stylised version of the constraint Jacobian structure that will be encountered in the robust optimal control problem formulations of this thesis. Focusing on the A block, the band-diagonal contribution stems from the discretisation of the dynamic constraints as in multiple-shooting or direct-collocation methods, to be discussed in the next section. The dense column is caused by having time as a decision variable and this variable affecting all dynamics. The dense rows are caused by elimination of Lyapunov variables as will be discussed in Chapter 5. The Figure shows how this representative sparsity pattern is effectively exploited after reordering.

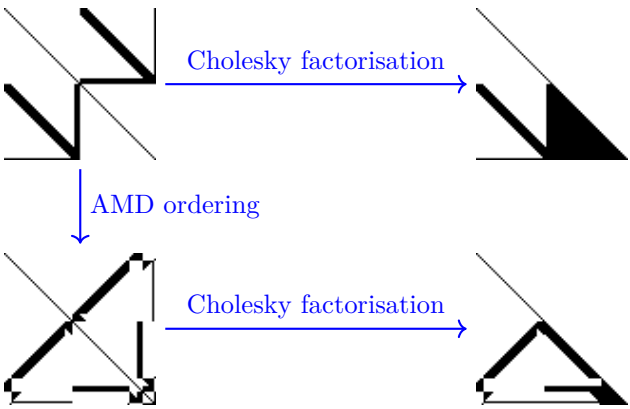


Figure 2.2: Structure preservation after reordering in Cholesky factorisation of a typical sparsity pattern.

Many codes for the $LDL^\top$ factorisation are based on *multifrontal* algorithms, an adaption of sparsity-aware Gaussian elimination that can benefit from

parallelism (Duff 1983). The Harwell Software Library (HSL 2011) maintains a set of high-performance factorisation methods, notably MA57 and MA97. A similar option is WSMP from IBM (Gupta 2001). These codes are commercial, but free for academic use, though redistribution is prohibited. An option in the public domain is the MUMPS solver (Amestoy 2000). While it is used extensively in the field of finite-element simulations, for the use in optimisation, it is by far not on-par with its commercially developed alternatives.

For dense subproblems, the open-source[1] LAPACK package is used (Anderson 1999), and matrix-vector operations are written in terms of BLAS routines (Blackford 2002), for which numerous open-source implementations exist. A prominent BLAS routine is `DGEMM` for matrix-matrix products. For this routine, the triple-nested for loop that defines the matrix product is re-ordered into blocks as to exploit the hierarchical memory layout of modern CPUs.

2.2.2 Algorithmic differentiation

A key point in using derivative-based optimisation is that knowledge of the gradient is required. The field of algorithmic differentiation (AD) – see e.g. Bartholomew-Biggs 2000; Griewank 1989 – deals with techniques to efficiently construct sensitivities with high precision.

When the function $F(x) : \mathbb{R}^n \to \mathbb{R}^m$ that needs to be differentiated is not of a black-box type, but instead an interpretable algorithm with run-time $\mathcal{T}$, algorithmic differentiation offers a mechanistic way to derive two algorithms from the original $F(x)$ that accurately evaluate the following sensitivities or directional derivatives:

$$\begin{array}{ll} \text{Forward sensitivity} & \mathrm{AD}_F^{\mathrm{fwd}}(x, s^{\mathrm{fwd}}) = J(x)s^{\mathrm{fwd}}, \quad s^{\mathrm{fwd}} \in \mathbb{R}^n \\ \text{Adjoint/reverse sensitivity} & \mathrm{AD}_F^{\mathrm{adj}}(x, s^{\mathrm{adj}}) = J^\top(x)s^{\mathrm{adj}}, \quad s^{\mathrm{adj}} \in \mathbb{R}^m, \end{array}$$

with $J = \frac{\partial F}{\partial x} : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ the Jacobian of F and with the run-time of either of the algorithms AD_f a small multiple of $\mathcal{T}$.

Obtaining a Jacobian from sensitivities A straightforward approach to recover all Jacobian entries is to seed the sensitivity function with columns of an identity matrix. In this way, the forward and reverse sensitivities correspond directly to columns and rows of the sought-after Jacobian respectively. For $m \ll n$, the obvious choice is to use m adjoint sensitivities, while in the $n \ll m$

[1] Note that there exist numerous commercial implementations of LAPACK and BLAS, too.

case, using n forward sweeps is cheapest. With this strategy, the cost for a total Jacobian is in the order of $\min(n, m) \cdot \mathcal{T}$.

As an example, consider a function with $n = 4$ inputs and $m = 3$ outputs. Its Jacobian can be written with 12 unknowns:

$$J = \begin{bmatrix} a & d & g & j \\ b & e & h & k \\ c & f & i & l \end{bmatrix}.$$

Evaluating the forward sensitivity $\mathrm{AD}_F^{\mathrm{fwd}}$ using the first column of I_n as seed results in a concrete vector of numbers v:

$$\begin{bmatrix} a & d & g & j \\ b & e & h & k \\ c & f & i & l \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = v.$$

In this way, the values in the first column of J are identified. In total four forward sensitivity sweeps or – alternatively – three adjoint sensitivity sweeps are needed to obtain the values of the full Jacobian.

If one knows the sparsity of J beforehand, the number of required sensitivities can potentially be drastically reduced. For example, when $n = m$ and J is known to be diagonal, a single sensitivity evaluation with seed $[1, 1, \ldots]^\top$ suffices. More generally, a colouring of the *column intersection graph* of the sparsity pattern of J provides a small set of seeds usable to obtain the full Jacobian (Curtis 1974; Gebremedhin 2005).

Extending the above example, assume now that some entries of J are known to be structurally zero:

$$J = \begin{bmatrix} & & g & j \\ b & & h & \\ & f & i & \end{bmatrix}.$$

The forward colouring of this structure reveals a set with two sensitivity seed vectors:

$$\left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\} \Leftrightarrow \begin{bmatrix} & & g & j \\ b & & h & \\ & f & i & \end{bmatrix}$$

The interpretation of the colouring is simple when inspecting the coloured Jacobian entries: at most a single entry of each colour may be present in each row of the Jacobian.

2

When one evaluates the forward sensitivity $\mathrm{AD}_F^{\mathrm{fwd}}$ using a coloured seed vector, one obtains the values for all the entries of that color in the Jacobian:

$$\begin{bmatrix} & & g & j \\ b & & h & \\ & f & i & \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b \\ f \\ 0 \\ j \end{bmatrix}.$$

In this way, the full Jacobian is constructed using only two sensitivity sweeps.

How to obtain sensitivities Consider two independent variables (or *inputs* in procedural terminology) x, y and two dependent variables (or *outputs*) z, w defined by functions f and g, all possibly vector-valued:

$$z = f(x, y) \qquad w = g(x, y). \tag{2.8}$$

The forward mode of algorithmic differentiation expresses how perturbations of the inputs influence the outputs as:

$$\dot{z} = \frac{\partial f}{\partial x}(x, y)\dot{x} + \frac{\partial f}{\partial y}(x, y)\dot{y} \qquad \dot{w} = \frac{\partial g}{\partial x}(x, y)\dot{x} + \frac{\partial g}{\partial y}(x, y)\dot{y}. \tag{2.9}$$

The differential perturbations $\dot{x}$ and $\dot{y}$ are commonly referred to as dot-quantities. Note how the dot-quantities appear at the right-hand of the partial derivatives; they are column vectors. This expression is equivalent with the *total differential* of the defining functions using differentials dx, dy:

$$dz = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy \qquad dw = \frac{\partial g}{\partial x}dx + \frac{\partial g}{\partial y}dy. \tag{2.10}$$

The adjoint (or reverse) mode starts from perturbations on the outputs and traces their origin back to perturbations on the inputs:

$$\bar{x} = \bar{z}\frac{\partial f}{\partial x}(x, y) + \bar{w}\frac{\partial g}{\partial x}(x, y) \qquad \bar{y} = \bar{z}\frac{\partial f}{\partial y}(x, y) + \bar{w}\frac{\partial g}{\partial y}(x, y). \tag{2.11}$$

The differential perturbations $\bar{z}$ and $\bar{w}$ are commonly referred to as bar-quantities or adjoints. Note how the bar-quantities appear at the left-hand of the partial derivatives; they are row vectors. In general, bar-quantities have a structure that is the transpose of the structure of their bar-less counterparts. It should be noted that in CasADi software (Andersson 2012), bar-quantities are defined as the transpose of the mathematical definition used in this thesis text, for efficiency reasons.

The mechanistic approach to obtaining sensitivities consists of first writing a computation as a sequence of operations acting on inputs and intermediate variables, e.g.

Input: x_1, x_2
Output: x_3, x_4
$x_3 \leftarrow f_1(x_1, x_2)$;
$x_4 \leftarrow f_2(x_1, x_2, x_3)$;

and then applying the above rules on the algorithm:

Forward mode	**Adjoint mode**
Input: x_1, x_2 **Forward seeds**: $\dot{x}_1$, $\dot{x}_2$ **Output**: x_3, x_4 **Forward sensitivities**: $\dot{x}_3$, $\dot{x}_4$ $x_3 \leftarrow f_1(x_1, x_2)$; $\dot{x}_3 \leftarrow \frac{\partial f_1}{\partial x_1}\dot{x}_1 + \frac{\partial f_1}{\partial x_2}\dot{x}_2$; $x_4 \leftarrow f_2(x_1, x_2, x_3)$; $\dot{x}_4 \leftarrow \frac{\partial f_2}{\partial x_1}\dot{x}_1 + \frac{\partial f_2}{\partial x_2}\dot{x}_2 + \frac{\partial f_2}{\partial x_3}\dot{x}_3$;	**Input**: x_1, x_2 **Adjoint seeds**: $\bar{x}_3$, $\bar{x}_4$ **Output**: x_3, x_4 **Adjoint sensitivities**: $\bar{x}_1$, $\bar{x}_2$ $x_3 \leftarrow f_1(x_1, x_2)$; $x_4 \leftarrow f_2(x_1, x_2, x_3)$; $\bar{x}_1, \bar{x}_2 \leftarrow 0$; $\bar{x}_1 \leftarrow \bar{x}_1 + \bar{x}_4 \frac{\partial f_2}{\partial x_1}$; $\bar{x}_2 \leftarrow \bar{x}_2 + \bar{x}_4 \frac{\partial f_2}{\partial x_2}$; $\bar{x}_3 \leftarrow \bar{x}_3 + \bar{x}_4 \frac{\partial f_2}{\partial x_3}$; $\bar{x}_1 \leftarrow \bar{x}_1 + \bar{x}_3 \frac{\partial f_1}{\partial x_1}$; $\bar{x}_2 \leftarrow \bar{x}_2 + \bar{x}_3 \frac{\partial f_1}{\partial x_2}$;

For a convenient collection of AD rules for matrix-valued functions, see Giles 2008. Of particular use is the following rule for matrix multiplication:

$$C \leftarrow AB \ ;$$

Forward	Adjoint		
$\dot{C} \leftarrow \dot{A}B + A\dot{B}$;	$\bar{A} \leftarrow B\bar{C}$; $\bar{B} \leftarrow \bar{C}A$;	or	$\bar{A}^\top \leftarrow \bar{C}^\top B^\top$; $\bar{B}^\top \leftarrow A^\top \bar{C}^\top$;

It should be noted that there is a simple connection between AD rules (2.9) and (2.11) via the multiplication rule. If one has an algorithm for forward AD,

$$\dot{z} \leftarrow \tfrac{\partial f}{\partial x}(x,y)\dot{x} + \tfrac{\partial f}{\partial y}(x,y)\dot{y};$$
,

one can apply the adjoint AD rules on top of this algorithm, creating bar-quantities of dot-quantities. If plain bar-quantities are introduced in the process to deal with variations of original variables, the result is an adjoint-over-forward algorithm, providing second order information. However, if the original variables are taken constant in the process, one obtains:

$$\bar{\dot{x}} \leftarrow \bar{\dot{z}}\tfrac{\partial f}{\partial x}(x,y) \; ;$$
$$\bar{\dot{y}} \leftarrow \bar{\dot{z}}\tfrac{\partial f}{\partial y}(x,y) \; ;$$
,

which is, after dropping the dots, exactly the adjoint AD algorithm shown earlier. Rather than being a tautological remark, there is a practical consequence to this connection:

Corollary 2.2.1. *An adjoint algorithm can be obtained by operating on a forward algorithm, as alternative to operating on the original algorithm, if the original quantities are taken constant. Bar-quantities of dot-quantities must be simplified to plain bar-quantities.*

This corollary will be used in Section 5.3.

Software There are several software tools available that perform algorithmic differentiation. Some packages (e.g. TAPENADE, *TAPENADE* 2013) focus on source-code-transformation: they can directly interprete the source-code of a C/C++/Fortran function, and produce an extended version with seed inputs and sensitivity outputs. Typically, no adaptions to source-code are needed. A second-class (e.g. ADOL-C Griewank 1999) works with operator-overloading: they provide an intelligent wrapper around numerical data-types for which all mathematical operations are overloaded such that a sequence of operations is obtained concurrently with numerical evaluation. Minimal adaptions to

source-code are needed. A third class (e.g. CasADi Andersson 2012) uses a hybrid: the user constructs a symbolic computational graph by operating on symbolic data-types that behave like numerical data-types, with source-code transformation applied in-memory to the constructed graph. Large adaptions to source-code are needed.

2.3 Optimal control problems

Recall the continuous-time models obtained through modelling (Section 2.1.1):

$$\dot{x} = f(x(t), u(t)),$$

with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$.

These models can be used for optimisation in a continuous-time optimal control problem (OCP) formulation:

$$\begin{array}{lllll}
\underset{x(\bullet),u(\bullet)}{\text{minimise}} & \int_0^T l(x(t),u(t))dt + E(x(T)) & & & \text{Objective} \\
\text{subject to} & \dot{x} = f(x(t),u(t)), & t \in [0,T] & & \text{Dynamic constraints} \\
& 0 \geq h(x(t),u(t)), & t \in [0,T] & & \text{Path constraints} \\
& 0 \geq r(x(T),x(0)). & & & \text{Boundary constraints}
\end{array}$$

In this notation, $x(\bullet)$ and $u(\bullet)$ corresponds to the entire state/control trajectory on $t \in [0,T]$. The optimisation problem is infinite-dimensional at this point since, in the mathematical sense, the value of time can take any real value between 0 and T. Further, $l(x,u)$ is a Lagrange term integrand, and E a Mayer term. The formulation is also trivial to extend to DAE systems.

The notation for path ($h \in \mathbb{R}^q$) and boundary ($r \in \mathbb{R}^b$) constraints is not limited to inequalities; it should be imagined that r can contain opposing entries to construct an equality. For example, periodic boundary conditions can be written as:

$$r = \begin{bmatrix} x(0) - x(T) \\ x(T) - x(0) \end{bmatrix}.$$

This is only to allow more compact canonical notation; doing this in practice with solvers is not a good idea, due to the fact that the opposing inequalities

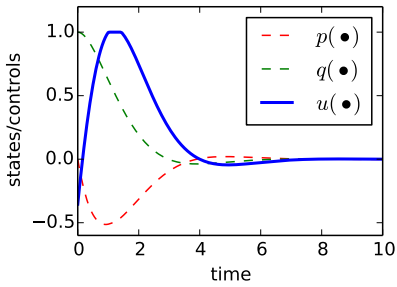
violate the so called linear independence constraint qualification (LICQ). Instead, one needs to impose them directly as equalities.

We will further omit $t \in [0, T]$ from the notation.

To make the discussion concrete, consider an optimal control problem for regulating a Van der Pol oscillator to the origin with minimal control input:

$$\underset{x(\bullet),u(\bullet)}{\text{minimise}} \quad \int_0^T \left[p^2 + q^2 + u^2\right] dt$$

$$\begin{aligned} \text{s.t.} \quad \dot{p} &= (1-q^2)p - q + u \\ \dot{q} &= p \\ -1 &\leq u(t) \leq 1 \\ p(0) &= 0, \\ q(0) &= 1, \end{aligned}$$



with $T = 10$ and the solution shown on the right.

While indirect methods to solve OCPs are more valuable to gain insights (e.g. for a linear quadratic regulator, LQR), this thesis is devoted to the direct approach for solving optimal control problems (*first discretise then optimise*), since it is more practical and flexible.

To make the presentation of the direct approach clearer, first get rid of the Lagrange term by augmenting the state space with a cost state:

$$\dot{c} = l(x(t), u(t)).$$

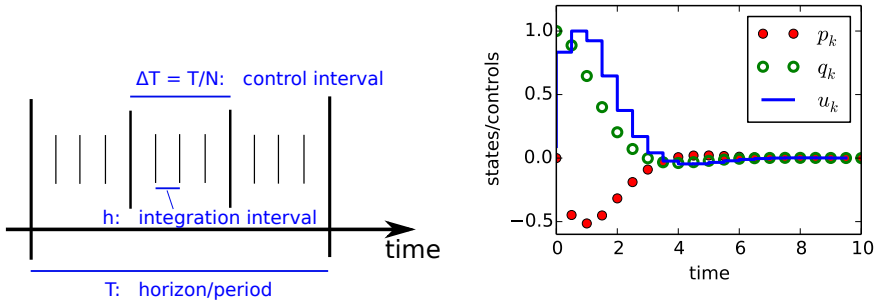
An initial constraint $c(0) = 0$ and replacing the Lagrange term by the Mayer term $c(T)$ completes the transformation.

The basic idea of a direct methods is simply to parametrise the state/control trajectory by a finite number of unknowns, such that an NLP of finite dimensions is obtained. More specifically, the following recipe is often followed:

- Introduce a time-grid $0 = t_0 \leq t_1 \leq \ldots \leq t_N = T$
- Consider N values of controls u_k that are held constant on each control interval $[t_k, t_{k+1}[$.
- Consider $N + 1$ values of state x_k for each t_k.
- Use an integrator block Φ to relate x_k to x_{k+1}, given u_k.

- Enforce path constraints only at each t_k.

The illustration of Figure 2.3a is good to have in mind when following this recipe. Figure 2.3b shows it when applied to a concrete example.



(a) Time scales in an optimal control problem, for a direct method an uniform time-grid

(b) Van der Pol example solved by a direct method, $N = 20$

Figure 2.3: Direct methods

2.3.1 Single-shooting

The most straightforward direct method is single-shooting (Sargent 1978):

$$\begin{aligned} \min_{x_0, u_\bullet} \quad & E(x_N) \\ \text{s.t.} \quad & 0 \geq h(x_k, u_k), \quad k = 0, 1, \ldots, N-1 \\ & 0 \geq r(x_N, x_0), \end{aligned}$$

with $u_\bullet$ the discrete control trajectory $u_0, u_1, \ldots, u_{N-1}$ and $x_1, x_2, \ldots x_N$ obtained by integrating:

$$\begin{aligned} x_1 &\leftarrow \Phi\left(f; x_0, u_0\right) ; \\ x_2 &\leftarrow \Phi\left(f; x_1, u_1\right) ; \\ &\vdots \\ x_N &\leftarrow \Phi\left(f; x_{N-1}, u_{N-1}\right) ; \end{aligned}$$

The objective Hessian of this formulation, and the Lagrange Hessian as a consequence, is in general dense: the Mayer term $E(x_N)$ can depend on the initial value x_0 and on the entire control trajectory.

2.3.2 Multiple-shooting

The distinction between single-shooting and multiple-shooting (Bock 1984) is subtle yet crucial. All states are now decision variables:

$$
\begin{aligned}
\min_{x_\bullet, u_\bullet} \quad & E(x_N) \\
\text{s.t.} \quad & x_{k+1} = \Phi\left(f; x_k, u_k\right), \quad k = 0, 1, \ldots, \tilde{N} \qquad \text{Coupling constraints} \\
& 0 \leq h(x_k, u_k), \quad k = 0, 1, \ldots, \tilde{N} \\
& 0 \leq r(x_N, x_0),
\end{aligned}
$$

The dimension of state space n is often much larger than that of control space m. Therefore, the NLP resulting from multiple-shooting is typically much larger than that of single-shooting.

Still, multiple-shooting can be superior in the following ways:

- The state trajectory can be fully initialised. Because of non-convexity, this can make a big difference.
- The integrator can start from a meaningful initial value at the start of each control interval, and hence explosion of unstable systems is averted.
- The problem is less non-linear because the non-linearity is spread out over the coupling constraints (Albersmeyer 2010).
- The integration can happen in parallel for each control interval.
- The method scales better with long horizons N.

The constraint Jacobian of the multiple-shooting formulation is large but sparse (block-diagonal with N blocks). The QP subproblems for an SQP method can be treated by generic sparse solvers as highlighted before, by special block-diagonal codes (qpDUNES, *qpDUNES Website* 2009 – Forces, Domahidi 2012), or by dense QP solvers (qpOASES, Ferreau 2009) after *condensing* (Bock 1984).

One interpretation of multiple-shooting is considering it as single-shooting with the intermediate expressions for $x_1, x_2, \ldots, x_N$ lifted (Albersmeyer 2010). A lifting transformation does not alter the solution, but may hugely impact the way how a scheme convergences towards that solution. Furthermore, the condensing of QP subproblems allows for an SQP method to get the convergence behaviour of the full (lifted) space while operating on the reduced space. A caveat is that the condensing transformation is unfavourable since it has run-time complexity

$O(N^2)$ (Andersson 2013), which is similar to that of constructing the Lagrange Hessian for the single-shooting problem.

2.3.3 Direct collocation

When a collocation integrator of degree d is used and its helper expressions lifted, one obtains a direct collocation method. To resolve the systems dynamics accurately, a larger number of intervals typically needs to be introduced into the NLP compared to shooting methods.

One benefit of a direct collocation method is that time integration is not resolved up to high-accuracy in each NLP step, but rather converged to, together with the optimality conditions. This can reduce the total computation time spent in integrating.

A downside is that blocks with a shape $n(d+1)$-by-$n(d+1)$ appear on the diagonal of the constraint Jacobian as opposed to n-by-n blocks for multiple-shooting which also might be less in number. However, this is less of a problem then might be suspected: while the multiple-shooting blocks $\frac{\partial \Phi}{\partial x}$ are typically dense, the blocks for collocation inherit much of the sparsity that was present in the dynamic system model.

2.3.4 Software

Software packages that provide a ready-to-use multiple-shooting solver include the closed-source MUSCOD-II (Leineweber 2003), open-source ACADO (*ACADO Toolkit* 2009–2013), which specialises in fast model predictive control (MPC), and the open-source JModelica (*JModelica.org* n.d.), which was extended with direct collocation, too (Magnusson 2012). Since the heart of this thesis lies in formulations and algorithms that go beyond the standard form, the tool CasADi (Andersson 2012) was chosen; rather than providing a ready-to-use optimal control framework, it helps in creating one from scratch using mathematical building blocks. An extensive discussion of the tool is reserved to Chapter 6.

Conclusion

This chapter introduced the terminology needed to discuss optimal control problem formulations and solution methods. There are no new results in this

chapter.

For handling stiff DAE systems, the implicit collocation integration scheme has been introduced in Section 2.1.2. This integrator, when used in a direct method to solve an OCP and its helper variables are lifted, leads to a direct collocation scheme in Section 2.3.3. The study of stability for steady-state equilibria in Section 2.1.3 led to the criteria of Equation (2.3) that resemble the algebraic Lyapunov equation $AP + PA^\top = Q$. The connection with the Lyapunov differential equations will be made clear in the next chapter. For limit-cycles, Floquet multipliers were put forward to quantify stability in Section 2.1.3.

The chapter further explained in Section 2.2.2 algorithmic differentiation as an efficient means to compute the gradients required for optimisation techniques of Section 2.2.1.

A decision was made not to use ready-to-use optimal control tools, but rather to use and extend a framework to facilitate the creation of new formulations, CasADi, which will be treated in Chapter 6.

Chapter 3

Systems with uncertainties and the Lyapunov Framework

The previous chapter introduced optimal control problems in a deterministic setting. For the remainder of this thesis, only periodic OCPs are considered. More specifically, we study the small-signal behaviour of systems on limit cycles resulting from nonlinear periodic OCP formulations.

This chapter shows how stochastic disturbances in periodic OCPs can be addressed by the *Lyapunov framework*. In the presence of white-noise Gaussian disturbances, it can guarantee that path constraints are *almost certainly* met, using an arbitrary degree of confidence, in the assumption that the linearisation around the limit cycle is adequate.

With the notion of stability intimately connected to perturbations of initial-conditions, the framework also allows for stability optimisation. In particular, open-loop stability can optimised for.

This chapter starts with the notion of state-covariance and its propagation along a nominal trajectory in Section 3.1. Using this notion, the Lyapunov framework, as developed by Houska 2007, is outlined in section 3.2. A final Section 3.3 highlights the treatment of systems with invariants, presenting both an existing and a novel technique.

3.1 Propagation of covariance

3.1.1 Some notes on covariance

Standard deviation σ and variance σ^2 are familiar concepts in one dimensional statistics. The multi-variate extension is straightforward. For a random variable vector $x \in \mathbb{R}^n$ and with $\mathrm{E}(\bullet)$ the expected-value operator, the *covariance* matrix[1] is defined as:

$$\Sigma_x \equiv \mathrm{Cov}(x,x) \triangleq \mathrm{E}\left((x-\bar{x})(x-\bar{x})^\top\right) = \mathrm{E}\left(xx^\top\right) - \bar{x}\bar{x}^\top, \tag{3.1}$$

with $\bar{x} \equiv \mathrm{E}(x)$.

A covariance matrix $\Sigma_x \in \mathbb{S}^n$ is positive-semidefinite and symmetric by definition.

3

Linearised mapping Consider the nonlinear mapping $g(x) : \mathbb{R}^n \to \mathbb{R}^m$.

The linearised covariance propagation rule through this mapping function is given by:

$$\Sigma_{g(x)} \sim \frac{\partial g}{\partial x}(\bar{x})\Sigma_x \frac{\partial g}{\partial x}^\top(\bar{x}), \qquad x \to \bar{x} \tag{3.2}$$

The proof follows from a Taylor expansion of $g(x)$, the covariance definition and the knowledge that $E(\bullet)$ is a linear operator.

Geometrical interpretation A covariance matrix Σ defines a tolerance ellipsoid in n-dimensional space. Its $s\sigma$ boundary is given by the set:

$$\left\{x_\mathrm{s} \in \mathbb{R}^n | x_\mathrm{s}^\top \Sigma^{-1} x_\mathrm{s} = \mathrm{s}^2\right\}. \tag{3.3}$$

The eigenvectors of Σ correspond to the principal axes of the $s = 1$ ellipsoid and the eigenvalues of Σ are the squares of the semi-axes.

This implicit definition does not lend itself well to plotting. For that purpose, one can transform the points x_u on a unit-hypersphere $\left\{x_\mathrm{u} \in \mathbb{R}^n | x_\mathrm{u}^\top x_\mathrm{u} = 1\right\}$, which one can easily write a parametrisation for, into points on the $s\sigma$ ellipsoid boundary with the following mapping:

[1] *Variance*, *cross-covariance* and *variance-covariance* matrix are alternative names in use for the same concept.

$$x_{\mathrm{s}} = sWx_{\mathrm{u}}, \tag{3.4}$$

where W is not unique but defined by a decomposition:

$$\Sigma = WW^{\top}. \tag{3.5}$$

The proof follows from substituting the mapping and decomposition into Equation (3.3), and using the identity $(AB)^{-1} = B^{-1}A^{-1}$ for invertible matrices. One can make W unique by requiring symmetry and positive-definiteness, as in the following example. This choice amounts to the *square root* of the matrix Σ. An alternative choice would be to use a Cholesky factorisation of Σ: requiring triangularity and a positive diagonal.

Figure 3.1 illustrates the geometrical interpretation for a numerical example where:

$$\Sigma = \begin{bmatrix} 20 & 5 \\ 5 & 10 \end{bmatrix}.$$

Its eigendecomposition is:

$$\Sigma = \begin{bmatrix} 0.924 & -0.383 \\ 0.383 & 0.924 \end{bmatrix} \begin{bmatrix} 22.1 & 0 \\ 0 & 7.93 \end{bmatrix} \begin{bmatrix} 0.924 & -0.383 \\ 0.383 & 0.924 \end{bmatrix}^{\top},$$

and

$$W = \begin{bmatrix} 0.924 & -0.383 \\ 0.383 & 0.924 \end{bmatrix} \begin{bmatrix} \sqrt{22.1} & 0 \\ 0 & \sqrt{7.93} \end{bmatrix} \begin{bmatrix} 0.924 & -0.383 \\ 0.383 & 0.924 \end{bmatrix}^{\top} = \begin{bmatrix} 4.42235 & 0.66544 \\ 0.66544 & 3.09147 \end{bmatrix}.$$

Suppose we only wish to consider a scalar subspace $\hat{x} = v^{\top}x$ with $v \in \mathbb{R}^n$. Using Rule (3.2), we find that $\Sigma_{v^{\top}x} = v^{\top}\Sigma_x v$. For projections orthogonal to the coordinate axes, $\Sigma_{\hat{x}}$ is formed by the element of Σ_x whose row and column corresponds to the axis on which we are projecting. In Figure 3.1, the result is plotted in red for projections on the first and second stochastic variable with values $\sqrt{20}$ and $\sqrt{10}$ respectively.

As the figure shows for a 1σ bound projected on the axes, we have the following proposition:

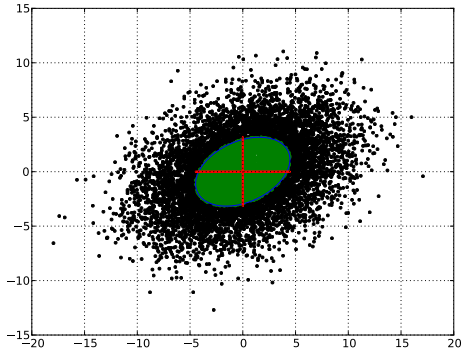


Figure 3.1: Illustration of a covariance ellipsoid. The set bounded by the 1σ-boundary is pictured solid green. The dots are random realisations of a variable with the given covariance.

Proposition 3.1.1. *The one-dimensional $s\sigma$ bound of the projection of a covariance matrix is identical to that found by projecting the full-dimensional $s\sigma$ bound.*

A proof is given in Appendix A.1.

Volume of the uncertainty ellipsoid The fraction of randomly sampled points that lies within the $s\sigma$ ellipsoid depends on the dimension n through the incomplete Gamma function P (Walck 2007), related to the χ^2-distribution:

$$f = P\left(\frac{n}{2}, \frac{s^2}{2}\right) \qquad \text{(Available in common scientific toolsets}^2\text{)} \qquad (3.6)$$

For the 1-dimensional case, we have the familiar values $68, 95, 99.7\%$ for $s = 1,2$ and 3 respectively. For increasing dimensions n, the fractions shrink fast. For $n = 10, s = 1$, for example, the fraction is a mere 0.017%. When considering the projection into a 1D subspace, we still obtain the 68% fraction of course.

[2] `Python: scipy.stats.chi2.cdf(s**2,n) Matlab: chi2cdf(s^2,n) , Mathematica: GammaRegularized[n/2,0,s^2/2]`

3.1.2 Discrete and continuous propagation

This section introduces state-covariance and derives the rules for its propagation through a nonlinear system. The approach taken is similar to that of the extended Kalman filter (EKF, Julier 1995; Julier 2004): the propagation rules are defined in terms of a first-order approximation (linear) of the system. This leads to the well-known continuous Lyapunov differential equations:

$$\dot{P}(t) = A(t)P(t) + P(t)A(t)^\top + Q(t), \tag{3.7}$$

with $P, Q \in \mathbb{S}^n$ and $A \in \mathbb{R}^n$. The interpretation of the matrices in this formula will follow in the remainder of this section. We refer to the textbook Gajić 1995 for properties of the Lyapunov matrix equations.

An alternative approach would be to consider full nonlinear propagation of a selected set of disturbed initial conditions, and defining covariance in terms of the resulting simulated perturbed states. This approach is similar to the idea of the unscented Kalman filter (UKF, Romanenko 2004).

It should be noted that for both approaches, propagation rules may be written in terms of a factorisation $\Sigma = LL^\top$ of the covariance (van der Merwe 2001).

Propagation of initial-state excursion As in Section 2.1.3, we consider an excursion $\delta x(0)$ on a nominal initial value $x^\star(0)$. Recall that a deterministic excursion evolves as:

$$\delta x(t) \sim S(t)\delta x(0), \qquad \delta x(0) \to 0 \tag{3.8}$$

with $S(t) \equiv \frac{\partial x(t)}{\partial x(0)}$ the sensitivity matrix, computable as

$$\dot{S}(t) = \frac{\partial f}{\partial x} S(t), \qquad S(0) = I_n. \tag{3.9}$$

Rather than tracing out how one particular disturbance propagates through the system dynamics linearised along a trajectory, let us inspect how statistical properties of a stochastic perturbation propagate. When considering zero mean Gaussian distributions, this question leads to the Lyapunov differential equations.

The propagation rule of Equation (3.2) applied to the excursion propagation of Equation (3.8) delivers a simple rule for propagation of covariance through

linearised system dynamics:

$$\Sigma_{\delta x(t)} = S\Sigma_{\delta x(0)}S^\top. \tag{3.10}$$

Combining the above equation, its time derivative and Equation (3.9) for the sensitivity matrix, one obtains

$$\dot{\Sigma}_{\delta x(t)} = \frac{\partial f}{\partial x}\Sigma_{\delta x(t)} + \Sigma_{\delta x(t)}\frac{\partial f}{\partial x}^\top. \tag{3.11}$$

Identifying the state-covariance $\Sigma_{\delta x(t)}$ with the Lyapunov matrix P, and the linearised system dynamics $\frac{\partial f}{\partial x}$ with A, one obtains:

$$\dot{P}(t) = A(t)P(t) + P(t)A(t)^\top. \tag{3.12}$$

There is only one term missing to obtain the Lyapunov equations.

3

Continuous injection of disturbance Consider an integrator that maps from initial state x_0 and constant stochastic disturbance w to a final state a time ΔT later:

$$x_\mathrm{f} = \Phi(\Delta T; x_0, w). \tag{3.13}$$

Linearised covariance propagation through this mapping results in:

$$\Sigma_{x_\mathrm{f}} = \frac{\partial \Phi}{\partial x}\Sigma_{x_0}\frac{\partial \Phi}{\partial x}^\top + \frac{\partial \Phi}{\partial w}\Sigma_w\frac{\partial \Phi}{\partial w}^\top. \tag{3.14}$$

The required sensitivities can be defined by matrix differential equations:

$$\frac{\partial \Phi}{\partial x} \equiv S_x \qquad \dot{S}_x = AS_x \qquad S_x(0) = I$$

$$\frac{\partial \Phi}{\partial w} \equiv S_w \qquad \dot{S}_w = AS_w + C \qquad S_w(0) = \mathbf{0},$$

with $A = \frac{\partial f}{\partial x}$ and $C = \frac{\partial f}{\partial w}$. In the limit of small ΔT, these matrices are constant and hence:

$$S_x \sim \exp(A\Delta T) = I + A\Delta T + O(\Delta T^2)$$

$$S_w \sim \exp(A\Delta T)\int_0^{\Delta T}\exp(-At)C dt = C\Delta T + O(\Delta T^2)$$

Take the limit of Equation (3.14) to obtain a time derivative:

$$\lim_{\Delta T \to 0} \frac{\Sigma_{x_f} - \Sigma_{x_0}}{\Delta T} = \frac{A\Sigma_{x_0}\Delta T + \Sigma_{x_0}A^\top \Delta T + C\Sigma_w C^\top \Delta T^2 + O(\Delta T^2)}{\Delta T}$$

The ΔT^2 factor of the term $C\Sigma_w C^\top$ is important. If the noise input Σ_w is to have any effect in a continuous setting, it needs a dimension of time:

$$\dot{P} = AP + PA^\top + C\Sigma'_w C^\top \qquad [\frac{1}{\text{time}}]$$

where the discrete and continuous variants for noise input are related as:

$$\Sigma_w = \frac{\Sigma'_w}{\Delta T}.$$

Summary For the nonlinear system $\dot{x} = f(t, x, u, w)$, regard its linearisation along a nominal trajectory $x^\star(t), u^\star(t)$ as a linear time-varying system:

$$\frac{d\delta x}{dt}(t) = A(t)\delta x(t) + C(t)w(t), \tag{3.15}$$

with $\delta x(t) = x(t) - x^\star(t)$ and:

$$A(t) = \frac{\partial f}{\partial x}(t, x^\star(t), u^\star(t), 0),$$

$$C(t) = \frac{\partial f}{\partial w}(t, x^\star(t), u^\star(t), 0).$$

The discrete-time equivalent makes use of an integrator, and assumes constant control effort and disturbance:

$$\delta x_{k+1} = \tilde{A}_k \delta x_k + \tilde{C}_k w_k, \tag{3.16}$$

with:

$$\tilde{A}_k = \frac{\partial \Phi}{\partial x}(f; t_k, x^\star_k, u^\star_k, 0),$$

$$\tilde{C}_k = \frac{\partial \Phi}{\partial w}(f; t_k, x^\star_k, u^\star_k, 0).$$

For P the state-covariance, the propagation rules are summarised in Table 3.1.

For the remainder of this thesis, it should be clear from the context whether the continuous or discrete variant is used, and tilde-notation is reserved for other concepts.

3.1.3 Numerical example

A simple numerical demonstration of covariance propagation is given in this section. The studied system has two states $x\,[\mathrm{m}]$ and $y\,[\mathrm{m/s}]$:

$$\dot{x} = y \qquad k = 0.3\,\mathrm{N/m}$$

$$m\dot{y} = -cy - kx + u + w, \qquad c = 0.1\,\mathrm{N \cdot s/m} \qquad m = 1\,\mathrm{kg}$$

and is controlled by:

$$u^\star(t) = \sin(5t). \quad [\mathrm{N}]$$

The covariance of input noise is given as:

$$\Sigma'_w = 1 \times 10^{-4}\,\mathrm{N^2 s}.$$

The nominal trajectory $x^\star(t)$ shown in Figure 3.2a, is obtained through simulation with the above control action, with $w \equiv 0$ and with initial values $(x = 1\,\mathrm{m}, y = 0\,\mathrm{m/s})$.

For a perturbed simulation, an equidistant time-grid is created running from $0\,\mathrm{s}$ to $80\,\mathrm{s}$ with spacing $\Delta T = 20 \times 10^{-3}\,\mathrm{s}$. On each interval k, the system is integrated using a constant disturbance w_k, randomly sampled from a Gaussian distribution with covariance:

$$\Sigma_w = \Sigma'_w / \Delta T = 5 \times 10^{-3}\,\mathrm{N}^2.$$

Continuous	Discrete
$\dot{P}(t) = A(t)P(t) + P(t)A(t)^\top + Q(t)$	$P_{k+1} = \tilde{A}_k P_k \tilde{A}_k^\top + \tilde{Q}_k$
$Q(t) = C(t)\Sigma'_w C(t)^\top$	$\tilde{Q}_k = \tilde{C}_k \Sigma_w \tilde{C}_k^\top$

Table 3.1: Propagation rules for state-covariance P

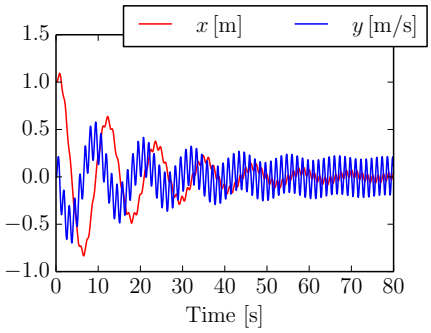
Figure 3.2b shows a few of these perturbed simulations.

The bottom row of Figure 3.2 shows for each sampling time an estimate of the standard deviation s for each state, estimated from a number N_r of perturbed simulations, along with the $\alpha = 2.5\%$ confidence interval of that estimate:

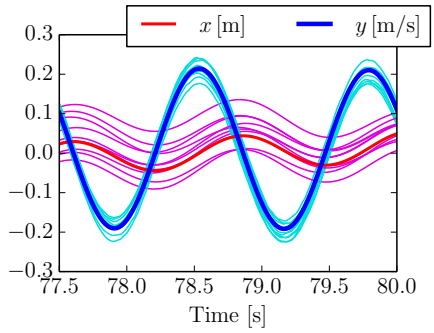
$$\sqrt{\frac{(N_\mathrm{r}-1)s^2}{\chi^2_{\alpha/2}}} \leq \sigma \leq \sqrt{\frac{(N_\mathrm{r}-1)s^2}{\chi^2_{1-\alpha/2}}}. \qquad \text{with } \chi^2 : \text{chi-squared distribution } [3]$$

The crucial point of this example is that the square-roots of the diagonal entries of the state-covariance P, as calculated by the continuous Lyapunov differential equations integrated from $P(0) = 0$, form a good prediction for the estimated standard deviations. As expected, the agreement becomes better as the number of perturbed simulations is increased.

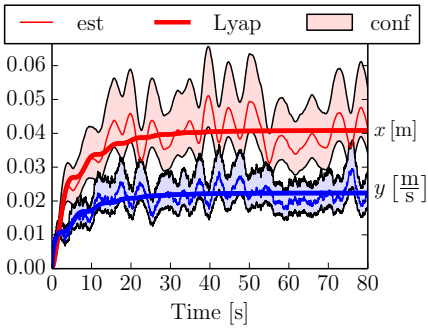
[3] Python: `scipy.stats.chi2.ppf`$(1-\alpha, N_\mathrm{r}-1)$



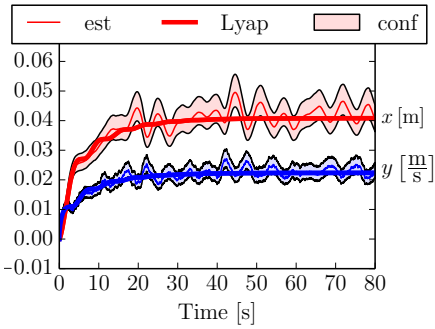
(a) Nominal system



(b) Nominal system overlaid with 10 perturbed simulations (zoom)



(c) Standard deviation for 40 simulations



(d) Standard deviation for 160 simulations

Figure 3.2: Results of a numerical covariance-propagation experiment

3.2 Outline of the Lyapunov framework

The *Lyapunov framework* we refer to is about inserting the Lyapunov differential equations into a periodic optimal control, and using the resulting $P(t)$ to robustify path constraints $0 \geq h(x(t), u(t)) \in \mathbb{R}^p$ and as part of the objective

function:

$$
\begin{array}{lrll}
\underset{x(\bullet),u(\bullet),P(\bullet)}{\text{minimise}} & J(x(\bullet),u(\bullet),P(\bullet)) & & \text{Objective} \\
\text{s.t.} \quad \dot{x}(t) & = & f(x(t),u(t)) & \text{System dyn.} \\
x(0) & = & x(T) & \text{Periodic state} \\
\dot{P}(t) & = & A(t)P(t)+P(t)A^T(t)+Q(t) & \text{Cov. propagation} \\
P(0) & = & P(T) & \text{Periodic cov.} \\
P(0) & \succeq & 0 & \text{Pos. def.} \\
0 & \geq & h_i(x(t),u(t)) + \gamma \overbrace{\sqrt{\underbrace{\frac{\partial h_i}{\partial x} P(t) \frac{\partial h_i}{\partial x}^T}_{\text{variance of } h_i(x)}}}^{\text{safety margin}}, \quad i=1,\ldots,p & \text{Path constr.}
\end{array}
$$

(tuning knob ↑ on γ)

where we remind the reader that the restriction $t \in [0,T]$ is omitted from the notation for brevity.

This canonical formulation, as proposed in Houska 2007, will be explained in this section in several parts.

An important aspect of the formulation is the periodicity constraint of $P(t)$. As explained in subsection 3.2.1, the periodic Lyapunov differential equations (PLDE) have a unique solution for a stable limit cycle of the nominal system dynamics f.

In the last constraint of the formulation, the Lyapunov matrix $P(t)$ is used to automatically construct as safety-margin on top of the nominal path constraints. Subsection 3.2.2 elaborates on this aspect of robustifying path constraints.

The Lyapunov matrix $P(t)$ can also appear in the objective of the formulation. Subsection 3.2.3 shows how this can be used for stability optimisation of limit cycles.

3.2.1 Periodic Lyapunov differential equation

Intuitive view For a periodic linear system, if the disturbance term $Q(t)$ is a cyclostationary process with the same period as the limit-cycle, the periodic Lyapunov differential equations (PLDE):

$$\begin{cases} \dot{P}(t) = A(t)P(t) + P(t)A(t)^\top + Q(t) \quad t \in [0,T] \\ P(0) = P(T), \end{cases} \tag{3.18}$$

admits a unique periodic positive definite solution $P^\star(t) = P^\star(t+T)$ if and only if the limit cycle is stable.

The interpretation is that a dynamic equilibrium is found between noise input and dissipation through the system dynamics linearised along the limit-cycle:

$$\underbrace{\dot{P}(t)}_{\text{accumulation}} = \underbrace{A(t)P(t) + P(t)A(t)^\top}_{\text{dissipation}} + \underbrace{Q(t)}_{\text{source}}. \tag{3.19}$$

The solution $P(t)$ forms a continuum of state-covariance ellipsoids along the limit cycle trajectory, tracing out a tube of uncertainty.

The more the limit-cycle is flirting with the boundary of stability or the more noise is injected, the bigger the resulting tube size will be. The trace is a simple metric for the size of the uncertainty ellipsoid $P(t)$: being invariant under orthogonal transformations, it equals the sum of the squared lengths of all semi-axes.

Rigorous view Regard the linear time-varying system with noise w:

$$\frac{dx}{dt}(t) = A(t)x(t) + C(t)w(t), \tag{3.20}$$

Consider its fundamental solution $G(t,\tau)$, a natural extension to the sensitivity matrix $S(t)$:

$$\frac{\partial G(t,\tau)}{\partial t} = A(t)G(t,\tau), \qquad G(\tau,\tau) = I. \tag{3.21}$$

An extensive discussion of existence and uniqueness for solutions of the periodic Lyapunov equations, including the indefinite cases, can be found in Bolzern 1988. Here, we focus on the positive definite case, with the Lyapunov lemma (Bolzern 1988):

Lemma 3.2.1. *The periodic Lyapunov differential equations admit a unique periodic solution $P(t) \succ 0$ if and only if the monodromy matrix $G(T,0)$ is asymptotically stable and the reachability Grammian $R(T)$ is positive definite:*

$$R(T) = \int_0^T \Gamma_T(\tau)\Gamma_T(\tau)^\top d\tau, \tag{3.22}$$

with:

$$\Gamma_t(\tau) = \begin{cases} G(t,\tau)C(\tau) & \tau \leq t \\ 0. & \text{otherwise} \end{cases} \tag{3.23}$$

The reachability criterion can always be satisfied by adding some regularisation to the DPLE: adding a small positive multiple of the unit matrix to the input term $Q(t)$ (Houska 2007).

The periodic solution can be written as:

$$P(t) = \int_{-\infty}^{\infty} \Gamma_t(\tau)\Gamma_t(\tau)^\top d\tau, \tag{3.24}$$

which can be proven to be identical to the state-covariance under white Gaussian noise disturbance, by *Itô calculus* (Chung 1990; Houska 2007). Coloured noise can be treated by adding a noise model to the nominal system.

The solution of DPLE has an alternative interpretation when considering not Gaussian noise but instead L_2-bounded disturbance sequences (Houska 2007):

$$W = \left\{ w(\bullet) | \sqrt{\int_{-\infty}^{\infty} w(\tau)^\top w(\tau) d\tau} \leq \gamma \right\}. \tag{3.25}$$

In this interpretation, the scaling factor γ gives the worst-case system state in any direction:

$$\forall v \in \mathbb{R}^n : \max_{w \in W} v^\top x(t) = \gamma \sqrt{v^\top P(t) v}. \tag{3.26}$$

For the nonlinear system $\dot{x} = f(t, x, u, w)$, we regard its linearisation along the limit-cycle $x^\star(t), u^\star(t)$ as a linear time-varying system:

$$\frac{d\delta x}{dt}(t) = A(t)\delta x(t) + C(t)w(t), \tag{3.27}$$

with:

$$A(t) = \frac{\partial f}{\partial x}(t, x^\star(t), u^\star(t), 0),$$

$$C(t) = \frac{\partial f}{\partial w}(t, x^\star(t), u^\star(t), 0).$$

The above interpretation of state-covariance and worst-case bound still stands, but is now approximate.

3.2.2 Robustification of bounds

The notion of state-covariance can be used to robustify constraints in state space, described by $h_i(x) \leq 0$ with h_i a scalar entry of the path constraints $0 \geq h$. For more compact notation, assume h is already scalar.

To settle the mind, one should imagine h as being a geometric constraint, but this is no restriction. Other possibilities, such as a constraint on energy expenditure could be imagined. Furthermore, the constraint that is robustified may originate from an NLP reformulation that casts the objective function in the form of a constraint.

The (scalar) variance of h follows immediately from the propagation rule of covariance:

$$\Sigma_{h(x)} = \frac{\partial h}{\partial x} P \frac{\partial h}{\partial x}^\top \tag{3.28}$$

It follows that, within linear approximation, the constraint $h(x) \leq 0$ can be made to be met with a probability p if it is replaced by the following equation:

$$h(x) + \gamma \sqrt{\frac{\partial h}{\partial x} P \frac{\partial h}{\partial x}^\top} \leq 0, \tag{3.29}$$

with $1 - \mathcal{C}(\gamma) = p$, using $\mathcal{C}$ here to denote the cumulative distribution function for a Gaussian distribution. This robustified path constraint may also be called a *simple chance constraint*. Figure 3.3 shows a covariance ellipsoid together with the $\sqrt{\frac{\partial h}{\partial x} P \frac{\partial h}{\partial x}^\top}$ term in blue, formed by projecting the ellipsoid in parallel with the linearised constraint onto the line perpendicular to the constraint. The test point in the figure would satisfy Equation (3.29) with $\gamma = 1$, but not with $\gamma = 2$.

3.2.3 Stability optimisation

Recall from Section 2.1.3 that the spectral radius of the monodromy matrix M provides a metric for stability. There are two drawbacks to this metric.

First, the spectral radius metric includes solely information about the system dynamics, and does not consider the system's susceptance to noise nor the quality of the noise. Both missing notions are critical in studying the asymptotic behaviour of a disturbed system. The spectral radius is still useful as a binary criterion: $\rho(M) < 1$ is a prerequisite to have any asymptotically stable behaviour at all. Yet in order to talk about relative stability, stability margin and

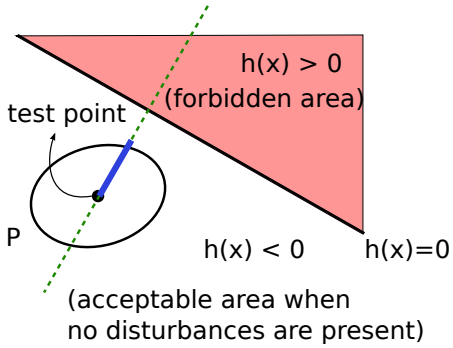


Figure 3.3: Depiction of a bound $0 \geq h(x)$ in state space, a 1-σ covariance ellipsoid and its projection towards the bound.

asymptotic behaviour, the Lyapunov matrix size $\mathrm{tr}(P)$ is a more valuable metric.

Second, the spectral radius is not differentiable, and hence difficult to use in the context of optimal control and derivative-based optimisation. On the other hand, with P being a decision variable, the Lyapunov stability metric $\mathrm{tr}(P)$ is trivially differentiable. The connection with the spectral radius is made in the form of the smoothed spectral abscissa (Diehl 2009b; Vanbiervliet 2009).

Using the Lyapunov formulation of Equation (3.17), a state and control trajectory can be optimised for open-loop stability by having the Lyapunov matrix appear in the objective functions:

$$J(x(\bullet), u(\bullet), P(\bullet)) = \text{any of} \begin{cases} \mathrm{tr}(P(0)) \\ \int_0^T \mathrm{tr}(P(t))dt \\ \int_0^T \mathrm{tr}(P(t))^2 dt. \end{cases} \tag{3.30}$$

3.3 Treatment of invariants

Conservation laws form the cornerstone of classical mechanics.

For conservative systems, energy and (angular) momentum are integrals of motions that constrain a system to evolve on a certain manifold in state space. Noether showed how these conserved quantities are connected to symmetry of space-time (Noether 1918). The Lagrange mechanics framework uses these

conserved quantities to derive the equations of motion, hence the system dynamics implicitly contains these invariants, i.e. functions of state that are constant along a trajectory:

$$C(x(t)) = 0$$

$$\frac{dC}{dt} = 0,$$

with $C : \mathbb{R}^n \to \mathbb{R}^q$.

In practice, using a conservative system in an optimal control setting destroys the integrals of motions: the action of control is a means for the system to dissipate or gain energy or (angular) momentum.

The invariants that may occur in dynamic systems usually have another source: they come from modelling in non-minimal coordinates. Violation of such an invariant would correspond to an impossible configuration; a point outside of the configuration manifold of state space. In formulating a problem with a free state variable, it is important to constrain the variable to lie on the manifold at some point of the trajectory. The dynamics will ensure it stays on it for all times.

However, there is something problematic about posing a boundary problem in addition to an invariance constraint:

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

$$x(0) = x(T)$$

$$C(x(0)) = 0.$$

Even if this problem has a theoretical solution, a numerical technique based on root-finding with an integrator Φ:

$$F(w) = \begin{bmatrix} w - \Phi(T; w) \\ C(w) \end{bmatrix} = 0$$

will struggle to find that solution for two reasons:

- Numerical integration only approximately conserves the invariants; the equalities cannot be met exactly.

- The system is overdetermined.

Concerning the first problem, there exist symplectic integrators that do conserve invariants up to machine precision regardless of integration accuracy, but such integrators are very much problem-specific, and thus conflicting with a desire to use an off-the-shelf integrator to solve every conceivable problem.

Concerning the second problem, in an optimal control setting this is equivalent to having linearly dependent in rows of the constraint Jacobian, a violation of the LICQ conditions that are essential in convergence proves for classical NLP solvers. In practice, this results in iterations with non-converging infeasibility of optimality conditions. NLP solvers have been proposed that can treat LICQ violations (Gill 2013), but again, it is much more convenient to be able to use mature off-the-shelf NLP solvers.

It is still possible to use standard integrators and NLP solvers if enough care is taken. One option is to do a change of coordinates, another to just leave out some of the redundant boundary conditions. These suggestions are add-hoc and require much insight into the system model to be successfully applied. The treatments provided in this section, on the contrary, require only knowledge of the constraint Jacobian and are generic.

Subsection 3.3.1 of this chapter shows a treatment from the literature, using projection of constraints. Subsection 3.3.2 proposes a method to propagate covariance directly in a reduced space, with an accompanying numerical example in Subsection 3.3.3.

3.3.1 Projection of constraints

Two different techniques are proposed in Sternberg 2012a to handle invariants $C : \mathbb{R}^n \to \mathbb{R}^q$, with $q < n$. This subsection merely presents these known results to prepare for the contribution in the next subsection.

The ingredients for the treatment are the following:

	Description	Definition		Size	
J	Invariant Jacobian	$J = \frac{\partial C}{\partial x}$		$\mathbb{R}^{q\times n}$	
Z	Compliant subspace Note: not unique, $ZZ^\top \neq I$	$\begin{cases} JZ = 0_{n\times(n-q)} \\ Z^\top Z = I_{n-q} \end{cases}$	Nullspace Orthon.	$\mathbb{R}^{n\times(n-q)}$	
$J^\dagger$	Pseudo-inverse Note: $JJ^\dagger = I$, $J^\dagger J \neq I$	$J^\dagger = J^\top(JJ^\top)^{-1}$		$\mathbb{R}^{n\times q}$	
V	Symmetric projector Note: $V = V^\top$	$V = I - J^\dagger J$		$\mathbb{R}^{n\times n}$	

In what follows, J is assumed to be of full row rank.

These objects have a simple geometrical interpretation. Due to invariants $C : \mathbb{R}^n \to \mathbb{R}^q$, candidate points in state space loose q degrees of freedom: they are constrained to move on a $(n-q)$ - dimensional manifold.

The Jacobian of the invariants provides a non-orthogonal basis for the forbidden directions. The compliant subspace provides an orthonormal basis for the allowed directions.

Optimal control problems Consider the following periodic optimal control problem with problematic constraints:

$$
\begin{array}{lllll}
\underset{x(\bullet),u(\bullet),T}{\text{minimise}} & J(x(\bullet),u(\bullet),T) & & & \\
\text{subject to} & \dot{x} = f(x(t),u(t)), & t\in[0,T] & \text{Dynamic constraints} \\
& 0 \geq h(x(t),u(t)), & t\in[0,T] & \text{Path constraints} \\
& 0 = x(0) - x(T), & & \\
& 0 = C(x(T)). & &
\end{array}
$$

The idea is to replace the last two constraints according to either of these methods (Sternberg 2012a):

Projection method	Null-space method
$0 = x(0) - x(T) - J^{\dagger} C(x(T))$ (3.31)	$\begin{cases} Z^{\top}(x(T))\,[x(0) - x(T)] \\ 0 = C(x(T)) \end{cases}$ (3.32)

Invariants give rise to eigenvalues 1 in the monodromy matrix M. These can be projected away as:

$$\tilde{M} = ZMZ^{\top}. \in \mathbb{R}^{(n-q)\times(n-q)} \qquad \square \tag{3.33}$$

Robust optimal control problems There is no uncertainty associated with the directions violating the invariants since such violations cannot occur. Therefore, with P the covariance matrix we have:

$$0 \equiv \Sigma_{C(x)} = JPJ^{\top} \in \mathbb{S}^{q} \qquad \square \tag{3.34}$$

Hence, the invariants cause eigenvalues 0 in the covariance matrix. The columns of $J^{\top}$ form a basis of the subspace for which P is identically zero.

The subspace of P that allows for uncertainty is given by the complement of $J^{\top}$: Z. Asymptotic stability of a limit-cycle hence requires:

$$Z^{\top} P Z \in \mathbb{S}^{(n-q)} \succ 0. \qquad \square \tag{3.35}$$

The parts of P that relate the subspaces $J^{\top}$ and Z must be zero as well, since any principal submatrix of a positive-semidefinite (PSD) matrix is PSD and hence must have a nonnegative determinant; see e.g. Horn 1986.

If invariants are present in the system dynamics f, the covariance periodicity constraint $P(0) = P(T)$ of the canonical Lyapunov Formulation (3.17) may be replaced by one of the following to avoid violating LICQ conditions (Sternberg 2012a):

Projection method	Null-space method
$P(0) - VP(T)V^{\top} = 0$ (3.36)	$\left[\begin{array}{c\|c} Z^{\top}\,[P(0) - P(T)]\,Z & Z^{\top} P(0) J^{\top} \\ \hline JP(0)Z & JP(0)J^{\top} \end{array}\right] = 0$ (3.37)

Practical formulations The projection method lends itself well to fit into the embedded discrete periodic Lyapunov form, which will be discussed in Chapter 5. One simply applies a transformation to the system dynamics and noise input matrices at the end of the period:

$$\bar{A}_{N-1} = V A_{N-1}$$

$$\bar{Q}_{N-1} = V Q_{N-1} V^\top.$$

Since $J^\dagger$ is continuous and differentiable when J is of full row-rank (Golub 1973), there are no particular difficulties associated with this formulation. Indeed, more generally, Wedin 1973 showed that the pseudo-inverse is differentiable on any constant-rank topological space.

A classical way to obtain a null-space basis Z is by using Householder reflections, as appear in a QR decomposition. Such algorithm contains if-statements and hence yield a basis that is not differentiable. Doležal 1964 showed that a basis exists that is differentiable provided that the rank of J remains constant. Techniques have been proposed to construct null-space bases that are locally smooth (Coleman 1984; Gill 1985), yet a universally smooth null-space basis cannot be constructed in general (Byrd 1986).

The quantities Z and $J^\dagger$ can be implicitly defined by adding their defining equations as part of the NLP. This approach entails the expense of extra decision variables, but comes with a guarantee of smoothness and avoids the need for embedded algorithms.

3.3.2 Covariance propagation on reduced space

Since the algorithms for embedded Lyapunov scale with $O(n^6)$ or $O(n^3)$, there is a big incentive to explore covariance propagation on a reduced covariance space, borrowing an idea from the area of extended Kalman filter research (Bonnabel 2009).

The following lemma shows the conversion between full space and reduced space.

Lemma 3.3.1. *The reduced covariance space $\hat{P}$ is obtained by transforming the full space P with the compliant subspace matrix Z. Conversely, the original full space P can be obtained by transforming with $Z^\top$:*

$$\left.\begin{aligned} P &\succeq 0 \\ \hat{P} &\triangleq Z^\top P Z \\ JPJ^\top &= 0 \\ JZ &= 0 \\ Z^\top Z &= I \end{aligned}\right\} \Rightarrow P = Z\hat{P}Z^\top$$

Proof. Multiplying the deduced statement from the left with a full-rank matrix F and $F^\top$ from the right, provides an equivalent transformed statement. Choosing $F = \begin{bmatrix} Z^\top \\ J \end{bmatrix}$ leads to four different statements that now demand proof:

$$\left[\begin{array}{l|l} Z^\top PZ = \cancelto{I}{Z^\top Z}\hat{P}\ \cancelto{I}{ZZ^\top} & Z^\top PJ^\top = \cancelto{I}{Z^\top Z}\hat{P}\ \cancelto{\mathbf{0}}{Z^\top J^\top} \\ \hline JPZ = \cancelto{\mathbf{0}}{JZ}\hat{P}\ \cancelto{I}{Z^\top Z} & \cancelto{\mathbf{0}}{JPJ^\top} = \cancelto{\mathbf{0}}{JZ}\hat{P}\ \cancelto{\mathbf{0}}{Z^\top J^\top} \end{array}\right] \tag{3.38}$$

All statements are trivially proven using the premises of the lemma.

□

Continuous time We start with the definition of the Lyapunov differential equations:

$$\dot{P} = AP + PA^\top + C\Sigma_w' C^\top$$

Multiply from the left and right to obtain:

$$Z^\top \dot{P} Z = Z^\top APZ + Z^\top PA^\top Z + Z^\top C\Sigma_w' C^\top Z.$$

Write $\hat{P} = Z^\top PZ$ and $P = Z\hat{P}Z^\top$:

$$\dot{\hat{P}} = Z^\top AZ\hat{P}\cancelto{I}{Z^\top Z} + \cancelto{I}{Z^\top Z}\hat{P}Z^\top A^\top Z + Z^\top C\Sigma_w' C^\top Z.$$

Discrete time We start with the definition of the discrete Lyapunov equations:

$$P_+ = APA^\top + C\Sigma_w C^\top$$

Multiply from the left and right to obtain:

$$Z_+^\top P_+ Z_+ = Z_+^\top A P A^\top Z_+ + Z_+^\top C \Sigma_w C^\top Z_+.$$

Write $\hat{P} = Z^\top P Z$ and $P = Z \hat{P} Z^\top$:

$$\hat{P}_+ = Z_+^\top A Z \hat{P} Z^\top A^\top Z_+ + Z_+^\top C \Sigma_w C^\top Z_+.$$

The rules for covariance propagation on the reduced space $\hat{P}$ can be summarised as follows:

Continuous	Discrete
$\dot{\hat{P}} = \hat{A}\hat{P} + \hat{P}\hat{A}^\top + \hat{C}\Sigma_w'\hat{C}^\top$	$\hat{P}_+ = \hat{A}\hat{P}\hat{A}^\top + \hat{C}\Sigma_w\hat{C}^\top$
$\hat{C} = Z^\top C$	$\hat{C} = Z_+^\top C$
$\hat{A} = Z^\top A Z$	$\hat{A} = Z_+^\top A Z$
$P = Z\hat{P}Z^\top$	
$\hat{P} = Z^\top P Z$	

The discrete propagation rule for covariance on the reduced space is not sensitive to the choice of (non-unique) null-space basis Z, and is only a means to compute the (unique) covariance in a more efficient way. Even at a discontinuity Z, the formulation is not problematic.

For the continuous case, the story is more complex since $\dot{\hat{P}}$ is an ill-defined concept at a discontinuity. Since we will focus anyway on the discrete form to benefit from the embedding of discrete periodic Lyapunov solvers, this case is not further investigated.

3.3.3 Numerical example

As an illustrating example, we compare two ways to model a pendulum, one in minimal coordinates, one without, each in a Lagrange framework (Hurtado 2011). The pendulum, illustrated in Figure 3.4, has mass $m = 1\,\mathrm{kg}$, tether length $r = 1\,\mathrm{m}$, and experiences viscous damping with coefficient $\alpha = 1\,\mathrm{N}\cdot\mathrm{s/m}$.

Using geometric insight, one can construct the following conversion formulas between the two sets of coordinates coordinates $\Theta = [\theta, \dot{\theta}]$ and $X = [x, y, \dot{x}, \dot{y}]$:

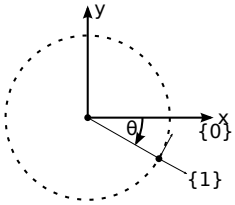


Figure 3.4: Coordinates and frames of the pendulum system.

	Forward mapping	Backward mapping
Full	$X = T_f(\Theta)$: $\begin{cases} x = r\cos\theta \\ y = -r\sin\theta \\ \dot{x} = -r\dot{\theta}\sin\theta \\ \dot{y} = -r\dot{\theta}\cos\theta \end{cases}$	$\Theta = T_b(X)$: $\begin{cases} \theta = \text{atan2}(-y, x) \\ \dot{\theta} = \frac{y\dot{x} - x\dot{y}}{r^2} \end{cases}$
Linearised	$\frac{\partial T_f}{\partial \Theta} = \begin{bmatrix} -r\sin\theta & 0 \\ -r\cos\theta & 0 \\ -r\dot{\theta}\cos\theta & -r\sin\theta \\ r\dot{\theta}\sin\theta & -r\cos\theta \end{bmatrix}$	$\frac{\partial T_b}{\partial X} = \begin{bmatrix} \frac{y}{x^2+y^2} & -\dot{y}/r^2 \\ \frac{-x}{x^2+y^2} & \dot{x}/r^2 \\ 0 & y/r^2 \\ 0 & -x/r^2 \end{bmatrix}^\top$

A Lagrange methodology leads to:

	Minimal	Non-minimal
Coord.	$q = [\theta]$	$q = [x, y]$
Kinetic	$T = m\frac{r^2\dot{\theta}^2}{2}$	$T = m\frac{\dot{x}^2+\dot{y}^2}{2}$
Potential	$V = -mrg\sin\theta$	$V = myg$
Force	$F_q = -\alpha r^2\dot{\theta}$	$F_q = -\alpha\dot{q}$
Invariant	-	$c = x^2 + y^2 - r^2$
Lagrange	$\frac{d\partial L}{dt\partial\dot{q}}^\top - \frac{\partial L}{\partial q}^\top = F_q$	$\frac{d\partial L}{dt\partial\dot{q}}^\top - \frac{\partial L}{\partial q}^\top + \frac{dc}{dq}^\top\mu = F_q$
Dynamics	$mr^2\ddot{\theta} = mrg\cos\theta + \alpha r^2\dot{\theta}$	$\begin{bmatrix} m & 0 & 2x \\ 0 & m & 2y \\ 2x & 2y & 0 \end{bmatrix}\begin{bmatrix} \ddot{q} \\ \mu \end{bmatrix} = \begin{bmatrix} -\alpha\dot{x} \\ \alpha\dot{y} - gm \\ -2\dot{x}^2 - 2\dot{y}^2 \end{bmatrix}$

Note how the non-minimal dynamics is free of trigonometric functions. This observation is relevant for rigid-body applications. Minimal parametrisations for orientation (e.g. Euler angles) can easily lead to page-long dynamic equations whereas non-minimal parametrisations (e.g. quaternions, full rotation-matrix) tend to lead to compact equations in spite of the larger state dimensions.

Regardless whether the index-1 DAE for the non-minimal dynamics is kept as DAE or symbolically inverted to become an ODE, the non-minimal system exhibits invariants viz. $C = [c, \dot{c}]^\top$. Using a high precision variable step-size integrator (tolerance of 1×10^{-15}) with equivalent initial conditions

$$\Theta_0 = [0\,\text{rad}, 0.1\,\text{rad/s}], \qquad X_0 = T_f(\Theta) = [1\,\text{m}, 0\,\text{m}, 0\,\text{m/s}, -1\,\text{m/s}], \quad (3.39)$$

we obtain the simulation results shown in Figure 3.5. Note that the invariants change a tiny amount (see Figure 3.5c) due to finite precision of the integrators.

One important property of systems with invariants in the context of this thesis, is that they give rise to special structure for state-covariance. Indeed, the invariants express a mathematical truth that exhibits no uncertainty. Writing the transformation between state-covariances for both system descriptions:

$$\Sigma_X = \frac{\partial T_f}{\partial \Theta} \Sigma_\Theta \frac{\partial T_f}{\partial \Theta}^\top, \quad (3.40)$$

and

$$\Sigma_\Theta = \frac{\partial T_b}{\partial X} \Sigma_X \frac{\partial T_b}{\partial X}^\top, \quad (3.41)$$

one can easily deduce that state-covariance of the non-minimal system has a rank of at most that of the minimal system. Figure 3.6a shows how the covariance spectra behave numerically with the following initial conditions:

$$\Sigma_\Theta^0 = \begin{bmatrix} 1\text{rad}^2 & 0 \\ 0 & 1\frac{\text{rad}^2}{\text{s}^2} \end{bmatrix},$$

$$\Sigma_X^0 = \frac{\partial T_f^0}{\partial \Theta} \Sigma_\Theta^0 \frac{\partial T_f^0}{\partial \Theta}^\top = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1\text{m}^2 & 0.1\text{m}^2/\text{s} & 0 \\ 0 & 0.1\text{m}^2/\text{s} & 0.01\text{m}^2/\text{s}^2 & 0 \\ 0 & 0 & 0 & 1\text{m}^2/\text{s}^2. \end{bmatrix}$$

Note that the nonzero eigenvalues do not match since the transformations are not normalised. That the above equality (3.41) does hold for the simulation, can be witnessed in Figure 3.6b.

Since the computational effort of Lyapunov schemes is very sensitive to the number of nominal states, it can be beneficial to propagate covariance on a reduced subspace. For the discrete Lyapunov equation, this would be:

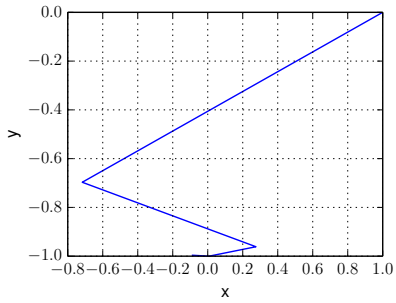
$$\Sigma_\Theta^+ = \frac{\partial T_b^+}{\partial X} \left[A \left(\frac{\partial T_f}{\partial \Theta} \Sigma_\Theta \frac{\partial T_f}{\partial \Theta}^\top \right) A^\top + Q \right] \frac{\partial T_b^+}{\partial X}^\top, \quad (3.42)$$

with A the linearised dynamics of the non-minimal system and Q a discrete source of disturbance that lies in the invariant subspace.

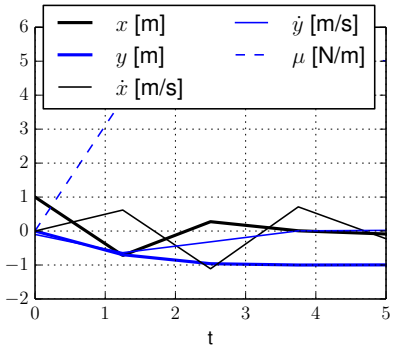
Regarding the non-minimal description as the main description of interest, and the minimal one just as an aid to make covariance propagation easier, one observes that solely the linearised mappings $\frac{\partial T_f}{\partial \Theta} = T'_f$ and $\frac{\partial T_b}{\partial X} = T'_b$ are needed, and not the exact mappings. The Θ parametrisation is in fact just one choice of T'_f and T'_b that obeys the following properties:

$$\frac{\partial C}{\partial X} T'_f = 0, \qquad \text{and} \qquad T'_b T'_f = I. \tag{3.43}$$

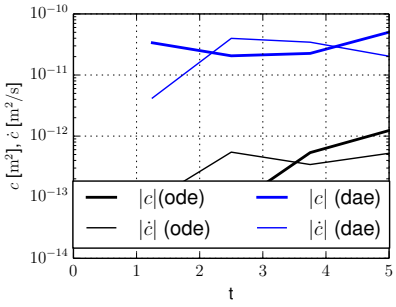
Hence the efficient Lyapunov equation for a non-minimal system requires a basis for the null-space of the invariant Jacobian (or rather an orthogonal complement to be precise) and a pseudo-inverse of that basis.



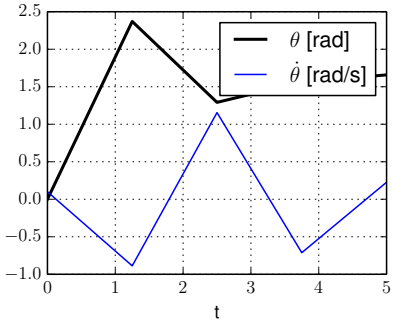
(a) Trajectory in $x - y$ space



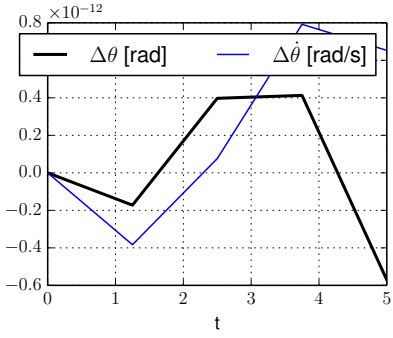
(b) Non-minimal state evolution



(c) Invariants evolution. Note that the invariants are exactly zero at $t = 0$ and hence have no representation in the logarithmic plot.

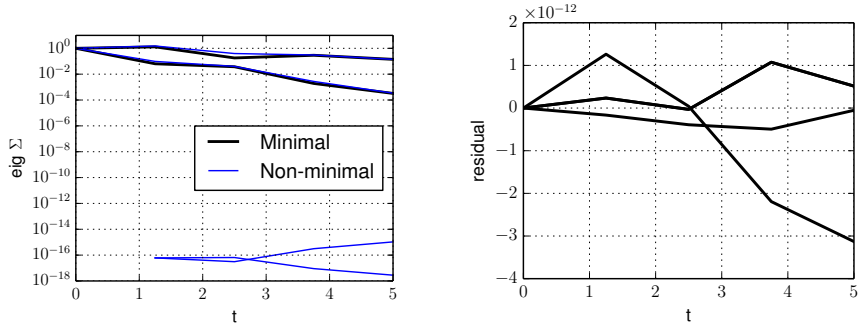


(d) Minimal state evolution



(e) Difference between $\Theta(t)$ and $T_b(X(t))$

Figure 3.5: Pendulum simulation in minimal and non-minimal coordinates



(a) Eigenvalues of covariance matrix

(b) Residual of Equation (3.41)

Figure 3.6: Simulation of pendulum covariances.

Conclusion

This chapter presented the *Lyapunov framework* (Bolzern 1988; Houska 2007, 2011b) as an optimal control problem formulation that allows one to robustify optimal control problems, and allows one to have stability as a design criterion by providing a smooth stability metric that accounts for system disturbances.

It should be noted a number of approximations are made when using the Lyapunov framework:

1. Only first-order information of the system linearised along a limit cycle is used to determine covariance.
2. Only first-order information of the path constraint function linearised around the limit cycle is used to *robustify*.
3. Nonlinear distortion may introduce bias onto the effective noise input, and is not accounted for. The net effect is that the mean of the perturbed state trajectory will be off from the computed limit cycle. We make no attempt here to quantify this offset.

Higher order approximations are considered in literature (Diehl 2006; Nagy 2007). This thesis, however, deals with getting the most out of the existing first-order formulation, in terms of efficiency. For quantification of its approximation errors, we refer to Houska 2007.

Furthermore, this chapter issued a warning for systems with invariants in the context of periodic optimal control. After laying out an existing strategy from the literature to treat such systems in the Lyapunov framework (Sternberg 2012a), it was noted that the invariants may in fact help to reduce the computational burden of DPLE solvers by propagating covariance only on a reduced subspace. This idea, which works with the null-space of the Jacobian of the invariants, is known to the field of extended Kalman filter research (Bonnabel 2009), but it has not been used in the context of robust optimal control problems.

Chapter 4

Positive definiteness preserving Lyapunov discretisation

This chapter reports on a numerical scheme with a desirable property to integrate the Lyapunov differential equations in the context of robustified OCP. In prior art, the continuous-time Lyapunov states are discretised in the same manner as the original (unrobustified) states. This straightforward technique fails to guarantee conservation of positive-semidefiniteness of the Lyapunov matrix under discretisation. In this chapter, a discretisation method coined PDPLD, is introduced that does come with such a guarantee.

This chapter starts with a motivation (Section 4.1), proceeds with Section 4.2 about formulation and the classical discretisation, and introduces a positive definiteness preserving Lyapunov discretisation scheme in Section 4.3. In a final section (Section 4.4), the scheme is demonstrated by means of a simple "toy" system. This chapter is loosely based on a CDC contribution (Gillis 2013).

4.1 Motivation

The continuous Lyapunov equation $\dot{P} = AP + PA^\top + Q$ is a celebrated equation that is excellent for proving theorems. In the covariance interpretation, it constitutes an evolution equation for covariance, devoid of the notion of

sensitivity equations. It appears as a natural and helpful tool to embed the notion of uncertainty in OCP. Without any modification to one's favourite integrators or OCP environment, one can just lump the elements of P and the right hand side in a vector, and augment the original state space with these. In fact, exploiting symmetry, one needs to add only $n(n+1)/2$ states and right hand sides into the system.

There is a fundamental problem with this approach however. Consider one uses forward Euler integrator on the Lyapunov equation:

$$P_{i+1} = P_i + \Delta t(A_i P_i + P_i A_i^\top + Q_i) \tag{4.1}$$

Manipulating this form leads to:

$$P_{i+1} = (I + A_i\Delta t)P_i(I + A_i\Delta t)^\top + \Delta t Q_i - (\Delta t)^2 A_i P_i A_i^\top. \tag{4.2}$$

The left term is positive definite if and only if P_i is positive definite, but the right term is negative definite, albeit with a factor $(\Delta t)^2$ of higher order. While there exist special integration schemes (Dieci 1994) that avoid these issues, popular schemes such as implicit Runge-Kutta suffer from it. We conclude that numerical integration error can accumulate to destroy the positive definiteness of P.

4.2 Formulation and classical discretisation of robust optimal control problems

4.2.1 Formulation

Consider the class of continuous-time T-periodic optimal control problems (OCP) that can be written as:

$$\begin{array}{lll} \underset{x(\bullet),u(\bullet),T}{\text{minimise}} & J(x(\bullet),u(\bullet),T) & \\ \text{subject to} & \dot{x}(t) = f(x(t),u(t),0), & t\in[0,T] \\ & x(0) = x(T) & \\ & 0 \geq h_i(x(t)), & t\in[0,T] \\ & 0 = \phi(x(0)) & \end{array} \tag{4.3}$$

where J constitutes the cost functional, h_i indicates the i^{th} entry in a set of scalar path constraints of length q, and $\phi : \mathbb{R}^n \to \mathbb{R}$ is a technical requirement to remove phase invariance of the solution.

The robustified equivalent of this equation is given by:

$$\begin{aligned}
&\underset{x(\bullet),u(\bullet),P(\bullet),T}{\text{minimise}} && J(x(\bullet),u(\bullet),T) \\
&\text{subject to} && \dot{x}(t) = f(x(t),u(t),0) \\
& && x(0) = x(T) \\
& && \dot{P}(\mathrm{t}) = AP(t) + P(t)A^\top + C\Sigma_w' C^\top \\
& && P(0) = P(T) \\
& && P(0) \succeq 0 \\
& && 0 \geq h_i(x(t)) + \gamma\sqrt{D_i(t)PD_i(t)^\top} \\
& && 0 = \phi(x(0)),
\end{aligned} \tag{4.4}$$

with $D_i(t) = \frac{\partial h_i(x(t))}{\partial x}$, $A = \frac{\partial f}{\partial x}(x(t),u(t),0)$ and $B = \frac{\partial f}{\partial u}(x(t),u(t),0)$. We are particularly interested in solutions for which P is positive-semidefinite and bounded over the period, as this corresponds to a stable trajectory Bolzern 1988; Houska 2010.

Problem 4.4 contains a positive-definiteness constraint $P(0) \succeq 0$. In Houska 2010 it is argued to drop this constraint such that the problem can be solved with off-the-shelf NLP solvers, and initialise the trajectory with a initial guess for trajectory $P(t)$ that is everywhere positive-definite.

The classical way to solve Problem (4.4) is by augmenting the state space (Houska 2010). One picks a set of Lyapunov states $y \in \mathbb{R}^l$ that parametrize the Lyapunov matrix $P \in \mathbb{R}^{n\times n}$ by means of a mapping $L : \mathbb{R}^l \to \mathbb{R}^{n\times n}$ and adds these states to the original state space.

Using this state-space augmentation technique, one effectively casts the robust Problem (4.4) into the form of the nominal Problem (4.3), but with enlarged dimensions.

The next section uses the symbolics of Problem (4.3), and applies as such equally well to non-robustified optimal control problems as to robustified problems formulated by state-space augmentation.

4.2.2 Direct transcription

In the family of direct methods, one proceeds by discretising the infinite-dimensional problem such that it becomes a finite-dimensional nonlinear problem. Consider a global time-grid $[t_0, t_1, \ldots, t_N]^\top$ of monotonously increasing time instants. The states sampled at these instants are denoted by $x_0, x_1, \ldots x_N$,

sampled control inputs as $u_0, u_1, \dots u_{N-1}$ and sampled disturbances as $w_0, w_1, \dots w_{N-1}$.

For any type of integrator, the action of integrating the system f can be formulated as a discrete-time system:

$$x_{k+1} = \Phi(t_{k+1} - t_k; x_k, u_k, w_k), \qquad k = 0 \dots N-1 \tag{4.5}$$

With this abstraction and assuming an evenly spaced time discretisation, Problem (4.3) can be discretised as:

$$\begin{aligned} \min_{x_\bullet, u_\bullet, \delta} \quad & J(x_\bullet, u_\bullet, \delta N) \\ \text{s.t.} \quad & x_{k+1} = \Phi(\delta; x_k, u_k, 0), \quad k = 0, \dots, N-1 \\ & x_0 = x_N \\ & 0 = \phi(x_0) \\ & 0 \geq h_i(x_k), \quad k = 0, \dots, N-1 \end{aligned} \tag{4.6}$$

with $\delta = \frac{T}{N}$.
If the function Φ is obtained by an underlying integrator, this formulation amounts to what is called a direct multiple-shooting method (Bock 1984). This numerical scheme was applied to solve the OCP with Lyapunov states in Bolzern 1988; Houska 2010.

4

4.2.3 Direct collocation method

A collocation method can be chosen as the integrator Φ to implement the multiple-shooting scheme.

Recalling the interpolating polynomial Π and helper states z from Section 2.1.2, one may write the collocation integrator as:

$$x_{k+1} = \Phi_{\text{coll}}(\delta; x_k, u_k, 0) : \begin{cases} x_{k+1} & = F(z_k) \\ 0 & = G(\delta; x_k, z_k, u_k, w_k), \end{cases} \tag{4.7}$$

with F an explicit part:

$$F(z_k) = \Pi(1; z_k), \tag{4.8}$$

and G a fully implicit part:

$$G(\delta; x, z, u, 0) = \begin{pmatrix} \Pi(0; z) - x \\ \frac{\partial \Pi}{\partial \rho}(\rho_1; z) - \delta f\left(\Pi(\rho_1; z), u, 0\right) \\ \cdots \\ \frac{\partial \Pi}{\partial \rho}(\rho_d; z) - \delta f\left(\Pi(\rho_d; z), u, 0\right) \end{pmatrix}. \tag{4.9}$$

The set of $n(d+1)$ nonlinear equations $G = 0$ implicitly defines a solution for the helper variables: $z_k^* = G^{-1}(\delta; x_k, u_k, w_k)$ that can be obtained by Newton iterations.

It is a crucial to specify on what level the implicit variables z are solved for:

- If the $G = 0$ root-finding takes place inside the integrator block Φ, we have a multiple-shooting method that happens to use collocation methods to integrate.
- If we replace the explicit function Φ in Equation (4.6) with the implicit form (F,G), we effectively pass on the burden of solving the nonlinear implicit functions at each control interval onto the global nonlinear transcribed optimal control problem, yielding a *direct collocation method*:

$$\begin{aligned} \min_{x_\bullet, u_\bullet, z_\bullet, \delta} \quad & J(x_\bullet, u_\bullet, \delta N) \\ \text{s.t.} \quad & x_{k+1} = F(z_k) \\ & 0 = G(\delta; x_k, z_k, u_k, 0) \\ & x_0 = x_N \\ & 0 = \phi(x_0) \\ & 0 \geq h_i(x_k), \quad k = 0 \ldots N-1. \end{aligned} \tag{4.10}$$

It is the direct collocation method that will be used in the remainder of this chapter.

While this method is applied successfully to non-robustified optimal control problems for a wide range of practical engineering problems (Magnusson 2012), its application to robust optimal control problems cast into non-robust form can be problematic due to numerical integration error not preserving positive definiteness.

4.3 Positive definiteness preserving Lyapunov Discretisation

It was noticed during numerical experiments in this thesis that the technique of dropping the positive definiteness constraint from Problem (4.4) and initialising with a positive definite trajectory $P(t)$ has issues. The scheme, discretised by multiple-shooting or direct collocation, may fail to converge since the integrator action is not guaranteed to preserve positive-definiteness.

We take a step back and reason about the propagation of covariance directly onto discretised Problem (4.6) in the original state space. In a linear approximation over one control interval, the state covariance matrix evolves as:

$$P_{k+1} = \frac{\partial \Phi}{\partial x} P_k \frac{\partial \Phi}{\partial x}^\top + \frac{\partial \Phi}{\partial w} \Sigma_w \frac{\partial \Phi}{\partial w}^\top. \tag{4.11}$$

This form is based on the assumption that the covariance of the disturbance $\Sigma'_w(t)$ is constant during one control interval. To make the form an exact discretisation of a continuous Lyapunov equation with time-varying $\Sigma'_w(t)$, the right term must be the reachability Grammian from Equation (3.22). While the method proposed below can be extended to provide a closer approximation to the time-varying case, we proceed by assuming that constant covariance is a faithful representation of reality in applications.

Applying the implicit function theorem to Equation (4.7) that defines Φ_{coll} in an implicit manner, leads to the following identities:

$$\frac{\partial \Phi_{\text{coll}}}{\partial x} = \underbrace{-\frac{\partial F}{\partial z}\left(\frac{\partial G}{\partial z}\right)^{-1}}_{\triangleq M} \frac{\partial G}{\partial x}, \tag{4.12}$$

$$\frac{\partial \Phi_{\text{coll}}}{\partial w} = M \frac{\partial G}{\partial w}. \tag{4.13}$$

Avoiding symbolic inversion, we propose to introduce a helper variable $M \in \mathbb{R}^{n \times n(d+1)}$ and obtain an implicit integrator scheme for both original and

Lyapunov states, which we will refer to as PDPLD:

$$\begin{cases} x_{k+1} = F(z_k) & \text{(4.14a)} \\ 0 = G(\delta; x_k, z_k, u_k, w_k) & \text{(4.14b)} \\ P_{k+1} = M_k \left(\dfrac{\partial G}{\partial x} P_k \dfrac{\partial G}{\partial x}^\top + \dfrac{\partial G}{\partial w} \Sigma_w \dfrac{\partial G}{\partial w}^\top \right) M_k^\top & \text{(4.14c)} \\ 0 = \dfrac{\partial F}{\partial z}^\top - \dfrac{\partial G}{\partial z}^\top M_k^\top & \text{(4.14d)} \end{cases}$$

The lemma that accompanies this scheme reads:

Lemma 1. *If $P_0 \succeq 0$ and $P_0, \ldots, P_N$ satisfy Equation* (4.14)*, then $P_1, \ldots, P_N \succeq 0$.*

Assuming $\Sigma'_w \succeq 0$ and starting from $P_0 \succeq 0$, the lemma follows by induction using Equation (4.14c) ■.

$$\begin{array}{lllll|l} \underset{x_\bullet, u_\bullet, z_\bullet, P_\bullet, M_\bullet, T}{\text{minimise}} & J(x_\bullet, u_\bullet, T) & & & & \# \\ \text{subject to} & x_{k+1} & = & F(z_k) & & nN \\ & 0 & = & G(T/N; x_k, z_k, u_k, 0) & & Nn(d+1) \\ & x_0 & = & x_N & & n \\ & P_{k+1} & = & M_k \left(\frac{\partial G}{\partial x} P_k \frac{\partial G}{\partial x}^\top + \frac{\partial G}{\partial w} \Sigma'_w \frac{\partial G}{\partial w}^\top \right) M_k^\top & & n^2 N \\ & P_0 & = & P_N & & n^2 \\ & P_0 & \succeq & 0 & & n \\ & \frac{\partial F}{\partial z}^\top & = & \frac{\partial G}{\partial z}^\top M_k^\top & & Nn^2(d+1) \\ & 0 & = & \phi(x_0) & & 1 \\ & 0 & \geq & h_i(x_k) + \gamma \sqrt{D_{i,k} P_k D_{i,k}^\top}, & & Nq \\ & & & i = 0 \ldots q-1, \quad k = 0 \ldots N-1. & & \end{array} \tag{4.15}$$

with $D_{i,k} = \frac{\partial h_i(x_k)}{\partial x}$.

Starting from a non-robustified optimal-control problem discretised using direct collocation as in Formulation (4.10), we propose now to robustify with PDPLD such that Formulation (4.15) is obtained with the following decision variables:

$$\begin{array}{ll} x_k \in \mathbb{R}^n,\, P_k \in \mathbb{R}^{n\times n}, & k = 0,\ldots,N \\ u_k \in \mathbb{R}^m,\, z_k \in \mathbb{R}^{n(d+1)},\, M_k \in \mathbb{R}^{n\times n(d+1)}, & k = 0,\ldots,N-1 \\ T \in \mathbb{R} & \end{array}$$

The benefit of the PDPLD integration scheme is illustrated below. Consider an autonomous nonlinear oscillator,

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -\kappa & -c \end{bmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ -\alpha y^2 \end{pmatrix}. \tag{4.16}$$

For this system, we will explore the marginally stable regime (damping $c = 0$), as well as an unstable regime (damping $c < 0$). It should be clear that this is just a simple artificial model for an unstable system. One cannot expect to construct such system using a mass suspended on a spring.

For $\alpha = 0$, the system matrix has eigenvalues $\left\{-\frac{c}{2} \pm \omega j\right\}$ with ω the damped frequency of the system:

$$\omega = \sqrt{\kappa - \frac{c^2}{4}}. \tag{4.17}$$

Writing the Lyapunov matrix as $P = \begin{bmatrix} P_{xx} & P_{yx} \\ P_{xy} & P_{yy} \end{bmatrix}$ and its vectorisation $\operatorname{vec}(P) = [P_{xx}, P_{yx}, P_{xy}, P_{yy}]^\top$, we have for $\alpha = 0$:

$$\operatorname{vec}\left(\dot{P}\right) = [A \otimes I_2 + I_2 \otimes A]\operatorname{vec}(P) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ -k & -c & 0 & 1 \\ -k & 0 & -c & 1 \\ 0 & -k & -k & -2c \end{bmatrix}\operatorname{vec}(P), \tag{4.18}$$

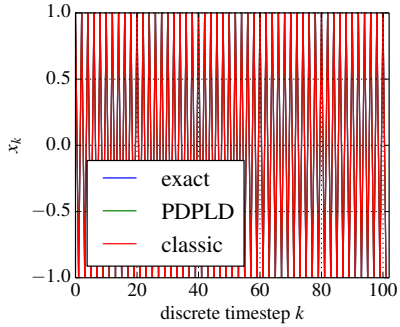
which simplifies by virtue of symmetry to:

$$\begin{bmatrix} \dot{P}_{xx} \\ \dot{P}_{xy} \\ \dot{P}_{yy} \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 \\ -k & -c & 1 \\ 0 & -2k & -2c \end{bmatrix}\begin{bmatrix} P_{xx} \\ P_{xy} \\ P_{yy}. \end{bmatrix} \tag{4.19}$$

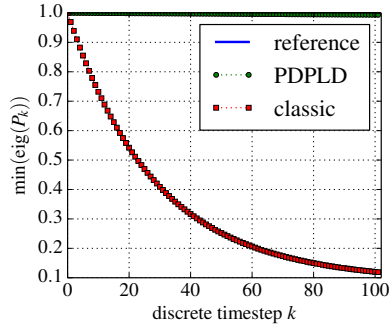
The eigenvalues of this system are $\{-c, -c \pm 2\omega j\}$, corresponding to a frequency doubling.

We compare *classical collocation* (integration scheme (4.7) applied to augmented state space (4.4)) and *PDPLD collocation* (integration scheme (4.14)) in

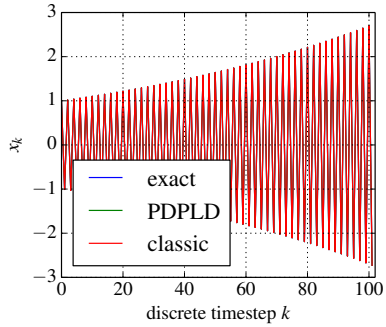
Figure 4.1 with a reference solution obtained from high accuracy (1×10^{-12}) variable step-size integration with `SUNDIALS` (Hindmarsh 2005). For the collocation schemes, a sampling rate of $\omega/2$ is chosen. It is obvious from the left-hand side plots in the figure that the nominal states are integrated quite well for any integrator using this coarse time grid. For the Lyapunov states at the right-hand side plots, a frequency doubling was anticipated by the above analysis. While the nominal states are sampled on a high and low point repeatedly, the Lyapunov plot captures the highs only. Deviations from the reference can be seen very clearly for the Lyapunov plots, a result of the frequency doubling.
Figure 4.1f demonstrates the gist of this chapter. For *classical collocation* in the non-linear case, a zero-crossing of the plotted minimal eigenvalue of P, a disruptive change in definiteness, can be observed. In contrast, in the same plot *PDPLD collocation* features a Lyapunov matrix that remains positive semi-definite.



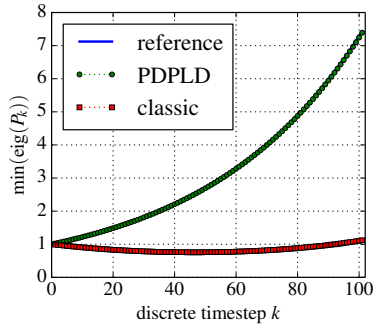
(a) Nominal states ($\alpha = 0, c = 0$)



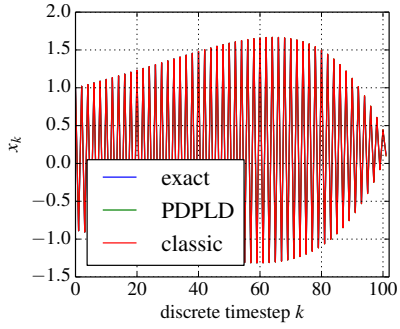
(b) Lyapunov states ($\alpha = 0, c = 0$)



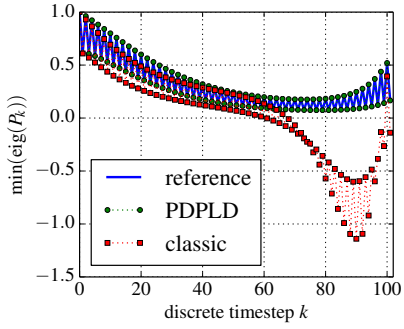
(c) Nominal states ($\alpha = 0, c = -0.02$)



(d) Lyapunov states ($\alpha = 0, c = -0.02$)



(e) Nominal states ($\alpha = -0.1, c = -0.02$)



(f) Lyapunov states ($\alpha = -0.1, c = -0.02$)

Figure 4.1: Comparison of different integrator schemes for an autonomous oscillator, sampled at $\frac{\omega}{2}$. At the left hand side figures, the three schemes lead to trajecories that are indistinguishable in the plot. The right-hand side plots show the minimum eigenvalue of the Lyapunov matrix.

4.4 Robust optimal control of a tutorial example

The remainder of this chapter shows the application of Formulation (4.15) to a simple robust optimal control problem that was specifically constructed to allow for intuitive interpretation of results.

The robust optimal control problem is solved by a homotopy in three consecutive steps. Each step reuses the primal and dual solution of the previous step where possible. An SQP method is ideally suited to execute such a homotopy.

Step 1: A nominal trajectory is identified by solving Problem (4.10) for the nominal system.

Step 2: P and M are introduced as decision variables as in Problem (4.15) but without adding robustifying margins to the path constraints. The result of this step is a trajectory of positive-definite $P_\bullet$ matrices that satisfy the periodic Lyapunov equations on the limit cycle obtained in Step 1.

Step 3: The full Problem (4.15) is tackled. By virtue of initialisation with $P_\bullet$ of the previous step, domain errors in the path constraints are averted.

4.4.1 Implementation details

The problem is formulated in the `Python` scripting environment. Gradients and Hessians are constructed by efficient sparsity-aware algorithmic differentiation provided by `CasADi` Andersson 2012. The nonlinear SQP solver `WORHP` (Büskens 2012) is used to solve the resulting nonlinear program with a tolerance of 1×10^{-10}. The linear solver used with `WORHP` is `MA57` (HSL 2011).

4.4.2 Problem statement

The system at hand consists of a point mass moving in a plane. The plane contains obstacles (super-ellipses) that the point may not intrude. The goal is to find a time-optimal periodic trajectory for the point mass in the plane, initialised with a trivial initial guess. The control inputs are coordinates of a guide-point. The point mass is connected with a spring to the guide-point, subject to friction that stabilises the system. The disturbances are forces on the point mass.

We define the following system components:

- States: $x = [p_x \quad p_y \quad v_x \quad v_y]^\top, \qquad n = 4$
- Controls: $u = [u_x \quad u_y]^\top, \qquad m = 2$
- Disturbances: $w = [w_x \quad w_y]^\top, \qquad p = 2$

The system dynamics $\dot{x} = f(x, u, w)$ is described by:

$$\dot{x} = \begin{pmatrix} v_x \\ v_y \\ -\kappa(p_x - u_x) - \beta v_x \sqrt{v_x^2 + v_y^2 + c^2} + w_x \\ -\kappa(p_y - u_y) - \beta v_y \sqrt{v_x^2 + v_y^2 + c^2} + w_y \end{pmatrix} \tag{4.20}$$

As parameters we use a spring constant $\kappa = 10$ and damping constants $c = 1$ and $\beta = 1$. Note that the system is constructed here in a dimensionless fashion. For a physical interpretation, one can choose a point mass of 1 kg and interpret all variables in matching SI units: states in m and m/s, controls in m, disturbances in N, κ in N/m, c in m/s and β in kg/m.

The path constraints are given by:

$$h_i(p_{x,k}) = \left(\frac{p_{x,k} - x_{0,i}}{a_i}\right)^{n_i} + \left(\frac{p_{y,k} - y_{0,i}}{b_i}\right)^{n_i} - 1 \tag{4.21}$$

with $n_i \in \mathbb{N}^+$. We use the following numerical values:

i	$x_{0,i}$	$y_{0,i}$	a_i	b_i	n_i
0	0	0	1	1	4
1	1	0.5	0.5	2	4

The objective function is chosen to be time-optimality, with added regularisation of control inputs:

$$J(x_\bullet, u_\bullet, T) = T + \frac{1 \times 10^{-2}}{2N} \sum_{k=0}^{N-1} ||u_k||_2^2 \tag{4.22}$$

The function to remove phase invariance from the periodic solution is chosen to be:

$$\phi(x) = p_x \tag{4.23}$$

The number of control intervals is chosen as $N = 40$.
A Legendre polynomial of degree $d = 5$ is chosen cf. Biegler 2010.

4.4.3 Homotopy results

Problem (4.10) is instantiated with the above particularities. All decision variables are initialised by zero, with two exceptions: $T = 4$, and the parts of the states x and helper variables z that correspond to coordinates in the plane are initialised by a circle of radius 3, encircling the obstacles by a wide margin. The initialisation is illustrated in Figure 4.2. Some iterates are shown in Figure 4.3 while the converged solution is shown in Figure 4.4. Active path constraints are highlighted by a line originating from the obstacle centre.

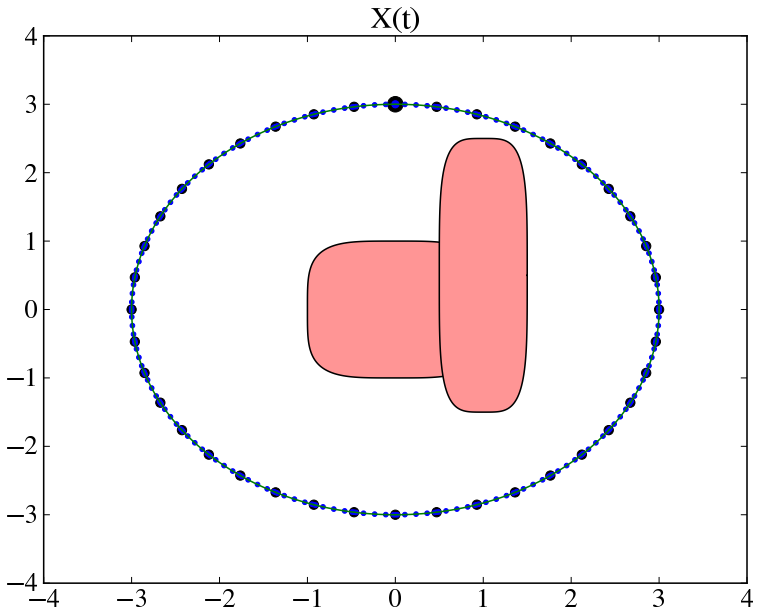
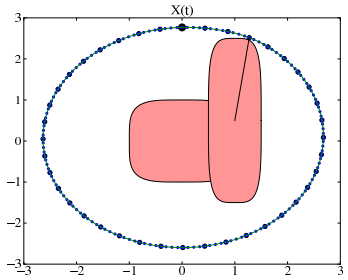
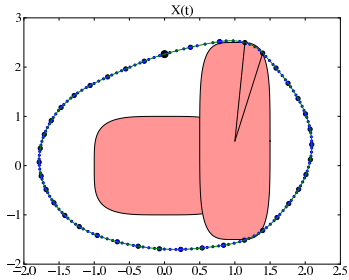


Figure 4.2: Illustration of the initial guess of Step 1. This plot shows the plane with the two obstacles. One can discern 40 black circles that correspond to $x_\bullet$ and tiny dots in between that correspond to $z_\bullet$. The polynomial interpolation of the collocation can be seen in the background. The highlighted dot at the top corresponds to $k = 0$ where the phase fix is active. The motion is clock-wise around the obstacles.

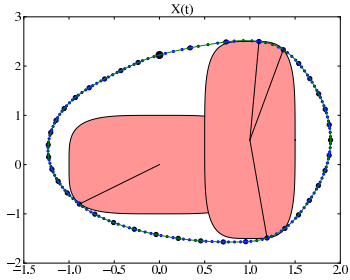
In the next step, P and M are introduced as in Problem (4.15) with non-robustified path constraints. The disturbance is chosen to have covariance



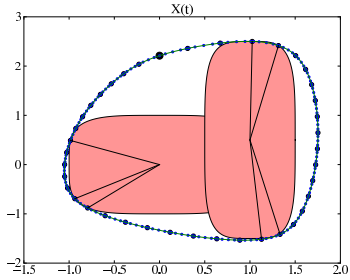
(a) Iteration 2.



(b) Iteration 14.



(c) Iteration 19.

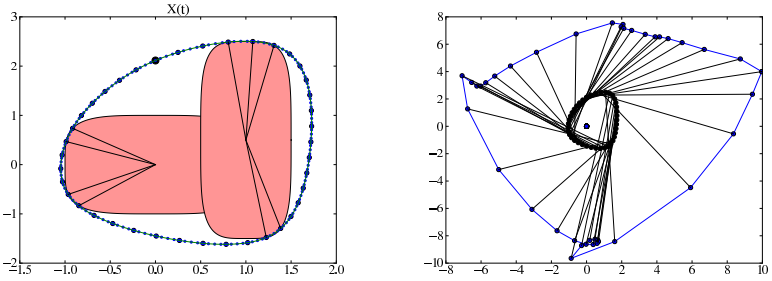


(d) Iteration 39.

Figure 4.3: Trajectories during WORHP iterations of Step 1.

$\Sigma'_w = I$. Figure 4.5 shows the solution to Step 2.
As an SQP method, WORHP is ideally suited to solve this problem using initialisation from the first step. Convergence is fast and clean:

```
Total number of variables ........................      5708
                     variables with lower bound only        1
                variables with lower and upper bound        7
                     variables with upper bound only        0
Total number of box constraints ..................       15
Total number of other constraints ................     5701
                              equality constraints     5621
      inequality constraints with lower bound only       80
 inequality constraints with lower and upper bound        0
      inequality constraints with upper bound only        0

Gradient (user)     81/5708     =   1.419%
Jacobian (user)  75641/32541308 =   0.232%
Hessian  (user)  64560/16293486 =   0.396%

NLP Method   Merit Function        QP Method     Interior-Point
NLP MaxIter  500                  QP MaxIter    80
LA solver    MA57 (Refine 10)     LA tolerance  1.00E-12
```



(a) State trajectory (b) Control inputs

Figure 4.4: Fully converged Step 1. The right figure shows control inputs. The springs that connect the guide-point with the point mass are drawn for each control interval.

```
Tolerances:
Optimality  (sKKT)   1.00E-09 (1.0E-03)   IP ComTol    2.00E-07
Feasibility          1.00E-06 (1.0E-03)   IP ResTol    5.00E-08
Complementarity      1.00E-03

Timeout               300.000  seconds

    ITER          OBJ              CON              sKKT        FLAGS   ALPHA      |DX|
[   0|   9]  1.6441388009E+00  1.6277658752E+00  6.5313617340E-01  Uin  0.000E+00  1.307E-01
[   1|  26]  1.6441388919E+00  1.2765463398E+00  8.5462742923E-02  Uin  1.000E+00  2.652E+00
[   2|   3]  1.6441399400E+00  1.1389230755E+00  1.3803220625E-02  Uin  1.000E+00  1.040E+00
[   3|   3]  1.6441989500E+00  9.3216333954E-01  2.1295424641E-02  Uin  1.000E+00  1.563E+00
[   4|   3]  1.6446167121E+00  5.5710105528E-01  3.4337133515E-02  Uin  1.000E+00  2.837E+00
[   5|   2]  1.6451938518E+00  1.0443783707E-01  9.1835032530E-02  Uin  1.000E+00  3.448E+00
[   6|   5]  1.6441480284E+00  6.8104348164E-03  6.1134851647E-03  Uin  1.000E+00  9.451E-01
[   7|   3]  1.6441404104E+00  2.4501547980E-05  6.5654692239E-05  Uaa  1.000E+00  2.243E-01
[   8|   3]  1.6441389405E+00  8.2494781006E-09  4.7324369525E-08  Ufa  1.000E+00  5.012E-03
[   9|   3]  1.6441389274E+00  1.2999839993E-07  3.8430480753E-10  Ufo  1.000E+00  6.191E-05

Final values after iteration 9:
Final objective value .............  1.6441389274E+00
Final constraint violation ........  1.2999839993E-07
Final KKT conditions ..............  3.8430480753E-10
Successful termination: Optimal Solution Found.
```

A last step adds the robustifying terms to the path constraints with $\gamma = 1$. Figure 4.6 shows the solution to Step 3.

Some remarks are in place when closely observing the solution of this application in Figure 4.7.

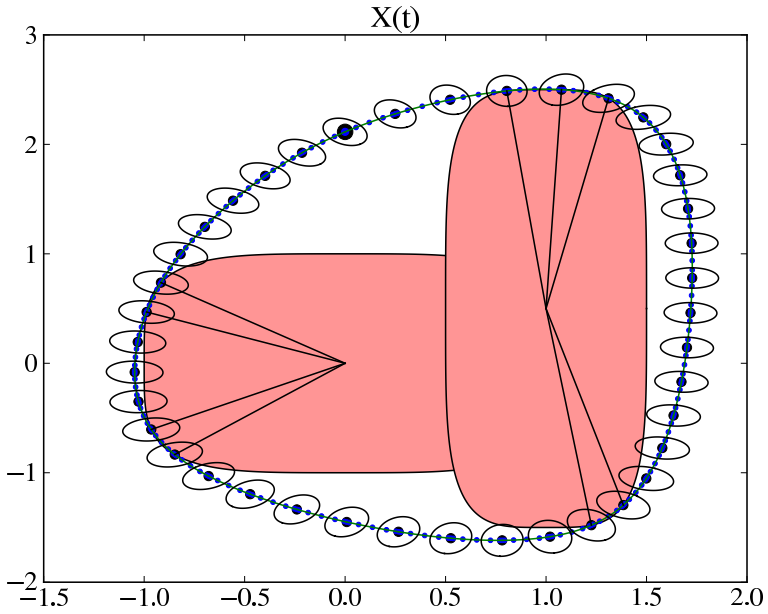


Figure 4.5: Illustration of converged Step 2: covariances propagated without robustification. The 1-sigma bounds of the uncertainty ellipsoids are highlighted.

First, it is apparent that the interpolated trajectory from Step 1 intersects with the obstacle. Indeed, during discretisation, the path constraints were only sampled at a few selected locations. There is no guarantee of non-intersection for other points on this curve.

Second, at the locations where the robustified path constraints are active, the uncertainty ellipsoids are not touching the obstacle. Indeed, a min-max formulation is required to obtain such geometrically sound result. It cannot be expected from the linearisation based robustifying formulation used here. The robustifying term of Equation (4.4) only accounts for uncertainty in the $\frac{\partial h_i}{\partial x}$ direction and is subject to linearisation errors of the nonlinear constraint function.

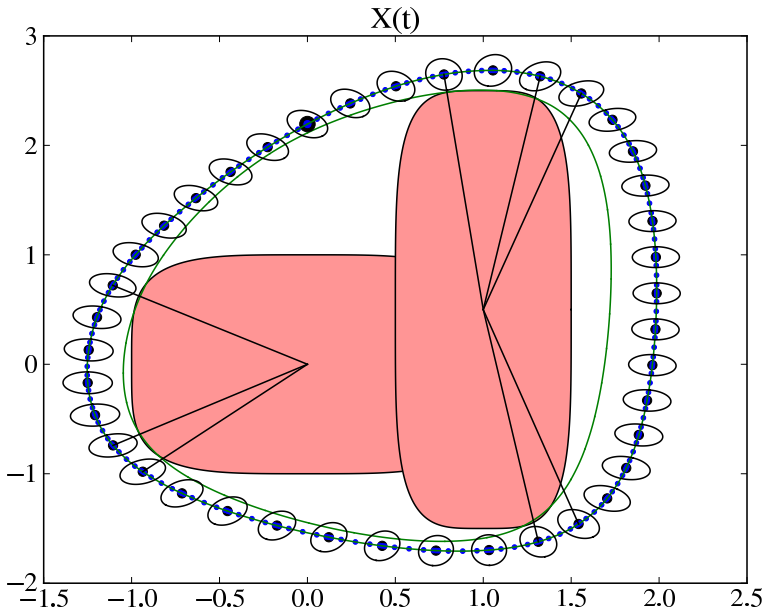


Figure 4.6: Illustration of converged Step 3. The solid green trajectory in the background meeting closely with the obstacle is the interpolated path of non-robustified Step 1.

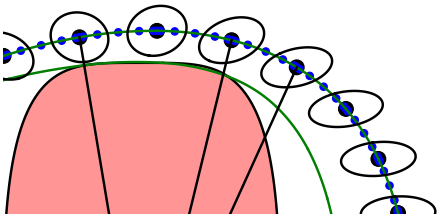


Figure 4.7: Focusing on the upper region of Figure 4.6.

Conclusion

This chapter was based on CDC contribution (Gillis 2013). Following Houska 2007, we intend to use a positive-definite initial guess of P_k to solve the discretised robust optimal control problem by a standard NLP, dropping the convex $P_0 \succ 0$

constraint.

It was noted that for such an approach to work reliably, the discretisation of the Lyapunov equation needs to guarantee preservation of positive-definiteness. The state-augmentation approach, which is the classic way in literature to solve robust optimal control problems (Houska 2010; Logist 2011b; Sternberg 2012d) does not come with such a guarantee. A simple example was constructed for which a standard integration technique, a collocation method, failed to maintain positive-definiteness even though the nominal states were accurately resolved.

While there exist special-purpose integrators that come with a guarantee (Dieci 1994), we proposed instead to add the Lyapunov equations directly in discrete form, making use of the derivative of the integrator of the nominal states, which is anyhow available in a derivative-based optimal control framework.

This scheme, positive definiteness preserving Lyapunov discretisation (PDPLD), was further worked out specifically for a direct collocation method and demonstrated using a tutorial robust optimal control example with four states and no invariants in the dynamics.

Chapter 5

Embedded Lyapunov solvers for optimal control

In Chapter 3, the usefulness of the periodic Lyapunov differential equations (PLDE), as a means to robustify optimal control problems (OCP) was established. Using a direct multiple-shooting transcription of the OCP into an NLP, one obtained:

$$\begin{aligned}
\min_{x_\bullet, u_\bullet, P_\bullet} \quad & J(x_\bullet, u_\bullet, P_\bullet) \\
\text{s.t.} \quad & 0 \geq h(x_k, u_k, P_k), \\
& x_{k+1} = \Phi\left(f; x_k, u_k\right), \\
& x_0 = x_N, \\
& P_{k+1} = \Phi\left(AP + PA^\top + Q; P_k, x_k, u_k\right), \\
& P_0 = P_N.
\end{aligned}$$

In Chapter 4, it was argued that a discrete propagation of covariance may be a better idea:

$$\begin{aligned}
\min_{x_\bullet, u_\bullet, P_\bullet} \quad & J(x_\bullet, u_\bullet, P_\bullet) \\
\text{s.t.} \quad & 0 \geq h(x_k, u_k, P_k), \\
& x_{k+1} = \Phi\left(f; x_k, u_k\right), \\
& x_0 = x_N, \\
& P_{k+1} = \frac{\partial \Phi}{\partial x} P_k \frac{\partial \Phi}{\partial x}^\top + Q_k, \\
& P_0 = P_N.
\end{aligned}$$

In both cases, the resulting NLP contains the Lyapunov/covariance matrices as decision variables, and includes constitutive equations involving these matrices (propagation and periodicity) as generic nonlinear constraints.

Such an approach is very natural from an optimal control practitioner's perspective, since conventional optimal control tools (enforcing standard formulation, AD on scalar-valued graphs/trees) readily allow the practitioner to formulate and solve this form.

The practitioner of numerical linear algebra – however – may recognise this approach as highly inefficient for large state dimensions. Indeed, the discrete periodic Lyapunov equations (DPLE) can be pulled out of the formulation and be solved by dedicated algorithms, with a more favourable computational complexity than what the generic KKT-system solver of the NLP solver can offer.

This chapter outlines how results from both fields of research can be combined to yield efficient solvers, embeddable in a differentiable expression graph, with a run-time complexity improved from $O(n^6N)$ to $O(n^3N)$ with n the state-dimension and N the horizon length.

However, the existence of such efficient embeddable DPLE solvers does not guarantee that the DPLE formulation is a superior way to solve robust OCPs. A notable result of this chapter is that there exist applications where the embedding of Lyapunov solvers is *not* beneficial. Implementation issues will be highlighted in the Chapter 6 on CasADi.

The search in numerical linear algebra literature, guided by a perspective defined in Section 5.1, is reported on in Section 5.2. After discussing algorithmic differentiation of embedded solvers in Section 5.3, the chapter proceeds with focusing on a direct solver and an iterative solver in Section 5.4, followed by a set of computational benchmarks in Section 5.5. Numerical aspects are briefly touched in Section 5.6, and Section 5.7 draws parallels with continuous-time Lyapunov approaches.

It should be noted that an efficient LQ-optimal control formulation for linear systems, albeit without the notion of embedding or algorithmic differentiation, was developed in Pittelkau 1993; Varga 1998, and hence forms the closest prior art to the subject at hand.

5.1 The optimal control perspective

The study of a robust periodic OCP, after discretisation and linearisation along a trajectory, boils down to the study of a discrete linear N-periodic parametric time-varying system driven by periodic white Gaussian noise:

$$\begin{cases} x_{k+1 \bmod N} & = A_k(p)x_k + C_k(p)w_k, \\ y_k & = H_k^\top(p)x_k. \end{cases} \tag{5.1}$$

For clarity of the presentation, we omit the parametric dependency in the following.

The stochastic property of interest for this system is the covariance Y_k of the output y_k of this system:

$$Y_k = H_k^\top P_k H_k, \tag{5.2}$$

where P_k is the solution of the DPLE:

$$P_{k+1 \bmod N} = A_k P_k A_k^\top + Q_k, \tag{5.3}$$

with $Q_k = C_k V C_k^\top$ and V the covariance of w_k.

The embeddable solver we wish to construct constitutes a mapping of a list of discrete system dynamics matrices and a list of noise inputs, to the list of covariance matrices that satisfy the DPLE:

$$P_\bullet = \texttt{DPLE\_solver}(A_\bullet, Q_\bullet). \tag{5.4}$$

5.1.1 Differentiability

Of paramount importance in this context is that the solver can be differentiated.

However, at the core of a concrete `DPLE_solver`, there might be a compiled numerical routine *without* source code that could be handled by source-code-transforming AD tools. Indeed, a routine may be closed-source or written in a legacy language unsupported by any such AD tool.

Even if the source code of such numerical routine were available to perform source-code-transforming AD on, the presence of switches (`if` statements) in the code may well render the obtained derivatives of the function discontinuous and hence not useful for derivative based optimisation purposes, as detailed in

Beck 1994, even while regions of validity can be computed as in Araya-Polo 2005.

Luckily, a DPLE solver – which can be seen as an algorithm to solve a linear system of particular system – exhibits self-similarity when applying derivative rules on the defining equation, Equation (5.3):

$$\dot{P}_{k+1 \bmod N} = A_k \dot{P}_k A_k^\top + \left[\dot{A}_k P_k A_k^\top + A_k P_k \dot{A}_k^\top + \dot{Q}_k\right], \tag{5.5}$$

where we understand $\dot{\bullet}$ here to mean the derivative with respect to some parameter of interest; recall that the object of study, Equation (5.1), is parametric.

Hence, the solver itself can be re-used to define its own derivative:

$$\dot{P}_\bullet = \texttt{DPLE\_solver}(A_\bullet, \dot{A}_\bullet P_\bullet A_\bullet^\top + A_\bullet P_\bullet \dot{A}_\bullet^\top + \dot{Q}_\bullet). \tag{5.6}$$

Note that a less ambiguous notation will be constructed later.

The property of self-similarity of derivatives makes it possible to infinitely differentiate the solver in a source-code-transforming AD tool.

The desire to exploit this property has an important consequence: Lyapunov solvers that rely on the solution being positive-definite (e.g. Cholesky/$LDL^\top$ based algorithms) are of limited use. Indeed, while the solution of the nominal problem $P_\bullet$ must always remain positive-definite for our stochastic interpretation, the solution of its derivative $\dot{P}_\bullet$ will easily become indefinite.

Since the derivative, in our OCP context, is typically evaluated a lot more than its progenitor, it makes little sense to embed a positive-definite-only solver, even when its performance should be massively better than its peers.

5.1.2 Structure

The systems of interest in this thesis are ODEs and DAEs and often exhibit sparsity in the system dynamics. In a shooting-discretisation with high-accuracy integrators (e.g. a high-order implicit collocation scheme), the resulting linear system typically exhibits dense dynamics, i.e. $A_k \in \mathbb{R}^{n \times n}$.

For this reason, we set out to implement solvers for just dense discrete Lyapunov equations. The sparse counterpart will nevertheless be treated as it arises naturally in the lifting re-formulation of the next section.

Sparse solvers may also be useful when integrator accuracy is low, as is the case in the PDE or network context, where an implicit Euler time-integration scheme is a common choice.

For systems of the implicit form $M(x,u)\dot{x} = f(x,u)$, the linearised system after high-accuracy integration again looses the structure of the continuous-time form.

There is in general no resulting implicit discrete form with clear benefits over an explicit form.

This is why the extension of the Lyapunov equations to implicit systems (a.k.a. descriptor systems), as proposed in Benner 2014a, is not considered.

There is still one type of structure discernible in Equation (5.3) that can be exploited. In case the system is not controllable in w, the exact solution of the Lyapunov equations is of low-rank. We will consider solvers that handle these low-rank discrete periodic Lyapunov equations (LRDPLE). These solvers are especially suitable if the dimension n_w of the noise input w_k is much smaller than the dimension n of state space (i.e. C_k is a tall and skinny matrix). When n_w does not scale with n, which may occur in some applications of robust OCP, such solvers may beat $O(n^3)$ complexity.

5.1.3 Initialisation and evaluation stages

Memory management is an expensive process that can easily dominate the run-time of an implementation of a numerical recipe. In an efficient optimal control tool implementation such as CasADi, care is taken not to dynamically allocate memory during evaluation of the NLP function and its derivatives.

To avoid this, CasADi adopts a two-stage approach. The structure of the NLP is calculated and memory is allocated once at initialisation time. This can drastically reduce the run-time needed for a single numerical evaluation. Even though the total run-time of one initialisation plus one evaluation may be similar or even larger than in one-shot approach, the total run-time for solving the OCP may be much reduced since the numerical evaluation step is required a huge number of times.

Not all solution techniques for DPLE in literature are suitable to exploit this two-stage approach with upfront memory-allocation. More importantly, libraries that offer a state of-the-art implementation to solve a single DPLE problem in a one-shot approach, may not be performant for our present purposes. This is why we adopt more of a first-principles approach when scouting the literature for efficient Lyapunov solvers.

5.2 An overview of Lyapunov solvers

5.2.1 Native DPLE solvers

Consider a canonical form of DPLE:

$$P_{k^+} = A_k P_k A_k^\top + Q_k, \quad k = 0, \ldots, N-1 \tag{5.7}$$

with $k^+ = (k+1) \mod N$.

The first class of solvers operates on the equivalent block diagonal form:

$$\underbrace{\begin{bmatrix} P_1 & 0 & 0 & 0 \\ & \ddots & & \\ 0 & & 0 & 0 \\ 0 & 0 & P_{N-1} & 0 \\ 0 & 0 & 0 & P_0 \end{bmatrix}}_{\mathrm{diag}(P_{k^+})} = \underbrace{\begin{bmatrix} A_0 & 0 & 0 \\ & \ddots & \\ 0 & 0 & A_{N-1} \end{bmatrix}}_{\mathrm{diag}(A_k)} \underbrace{\begin{bmatrix} P_0 & 0 & 0 \\ & \ddots & \\ 0 & 0 & P_{N-1} \end{bmatrix}}_{\mathrm{diag}(P_k)} \underbrace{\begin{bmatrix} A_0 & 0 & 0 \\ & \ddots & \\ 0 & 0 & A_{N-1} \end{bmatrix}^\top}_{\mathrm{diag}(A_k)^\top} + \underbrace{\begin{bmatrix} Q_0 & 0 & 0 \\ & \ddots & \\ 0 & 0 & Q_{N-1} \end{bmatrix}}_{\mathrm{diag}(Q_k)}. \tag{5.8}$$

A crucial observation is that, since the set of equations is linear in $P_\bullet$, it can be rewritten in the familiar form of a (sparse) linear system. To see this, write the unknowns $P_\bullet$ in the flattened (vector) form,

$$\mathrm{vec}\,(P_{k^+}) = \mathrm{vec}\left(A_k P_k A_k^\top\right) + \mathrm{vec}\,(Q_k), \quad k = 0, \ldots, N-1$$

and use the Kronecker product relation to obtain:

$$\mathcal{P}_{k^+} = (A_k \otimes A_k)\,\mathcal{P}_k + \mathcal{Q}_k, \quad k = 0, \ldots, N-1$$

with $\mathcal{P}_k = \mathrm{vec}\,(P_k)$ and $\mathcal{Q}_k = \mathrm{vec}\,(Q_k)$.

For the case $N = 1$, we have the linear system:

$$(I_{n^2} - A_0 \otimes A_0)\,\mathcal{P}_0 = \mathcal{Q}_0. \tag{5.9}$$

5

For the general case, the system reads:

$$\begin{bmatrix} -In^2 & 0 & 0 & 0 & A_{N-1}\otimes A_{N-1} \\ A_0\otimes A_0 & -I_{n^2} & 0 & 0 & 0 \\ 0 & A_1\otimes A_1 & -I_{n^2} & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & A_{N-2}\otimes A_{N-2} & -I_{n^2} \end{bmatrix} \begin{bmatrix} \mathcal{P}_0 \\ \mathcal{P}_1 \\ \vdots \\ \mathcal{P}_{N-1} \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{N-1} \\ \mathcal{Q}_0 \\ \mathcal{Q}_1 \\ \vdots \\ \mathcal{Q}_{N-2} \end{bmatrix}. \tag{5.10}$$

Keeping in mind dense system matrices, the size of this linear system is $O(n^2N)$. It can be reduced to $O(Nn(n+1)/2)$ by taking symmetry of P into account. Regardless of the symmetry reduction, the time to factorise the (dense) Kronecker blocks is of order $O(n^6)$. The run-time complexity of this `simple` solver is what we set out to beat. The key lies in never forming or factorising the equivalent linear system (5.10).

The periodic Schur decomposition (Bojanczyk 1992; Sreedhar 1994) is key to an important class of DPLE solvers. As elaborated on in Section 5.4.1, a decomposition and reordering is employed to bring the periodicity of matrix form (5.8) from the matrix-level to a block-level. A backwards-substitution technique is used to delegate the numerical work to a solver for periodic matrix equations of state-size at most 2, resulting in a run-time complexity of $O(n^3N)$. Since the `slicot` library (Van Huffel 2004) was used for obtaining the periodic Schur decomposition, our implementation of this solver is referred to as `slicot`. It should be noted that the `simple` solver can exploit sparsity in the system dynamics, while the `slicot` solver is incapable of this feat.

5.2.2 Transformations from DPLE to DLE

Another class of DPLE solvers relies on first transforming the DPLE as discrete Lyapunov equation (DLE, also known as Stein equation):

$$P = APA^\top + Q \tag{5.11}$$

Form A The first transformation technique, `lifting` (form A), results in a large but sparse DLE:

$$\begin{bmatrix} P_0 & & \\ & \ddots & \\ & & P_{N-1} \end{bmatrix} = \begin{bmatrix} A_0 & & A_{N-1} \\ & \ddots & \\ & A_{N-2} & \end{bmatrix} \begin{bmatrix} P_0 & & \\ & \ddots & \\ & & P_{N-1} \end{bmatrix} \begin{bmatrix} A_0 & & A_{N-1} \\ & \ddots & \\ & A_{N-2} & \end{bmatrix}^\top + \begin{bmatrix} Q_{N-1} & & \\ & Q_0 & \\ & & \ddots \\ & & & Q_{N-2} \end{bmatrix}. \tag{5.12}$$

Form B An alternative form is given by `lifting` (form B):

$$\begin{bmatrix} P_{N-1} & & \\ & \ddots & \\ & & P_0 \end{bmatrix} = \begin{bmatrix} & & A_{N-2} \\ & \iddots & \\ A_{N-1} & & \end{bmatrix} \begin{bmatrix} P_{N-1} & & \\ & \ddots & \\ & & P_0 \end{bmatrix} \begin{bmatrix} & & A_{N-2} \\ & \iddots & \\ A_{N-1} & & \end{bmatrix}^\top + \begin{bmatrix} Q_{N-2} & & \\ & \ddots & \\ & & Q_{N-1} \end{bmatrix}. \tag{5.13}$$

A second transformation technique, `condensing`, results in a smaller but dense DLE:

$$P = \bar{A} P \bar{A}^\top + \bar{Q}, \tag{5.14}$$

where $\bar{A}$ is the product of system matrices in reverse order:

$$\bar{A} = A_{N-1} \dots A_0, \tag{5.15}$$

and $\bar{Q}$ the total accumulation of input noise over one period:

$$\bar{Q} = Q_{N-1} + A_{N-1} Q_{N-2} A_{N-1}^\top + \dots + (A_{N-1} \dots A_1)\, Q_0\, (A_{N-1} \dots A_1)^\top. \tag{5.16}$$

The values of the DPLE covariances can easily be reconstructed by recursion from the DLE solution $P^\star$:

$$\begin{aligned} P_0 &= P^\star \\ P_1 &= A_0 P_0 A_0^\top + Q_0 \\ &\vdots \\ P_{N-1} &= A_{N-2}^\top P_{N-2} A_{N-2}^\top + Q_{N-2}. \end{aligned} \tag{5.17}$$

Both these transformation techniques can be generalised to the low-rank case.

After a condensing/lifting transformation to a DLE, and when setting the period to 1, the DLE can be solved by any of the above native DPLE solvers.

For example, consider the composition of lifting, promoting to DPLE with period 1, and solving with the `simple` solver, denoted `lifting.dple.simple`.

Since the lifted A matrix is nN-by-nN, a naive implementation of a linear solver, not exploiting sparsity of the right hand side, will lead to a system of size n^2N^2 to be solved inside the `simple` solver with complexity $O(n^6N^6)$. When this property is properly exploited, a reduced system of size n^2N can be formed that is completely equivalent to the original `simple` solver's system of Equation (5.10) and hence exhibit run-time complexity $O(n^6N)$.

On the other hand, after lifting, the use of a dense DLE solver (e.g. `lifting.dple.slicot`) would manage a complexity $O(n^3N^3)$, while the use of sparse DLE solver (e.g. `lifting.smith`) can reach $O(n^3N)$, the same as the dedicated `slicot` solver.

If neither dedicated DPLE solvers nor sparse DLE solvers are available, then condensing is the only efficient transformation option. If either is available, however, then condensing run-time performance is of the same order and other criteria such as memory use and numerical accuracy come into play. Note that the condensing approach costs not $O(n^3)$, but $O(n^3N)$ because of the accumulation phase of the condensing transformation.

5.2.3 DLE solvers

The direct techniques to solve a DLE are based on the Schur decomposition, of which the periodic extension was already discussed in Section 5.2.1. Common to all is that the resulting block-triangularity of the equivalent linear system is exploited, sparsity of $A \in \mathbb{R}^{n\times n}$ is not exploited, and complexity is $O(n^3)$. The classical Bartels-Stewart algorithm, defined for continuous Lyapunov algebraic equations (CLE), was applied to DLE by Barraud 1977. In the same work it is noted that balancing of A (Parlett 1969) may improve accuracy. The Hessenberg-Schur method (Golub 1979) is an adaption over the classical algorithm in which complex arithmetic is avoided. The work of Hammarling 1982 is another notable result in the field, but cannot be used in this context because it is based on a Cholesky factorisation, requiring positive-definiteness.

Iterative solvers are the counterpart of the direct methods and have the advantage that they facilitate sparsity and low-rank exploitation, potentially circumventing the $O(n^3)$ restriction. The most basic algorithm is the *Smith iteration*, which relies on the fact that the Lyapunov operator $f(X) = AXA^\top + Q$ will exhibit a fixed point if A is stable. This type together with low-rank extensions will be elaborated on in a separate section, Section 5.4.2.

Iterative schemes from the CLE literature, such as the sign method by Roberts 1980, can be applied after performing a Cayley transform (Benner 2002), as follows:

Theorem 5.2.1. *Under conditions of invertibility, P is a solution to the DLE:*

$$P = APA^\top + Q, \tag{5.18}$$

if and only if P is a solution of the CLE:

$$0 = \hat{A}P + P\hat{A}^\top + \hat{Q}, \tag{5.19}$$

with:

$$\hat{A} = (A - I)^{-1}(A + I) \tag{5.20}$$

$$\hat{Q} = 2(A - I)^{-1}Q(A - I)^{-T}. \tag{5.21}$$

$$\tag{5.22}$$

The proof follows from left- and right-multiplying the CLE with $(A - I)$ and $(A - I)^\top$ respectively.

5.2.4 Overview schematic

In the schematic below, the four corner blocks denote the four classes of discrete Lyapunov solver interfaces, while the smaller blocks give the various possibilities to transform from one class to another class. The contents of each class block states the derivative rules that will be obtained in the Section 5.3. The headers of each box enlist the various solvers that have been implemented, by means of the labels used in CasADi. It should be noted that multiple variants of numerical algorithms can hide under one such label. In particular, the `smith` solver of the upper-right class block offers an option in CasADi to choose between the regular (Algorithm 3) and squared Smith iteration (Algorithm 4) of Section 5.4.2.

slicot, simple
$P_k \leftarrow$ DPLE_solver(A_k, Q_k)
$\dot{P}_k \leftarrow$ DPLE_solver$(A_k, \dot{A}_k P_k A_k^\top + A_k P_k \dot{A}_k^\top + \dot{Q}_k)$
$\bar{Q}_{N-1-k} \leftarrow$ DPLE_solver$(A_{N-1-k}^\top, \bar{P}_{N-1-k})$
$\bar{A}_k \leftarrow 2 P_k A_k^\top \bar{Q}_k$
lifting
lifting
dple
$N = 1$
condensing
condensing
smith,simple
$P \leftarrow$ DLE_solver(A, Q)
$\dot{P} \leftarrow$ DLE_solver$(A, \dot{A} P A^\top + A P \dot{A}^\top + \dot{Q})$
$\bar{Q} \leftarrow$ DLE_solver$(A^\top, \bar{P})$
$\bar{A} \leftarrow 2 P A^\top \bar{Q}$
dle
$Q = CVC^\top$
$Y = H^\top P H$
lrdle
$C = H = I$
lifting
lifting
lrdple
$N = 1$
condensing
condensing
smith,simple
$Y \leftarrow$ LRDLE_solver(A, V, C, H)
dple
$Q = CVC^\top$
$Y = H^\top P H$
lrdle
$C = H = I$
simple
$Y_k \leftarrow$ LRDPLE_solver(A_k, V_k, C_k, H_k)

5.3 Embedding Lyapunov solvers in an AD framework

As was shown in Section 5.2, the discrete periodic Lyapunov equations can be formulated as a linear system. Any DPLE solver being just a clever way to solve this linear system, we will first focus on algorithmic differentiation of linear systems in general.

5.3.1 Linear solvers

Consider an invertible matrix $A \in \mathbb{R}^{n\times n}$ and column vectors $x, b \in \mathbb{R}^n$ such that

$$Ax = b.$$

Define a function `linear_solver` that captures the linear solve operation of this system:

$$x = \texttt{linear\_solver}(A, b). \tag{5.23}$$

Taking the differential of the implicit form leads to:

$$A\dot{x} = \dot{b} - \dot{A}x, \tag{5.24}$$

which is self-similar to the original since the forward mode AD can be written as:

$$\dot{x} = \texttt{linear\_solver}(A, \dot{b} - \dot{A}x). \tag{5.25}$$

Conversely, the adjoint mode AD of the solver is given by:

$$\begin{aligned} \bar{b}^\top &= \texttt{linear\_solver}(A^\top, \bar{x}^\top). \\ \bar{A} &= -x\bar{b}. \end{aligned} \tag{5.26}$$

Proof. Left-multiply the differential form of Equation (5.24) by the inverse:

$$\dot{x} = A^{-1}\dot{b} - A^{-1}\dot{A}x.$$

Transform using a flattened form $\mathcal{A} = \mathrm{vec}\,(A)$:

$$\dot{x} = A^{-1}\dot{b} - (A^{-1} \otimes x^\top)\dot{\mathcal{A}}.$$

It is now trivial to identify $\frac{\partial f}{\partial b}$ and $\frac{\partial f}{\partial \mathcal{A}}$ in the definition of forward AD applied on the vector-valued linear solve operation $x = f(\mathcal{A}, b)$:

$$\dot{x} = \frac{\partial f}{\partial b}\dot{b} + \frac{\partial f}{\partial \mathcal{A}}\dot{\mathcal{A}}.$$

Now, the definition of adjoint AD,

$$\bar{b} = \bar{x}\frac{\partial f}{\partial b} \qquad \bar{\mathcal{A}} = \bar{x}\frac{\partial f}{\partial \mathcal{A}},$$

can be used to produce:

$$\bar{b} = \bar{x}A^{-1} \qquad \bar{\mathcal{A}} = -\bar{x}(A^{-1} \otimes x^\top).$$

The remainder of the proof just involves rewriting this result:

$$\begin{aligned}
\bar{b}^\top = A^{-T}\bar{x}^\top \qquad & \bar{\mathcal{A}}^\top = -(A^{-T} \otimes x)\bar{x}^\top & \Leftrightarrow \\
A^T\bar{b}^\top = \bar{x}^\top \qquad & \bar{\mathcal{A}}^\top = -\text{vec}\left(A^{-T}\bar{x}^\top x\right) & \Leftrightarrow \\
& \bar{A}^\top = -A^{-T}\bar{x}^\top x^\top & \Leftrightarrow \\
& \bar{A}^\top = -\bar{b}^\top x^\top & \Leftrightarrow \\
& \bar{A} = -x\bar{b}.
\end{aligned}$$

□

It is useful to provide a more intuitive interpretation of the adjoint rules presented in Equation (5.26). Start with writing the forward rule as an algorithm:
$\dot{c} \leftarrow \dot{b} - \dot{A}x$;
$\dot{x} \leftarrow$ `linear_solver`$(A, \dot{c})$. ;

Applying Corollary 2.2.1 to this leads to:
$\bar{c}^\top \leftarrow$ `linear_solver`$(A^\top, \bar{x}^\top)$;
$\bar{b} \leftarrow \bar{c}$;
$\bar{A} \leftarrow -x\bar{c}$, ;

which is an alternative way of writing the adjoint rules. The interpretation of this notation is the following: first, the variation of system output ($\bar{x}$) is back-propagated onto a variation of the collective right-hand side $\bar{c}$. Next, the variation of the collective right-hand side $\bar{c}$ is propagated onto its constituents, as found by the forward rule.

A key observation in this section is that the particular numerical factorisation used in solving the linear system (such as LU-decomposition) plays no role in deriving the AD rules. Indeed, if A was factorised as LU with L, U respectively lower and upper triangular matrices, the forward derivative is simply found by two triangular solves as $U\dot{x} = L^{-1}(\dot{b} - \dot{A}x)$. Crucially, we do not need knowledge of $\frac{\partial L}{\partial A}$. L and U need in fact not be differentiable.

Any Lyapunov solver, being a particular linear solver, shows the same advantage.

5.3.2 Lyapunov solvers

Let us focus first on the DLE solver:

$$P = \texttt{DLE\_solver}(A, Q).$$

The forward AD rule is easily derived as in Section 5.1.1:

$$\dot{P} = \texttt{DLE\_solver}(A, \dot{A}PA^\top + AP\dot{A}^\top + \dot{Q}). \tag{5.27}$$

To obtain the adjoint rule, recall that the DLE can be written as a linear system:

$$\operatorname{vec}(Q) = \underbrace{[I - A \otimes A]}_{\triangleq \tilde{A}} \operatorname{vec}(P). \tag{5.28}$$

Applying the adjoint AD rule for this linear system gives:

$$\tilde{A}^\top \operatorname{vec}\left(\bar{Q}\right)^\top = \operatorname{vec}\left(\bar{P}\right)^\top, \tag{5.29}$$

which is equivalent to:

$$\operatorname{vec}\left(\bar{P}\right)^\top = \left[I - A^\top \otimes A^\top\right] \operatorname{vec}\left(\bar{Q}\right)^\top \tag{5.30}$$

and hence similar to the original Equation 5.28. Noting that P and Q are symmetric, the first part of the adjoint AD rule can be written as:

$$\bar{Q} = \texttt{DLE\_solver}(A^\top, \bar{P}). \tag{5.31}$$

The second part of adjoint AD rule is easily obtained using Corollary 2.2.1 as in the previous subsection. The collective right-hand side reads:
$\dot{V} \leftarrow \dot{A}PA^\top + AP\dot{A}^\top + \dot{Q}.$;

Applying adjoint AD on this rule results in:
$\bar{Q} \leftarrow \bar{V}$;
$\bar{A} \leftarrow PA^\top\bar{V} + (\bar{V}AP)^\top = 2PA^\top\bar{Q}.$;

For the DPLE solver, let us first introduce a less ambiguous notation. An under-bracket is used to construct a list of length N from an expression involving k, with k running from 0 to up to $N-1$:

$$\underbrace{P_k} = \texttt{DPLE\_solver}(\underbrace{A_k}, \underbrace{Q_k}). \tag{5.32}$$

The forward rule is again obtained as in Section 5.1.1:

$$\underbrace{\dot{P}_k} = \texttt{DPLE\_solver}(\underbrace{A_k}, \underbrace{\dot{A}_k P_k A_k^\top + A_k P_k \dot{A}_k^\top + \dot{Q}_k}). \tag{5.33}$$

The backward rule features a transpose as for the DLE case, but also a time-reversal:

$\underbrace{\bar{Q}_{N-1-k}} \leftarrow \texttt{DPLE\_solver}(\underbrace{A_{N-1-k}^\top}, \underbrace{\bar{P}_{N-1-k}})$;
$\bar{A}_k \leftarrow 2P_k A_k^\top \bar{Q}_k.$;

The proof is based on the equivalence of two forms of `lifting` transformations of Section 5.2.2.

Proof. First, write a DLE equivalent using `lifting` form A:

$$\begin{bmatrix} P_0 & & \\ & \ddots & \\ & & P_{N-1} \end{bmatrix} = \texttt{DLE\_solver}\left(\begin{bmatrix} A_0 & & A_{N-1} \\ & \ddots & \\ & A_{N-2} & \end{bmatrix}, \begin{bmatrix} Q_{N-1} & & \\ & Q_0 & \\ & & \ddots \\ & & Q_{N-2} \end{bmatrix}\right).$$

Next, apply the adjoint rule for the DLE:

$$\begin{bmatrix} \bar{Q}_{N-1} & & \\ & \bar{Q}_0 & \\ & & \ddots \\ & & \bar{Q}_{N-2} \end{bmatrix} = \texttt{DLE\_solver}\left(\begin{bmatrix} & & A_0^\top \\ & \iddots & \\ A_{N-1}^\top & & \end{bmatrix}, \begin{bmatrix} \bar{P}_0 & & \\ & \ddots & \\ & & \bar{P}_{N-1} \end{bmatrix}\right).$$

The last step is a time transformation $k \to (N-2-k) \bmod N$:

$$\begin{bmatrix} \bar{Q}_0 & & \\ & \ddots & \\ & & \bar{Q}_{N-1} \end{bmatrix} = \texttt{DLE\_solver}\left(\begin{bmatrix} & & A_{N-2}^\top \\ & \iddots & \\ A_{N-1}^\top & & \end{bmatrix}, \begin{bmatrix} \bar{P}_{N-2} & & \\ & \ddots & \\ & \bar{P}_0 & \\ & & \bar{P}_{N-1} \end{bmatrix} \right),$$

which can readily be written as the proposed adjoint rule using `lifting` form B. □

For the low-rank solver variants, keeping in mind Equation (5.2), we adopt the a signature with two extra inputs:

$$Y = \texttt{LRDLE\_solver}(A, V, C, H), \tag{5.34}$$

with:

$$P = APA^\top + CVC^\top \tag{5.35}$$

$$Y = H^\top PH. \tag{5.36}$$

The advantage of including H in the signature is that the low-rank but dense P does not need to be constructed explicitly, saving memory and computation time.

A strong caveat concerning the low-rank solver is that the derivative of an `LRPLE` is not sufficiently self-similar to qualify as embeddable. To see this, write the derivative of its definition as:

$$\dot{P} = A\dot{P}A^\top + \left[\dot{A}PA^\top + AP\dot{A}^\top + \dot{C}VC^\top + C\dot{V}C^\top + CV\dot{C}^\top\right], \tag{5.37}$$

and regroup to transform it in low-rank form:

$$\dot{P} = A\dot{P}A^\top + \tilde{C} \begin{bmatrix} P/2 & & & & \\ & -P/2 & & & \\ & & V/2 & & \\ & & & -V/2 & \\ & & & & \dot{V} \end{bmatrix} \tilde{C}^\top, \tag{5.38}$$

with

$$\tilde{C} = \begin{bmatrix} \dot{A} + A & \dot{A} - A & \dot{C} + C & \dot{C} - C & C \end{bmatrix}. \tag{5.39}$$

A first problem with this form is that it is not numerically stable due to the subtractions. A second problem is that the matrix $\tilde{C}$ is a lot thicker than C (wide instead of tall), nibbling away at the usefulness of a low-rank approach.

One may remark that the P in Equation (5.38) might itself be low-rank and hence could be replaced a by low-rank factorisation $DWD^\top$, yielding a forward rule that may still have a tall-and-skinny $\tilde{C}$. While adding D and W as outputs to the `LRDLE` signature seems to settle this issue, this is not the case. The low-rank factors are internal to the numerical process and may not be differentiable, excluding the possibility to construct an infinitely differentiable graph with an embedded `LRDLE` solver. The proposed `LRDLE` solver is hence only once differentiable in general.

Rather than working with these extra outputs, and still suffering from numerical stability issues of Equation (5.39), we chose for our solver a simple Smith-type – essentially switch-free – algorithm, on which source-code-transforming AD can be applied.

In summary, DLE and DPLE can easily by embedded in matrix-valued expression graphs, whereas the low-rank versions cannot.

5.4 Selected numerical algorithms for Lyapunov solvers

This section highlights two numerical algorithms in detail. There is no contribution here other than the way of presenting. The first is the periodic Schur solver, a direct DPLE solution method. The second is the family of Smith-type iterative algorithms, suitable for solving DLEs.

5.4.1 Periodic Schur solver

In this subsection, an implementation of a periodic Schur based solver is discussed in depth. The ideas in this section are much indebted to Varga 1997.

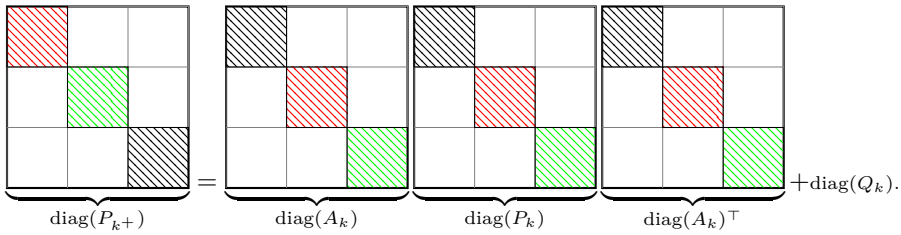
Recall the canonical form of the PDLE problem from Equation (5.7):

$$P_{k^+} = A_k P_k A_k^\top + Q_k, \quad k = 0, \ldots, N-1. \tag{5.40}$$

In our argument, we will make use of the following equivalent block diagonal form:

$$\begin{bmatrix} P_1 & & \\ & \ddots & \\ & & P_{N-1} \\ & & & P_0 \end{bmatrix} = \begin{bmatrix} A_0 & & \\ & \ddots & \\ & & A_{N-1} \end{bmatrix} \begin{bmatrix} P_0 & & \\ & \ddots & \\ & & P_{N-1} \end{bmatrix} \begin{bmatrix} A_0 & & \\ & \ddots & \\ & & A_{N-1} \end{bmatrix}^\top + \begin{bmatrix} Q_0 & & \\ & \ddots & \\ & & Q_{N-1} \end{bmatrix}.$$

A visual representation of the above form is illustrated below for $N = 3$. Each color corresponds to blocks at different discrete times k:



Next, consider the periodic Schur form:

Definition The real periodic Schur form of an N-periodic matrix sequence $H_0, H_1, \ldots, H_{N-1} \in \mathbb{R}^{n\times n}$ is defined by the relation $Z_k^\top H_k Z_{k^+} = T_k$ for $k = 0, \ldots, N-1$ with $k^+ = (k+1) \bmod N$. Here, $Z_0, \ldots, Z_{N-1}$ are n-by-n orthonormal matrices, T_0 is upper Hessenberg (i.e. upper triangular, but with extra elements on one band below the diagonal) and $T_1, \ldots, T_{N-1}$ are upper triangular.

Numerical routines for performing a periodic Schur decomposition with $O(n^3 N)$ complexity are readily available in control libraries, such as in the open-source SLICOT project (Van Huffel 2004) as algorithms `mb03{vd,vy,wdd}`.

5

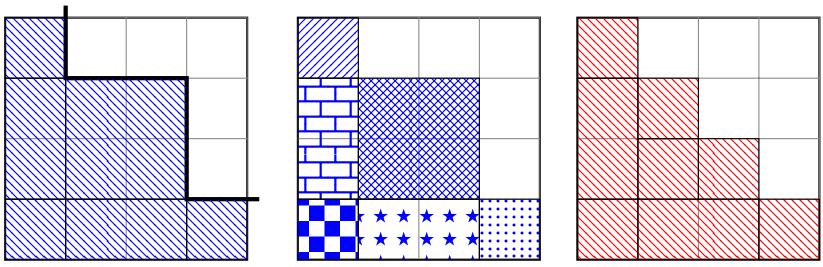
Using a Schur decomposition of $H_\bullet = A_\bullet^\top$, one can transform Equation (5.40) by left- and right-multiplying with $Z_{k^+}^\top$ and Z_{k^+} respectively:

$$Z_{k^+}^\top P_{k^+} Z_{k^+} = Z_{k^+}^\top A_k \overbrace{Z_k Z_k^\top}^{I_n} P_k \overbrace{Z_k Z_k^\top}^{I_n} A_k^\top Z_{k^+} + Z_{k^+}^\top Q_k Z_{k^+}, \tag{5.41}$$

which can be written as:

$$\tilde{P}_{k+} = \tilde{A}_k \tilde{P}_k \tilde{A}_k^\top + \tilde{Q}_k, \quad k = 0, \ldots, N-1. \tag{5.42}$$

with $\tilde{A}_0$ lower Hessenberg and $\tilde{A}_1, \ldots, \tilde{A}_{N-1}$ lower triangular. As visualised in Figure 5.1, the line separating the zeros from the nonzeros of $\tilde{A}_0$ takes the form of a staircase running on the diagonal. This staircase has p steps, which are of size 1 or 2. The staircase is the key to form a partitioning with p squares (1-by-1 or 2-by-2) on the diagonal and corresponding rectangles on the off-diagonals, as illustrated in Figure 5.1b.



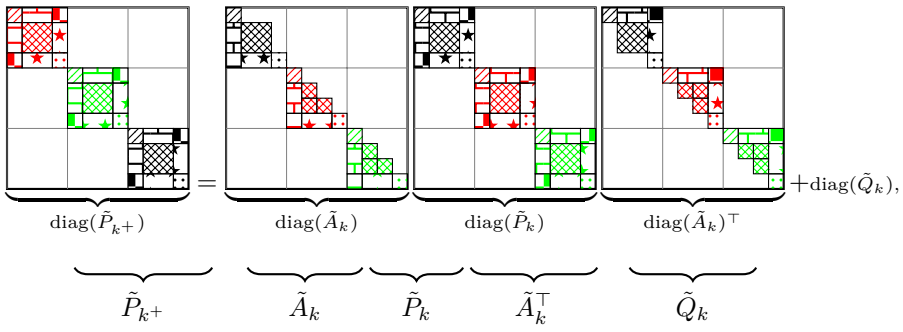
(a) Structure of $\tilde{A}_0$. The staircase stepsizes shown are $(1, 2, 1)$, $p = 3$.

(b) Staircase partitioning of $\tilde{A}_0$

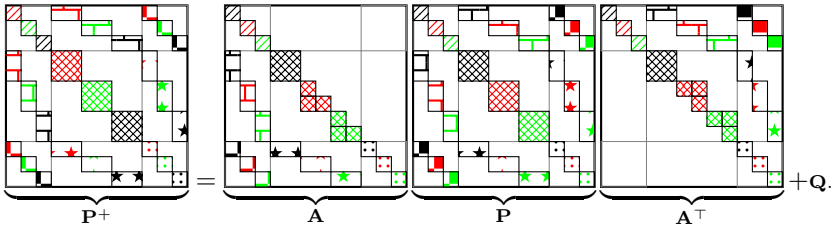
(c) Structure of $\tilde{A}_1, \tilde{A}_2, \ldots$

Figure 5.1: Illustration of the typical structure of Schur-transformed system matrices for an example system with $n = 4$.

Starting from the staircase partitioning of the Schur-transformed block diagonal form of the discrete periodic Lyapunov equations,

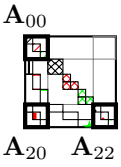


a reordering, further referred to as `staircaseOrdering`, is performed using the staircase step sizes as follows:



At first sight, this reordering makes matters worse: blocks appear farther from the diagonals than was originally the case. However, there is a crucial detail to be discerned: the periodicity of the matrix equations was transformed to a block level, hinting that the transformed problem might be decomposable into a sequence of lower-order problems.

This intuition can be formalised as follows. Write the reordered Lyapunov equations as a multiplication of its blocks, newly formed by the reordering:

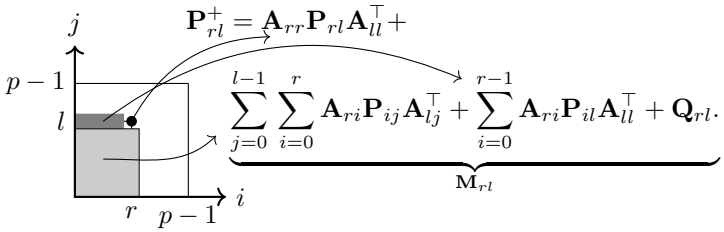


$$\mathbf{P}^+_{rl} = \mathbf{Q}_{rl} + \sum_{j=0}^{p-1}\sum_{i=0}^{p-1} \mathbf{A}_{ri}\mathbf{P}_{ij}\mathbf{A}_{lj}^\top, \quad r,l = 0,\ldots,p-1 \tag{5.43}$$

Owing to the block triangularity of $\mathbf{A}$, we have $y > x \implies \mathbf{A}_{xy} = 0$ and some of the summands can be removed:

$$\mathbf{P}^+_{rl} = \mathbf{Q}_{rl} + \sum_{j=0}^{l}\sum_{i=0}^{r} \mathbf{A}_{ri}\mathbf{P}_{ij}\mathbf{A}_{lj}^\top, \quad r,l = 0,\ldots,p-1 \tag{5.44}$$

Further splitting up the summation with the help of a summation diagram leads to:



(5.45)

In this form, it is clear that we can progressively solve low-order problems of the form $\mathbf{P}_{rl}^{+} = \mathbf{A}_{rr}\mathbf{P}_{rl}\mathbf{A}_{ll}^{\top} + \mathbf{M}_{rl}$, since $\mathbf{M}_{rl}$ does not depend on $\mathbf{P}_{ij}$ with $i > r$ or $j \geq l$. The low-order problems are discrete periodic Sylvester equations,

$$Y_{k^+} = E_k Y_k F_k^{\top} + G_k, \quad k = 0, \ldots, N-1 \tag{5.46}$$

which can be easily be solved by constructing an equivalent (sparse) linear system similar to Equation (5.10) – see Procedure 1. In this case, the maximum dense block size appearing in the linear system is of size 2×2 instead of the previous n^2.

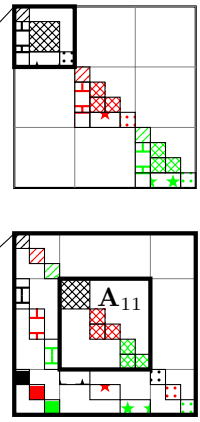
An implementation of the above ideas outlined as Algorithms 1 and 2.

Input: $[E_0, E_1, \ldots, E_{N-1}] \in [\mathbb{R}^{n\times n}]$, $[F_0, F_1, \ldots, F_{N-1}] \in [\mathbb{R}^{m\times m}]$,
$[G_0, G_1, \ldots, G_{N-1}] \in [\mathbb{R}^{n\times m}]$
Output: $[Y_0, Y_1, \ldots, Y_{N-1}]$
Solve

$$\underbrace{\begin{bmatrix} -I_{nm} & 0 & 0 & 0 & E_{N-1}\otimes F_{N-1} \\ E_0\otimes F_0 & -I_{nm} & 0 & 0 & 0 \\ 0 & E_1\otimes F_1 & -I_{nm} & 0 & 0 \\ & & \ddots & \ddots & \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & E_{N-2}\otimes F_{N-2} & -I_{nm} \end{bmatrix}}_{\mathcal{F}} \begin{bmatrix} \operatorname{vec}(Y_0) \\ \operatorname{vec}(Y_1) \\ \\ \vdots \\ \operatorname{vec}(Y_{N-1}) \end{bmatrix} = \begin{bmatrix} \operatorname{vec}(G_{N-1}) \\ \operatorname{vec}(G_0) \\ \operatorname{vec}(G_1) \\ \\ \vdots \\ \operatorname{vec}(G_{N-2}) \end{bmatrix};$$

Algorithm 1: dpse

Input: $[A_0, A_1, \ldots, A_{N-1}]$, $[Q_0, Q_1, \ldots, Q_{N-1}]$
Output: $[P_0, P_1, \ldots, P_{N-1}]$
$Z_\bullet, T_\bullet \leftarrow \texttt{periodicSchur}(A_\bullet^\top)$;
for $k \leftarrow 0, 1, \ldots, N-1$ **do**
  $\tilde{A}_k \leftarrow T_k^\top$;
  $\tilde{Q}_k \leftarrow Z_{k^+}^\top Q_k Z_{k^+}$;
$s \leftarrow \texttt{staircaseSteps}(\tilde{A}_0)$;
$\mathbf{Q} \leftarrow \texttt{staircaseOrdering}(s; \tilde{Q}_\bullet)$;
$\mathbf{A} \leftarrow \texttt{staircaseOrdering}(s; \tilde{A}_\bullet)$;
for $l \leftarrow 0, 1, \ldots, \mathit{size}(s) - 1$ **do**
  for $i \leftarrow 0, 1, \ldots, l-1$ **do**
    $F_i \leftarrow \sum_{j=0}^{l-1} \mathbf{P}_{ij} \mathbf{A}_{lj}^\top$;
  for $r \leftarrow 0, 1, \ldots, l$ **do**
    if $r = l$ **then**
      $F_l \leftarrow \sum_{j=0}^{l-1} \mathbf{P}_{lj} \mathbf{A}_{lj}^\top$;
    $G \leftarrow \sum_{i=0}^{r-1} \mathbf{A}_{ri} \mathbf{P}_{il}$;
    $M \leftarrow \mathbf{Q}_{rl} + \sum_{i=0}^{r} \mathbf{A}_{ri} F_i + G \mathbf{A}_{ll}^\top$;
    $\mathbf{P}_{rl} \leftarrow \mathbf{P}_{lr}^\top \leftarrow \operatorname{diag}\left(\texttt{dpse}(\mathbf{A}_{rr}, \mathbf{A}_{ll}, M)\right)$;
$\tilde{P}_\bullet \leftarrow \texttt{staircaseOrdering}^{-1}(s; \mathbf{P})$;
for $k \leftarrow 0, 1, \ldots, N-1$ **do**
  $P_k \leftarrow Z_k \tilde{P}_k Z_k^\top$;



Solver 2: lyap

5.4.2 Smith-type solvers

This short section presents results from Smith 1968 and Li 2012, investigating the applicability to the current goal of creating efficient embeddable Lyapunov solvers.

In its most simple form, the Smith iteration scheme reads:

Input: A, Q
Output: P
$X_{-1} \leftarrow 0$;
$X_0 \leftarrow Q$;
$k \leftarrow 0$;
while $||X_k - X_{k-1}|| > \epsilon$ **do**
  $X_{k+1} \leftarrow AX_kA^\top + Q$;
  $k \leftarrow k + 1$;
end
$P \leftarrow X_k$;

Solver 3: Simple Smith iteration

The run-time complexity of one iteration in this scheme can easily be shown to be:

$A \in \mathbb{R}^{n\times n}$ (dense)	A from `lifting` a dense (n, N)-DPLE
$O(n^3)$	$O(n^3N)$

Notice how the complexity of `lifting.smith` is on par with the `slicot` solver. When the discretised A is sparse (which does not typically occur for our applications), Smith may outperform `slicot`.

The order of convergence for this algorithm is linear, while the spectral radius of A determines the convergence factor. Obviously, stability of A is required to attain a solution.

Convergence can be improved to be quadratic by the method of frequency doubling yielding the squared Smith method:

Input: A, Q
Output: P
$X_{-1} \leftarrow 0$;
$X_0 \leftarrow Q$;
$Q_0 \leftarrow Q$;
$A_0 \leftarrow A$;
$k \leftarrow 0$;
while $||X_k - X_{k-1}|| > \epsilon$ **do**
  $X_{k+1} \leftarrow A_k X_k A_k^\top + Q_k$;
  $Q_{k+1} \leftarrow A_k Q_k A_k^\top + Q_k$;
  $A_{k+1} \leftarrow A_k^2$;
  $k \leftarrow k + 1$;
end
$P \leftarrow X_k$;

Solver 4: Squared Smith iterations (with frequency doubling)

Here the number of iterations may be drastically reduced, while the per-iteration cost is only a small multiple of the normal Smith algorithm.

A remark is in order about the term $A_{k+1} \leftarrow A_k^2$. While in general, squaring will destroy sparsity, it is not an issue here since we work with a DLE originating from either a lifting or condensing transformation of an N-periodic DPLE with dense n-by-n system dynamics. For the condensing case, A_k in the algorithm is already dense. For the lifting case, the structure of A_k, shown in Equations (5.12) and (5.13) is shifted under squaring, without additional fill-in.

For a low-rank variant of the Smith solver algorithm, write the iterations using $Q = CVC^\top$ as noise input:

$$X_0 = CVC^\top \tag{5.47}$$

$$X_1 = CVC^\top + ACVC^\top A^\top = \overbrace{\begin{bmatrix} C & AC \end{bmatrix}}^{D_1} \begin{bmatrix} V & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} C \\ AC \end{bmatrix} \tag{5.48}$$

$$X_2 = \overbrace{\begin{bmatrix} C & AC & A^2C \end{bmatrix}}^{D_2} \begin{bmatrix} V & 0 & 0 \\ 0 & V & 0 \\ 0 & 0 & V \end{bmatrix} \begin{bmatrix} C \\ AC \\ A^2C \end{bmatrix} \tag{5.49}$$

It may be that the matrix D_k never attains structural full row-rank as k increases. In that case the solution to the DLE is of structural low-rank itself. Notice the similarity between D_k and the controllability matrix: it is as if the noise input

is attempting to affect the system in the whole state space, spreading out its influence at each iteration. Failing to do so (controllability not full row-rank), corresponds to a low-rank Lyapunov solution, or in other words, the existence of directions in state space that have no uncertainty associated.

Apart from structural rank considerations, solutions to low-rank Lyapunov equations tend to have low numerical rank (Tippett 2000). This feature can be exploited by numerically compacting the factors D_k and V each iteration, e.g. using a QR decomposition, trading accuracy for speed/memory.

However, we will proceed with an exact algorithm in this context. Rather than constructing the dense X as an intermediate step, we can construct immediately the output Y. Also, there is no need to store the growing matrix D_k, unless when performing adjoint AD on the algorithm.

Input: A, V, C, H
Output: Y
$C_0 \leftarrow C$;
$Y_0 \leftarrow 0$;
$k \leftarrow 0$;
while $||C_k V C_k^\top|| > \epsilon$ **do**
  $Y_{k+1} \leftarrow Y_k + H^\top C_k V C_k^\top H$;
  $C_{k+1} \leftarrow A C_k$;
  $k \leftarrow k + 1$;
end
$Y \leftarrow Y_k$;

Solver 5: Low-rank Smith iterations

The per-step complexity is given by:

$A \in \mathbb{R}^{n \times n}$ (dense)	A from lifting a dense (n, N)-DPLE
$O(n^2)$	$O(n^2 N)$,

where the crucial point is that the dimension of the noise does not scale with n.

The low-rank Smith algorithm does not benefit from frequency doubling like the full-rank version does. Indeed, the noise input Q_k updated by:

$Q_{k+1} \leftarrow A_k Q_k A_k^\top + Q_k,$;

would correspond to $D_k V_k D_k^\top$ with $D_0 = C, V_0 = V$ and an update form:

$D_{k+1} \leftarrow [D_k, A_k D_k]$;

$V_{k+1} \leftarrow \begin{bmatrix} V_k & 0 \\ 0 & V_k \end{bmatrix}.$;

The computational costs that these updates entail is exponential in the iteration

number and may easily dominate the iteration cost of a tentative frequency-doubling low-rank Smith iteration.

One might just as well modify Algorithm 5 to check convergence only after each doubling of the iteration counter k, to effectively obtain the same behaviour and run-time as the tentative frequency-doubling variant.

Frequency-doubling low-rank Smith-type algorithms do appear frequently in the literature, but only in combination with numerical rank-compactors, yielding algorithms that balance accuracy and growth of the iteration expense such as in Benner 2014b; Li 2012, 2013; Sadkane 2012. Recall from Section 5.3.2 that LRDLE has no satisfactory high-level AD rule, unless for first-order if the undifferentiable guts of the numerical algorithm, i.e. the compacted versions of D_k and V_k in this case, are exposed. This unfortunate fact, together with the desire to avoid memory allocation as discussed in Section 5.1.3, led us to not further explore this particular area at present.

5.5 Benchmarks for Lyapunov solvers

This section serves to analyse the performance of several embedded discrete periodic Lyapunov solvers proposed in this chapter.

Subsection 5.5.1 treats the solvers as isolated entities, while Subsection 5.5.2 considers their performance in an OCP context, i.e. when they are embedded.

5.5.1 The stand-alone case: solution of DPLE

In this section, the performance of several DPLE solvers is analysed for different sizes of state space and horizon length.

The matrices passed on the solvers are dense and parametrically defined as:

$$A_k = \frac{-1}{2N} I_n \qquad k = 0, 1, \ldots, N-1$$

$$Q_k = \frac{1}{N} I_n \qquad k = 0, 1, \ldots, N-1$$

A break-down of computational time into initialisation (`init`) and evaluation (`eval`) time, as well as the total memory usage is reported in Figure 5.2 for

sweeps along the n or N parameter. This experimentally obtained data is summarised in Table 5.1.

	`simple`	`slicot`
Evaluation time	$O(n^6N)$	$O(n^3N)$
Memory	$O(n^4N)$	$O(n^2N)$
Initialisation time	$O(n^4N)$	$O(n^2N)$

Table 5.1: Resource usage complexities for different solvers.

In the `slicot` case, a further breakdown into external routines is provided in Figure 5.2: the time spent in the periodic Schur decomposition (`psd`) and the linear solver used for low-order sub-problems (`lin`) is shown in the bottom-left figure.

The combined time of these external routines is rather low compared to the total evaluation time. Referring to Algorithm (2), there is only ordering and linear algebra besides the external routines. Since ordering is a cheap operation, this means that the linear algebra inside our implementation (mainly sub-matrix multiplication) is underperforming. For increased performance, a BLAS routine such as `dgemm` should be used.

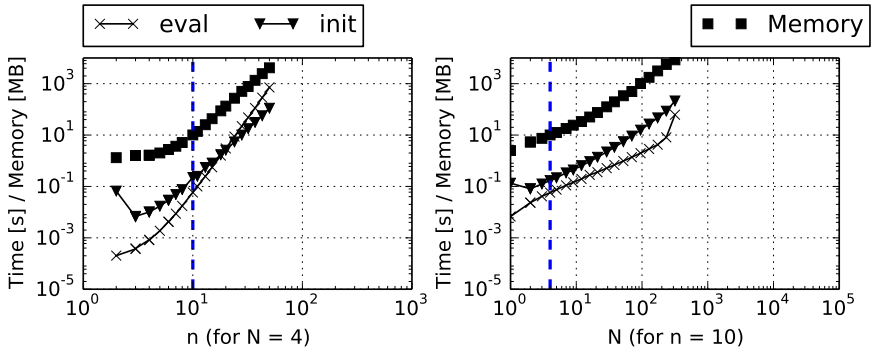
Figure 5.3 shows a convenient overlay of computation times for different solvers, and in addition gives the throughput speed of algorithmic differentiation. As expected, the evaluation times for sensitivities are a small multiple of the original evaluation cost. Since the timings are amortised over 64 concurrent sweeps, the factor may in fact be lower than 1, as will be clarified in Section 6.3.1.

Lastly, we show in Figure 5.4 the timings of function evaluations for a wider range of solvers. The matrices $A_\bullet$ used for these plots are different from those above, in order to guarantee a stable dynamics such that Smith iterations may converge. Also, for the Smith-type solvers, a fixed number of iterations is performed.

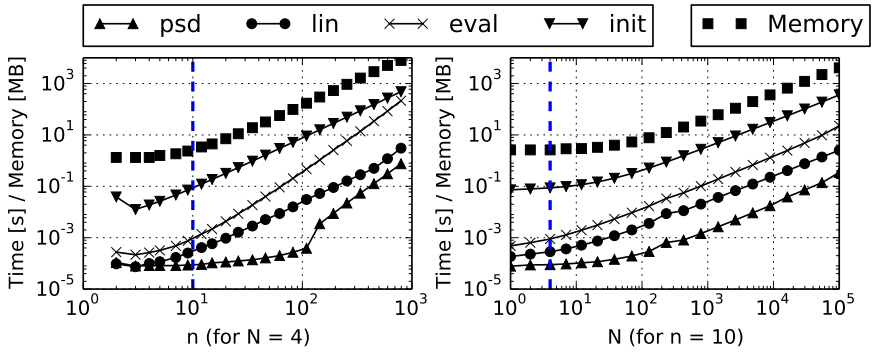
The particular matrices used for these benchmarks did not result in differences in accuracy between the various solver. For a comparative numerical experiment concerning ill-conditioned matrices, we refer to Section 5.6.

5.5.2 The embedded case: solution of robust OCP

In this section we report on benchmark results for robust OCP scenarios for several formulations based on a multiple-shooting discretisation on N intervals. For each combination of scenario and formulation, we report on



(a) `simple` solver



(b) `slicot` solver

Figure 5.2: Scaling of computational resources

the computational resources – time and memory – that are needed to perform one step in resulting the NLP.

The scenarios have little physical interpretation yet represent structures that are typical of robustified OCPs.

5

Model All scenarios are loosely based on a tracking OCP problem using the nonlinear ODE model of a chain of q masses connected by springs described in Wirsching 2006, but modified such that the input serves directly as a position instead of a velocity.

For each ball, the model has one position coordinate $p \in \mathbb{R}^2$ and one velocity coordinate $v \in \mathbb{R}^2$:

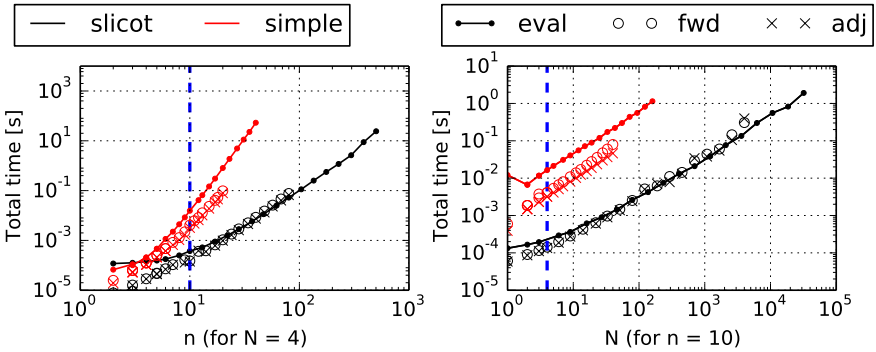


Figure 5.3: Timings for a single function evaluation, for single sensitivity sweeps, and for sensitivity sweeps amortised over 64 evaluations.

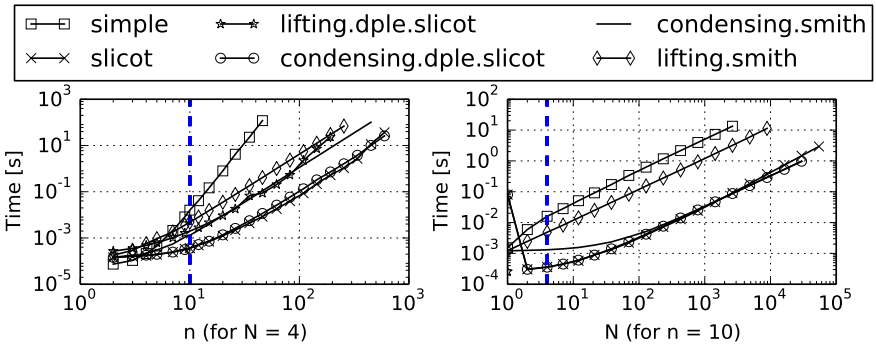


Figure 5.4: Timings for a wider range of solvers.

$$
\begin{cases}
\dot{p}^0 = v^0 \\
\dot{p}^1 = v^1 \\
\vdots \\
\dot{p}^{q-1} = v^{q-1} \\
\dot{v}^0 = \frac{F(p^1-p^0)-F(p^0)}{m} + [0-g]^\top - cv^0 \\
\dot{v}^1 = \frac{F(p^2-p^1)-F(p^1-p^0)}{m} + [0-g]^\top - cv^1 \\
\vdots \\
\dot{v}^{q-2} = \frac{F(p^{q-1}-p^{q-2})-F(p^{q-2}-p^{q-3})}{m} + [0-g]^\top - cv^{q-2} \\
\dot{v}^{q-1} = \frac{F(u-p^i)-F(p^{q-1}-p^{q-2})}{m} + [0-g]^\top - cv^{q-1},
\end{cases}
\tag{5.50}
$$

with $u \in \mathbb{R}^2$ the system input and with a spring-force given as:

$$F(\Delta p) = D\left[1 - L/||\Delta p||_2\right]\Delta p, \tag{5.51}$$

resulting in a system $\dot{x} = f(x,u)$ of size $x \in \mathbb{R}^n$ with $n = 4q$ states.

The system settings are:

T	Time period [s]	10
D	Spring constant [N/m]	0.1
g	Gravity [m/s^2]	9.81
L	Rest length [m]	0.033
m	Mass [kg]	0.15/n
c	Damping. [N $\cdot$ s/m]	0.1/n/T

For the multiple-shooting discretisation, we use an explicit fourth order Runge-Kutta scheme built-in in `CasADi` ('`rk`'). Using an integrator setting of one finite element, this integrator is actually wildly inaccurate for the system settings. This inaccuracy, however, does not affect the relative amount of resources required per NLP step and thus is irrelevant for the purpose of this section.

For the discrete formulation, we write the integrator using states and controls on a multiple-shooting interval k as:

$$x_{k+1} = \Phi(x_k, u_k) \tag{5.52}$$

As before, it is assumed that $\frac{\partial \Phi}{\partial x}$ is dense. For the low-accuracy one-element explicit '`rk`' integrator in `CasADi` , this is not actually the case. For the purpose of the benchmark, we make it dense artificially.

Formulation The formulation for the robust OCP benchmark is given as a boilerplate, and expressions behind the symbols ♠, ★ and ♣ will be specified later:

$$\begin{aligned}
&\underset{p_\bullet, v_\bullet, u_\bullet, \spadesuit}{\text{minimise}} && \textstyle\sum_{k=0}^{N-1} ||u_k - \bar{u}||_2^2 + \sum_{k=0}^{N-1} ||p_k^{\lfloor n/2 \rfloor} - \bar{p}_k||_2^2 && \text{Tracking objective} \\
&\text{s.t.} && 0 = \Phi(x_k, u_k) - x_{k+1} \quad k = 0, \ldots, \tilde{N} && \text{Shooting} \\
& && 0 = x_0 - x_N && \text{Periodicity} \\
& && \bigstar && \text{Robust path constraints} \\
& && \clubsuit, && \text{Lyapunov constraints}
\end{aligned} \tag{5.53}$$

where $\tilde{N} = N - 1$ and $\lfloor \bullet \rfloor$ rounds down to an integer. $p_k^{\lfloor n/2 \rfloor}$ selects the position vector of the mass in the middle of the chain, at time instance k.

Five variants of this formulation related to the Lyapunov equations are considered:

Variant 1: con The continuous form of the Lyapunov equations ($\dot{P} = \frac{\partial f}{\partial x} P + P \frac{\partial f}{\partial x}^\top + Q$) is used, and integrated using a version of Φ with an augmented state space.

$$Q = I_n T$$

$$\spadesuit = P_\bullet$$

$$\clubsuit = \begin{cases} 0 = \Phi'(P_k, x_k, u_k) - P_{k+1} & k = 0, \ldots, \tilde{N} \quad \text{Lyap. integrator} \\ 0 = P_0 - P_N & \text{Lyap. periodicity} \end{cases}$$

Variant 2: con sym Same as above, but exploiting symmetry of P: only the below-diagonal entries of P are decision variables.

Variant 3: dis The discrete form of the Lyapunov equations is used.

$$Q = I_n N$$

$$\spadesuit = P_\bullet$$

$$\clubsuit = \begin{cases} 0 = \frac{\partial \Phi}{\partial x} P_k \frac{\partial \Phi}{\partial x}^\top + Q - P_{(k+1) \bmod N} & k = 0, \ldots, \tilde{N} \end{cases}$$

Variant 4: dis sym Same as above, but exploiting symmetry of P.

Variant 5: slicot The discrete form of the Lyapunov equations is used, with $P_\bullet$ eliminated: $P_\bullet = \texttt{DPLE\_solver}(\frac{\partial \Phi}{\partial x}(x_\bullet, u_\bullet), I_n/N)$. We have $\spadesuit = \{\}$ and $\clubsuit = \{\}$.

Scenarios Concerning the robust path constraints, we consider the following four scenarios with $\alpha = 1 \times 10^6$:

Scenario 1: 1 There is exactly one robust path constraint. The problem of stability optimisation with a $\mathrm{tr}(P)$ in the objective can be cast into this form.

$$\bigstar = \begin{cases} \mathrm{tr}(P_0) \leq \alpha \end{cases}$$

Scenario 2: N The number of robust path constraints scales with N. The problems for robustified tethered flight correspond to this scenario: they

have a fixed number of robust path constraints for each shooting-interval, while choices for model fidelity or parametrisation may change the dimension of n.

$$\bigstar = \{ \quad \operatorname{tr}(P_k) \leq \alpha \quad k = 0, \ldots, \tilde{N}$$

Scenario 3: n The number of robust path constraints scales with n. This scenario arises when the system is a composition of subsystems, each requiring robust path constraints at a single point in time.

$$\bigstar = \{ \quad (P_0)_{jj} \leq \alpha \quad j = 0, \ldots, n-1$$

Scenario 4: Nn The number of robust path constraints scales with Nn. This scenario arises when the system is a composition of subsystems, each requiring robust path constraints over the entire horizon.

$$\bigstar = \begin{cases} \quad (P_k)_{jj} \leq \alpha \quad \begin{cases} k = 0, \ldots, \tilde{N} \\ j = 0, \ldots, n-1 \end{cases} \end{cases}$$

Results All combinations of scenarios and Lyapunov-variants are solved with `IPOPT` for varying dimensions of n and N.

The comparison of `con` and `dis` with their symmetric counterparts reveals that, when applicable, exploiting symmetry is always a good idea. Speed-ups of a factor 2-4, and memory savings of a factor 2-6 are typical. To avoid visual clutter in the benchmark results, the non-symmetric variants are further omitted.

Figure 5.5 shows the empirically obtained complexities of computational resources. They are remarkably simple to summarise:

1. For the `n` and `1` scenarios, the `slicot` approach is unambiguously superior.
2. For the `N` and `nN` scenarios, the `slicot` approach gives a better complexity in n, but a worse complexity in N. In other words, the approach without embedded Lyapunov solvers (`con`/ `dis`) *might* perform better for small systems with long horizons.

	`con/ dis`	`slicot`
Eval. time	$O(n^6N)$	$O(n^3N)$
Memory	$O(n^5N)$	$O(n^2N)$

(a) Scenario `1`

	`con/ dis`	`slicot`
Eval. time	$O(n^6N)$	$O(n^3N^2)$
Memory	$O(n^5N)$	$O(n^2N^2)$

(b) Scenario `N`

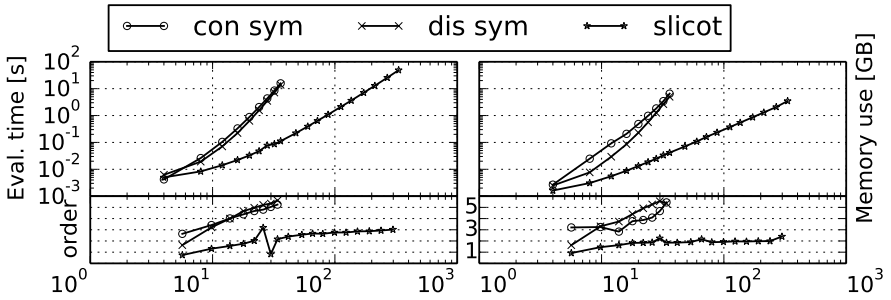
	`con/ dis`	`slicot`
Eval. time	$O(n^6N)$	$O(n^4N)$
Memory	$O(n^5N)$	$O(n^3N)$

(c) Scenario `n`

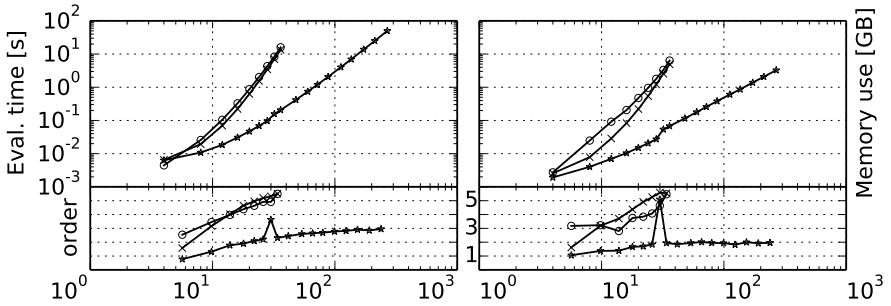
	`con/ dis`	`slicot`
Eval. time	$O(n^6N)$	$O(n^4N^2)$
Memory	$O(n^5N)$	$O(n^3N^2)$

(d) Scenario `Nn`

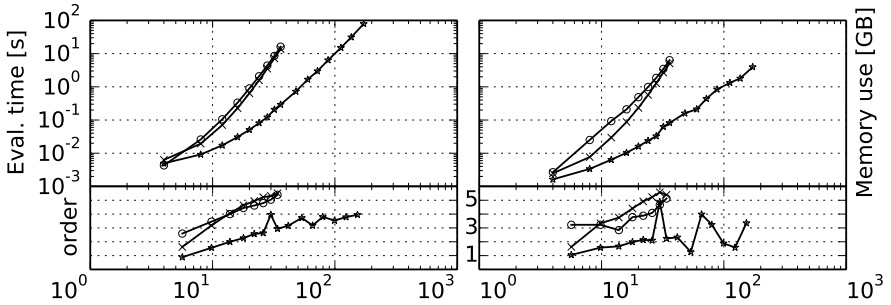
Figure 5.5: Complexity of computational resources summary.



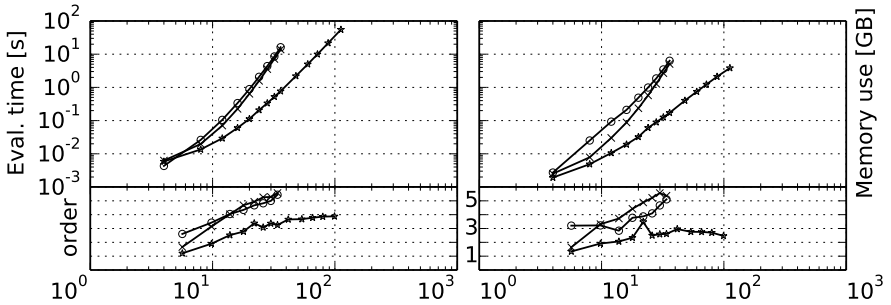
(a) Scenario `1`



(b) Scenario `N`

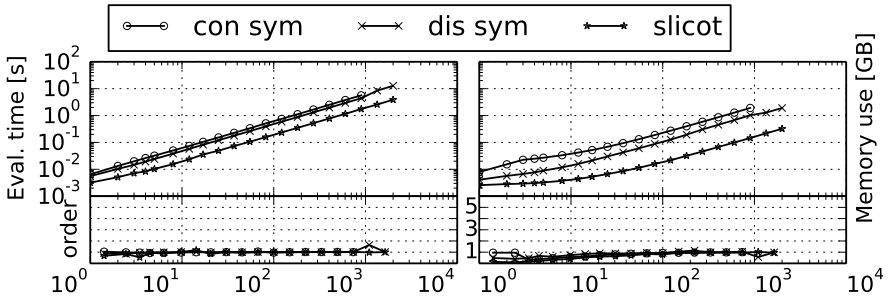


(c) Scenario `n`

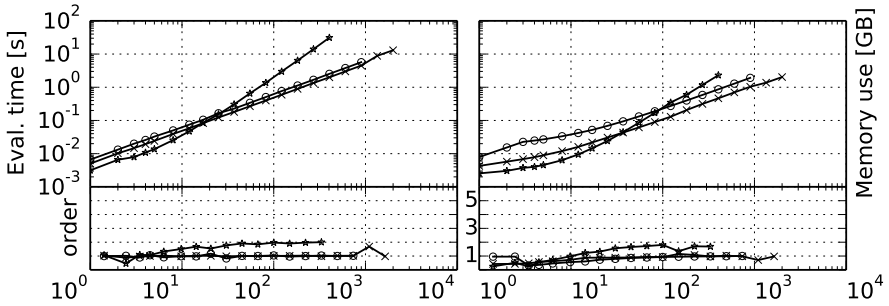


(d) Scenario `Nn`

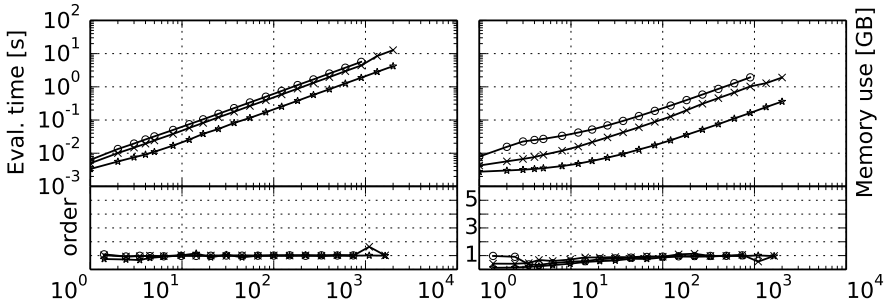
Figure 5.6: Scaling of computational resources for robust OCP with varying n on the horizontal axis for $N = 4$.



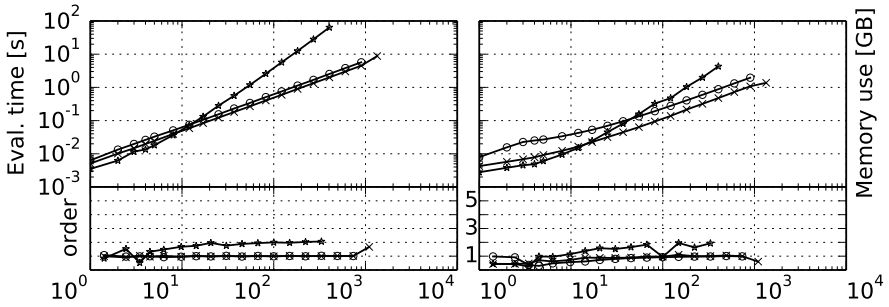
(a) Scenario `1`



(b) Scenario `N`



(c) Scenario `n`



(d) Scenario `Nn`

Figure 5.7: Scaling of computational resources for robust OCP with varying N on the horizontal axis for $n = 2$.

5.6 Notes on the numerical aspects of the Lyapunov equations

5.6.1 Conditioning of the Lyapunov equations

From the point of view of numerical stability, the demarcation between the DPLE solvers lies in the treatment of the periodicity.

For the native DPLE solvers of Section 5.2.1 as well as the lifting formulation of Section 5.2.2, the system dynamics matrices A_i appear separately in the equivalent linear system. This linear system, Equation (5.10), will be referred to as:

$$S_{\mathrm{p}}(A_\bullet)\mathrm{vec}(\mathcal{P}_\bullet) = \mathrm{vec}(\mathcal{Q}_\bullet). \tag{5.54}$$

For the condensing approach of Section 5.2.2, the system dynamics matrices A_i are multiplied out as $\bar{A} = A_{N-1} \dots A_0$, giving rise to the smaller equivalent linear system of the form of Equation (5.9):

$$S_{\mathrm{c}}(\bar{A})\mathcal{P}_0 = \mathcal{Q}_0. \tag{5.55}$$

We wish to compare the condition numbers $\kappa(S_{\mathrm{p}})$ and $\kappa(S_{\mathrm{c}})$ of these equivalent linear systems, where the condition number κ is given by:

$$\kappa(X) = ||X||_2 \, ||X^{-1}||_2, \tag{5.56}$$

for an arbitrary matrix X, and where the matrix norm $||X||_2$ equals the largest singular value $\sigma_{\max}(X)$ for square matrices X.

In the following text, we use matrix properties that are well-known and can be found in any good textbooks such as Horn 1986; Ipsen 2009; Laub 2004.

Using the triangle inequality and the permutation invariance of the spectral norm, we can obtain for the periodic equivalent system:

$$||S_{\mathrm{p}}||_2 \leq 1 + \max_i \sigma_{\max}(A_i \otimes A_i) = 1 + \max_i \sigma^2_{\max}(A_i), \tag{5.57}$$

where we have used the property $||A \otimes B||_2 = ||A||_2 \, ||B||_2$.

Conversely, for the condensed equivalent system, we obtain:

$$||S_{\mathrm{c}}||_2 \leq 1 + \Pi_i \sigma^2_{\max}(A_i), \tag{5.58}$$

where we have used the sub-multiplicative property of the spectral norm.

Comparing Equations (5.57) and (5.58) hint that the condensed form is more prone to ill-conditioning when the matrices A_i have large singular values. We offer no definite proof, since $||S_{\mathrm{c}}^{-1}||_2$ does not have a simple expression in terms of A_i; it is estimated numerically in practice (Gahinet 1990). The same remark holds for the periodic case (Varga 1997).

Instead, we offer a compelling example inspired by Tippett 2000.

We consider a periodic system with $n = 2$ states and with period $N = 2$.

The system dynamic matrices are:

$$A_0 = \begin{bmatrix} 20.2601 & -15.5336 \\ 30.8880 & -23.6821 \end{bmatrix}, \quad A_1 = \begin{bmatrix} 15.5420 & 23.6949 \\ 20.2536 & 30.8782 \end{bmatrix}.$$

The condition numbers for the equivalent linear systems differ by three orders of magnitude:

X	$\|\|X\|\|_2$	$\|\|X^{-1}\|\|_2$	$\kappa(X)$
S_{p}	2.16717×10^3	2.47078×10^7	5.35462×10^{10}
S_{c}	4.69448×10^6	2.46521×10^7	1.16722×10^{14} .

This difference in conditioning is evident when solving the DPLE with a concrete right hand side such as $Q_\bullet = I$. The table below summarises the residual error $||P_1^\star - A_0 P_0^\star A_0^\top - Q_0, P_0^\star - A_1 P_1^\star A_1^\top - Q_1||_{\max}$ of the solution $P_\bullet^\star$:

Solver approach	Solver	Residual
Periodic	`simple`	1.50513×10^{-6}
	`slicot`	1.22392×10^{-6}
	`lifting.simple`	1.50513×10^{-6}
	`lifting.dple.slicot`	1.35795×10^{-6}
Condensing	`condensing.simple`	2.52847×10^{-3}
	`condensing.dple.slicot`	1.89784×10^{-3}.

Clearly, the accuracy of the periodic approach is superior by three orders of magnitude in this example, as the condition number suggested. Also note that the `slicot` algorithm is stable: the residual error is of the same order as that of the naive factorisation of the equivalent linear system.

The construction of this numeric example is as follows. We started out with constructing a rank-1 matrix:

$$\bar{A} = \sigma y z^\top, \tag{5.59}$$

with $y, z \in \mathbb{R}^n$ and $||y||_2 = ||z||_2 = 1$. This matrix has one nonzero eigenvalue:

$$\lambda = \sigma y^\top z. \tag{5.60}$$

For the example above, we chose $y \in \mathbb{R}^2$ to be a normalised random vector and z to be very nearly orthogonal such that $y^\top z$ is small:

$$y = \begin{bmatrix} 6.08782 \times 10^{-1} \\ 7.93338 \times 10^{-1} \end{bmatrix}, \quad z = \begin{bmatrix} 7.93590 \times 10^{-1} \\ -6.08452 \times 10^{-1} \end{bmatrix}.$$

Further, we chose $\lambda = 9.00000 \times 10^{-1}$ and obtained $\sigma = 2.16667 \times 10^3$ as a consequence of Equation (5.60).

For the rank-1 structure of $\bar{A}$, there exist simple expressions for the condition number factors (Tippett 2000):

$$|1 - \sigma^2| \leq ||\bar{A} \otimes \bar{A} - I||_2 \leq 1 + \sigma^2$$

$$4.694477 \times 10^6 \leq 4.694478 \times 10^6 \leq 4.694479 \times 10^6,$$

and

$$||(\bar{A} \otimes \bar{A} - I)^{-1}||_2 \approx 1 + \frac{\sigma^2}{1 - \lambda^2}$$

$$2.46521 \times 10^7 \approx 2.47078 \times 10^7.$$

Next, we worked backward to construct A_0 and A_1 such that $\bar{A} = A_1 A_0$:

$$A_1 = \sqrt{\sigma} y w^\top, \quad A_0 = \sqrt{\sigma} w z^\top, \tag{5.61}$$

5

with w a random normalised vector:

$$w = \begin{bmatrix} 5.48463 \times 10^{-1} \\ 8.36175 \times 10^{-1} \end{bmatrix}.$$

Since the large singular value σ was distributed evenly on A_0 and A_1, the difference between Equations (5.57) and (5.58) was maximally exploited:

$$\left\| \begin{bmatrix} -I & A_1 \otimes A_1 \\ A_0 \otimes A_0 & -I \end{bmatrix} \right\|_2 \leq 1 + \max\left\{\sigma_{\max}^2(A_0), \sigma_{\max}^2(A_1)\right\}$$
$$= 1 + \max\left\{\sqrt{\sigma}^2, \sqrt{\sigma}^2\right\} = 1 + \sigma = 2167.67,$$

From numerical evaluation, $||S_{\mathrm{p}}^{-1}||_2$ seems to be similar in magnitude than $||S_{\mathrm{c}}^{-1}||_2$, but we cannot make any hard statements unfortunately even in this simple case.

5.6.2 Conditioning of the KKT system for the robust optimal control formulation

The action of eliminating variables of an optimal control problem may adversely affect the conditioning of the KKT system.

Let us consider a partitioning of decision variables $x \in \mathbb{R}^{p+q}$ into $x_p \in \mathbb{R}^p$ and $x_q \in \mathbb{R}^q$. We partition the constraints $Ax = c$ accordingly:

$$A_{pp}x_p + A_{pq}x_q = c_p$$
$$A_{qp}x_p + A_{qq}x_q = c_q.$$

Eliminating variables x_q corresponds to working with the Schur complement S of A_{qq}:

$$\underbrace{\left(A_{pp} - A_{pq}A_{qq}^{-1}A_{qp}\right)}_{S} x_q = c_p - A_{pq}A_{qq}^{-1}c_q,$$

where A_{qq} is required to be invertible.

If we take a very simple Hessian approximation $H = I$, the KKT-matrix for the partitioned system reads:

$$H_{\mathrm{f}} = \begin{bmatrix} I_p & & A_{pp}^\top & A_{qp}^\top \\ & I_q & A_{pq}^\top & A_{qq}^\top \\ A_{pp} & A_{pq} & & \\ A_{qp} & A_{qq} & & \end{bmatrix}, \tag{5.62}$$

while it takes the following form after elimination:

$$H_{\mathrm{e}} = \begin{bmatrix} I_p & S^\top \\ S & \end{bmatrix} = \begin{bmatrix} I & & \\ & I & -A_{pq}A_{qq}^{-1} \end{bmatrix} H_{\mathrm{f}} \begin{bmatrix} I & & \\ & I & -A_{pq}A_{qq}^{-1} \end{bmatrix}^\top. \quad (5.63)$$

Using known matrix norm properties, we can find:

$$||H_{\mathrm{e}}||_2 \leq 1 + ||S||_2, \quad (5.64)$$

and

$$||H_{\mathrm{e}}^{-1}||_2 = \left\| \begin{bmatrix} 0 & S^\dagger \\ \left(S^\dagger\right)^T & -\left(SS^\top\right)^{-1} \end{bmatrix} \right\|_2 \leq ||S^\dagger||_2 + ||\left(SS^\top\right)^{-1}||_2. \quad (5.65)$$

As in the previous section, it is the spectral norm of the inverse that prohibits an elegant symbolic analysis of the condition number: we have $||S^\dagger||_2 = 1/\sigma_{\min}(S)$, but a simple expression for the second norm was not found.

For the left factor of the condition number definition $\kappa(X) = ||X||_2\,||X^{-1}||_2$, we can see a significant difference between the normal case:

$$1 + \left\| \begin{bmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{bmatrix} \right\|_2 \leq 1 + ||A_{pp}||_2 + ||A_{pq}||_2 + ||A_{qp}||_2 + ||A_{qq}||_2,$$

and the eliminated case:

$$1 + \left\| \left[A_{pp} - A_{pq}A_{qq}^{-1}A_{qp}\right] \right\|_2 \leq 1 + ||A_{pp}||_2 + ||A_{pq}||_2\,||A_{qq}^{-1}||_2\,||A_{qp}||_2.$$

It appears that the formulation with elimination is more sensitive to large singular values for the submatrices. For example, the singular values of A_{qq}^{-1} should be small enough.

In our case of the robust optimal control formulation, $A_{qp}x_p + A_{qq}x_q = c_q$ should seen as the DPLE equations with $x_q = \mathrm{vec}(\mathcal{P}_\bullet)$, A_{qq} being S_{c} or S_{p} from Equations (5.57) and (5.57), and A_{qp} stemming from the dependence of $A_\bullet$ and $Q_\bullet$ on states and controls. This means that the choice of Lyapunov solver class, as explained Section 5.6.1, plays a role in conditioning of the KKT system.

5

5.7 Parallels with continuous-time approaches

For the sake of completeness, we make the connection with the continuous-time Lyapunov differential equations. Consider an integrator for the nominal system

coupled together with a Lyapunov integrator:

$$x(t) = \Phi\left(t, f; x_0, u(\bullet)\right) \tag{5.66}$$

$$R(t) = \Phi\left(t, AP + PA^\top + V; P(0) = 0, V(\bullet), x(\bullet), u(\bullet)\right). \tag{5.67}$$

If the system is integrated all the way to T (as in single-shooting), the periodicity constraint becomes a DLE:

$$P = \frac{\partial x(T)}{\partial x_0} P \frac{\partial x(T)}{\partial x_0}^\top + R(T). \tag{5.68}$$

Solving for $P^\star$ with any of the discussed solvers, one can consequently obtain an expression for the Lyapunov matrix:

$$P(t) = \Phi\left(t, AP + PA^\top + V; P^\star, V(\bullet), x(\bullet), u(\bullet)\right). \tag{5.69}$$

When an implicit integration scheme is applied to the continuous Lyapunov differential equations, a separable continuous Lyapunov algebraic equation (CLE) appears that can be solved with similar techniques as DLEs. Such structure exploiting Lyapunov integrator was worked out in Sternberg 2012b,c, and using such dedicated integrator together with the described method leads to a complexity $O(n^3)$. This combination, which was proposed in private communication by Boris Houska, is essentially equivalent to our `condensing` transformation, with R being obtained by integration rather than through accumulation and $\frac{\partial x(T)}{\partial x_0}$ obtained by a single-shooting approach rather than a multiplication of multiple-shooting results. The equivalence implies that the method is prone to the ill-conditioning associated with the condensing approach, as discussed in Section 5.6.

It should be noted that both a dedicated Lyapunov integrator and DLE solver are needed to avoid $O(n^6)$ complexity with this approach. In contrast, the discrete approach proposed in this thesis does not need the notion of a Lyapunov integrator to achieve the same order of performance, but instead requires sensitivities of a (classic) integrator.

Conclusion

This chapter expanded on the novel idea of the previous chapter of using the *discrete* periodic Lyapunov equations (DPLE) rather than the continuous form to solve robust optimal control problems.

In the previous chapter, the DPLE ended up as equality constraints in the NLP. In contrast, this chapter explored the idea to eliminate the Lyapunov matrix from the decision variables, opening up the potential to use efficient DPLE solvers from the literature.

Although the literature on stand-alone Lyapunov solvers is vast, only a small part of results was found to be immediately usable in our context of robust optimal control.

Notably, low-rank solvers are problematic to embed, and positive definiteness cannot be assumed about the noise input, excluding Cholesky based algorithms.

Despite these setbacks, a periodic Schur-based algorithm from the literature (Varga 1997) was identified and used to improve run-time complexity for the stand-alone DPLE solver from a naive $O(n^6)$ to an efficient $O(n^3)$ with n the number of states. In the benchmarks section, this theoretical complexity for the stand-alone use case is confirmed with experimental measurements.

The Schur-based algorithm was implemented to perform well in an optimal control framework. In particular, differentiation rules for DPLE where identified such that the solver, named the `slicot` solver, is embeddable in an infinitely-differentiable computational graph.

For the formulations of robust OCP, the benchmarks section revealed a subdivision in performance between two classes of scenarios. In the case that the number of robust constraints does not scale with the horizon length, elimination of covariances P and use of dedicated Lyapunov solvers is unambiguously good for efficiency: it improves the complexity from $O(n^6N)$ to $O(n^3N)$ with N the horizon length.

In the case that the number of robust constraints *does* scale with horizon length, the classic approach without embedded Lyapunov solvers, with P retained as decision variables, *may* give a better performance: the complexity changes from $O(n^6N)$ to $O(n^3N^2)$.

In related work, in Sternberg 2012c, the author has proposed a type of Lyapunov integrator for robust OCP applications that exploits some of the Lyapunov structure, yet still retains the $O(n^6)$ complexity for robust OCP because of the periodicity constraints. An unpublished extension of this formulation with a DLE solver would bring down this complexity and correspond to a condensing transformation as presented in this chapter. The condensing transformation, however, was shown in an example to be vulnerable to ill-conditioning.

Part II

Software and Applications

Chapter 6

CasADi

CasADi is a tool that allows one to easily prototype formulations and algorithms in the field of dynamic optimisation (Andersson 2012; Andersson 2013). It offers a set of mathematical components – such as linear solvers, implicit function solvers, numerical integrators – that can be embedded into symbolic sparse-matrix-valued computational expression graphs. These graphs are sorted into algorithms/functions, on which forward and adjoint source-code-transforming algorithmic differentiation is applied in order to efficiently obtain gradients, Jacobians, Hessians, ….

One typically constructs large-scale non-linear programs which are passed within the CasADi framework to derivative-based optimisation routines such as IPOPT (an interior point method, Wächter 2006a), WORHP (an SQP method, Büskens 2012) and SNOPT (an SQP method, Gill 2005). CasADi combines the ease of use of a scripting language (through the Python interface – van Rossum 2001) with the performance of a compiled language: all code is written in cross-platform C++. After mounting an admittedly steep learning curve, a user can obtain prototyping-style speed of development yet benefit from state-of-the-art performance. Code generation of functions is possible to obtain code for embedded systems for running NMPC loops.

Core values of the CasADi project are modularity, maintainability and efficiency and its target audience consists of researchers in the field of dynamic optimisation, application-experts with complex modelling needs, and developers of dynamic optimisation toolsets.

Below is a comparison of CasADi with similar tools:

CasADi	*ACADO Toolkit* 2009–2013
• Write your own solver using a pool of building-blocks • No limitations on formulation • Good at large-scale OCP • Easy to extend	• Black-box solver • Standard-form OCP • Good at small-scale real-time NMPC • Easy to get started

CasADi	Other operator-overloading AD tools e.g. ADOL-C, Griewank 1999
• Scalar and matrix graphs for fine-grained memory-speed trade-off • Sensitivities up to arbitrary order (source-code transforming of the computational graphs) • Code generation facilities • Parallelisation facilities • Batteries included: NLP-solvers, integrators, …	• Scalar graphs only, checkpointing • Little changes to the original code needed

CasADi is developed by Joel Anderson, myself and Greg Horn under a permissive open-source license. This chapter serves to highlight some of my contributions to the project which were needed to solve the problems in this thesis.

Section 6.1 walks the reader through some essential CasADi concepts and constructs by working towards an optimal control problem. Section 6.2 highlights the contribution of structured indexing, while Section 6.3 deals with that of Lyapunov solver implementation, including benchmarks and an example. The final section (6.4) treats hierarchical sparsity seeding, a contribution presented at the CSC2014 conference.

6

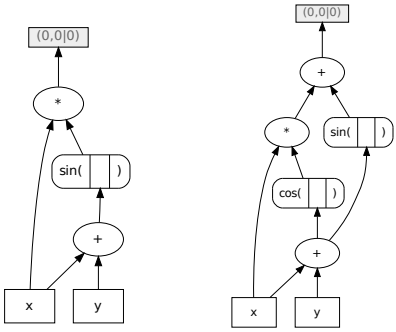
6.1 A compact introduction

This section is also accessible in extended form as an interactive video-tutorial (Gillis 2014).

CasADi, with the first three letters of its name hinting at "computer algebra system" (CAS), features symbolic primitives which can be composed with common mathematical operations into expression graphs:

```
from casadi import *

x = SX.sym("x")
y = SX.sym("y")
z = x*sin(x+y)

print "z=", z
print "J=", jacobian(z,x)
```

```
z= (x*sin((x+y)))
J= (sin((x+y))+(x*cos((x+y))))
```



In contrast to the impression one might get from the printout on the left, and in contrast to many popular CAS packages (Maple, Maplesoft 2004 – Matlab symbolic, McAndrew 1999 – sympy, SymPy Development Team 2014), CasADi is shown on the right to share sub-expressions to form computational graphs, not just trees. For the large-scale applications that are targeted, this distinction is crucial.

By creating a function, the expression graph is sorted into an algorithm: an ordered list of operations working on a random-access work vector. The function can be evaluated numerically or symbolically:

```
f = SXFunction([x,y],[z])
f.init(); print f
```

```
 Inputs (2):
  0. 1-by-1 (dense)
  1. 1-by-1 (dense)
 Output: 1-by-1 (dense)
@0 = input[0][0];
@1 = input[1][0];
@1 = (@0+@1);
@1 = sin(@1);
@0 = (@0*@1);
output[0][0] = @0;
```

```
f.setInput(1.2,0)
f.setInput(3.4,1)
f.evaluate()

print f.getOutput(0)
print f([1.2,3.4])
print f([1.2,x+y])
```

```
-1.19243
[DMatrix(-1.19243)]
[SX((1.2*sin((1.2+(x+y)))))]
```

Note that live-variables are used: the work-vector variables (`@0, @1, …`) are re-used.

Scalar expressions (SX) can be composed into matrices for modelling convenience. A special feature of CasADi is the notion of matrix-valued expression (MX) graphs. MX operations are not expanded down to scalar operations, but retained as a single node in the expression graph. This allows for a) greater efficiency in memory (similar to checkpointing, Bartholomew-Biggs 2000) and b) the embedding of numerical processes into the graph.

```
A = SX.sym("A",2,2)
B = SX.sym("B",2)
print mul(A,B), "\n", solve(A,B)
```

```
[((B_0*A_0)+(B_1*A_2)), ((B_0*A_1)+(B_1*A_3))]
[((B_0*(A_3/((A_0*A_3)-(A_2*A_1))))+(B_1*((-A_2
    )/((A_0*A_3)-(A_2*A_1))))), ((B_0*((-A_1)
    /((A_0*A_3)-(A_2*A_1))))+(B_1*(A_0/((A_0*
    A_3)-(A_2*A_1)))))]
```

```
A = MX.sym("A",2,2)
B = MX.sym("B",2)
print mul(A,B)
print solve(A,B)
```

```
(zeros(2x1, dense)
    +mul(A'', B))
(A\B)
```

Jacobians of expressions are found by first constructing a function. Functions for forward-mode and backward-mode sensitivities are derived from this.

```
u = SX.sym("u")
p,q,c = SX.sym("[p,q,c]")
ode = vertcat([
  (1 - q**2)*p - q + u,
  p, p**2+q**2+u**2])
x = vertcat([p,q,c])
f = SXFunction([x,u],[ode])
f.init()
ffwd = f.derivative(1,0)
fadj = f.derivative(0,1)
```

```
*********f*********|**********ffwd**********|**********fadj**********
 Inputs (2):       | Inputs (customIO: 4): | Inputs (customIO: 3):
  0. 3-by-1        |  0. (der_0)  3-by-1   |  0. (der_0)  3-by-1
  1. 1-by-1        |  1. (der_1)  1-by-1   |  1. (der_1)  1-by-1
 Output: 3-by-1    |  2. (fwd0_0)  3-by-1  |  2. (adj0_0)  3-by-1
@0 = input[0][1]; |  3. (fwd0_1)  1-by-1  | Outputs (customIO: 3):
@1 = sq(@0);       | Outputs (customIO: 2): | 0. (der_0)  3-by-1
@2 = 1;            |  0. (der_0)  3-by-1   |  1. (adj0_0)  3-by-1
@2 = (@2-@1);      |  1. (fwd0_0)  3-by-1  |  2. (adj0_1)  1-by-1
@1 = input[0][0]; |@0 = input[0][1];      |@0 = input[0][1];
@2 = (@2*@1);      |@1 = sq(@0);           |@1 = sq(@0);
@2 = (@2-@0);      |@2 = 1;                |@2 = 1;
@3 = input[1][0]; |@2 = (@2-@1);          |@2 = (@2-@1);
@2 = (@2+@3);      |@1 = input[0][0];      |@1 = input[0][0];
output[0][0] = @2; |@3 = (@2*@1);         |@3 = (@2*@1);
output[0][1] = @1; |@3 = (@3-@0);         |@3 = (@3-@0);
@1 = sq(@1);       |@4 = input[1][0];      |@4 = input[1][0];
...                |...                    |...
```

6

Seeding these with slices of the unit matrix leads to respectively the columns and rows of the Jacobian. Actually, CasADi computes the sparsity during initialisation of the function, and can exploit this (graph colouring) for smarter seeding. More details about this will be given in Section 6.4.

```
I = SX.eye(3)
J = jacobian(ode,x)
J.sparsity().spy()
print J
```

```
**.
*..
**.

[[(1-sq(q)), ((p*(-(q+q)))
     +-1), 00],
 [1, 00, 00],
 [(p+p), (q+q), 00]]
```

```
for i in range(3):
  print ffwd([ x,u, I[:,i],0 ]) [1]

for i in range(3):
  print fadj([ x,u, I[:,i] ]) [1]
```

```
[(1-sq(q)), 1, (p+p)]
[((p*(-(q+q)))-1), 0, (q+q)]
[0, 0, 0]
[(1-sq(q)), (-1+((q+q)*(-p))), 0]
[1, 0, 0]
[(p+p), (q+q), 0]
```

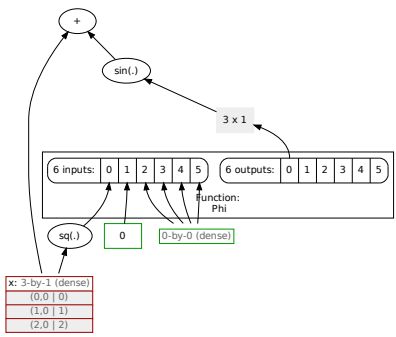
An integrator is one of the many classes of functions available. Just as any other CasADi function, it can be evaluated numerically or be embedded into an MX graph. To make syntax uniform, an `SXFunction` with a DAE input/output scheme is preferred over positional arguments.

```
f = SXFunction(daeIn(x=x,p=u),
               daeOut(ode=ode))
f.init()

tf = 10.0; N = 20; dt = tf/N

Phi = Integrator("cvodes",f)
Phi.setOption("name","Phi")
Phi.setOption("tf",dt)
Phi.init()

x0 = vertcat([0,1,0])

print Phi(x0=x0)["xf"]
```

```
[-0.494018, 0.876096, 0.500984]
```

```
X0 = MX.sym("x",3)
print sin(Phi(x0=X0**2)["xf"])+X0
```

```
(sin(function("Phi").call([sq(x), 0,
      0x0, 0x0, 0x0, 0x0]){0})+x)
```



We finish this introduction by showing how to use a multiple-shooting strategy to solve the OCP from Section 2.3 by multiple-shooting in CasADi. The required transformation into standard NLP form reads:

$$\begin{aligned} \underset{x_\bullet, u_\bullet}{\text{minimise}} \quad & c_N \\ \text{subject to} \quad & x_{k+1} - \Phi(x_k, u_k) = 0 \\ & p_0 = 0, q_0 = 1, c_0 = 0 \\ & -1 \le u_k \le 1 \end{aligned} \quad \rightarrow \quad \begin{aligned} \underset{X}{\text{minimise}} \quad & F(X, P) \\ \text{subject to} \quad & \text{lbx} \le X \le \text{ubx} \\ & \text{lbg} \le G(X, P) \le \text{ubg} \end{aligned}$$

In CasADi, we start by constructing the long vector of decision variables X. A *structure*, to be explained in the next section, is preferred over a manual `MX.sym` construction to have convenient slicing keyword-enabled slicing in addition to just numerical indexing.

```
X = struct_symMX([ (
      entry("x",repeat=N+1,shape=3),
      entry("u",repeat=N)
     ) ])

g = [ X["x",k+1] -
      Phi(x0=X["x",k],p=X["u",k])["xf"]
         for k in range(N) ]

obj = X["x",N,2]

nlp = MXFunction( nlpIn(x=X),
        nlpOut(g=vertcat(g),f=obj) )
S = NlpSolver("ipopt",nlp)
```

```
S.init()
S.setInput(0,"lbg")
S.setInput(0,"ubg")

lbx = X(-inf)
ubx = X(inf)
lbx["u",:] = -1
ubx["u",:] = 1
lbx["x",0] = x0
ubx["x",0] = x0

S.setInput(lbx,"lbx")
S.setInput(ubx,"ubx")
S.evaluate()
```

The resulting NLP is solved by IPOPT. Note from the statistics in the following printout that IPOPT reduced the total number of decision variables by eliminating p_0, q_0, c_0 since they are fixed. This behaviour can be influenced by setting options prior to initialising.

```
Number of nonzeros in equality constraint Jacobian...:   253
Number of nonzeros in Lagrangian Hessian.....:   115

Total number of variables.....:    80
              variables with lower and upper bounds:    20
Total number of equality constraints.....:    60

iter    objective    inf_pr   inf_du lg(mu) ||d|| lg(rg) alpha_du alpha_pr ls
   0  0.0000000e+00 8.76e-01 1.52e-02 -1.0 0.00e+00  -  0.00e+00 0.00e+00  0
```

```
  1  2.5486930e+00 7.94e-02 1.65e+00 -1.7 1.13e+00   -  4.89e-01 9.47e-01H 1
  2  2.9354426e+00 1.47e-03 1.07e-01 -1.7 4.00e-01   -  1.00e+00 1.00e+00h 1
  3  2.9329088e+00 5.01e-04 1.66e-03 -2.5 2.40e-02   -  1.00e+00 1.00e+00h 1
  4  2.9331084e+00 2.21e-04 3.18e-04 -3.8 2.00e-02   -  1.00e+00 1.00e+00h 1
  5  2.9330627e+00 3.88e-05 2.63e-05 -5.7 8.33e-03   -  1.00e+00 1.00e+00h 1
  6  2.9330138e+00 2.73e-06 7.18e-07 -5.7 2.28e-03   -  1.00e+00 1.00e+00h 1
  7  2.9330077e+00 1.57e-08 4.21e-09 -8.6 1.75e-04   -  1.00e+00 1.00e+00h 1
  8  2.9330077e+00 1.98e-11 4.18e-12 -8.6 8.47e-07   -  1.00e+00 1.00e+00h 1

Optimal Solution Found.
time spent in eval_f: 0.017805 s. (10 calls, 1.7805 ms. average)
time spent in eval_grad_f: 0.016499 s. (10 calls, 1.6499 ms. average)
time spent in eval_g: 0.018139 s. (10 calls, 1.8139 ms. average)
time spent in eval_jac_g: 0.084771 s. (11 calls, 7.70645 ms. average)
time spent in eval_h: 0.230212 s. (9 calls, 25.5791 ms. average)
time spent in main loop: 0.37315 s.
```

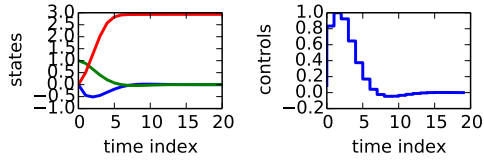
The given example enjoys textbook-style convergence: full steps are taken in nearly all iterations (cf. `alpha` and `ls` column), regularisation of the Hessian was nowhere needed (cf. `lg(rg)`), the barrier parameter used for the interior-point aspect of the method is brought down quickly (cf. `lg(mu)`), and convergence criteria all decrease quadratically (constraint violation `inf_pr`, optimality `inf_du`, step size `||d||`).

It should be noted that the required Jacobians and Hessians are constructed in the background and passed on to the IPOPT automatically. The constraint Jacobian features the block-diagonal structure that can be expected from multiple-shooting formulations:

```
J = S.jacG().output()
print J.shape
J[:12,:20].sparsity().spy()
```

```
(60, 83)
**.**.............
**.*.*............
****..*...........
....**.**.........
....**.*.*........
....****..*.......
........**.**.....
........**.*.*....
........****..*...
............**.**.
............**.*.*.
............****..*.
```

```
sol = X(S.getOutput())

figure()
plot(horzcat(sol["x",:]).T)
ylabel("states")

figure()
step(range(N),sol["u",:])
ylabel("controls")
```



6.2 Structured indexing of decision variables

This section highlights an inconvenience with the CasADi approach to work with decision variables, motivates why structured indexing is a solution, and describes formally how that feature works.

6.2.1 Motivation

Many optimisation environments (ACADO, *ACADO Toolkit* 2009–2013 – yalmip, Löfberg 2004 – cvxpy Diamond 2014 – …) do not require the notion of functions. One simply declares all decision variables separately at will, and the expressions composed with them are immediately passed to an optimising routine.

CasADi, on the other hand, requires the creation of functions, for which a list of decision variables is needed explicitly. In fact, for MX, the input to a function must be a single symbolic primitive. All the needed decision variables must be derived from the single parent variable through indexing, slicing or splitting:

```
from optoy import *

x = var()
y = var()

f = (1-x)**2 + \
    100*(y-x**2)**2

g = [x**2+y**2 <=1]

print "cost = ", \
  minimize(f, g)

print "x = ", x.sol
print "y = ", y.sol
```

```
cost = 0.0456748
x =  0.786415
y =  0.617698
```

```
from casadi import *

X = MX.sym("X",2)
x = X[0]; y = X[1]

f = (1-x)**2+100*(y-x**2)**2
g = x**2+y**2
F = MXFunction(nlpIn(x=X),nlpOut(f=f,g=g))
F.init()

nlp = NlpSolver("ipopt",F)
nlp.init()
nlp.setInput(-inf,"lbg")
nlp.setInput(1, "ubg")
nlp.evaluate()

print "cost = ", nlp.getOutput("f")
print "x = " , nlp.getOutput("x")[0]
print "y = " , nlp.getOutput("x")[1]
```

```
cost =  0.0456748
x =  0.786415
y =  0.617698
```

Clearly, the syntax on the right (CasADi) is more verbose and more difficult to learn than the shorter, more classic syntax on the left. The right syntax is however more clean from an implementation point of view (no need for side-effects such as global variable counters), and more powerful since the notion of embeddable functions allows the user to construct large-scale problems more efficiently.

This section documents a feature added to CasADi that relieves users from thinking about indexing, and avoids littering code with "magic" integers.

It should be noted that it is fairly trivial to make a package on top of CasADi that offers the shorter syntax. In fact, `optoy` from the left code listing above is such package (Gillis 2013a), coded in less than 300 lines, and using the feature of the present section under the hood. The related `doptoy` package provides compact syntax for optimal control problems.

6.2.2 Mechanism

Broadly speaking, the structured indexing feature is about defining the structure of variables with a formal grammar, and associating this structure with CasADi vectors and matrices such that indexing becomes structure-aware.

In general, a structure consists of an ordered mapping between labels and entries. Each entry can be an endpoint (having a shape or sparsity), or can be another structure. In addition, each entry can be repeated a number of times, and such repetitions can be repeated themselves, and so on, *ad infinitum*.

To make the discussion more concrete, consider a structure for a direct-collocation problem with 3 states, one control input, and some matrix parameter. A horizon of $N = 4$ and a low order collocation scheme ($d = 2$) is used.
Endpoint entries that are scalar can simply be defined by means of a string:

```
states = struct([ "p", "q", "c" ])
```

More complex entries are defined with the `entry` keyword:

```
s = struct([ entry("x",repeat=[N+1,d],struct=states),
             entry("u",repeat=N),
             entry("p",shape=(2,2)) ])
```

The flattened version of this structure has size:

```
print s.shape, (N+1)*d*3+N+2*2
```

```
(38, 1) 38
```

The structure defines how the hierarchical layout is mapped onto the linear layout of the flattened equivalent. For example, the flat/linear index 33 is associated with the control input at $k = 3$:

```
print s.getCanonicalIndex(33)
print s.i["u",3,0]
```

```
('u', 3, 0)
33
```

Objects that are associated with a structure can be indexed with a so-called "powerindex". This is a tuple of indexing objects that is used to descend the hierarchy of the structure, nibbling away from the start of the tuple as the algorithm traverses downwards.

There are three contexts that can be encountered while descending the structure definition by means of a powerindex:

- The dictionary context. This is always the first context to be encountered in the hierarchy. In this context, the currently active index is one of:
  - A label (`String` class): return an entry associated with the label
  - A list of labels: return a list of multiple entries
  - The empty dictionary (`{}`): returns the mapping as a dictionary
  - The ellipsis (`...`): returns a list of all entries

  If the powerindex is exhausted in this context, returns the flattened version of the remaining structure.
- The list context, encountered when traversing down the repeats of an entry. In this context, the currently active index is one of:
  - An integer: returns the contents of the list at the requested position
  - A slice (e.g. `:`, `2:`, `[0,3]`): returns a list with the elements requested by the slice

  If the powerindex is exhausted in this context, a (nested) list is returned for all remaining repeats.

- The endpoint context. This context is reached when the hierarchy is fully traversed, and there are no more repeats, or structures. The remaining indices in the powerindex are passed on to the endpoint object, which can be any of SX, MX, or a numerical array (IMatrix, DMatrix).

In all three contexts, an operation can be used as active index. It is ignored when traversing down the hierarchy, but is called on the return value of what comes after it.

To continue with our example, associate a DMatrix with our structure:

```
w = DMatrix(range(38)); W = s(w)
```

In a first example, the powerindex `("p",:,0)` is applied:

Context	Act.	Comments
Dict., keys: [”x”,”u”,”p”]	”p”	select the matrix parameter
Endpoint: 2-by-2 DMatrix	:	(:,0) is passed on to the endpoint object; it will select the first column

Both getting and setting can be used with a powerindex:

```
print W["p",:,0]
W["p",:,0] = vertcat([-1,-2])
```

```
[34, 35]
```

As second example, consider the powerindex `("x",:,0,"p")`:

Context	Act.	Comments
Dict., keys: [”x”,”u”,”p”]	”x”	select states
List, length: $N+1$	:	return a list of whatever is found by traversing deeper
List, length: d	0	select the first point of the collocation points
Dict., keys: [”p”,”q”,”c”]	”p”	select the ”p” state

```
print W["x",:,0,"p"]
W["x",:,0,"p"] = 0
```

```
[DMatrix(0), DMatrix(6), DMatrix(12), DMatrix(18), DMatrix(24)]
```

As third and last example, consider the powerindex `("x",[0,-1],horzcat)`:

Context	Act.	Comments
Dict., keys: [”x”,”u”,”p”]	”x”	select states
List, length: $N+1$	[0,-1]	return a list with first and last entry
List, length: d	horzcat	Do a horizontal concatenation of whatever is returned below
List, length: d	Exhausted	a list is returned
Dict., keys: [”p”,”q”,”c”]	Exhausted	return a vector (column matrix) of p, q and r.

```
print W["x",[0,-1],horzcat]
W["x",[0,-1],horzcat] = blockcat([[77,1],[77,2],[77,3]])
```

```
[DMatrix(
[[0, 3],
 [1, 4],
 [2, 5]]), DMatrix(
[[0, 27],
 [25, 28],
 [26, 29]])]
```

The following output shows the state of the numerical matrix `w`, before and after the assignments of the above examples.

```
****canonical index***** || *@ start** || **ex 1** || **ex 2** || **ex 3**
            [x,0,0,p,0] || 0         || 0       || 0       || 77
            [x,0,0,q,0] || 1         || 1       || 1       || 77
            [x,0,0,c,0] || 2         || 2       || 2       || 77
            [x,0,1,p,0] || 3         || 3       || 3       || 1
            [x,0,1,q,0] || 4         || 4       || 4       || 2
            [x,0,1,c,0] || 5         || 5       || 5       || 3
            [x,1,0,p,0] || 6         || 6       || 0       || 0
            [x,1,0,q,0] || 7         || 7       || 7       || 7
            [x,1,0,c,0] || 8         || 8       || 8       || 8
            [x,1,1,p,0] || 9         || 9       || 9       || 9
            [x,1,1,q,0] || 10        || 10      || 10      || 10
            [x,1,1,c,0] || 11        || 11      || 11      || 11
            [x,2,0,p,0] || 12        || 12      || 0       || 0
            [x,2,0,q,0] || 13        || 13      || 13      || 13
            [x,2,0,c,0] || 14        || 14      || 14      || 14
            [x,2,1,p,0] || 15        || 15      || 15      || 15
            [x,2,1,q,0] || 16        || 16      || 16      || 16
            [x,2,1,c,0] || 17        || 17      || 17      || 17
            [x,3,0,p,0] || 18        || 18      || 0       || 0
            [x,3,0,q,0] || 19        || 19      || 19      || 19
            [x,3,0,c,0] || 20        || 20      || 20      || 20
            [x,3,1,p,0] || 21        || 21      || 21      || 21
            [x,3,1,q,0] || 22        || 22      || 22      || 22
            [x,3,1,c,0] || 23        || 23      || 23      || 23
            [x,4,0,p,0] || 24        || 24      || 0       || 77
            [x,4,0,q,0] || 25        || 25      || 25      || 77
```

```
[x,4,0,c,0] || 26       || 26       || 26       || 77
[x,4,1,p,0] || 27       || 27       || 27       || 1
[x,4,1,q,0] || 28       || 28       || 28       || 2
[x,4,1,c,0] || 29       || 29       || 29       || 3
   [u,0,0] || 30        || 30       || 30       || 30
   [u,1,0] || 31        || 31       || 31       || 31
   [u,2,0] || 32        || 32       || 32       || 32
   [u,3,0] || 33        || 33       || 33       || 33
     [p,0] || 34        || -1       || -1       || -1
     [p,1] || 35        || -2       || -2       || -2
     [p,2] || 36        || 36       || 36       || 36
     [p,3] || 37        || 37       || 37       || 37
```

We highlight one extra feature that is important for OCP problems, i.e. the ability to interleave variables. The decision variable structure used in this example has all states, followed by all controls. An NLP constructed using this order would not have a constraint Jacobian with a desirable block structure. To obtain that, one must order the decision variables to be: state and controls at first control interval, state and controls at second control interval, …

This can be done by adding an extra grouping bracket in the structure definition:

```
s = struct([
            (
               entry("x",repeat=[N+1,d],struct=states),
               entry("u",repeat=N)
            ),
            entry("p",shape=(2,2)) ])
```

Note that METIS, a fill-in reducing matrix ordering package (Karypis 2009), can be used together with IPOPT to obtain desirable structure regardless of variable order.

While this section outlines the key idea of structures, there are many more features available. For an extensive treatment of all available features, see the online tutorial (Gillis 2013b).

6.3 Lyapunov solvers in CasADi

This section treats Lyapunov solvers in CasADi. Implementation details are given in Subsection 6.3.1, followed by a worked out concrete example in 6.3.2.

6.3.1 Implementation details

Multiple right-hand sides As hinted at in the tutorial of Section 6.1, CasADi composes Jacobians by making calls to a derivative function. In practice, the derivative function can be requested to provide up to 64 forward or adjoint sensitivities at once.

If the prototype for the Lyapunov solver were used verbatim in the implementation of the derivative, there would be a missed opportunity to share workload among the 64 requested sensitivities:

Input: $A, Q, \dot{A}_0, \dot{Q}_0, \dot{A}_1, \dot{Q}_1, \ldots, \dot{A}_{63}, \dot{Q}_{63}$
Output: $P, \dot{P}_0, \dot{P}_1, \ldots, \dot{P}_{63}$
$P \leftarrow \texttt{DLE\_solver}(A, Q)$;
$\dot{P}_0 \leftarrow \texttt{DLE\_solver}(A, \dot{A}_0 P A^\top + A P \dot{A}_0^\top + \dot{Q}_0)$;
$\vdots$
$\dot{P}_{63} \leftarrow \texttt{DLE\_solver}(A, \dot{A}_{63} P A^\top + A P \dot{A}_{63}^\top + \dot{Q}_{63})$;

For this reason, the Lyapunov solvers have a prototype under the hood that handles multiple right hand-sides:

$$P_0, P_1, \ldots, P_{\text{nrhs}-1} = \texttt{DLE\_solver}(A, Q_0, Q_1, \ldots Q_{\text{nrhs}-1})$$

With this prototype, a derivative can be defined that – in the case of the `slicot` solver – re-uses the Schur factorisation of A for all sensitivity directions:

Input: $A, Q, \dot{A}_0, \dot{Q}_0, \dot{A}_1, \dot{Q}_1, \ldots, \dot{A}_{63}, \dot{Q}_{63}$
Output: $P, \dot{P}_0, \dot{P}_1, \ldots, \dot{P}_{63}$
$P \leftarrow \texttt{DLE\_solver}(A, Q)$;
$\dot{P}_0, \ldots, \dot{P}_{63} \leftarrow$
$\texttt{DLE\_solver}(A, \dot{A}_0 P A^\top + A P \dot{A}_0^\top + \dot{Q}_0 + \ldots + \dot{A}_{63} P A^\top + A P \dot{A}_{63}^\top + \dot{Q}_{63})$;

Low-rank stopping criterion In implementing Algorithm 5 of Chapter 3, the low-rank Smith solver, there is an important detail to mention about the computation of the stopping criterion.

The criterion uses $||C_k V C_k^\top||$, where $C_k \in \mathbb{R}^{n \times n_w}$ and $V \in \mathbb{S}^{n_w}$ with $n_w \ll n$. For dense matrices, computing the result under the norm takes n^2 storage space. This storage requirement may be infeasible to meet for large scale LRDLEs. Indeed, large scale considerations led to the inclusion of H in the LRDLE signature, to avoid holding P in memory.

The element-wise ∞-norm was chosen to be able to compute the norm of the product without storing the product result; while producing the numerical values for the nonzeros of the product result, one can simply store the largest such value in absolute terms and discard values that are lower. The algorithm, `norm_inf_mul_nn`, is based on Bank 1993 and its implementation in SciPy (Jones 2001–2014).

Sparsity calculations As mentioned before, CasADi has an initialisation step for functions in which sparsities are computed in order to allocate memory. Consider a `DLE_solver`. Keeping in mind that A may be sparse due to a `lifting` transformation, it is important to properly compute the sparsity of P for efficiency.

This is done by a simple Smith-iteration as in Algorithm 3, in which the inputs/outputs are here understood to be sparsities instead of numerical matrices, and the norm is understood to be a 0-norm (count of nonzeros): the algorithm is used symbolically instead of numerically. For the low-rank case, similar to the above remark, the 0-norm of a product is computed without holding the product in memory.

Symmetry of inputs/outputs Symmetry of Q is enforced by taking a mapping $Q \to (Q + Q^\top)/2$ and likewise for output P. This is needed for consistency in the unit-tests, in which functions are seeded with random matrices to perform numerical testing. Recall that working with Cholesky factors is prohibited as discussed in Section 5.2.3.

6.3.2 A worked out example

We use the setup of Section 4.4.2 to demonstrate the use of an embedded Lyapunov solver.

First, we define the model $\dot{x} = f(x, u, w)$ and prepare for time-scaling:

```
xy = struct(["x","y"])
x  = struct_symSX([entry("p",struct=xy), entry("v",struct=xy)])
```

```
u  = struct_symSX(xy)
w  = struct_symSX(xy)

rhs = struct_SX(x)
rhs["p"] = x["v"]
rhs["v"] = -10*(x["p"]-u) - x["v"]*sqrt(sum_square(x["v"])+1) + w

T = SX.sym("T") # Time-period

f = SXFunction(daeIn(x=x,p=vertcat([T,u,w])),daeOut(ode=T*rhs))
f.init()
```

Path constraints are given by super-ellipsoids and robustified with a covariance (unknown and hence symbolic at this point) and $\gamma = 1$:

```
hyper = [ ( vertcat([1,1]), vertcat([0,0 ]) , 4),
          ( vertcat([0.5,2]), vertcat([1,0.5]) , 4)]

h     = [ sumAll(((x["p"]-p)/s)**n) - 1 for s,p,n in hyper ]

P = SX.sym("P",4,4)
margin = [ sqrt(quad_form(jacobian(i,x).T,P)) for i in h_nom ]

h_robust = SXFunction([x,P],[ n - m for n,m in zip(h, margin) ])
h_robust.init()
```

Next, an integrating block Φ is constructed for a multiple-shooting setup:

```
N = 20; Tref = 4.0
ts = [i*1.0/N for i in range(N)]

Phi = Integrator("rk",f)
Phi.setOption("number_of_finite_elements",2)
Phi.setOption("tf",1.0/N)
Phi.init()
```

Sensitivity functions of the integrating block will later be needed:

```
APhi = Phi.jacobian("x0","xf"); APhi.init()
CPhi = Phi.jacobian("p","xf") ; CPhi.init()
```

The NLP is constructed as usual:

6

```
V = struct_symMX([ entry("X",struct=x,repeat=N+1),
                   entry("U",struct=u,repeat=N), "T"])

g_coupling = []
As = []; Qs = []
for k in range(N):
  pk = vertcat([V["T"],V["U",k],DMatrix.zeros(w.shape)])
  xkp = Phi(x0=V["X",k],p=pk)["xf"]

  g_coupling.append(xkp-V["X",k+1])

  A = APhi(x0=V["X",k],p=pk)["jac"]
  As.append(A)

  C = CPhi(x0=V["X",k],p=pk)["jac"][:,-2:]
  Qs.append(mul(C,C.T)*50)


dple = DpleSolver("slicot",[i.sparsity() for i in As],
                           [i.sparsity() for i in Qs])
dple.setOption("linear_solver","csparse")
dple.init()

Ps = horzsplit(dple(a=horzcat(As),v=horzcat(Qs))["p"],x.shape[0])

g_obstacle = [ h_robust([V["X",k],Ps[k]]) for k in range(N) ]

g = struct_MX([
    entry("periodic", expr=V["X",0]-V["X",-1]),
    entry("coupling", expr=g_coupling),
    entry("obstacle", expr=g_obstacle)
  ])

F = V["T"] + 1e-2*sumAll(vertcat(V["U"])**2)/2/N

nlp = MXFunction(nlpIn(x=V),nlpOut(f=F,g=g))
```

Finally, an IPOPT solver is constructed and its bounds and initial values populated:

```
solver = NlpSolver("ipopt",nlp)
solver.setOption("hessian_approximation","limited-memory")
solver.init()
```

```
V0 = V(0.0)

V0["T"] = Tref

for k in range(N+1):
  V0["X",k,"p","x"] = 3*sin(2*pi*k/N)
  V0["X",k,"p","y"] = 3*cos(2*pi*k/N)

solver.setInput(V0,"x0")

lbg = g(-inf);  ubg = g(inf)
lbg["coupling"] = ubg["coupling"] = 0
lbg["periodic"] = ubg["periodic"] = 0
lbg["obstacle"] = 0

solver.setInput(lbg,"lbg")
solver.setInput(ubg,"ubg")

solver.evaluate()
```

To conclude, we plot the covariance ellipsoid on the solution trajectory:

```
sol = V(solver.getOutput())

plot(sol["X",:,"p","x"],sol["X",:,"p","y"])

Pf = MXFunction(nlpIn(x=V),Ps)
Pf.init()

Pf.setInput(sol)
Pf.evaluate()

circle = array([[sin(x),cos(x)] for x in linspace(0,2*pi,100)]).T

for k in range(N):
  w,v = numpy.linalg.eig(Pf.getOutput(k)[:2,:2])
  W   = mul(v,diag(sqrt(w)))
  e = mul(W,circle)
  plot(e[0,:].T + sol["X",k,"p","x"],
       e[1,:].T + sol["X",k,"p","y"],'r')
```

6

6.4 Hierarchical seeding for efficient sparsity pattern recovery

This section is based on an extended abstract contribution in Gillis 2014a.

Following up on Section 2.2.2, a colouring of the *column intersection graph* of the sparsity pattern of $J = \frac{\partial f}{\partial x}$ provides a small set of seeds usable to obtain the full Jacobian. We denote such colouring as $col(J)$ and use an existing distance-2 unidirectional algorithm (Gebremedhin 2005).

The potentially dramatic speed-up requires first the sparsity pattern to be obtained. We will assume that we can derive the following bitvector-valued dependency functions (Giering 2006) from the original algorithm f:

$$\begin{array}{lll} \text{Forward dependency} & \text{dep}_f^{\text{fwd}}(d^{\text{fwd}}) \in \mathbb{B}^m, & d^{\text{fwd}} \in \mathbb{B}^n \\ \text{Adjoint/reverse dependency} & \text{dep}_f^{\text{adj}}(d^{\text{adj}}) \in \mathbb{B}^n, & d^{\text{adj}} \in \mathbb{B}^m, \end{array}$$

with $\mathbb{B}$ the Boolean set $\{0, 1\}$. A zero in the dependency function output means that any seed s with sparsity as in the input d, when supplied to the corresponding sensitivity function, would result in a zero sensitivity output in that same location.

A straightforward technique to recover the full sparsity pattern is to seed the dependency functions with slices of a unit matrix. The run-time τ of the dependency functions is typically orders of magnitude smaller than the run-time $\mathcal{T}$ of the original function f. However, for large sparse matrices, the sparsity calculation run-time $\tau \min(n, m)$ could dominate the calculation of the Jacobian. In this work, we propose a hierarchical bitvector-based technique to recover the sparsity pattern faster for highly sparse cases, as would be the case in e.g. multiple-shooting based optimal control problem transcriptions.

The colouring of a sparse Jacobian allows one to recover more information from a single *sensitivity* sweep. A crucial observation is that it can do exactly the same for *dependency* sweeps.

The proposed algorithm starts with obtaining the sparsity pattern in a coarse resolution, performing a colouring of this coarse resolution, and hence potentially reducing the number of fine-grained dependency sweeps needed to obtain a fine-grained image of the sparsity. The algorithm (Algorithm 6) performs this refinement in a recursive way until the full sparsity is recovered. A schematic depiction of this process is given in Figure 6.1.

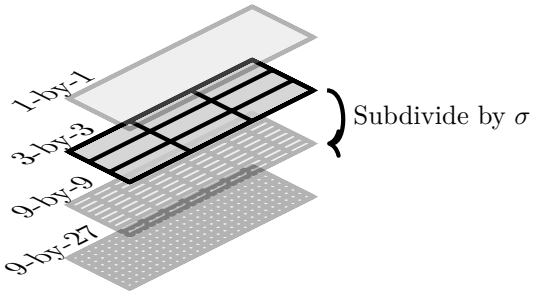


Figure 6.1: Schematic depiction of a hierarchy of sparsities, using a concrete example with $n = 27$ and $m = 9$ and $\sigma = 4$.

Let us walk through Algorithm 6 using the example of Figure 6.1. For these dimensions, there exist four levels of the sparsity hierarchy. Suppose we have sparsity:

$$r = \begin{pmatrix} \cdot & \star & \star \\ \star & \cdot & \star \\ \star & \cdot & \star \end{pmatrix},$$

at the start of the second of the necessary three iterations of the main loop of line 1. In this representation, dots are structural zeros and stars are nonzeros. It is clear that the forward mode gives a superior colouring in line 2-4:

$$\operatorname{col}(r) = \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\} \qquad \text{(fwd)}$$

$$\operatorname{col}(r^\top) = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\} \qquad \text{(adj)}$$

At this point, we can construct the block-sparsity matrix S of line 6 that will eventually become the start of the next main-loop iteration:

6

Input : $\sigma \in \mathbb{N}, \sigma > 1$ subdivision factor
Input : Dimensions n and m of Jacobian
Init : $(N, M) \leftarrow (n, m)$; $r \leftarrow [1]$; `/* Initialise with a scalar */`
1 **while** $N > 1$ *and* $M > 1$ **do**
2 fwd $\leftarrow$ col(r); adj $\leftarrow$ col$(r^\top)$; `/* Colouring of the coarse pattern */`
3 **if** *adj is cheaper* **then** seed $\leftarrow$ adj ; $(N, M, n, m, r) \leftarrow (M, N, m, n, r^\top)$; mode $\leftarrow$ 'adj';
4 **else** seed $\leftarrow$ fwd; mode $\leftarrow$ 'fwd' ;
5 $(\nu, \mu) \leftarrow$ dimensions of r; $(N, M) \leftarrow (\lceil N/\sigma \rceil, \lceil M/\sigma \rceil)$;
6 $S \leftarrow$ block matrix with ν-by-μ empty cells of shape $n/(N\nu)$-by-$m/(M\mu)$;
7 **foreach** $s \in$ *seed* **do**
8 $d \leftarrow$ block_dep $\left(\text{mode}; s \otimes I_{m/(M\mu)} \otimes v^M\right)$; `/* Block sparsity seeding */`
9 $d \leftarrow (I_{n/N} \otimes h^N)d$; `/* Block sparsity aggregate */`
10 **foreach** j *in nonzero locations of* s **do**
11 **foreach** i *in nonzero locations of column* j *of* r **do**
12 $S_{i,j} \leftarrow$ rows $ni/(N\nu)$ to $n(i+1)/(N\nu)$ of d; `/* Store result */`
13 **end**
14 **end**
15 **end**
16 **if** *mode* $=$ *'adj'* **then** $S \leftarrow S^\top$; $(N, M, n, m) \leftarrow (M, N, m, n)$;
17 $r \leftarrow S$;
18 **end**
Output : Jacobian sparsity r

Algorithm 6: Hierarchical sparsity recovery algorithm. In this code, v^n is a column vector of dimension n with all entries 1, and h^n its transpose. block_dep splits up its bitmatrix argument into columns, feeds these to $\text{dep}_f^{\text{fwd}}$ or $\text{dep}_f^{\text{adj}}$ depending on the mode, and lumps the results back together to form a new bitmatrix.

$$S = \begin{pmatrix} \cdot & \cdot & \cdot & ? & ? & ? & ? & ? \\ \cdot & \cdot & \cdot & ? & ? & ? & ? & ? \\ \cdot & \cdot & \cdot & ? & ? & ? & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \\ ? & ? & ? & \cdot & \cdot & \cdot & ? & ? \end{pmatrix}$$

Consider now the loop over all colours in line 7 with seed = col(r). For the first color, the following block sparsity seed is constructed to be used as argument

for the block-dependency function in line 8:

$$\begin{bmatrix}1\\1\\0\end{bmatrix} \otimes \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} \otimes \begin{bmatrix}1\\1\\1\end{bmatrix} = \left[\begin{array}{ccc}1&0&0\\1&0&0\\1&0&0\\0&1&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\\0&0&1\\\hline 1&0&0\\1&0&0\\1&0&0\\0&1&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\\0&0&1\\\hline 0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\end{array}\right],$$

Suppose that the block-dependency function evaluation results in:

$$d = \left[\begin{array}{ccc}0&0&0\\0&1&0\\1&0&0\\\hline 0&1&0\\1&0&0\\0&0&0\\\hline 0&0&0\\1&1&1\\0&0&1\end{array}\right]$$

The result of the block-dependency is aggregated in line 9. In our present case, the aggregation is a non-operation since we have already reached the final granularity in terms of rows at this point.

In the loop of line 10, the non-zeros of d are assigned to their locations in S:

$$S = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & ? & ? & ? \\ \cdot & \cdot & \cdot & \cdot & \star & \cdot & ? & ? & ? \\ \cdot & \cdot & \cdot & \star & \cdot & \cdot & ? & ? & ? \\ \cdot & \star & \cdot & \cdot & \cdot & \cdot & ? & ? & ? \\ \star & \cdot & \cdot & \cdot & \cdot & \cdot & ? & ? & ? \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & ? & ? & ? \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & ? & ? & ? \\ \star & \star & \star & \cdot & \cdot & \cdot & ? & ? & ? \\ \cdot & \cdot & \star & \cdot & \cdot & \cdot & ? & ? & ? \end{pmatrix}$$

This is repeated for the second colour and the next main-loop iteration is started with $r = S$.

Figure 6.2 shows the steps towards detecting some other example structures. The hierarchical sparsity algorithm takes $\log_\sigma(n)$ steps with intermediate sparsity patterns r of increasing granularity to fully resolve the sparsity of an n-by-n matrix. The examples have $n = 27, \sigma = 3$ and hence the number of steps is $\log_3(27) = 3$. Each intermediate sparsity r is visualised from bottom to top, alongside its forward colouring. The total number of dependency sweeps needed to resolve the structure equals σ times the sum of the lengths of the colouring sets.

As illustrated in Figure 6.2a, the sparsity of an n-by-n block-diagonal matrix with σ-by-σ blocks leads to intermediate sparsity patterns with an identity matrix structure, which has a trivial monochrome colouring. Since each colouring set has length one, the total sum of set lengths is $\log_\sigma(n)$. The total number of dependency sweeps that must be performed to resolve the full block-diagonal structure is hence $\sigma \log_\sigma(n)$ as opposed to n when no hierarchical sparsity strategy is employed.

As illustrated in Figure 6.2b, the intermediate sparsity patterns produced while resolving the structure of band-diagonal matrices of bandwidth σ are tri-diagonal. This implicates that the colouring sets are larger than in the previous block-diagonal case. The total number of required sweeps is $\sigma(3 \log_\sigma(n) - 2)$.

Figure 6.2d illustrates a structure that corresponds to the direct transcription method applied to a time-optimal control problem: a diagonal part corresponding to states x_k, a part above the diagonal corresponding to x_{k+1} and a dense column due to dependency on a common unknown time parameter. The total number of required sweeps is of $O(\log(n))$ as in the previous band-diagonal case.

Last, Figure 6.2c illustrates the operation of the algorithm on a dense matrix. The total number of required sweeps is:

$$\sum_{i=1}^{\log_\sigma(n)} \sigma^i = \frac{\sigma(1 - \sigma^{\log_\sigma(n)})}{1 - \sigma} = \frac{\sigma}{1 - \sigma}(1 - n).$$

In conclusion, the asymptotic run-time of the hierarchical sparsity algorithm is a factor $\sigma/(\sigma - 1)$ worse than the straightforward approach for a fully dense Jacobian (i.e. worst-case). However, for structures that are highly sparse (block-diagonal, banded, optimal control structure), the run-time improves in complexity from $O(n)$ to $O(\log(n))$.

For ease of presentation, the proposed algorithm is restricted for n and m integer powers of σ. The extension for general dimensions, together with a

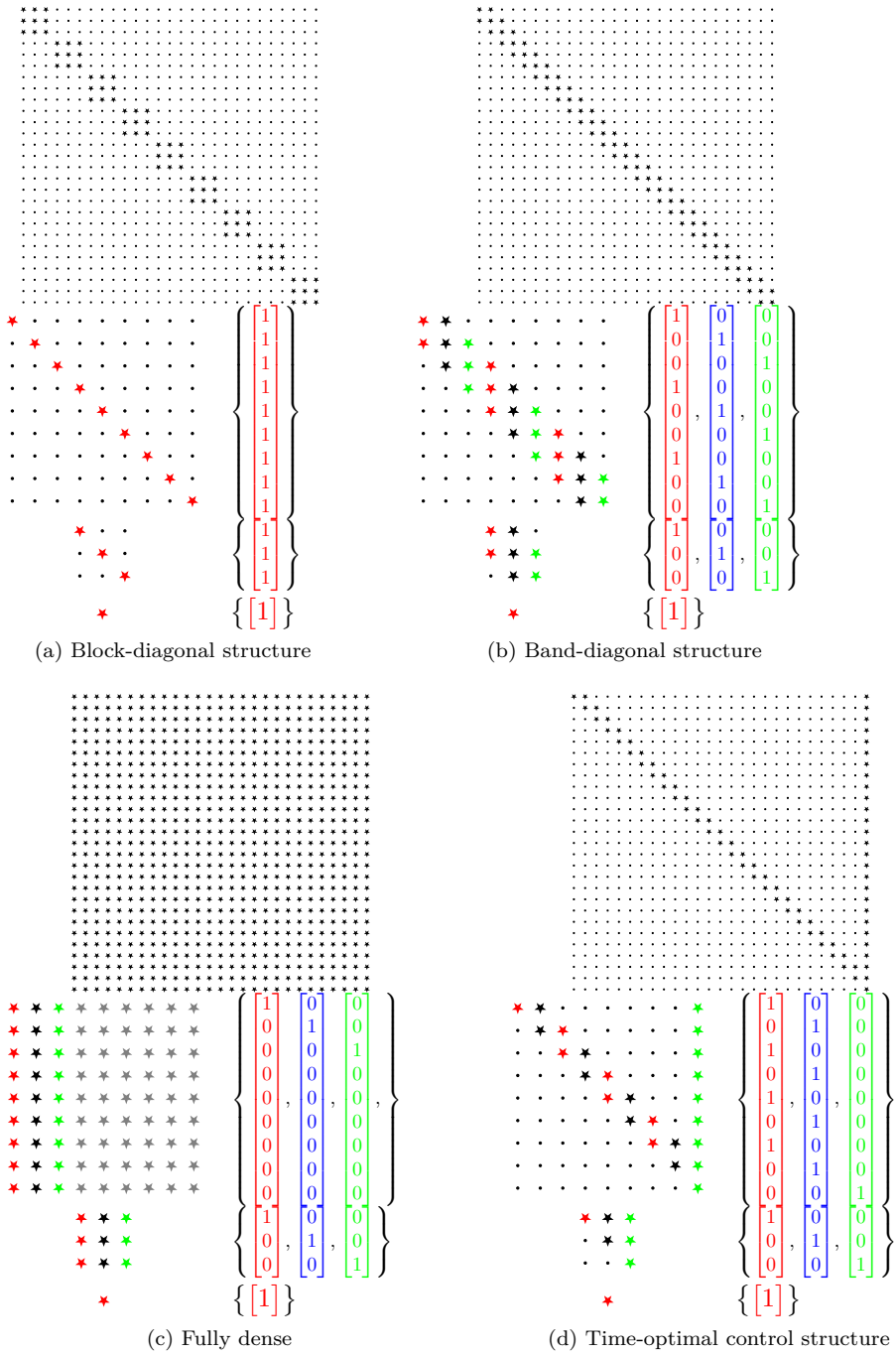
variant for star-colouring for symmetric Jacobians, was implemented in the `CasADi` framework. In that framework, 64 dependency sweeps are evaluated concurrently and hence a subdivision factor of $\sigma = 64$ was chosen.

The following table lists run-time results for n-by-n block-diagonal matrices with block size 4-by-4 and shows a clear benefit for the proposed algorithm in practice for large-scale systems:

	Straightforward approach	Proposed algorithm	
$n = 256$	0.11ms	0.6ms	0.9ms
$n = 16384$	328ms	84.0s	1.02s.

There is one particular type of sparse structure that will not benefit from the implemented hierarchical seeding algorithm: a Jacobian that is mostly banded but possess both a few extra dense rows and dense columns. Such structure may arise in time-optimal formulations with path constraints that depend on all states such as in Figure 8.11 of Section 8.4 of the quadcopter chapter.

In principle, it is easy to extend the algorithm and its implementation with bidirectional colouring facility in order to obtain $O(\log(n))$ complexity for this type of structure as well.



(a) Block-diagonal structure

(b) Band-diagonal structure

(c) Fully dense

(d) Time-optimal control structure

Figure 6.2: The hierarchical seeding algorithm applied to some selected sparsity patterns with $\sigma = 3$.

Conclusion

6

In this chapter, CasADi was presented – by means of a tutorial – as a generic tool to quickly develop optimal control problem formulations with state-of-the-art performance. It contrasts with existing AD tools such as Griewank 1999 in that its notion of speed-optimised scalar-valued graphs and memory-optimised matrix-valued graphs that can be embedded as functions allows the user to make a speed-memory trade-off, of particular importance for large-scale optimal control applications. It also provides easily accessible interfaces to state-of-the-art numerical codes such as NLP solvers and integrators.

CasADi contrasts with *ACADO Toolkit* 2009–2013 in that it is tailored for large-scale applications and offers a flexible do-it-yourself approach to optimal control rather than the black-box approach with a prescribed interface in ACADO.

CasADi is for the major part the work of Andersson 2013, but several ideas were contributed during the course of PhD research, of which this chapter highlights a few.

The first discussed contribution, structured indexing, reduces the amount of work needed to develop formulations with CasADi: it brings down the burden of decision variable management of CasADi more to the level of popular optimisation environments (*ACADO Toolkit* 2009–2013; Diamond 2014; Löfberg 2004) without hindering the notion of embeddable functions that distinguishes CasADi from its peers.

Another contribution, presented at CSC (Gillis 2014a), is a reduction in computational complexity of Jacobian sparsity detection for large-scale optimisation problems, such as may arise with robustification with the Lyapunov framework when the Lyapunov matrices are retained as decision variables.

For robust OCP with Lyapunov elimination, a complete tutorial example was worked out to demonstrate the use of the embedded Lyapunov solvers of last chapter.

Chapter 7

Steady-state analysis and control of towed aeroplanes

This chapter reports on steady-state analysis for aeroplanes that are towed around on a carousel.

The chapter starts with an introduction into *airborne wind energy* (AWE) in Section 7.1 and proceeds by highlighting how an experimental setup is modelled (Section 7.2). Contributions follow in the form of a rigorous steady-state analysis for varying tether lengths and presentation of software tools and results (Section 7.3), in the form of safety-optimisation of setpoints and controllers in Section 7.4 by means of the Lyapunov framework, and by a detailed exploration of formulations (Section 7.5).

7.1 Airborne wind energy and the Highwind carousel setup

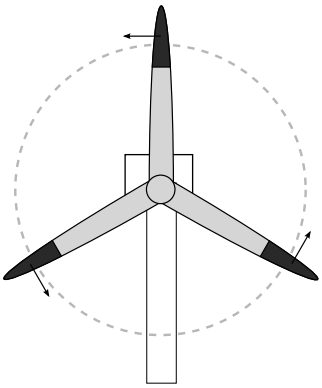
The perennial fusion reaction at the Sun's core bathes our solar system in a huge energy flux. This flux sustains gradients in energy content on the surface of planet Earth and these gradients are the ultimate driving forces for all macroscopic inanimate and animate motions in the universe cf. Tuisku 2009. Indeed, the second law of thermodynamics gives a natural direction to all irreversible transport processes across such energy gradients by demanding a net increase in entropy as argued by Carnot 1872. As a grain of sand over time

loses altitude on an eroding mountain top, its potential energy is dissipated as heat: the randomly-wiggling motion of atoms in the grain and surrounding rock is ever so slightly increased as the grain experiences friction on its path. It is very unlikely for the atoms in surrounding rock to coordinate their movement suddenly giving a macroscopic kick on the grain to move it upwards again. When wind exerts dynamic pressure on the grain, the energy gradient changes direction and it might become thermodynamically favourable for the grain to roll upwards, forming dunes.

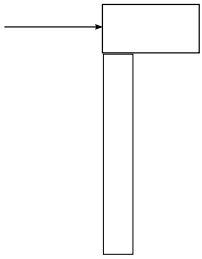
7

Wind is a spontaneous process transporting energy from regions of high atmospheric pressure (caused by solar heating) to regions of lower pressure, with Coriolis forces complicating the motion path. Gradients in kinetic energy can be exploited to harvest power from the wind. For example, birds are aided by such gradients during dynamic soaring (Cone 1964), and virtually all wind energy harvesting contraptions exploit the difference in speed between the Earth's surface and the wind. Any aerodynamic surface that has a different velocity than the uniformly flowing wind is subject to dynamic pressure by the wind, resulting in a net force on a point. If that point is allowed to move in a direction that forms a positive dot product with the force vector, work is delivered by the wind on the surface, and the wind itself undergoes a local change in velocity. A free-floating surface would simply accelerate until it reaches the same velocity as the wind. A force-transmitting connection to the ground is needed for a power harvesting surface to maintain a speed difference and extract power from the wind.

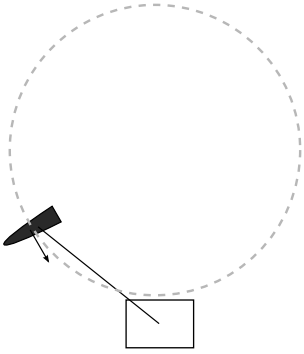
In a classical wind-turbine (depicted in Figure 7.1a), the work is delivered by a force induced on the blades tangential to a circle of rotation. A counterforce is delivered by a generator on the top of the supporting pole, which converts the harvested energy. A necessary side-effect of this design is a horizontal force component acting downwind on the pole (Figure 7.1b). This force delivers no work, but must be structurally supported by the pole and its foundations. Furthermore, a large portion of the useful force is induced by a small outer fast-moving region on the blade. The remainder of the blade can be seen as a necessity to transmit the force to the generator. In the kite-power design, the steel tower with concrete base is replaced by a tether (Figure 7.1c). The horizontally acting force is efficiently counteracted by the tension in the tether (depicted in Figure 7.1d). There are two main ideas for extracting power from the resulting fast-moving tethered wing. The first idea is to build a small on-board propeller and generator to extract work from the tangential force, sending electric energy downwards over the tether. The on-board generator can be made compact because it can revolve at high speeds. A downside is the weight added to the wing, weight that must be compensated by sacrificing part of the useful aerodynamic forces. In the second idea, work is extracted by



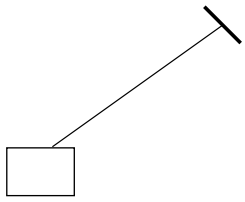
(a) Classical wind-turbine with aerodynamic forces doing work on the blade tips (front view)



(b) Classical wind-turbine with a horizontal force acting on the pole (side view)



(c) Kite-power concept with a wing tethered to a ground station (front view)



(d) Kite-power concept with a wing tethered to a ground station (side view)

Figure 7.1: Comparison of wind-turbines and kite-power concepts.

allowing the tensioned tether to roll out over a drum with generator. Rolling out while tether tension is high, and pulling in when tension is low. Controllable flight surfaces on the wing allow for creating such a change in tension. This idea is referred to as *pumping*.

Airborne wind energy, as defined by Diehl 2013, is a common name for wind

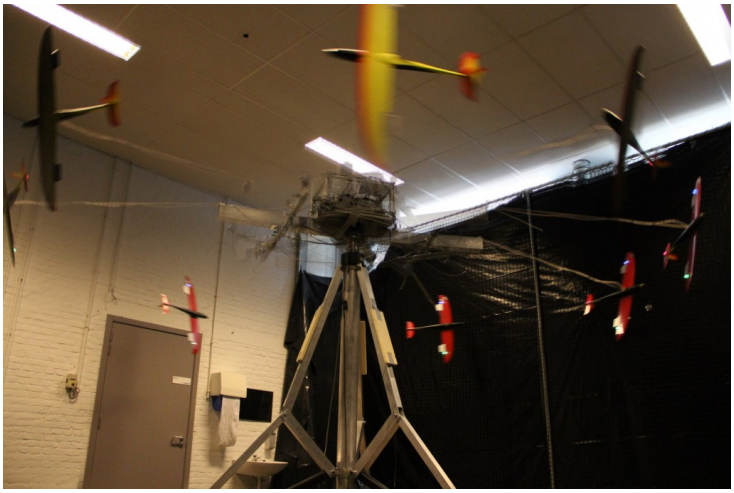


Figure 7.2: Stitched photograph of the Highwind experimental setup.

power technologies that do not require a rigid support structure for operation, among which kite-power is an example. The main benefit of these technologies is their ability to tap into wind streams high above the ground, where wind is generally stronger and more consistent. For kite-power in particular, we are effectively replacing a huge amount of concrete and steel by a tether and advanced control. The pumping-flavour of kite-power is investigated by the Leuven/Freiburg Highwind group. Master thesis students (Geebelen 2010; Ibens 2009) investigated a concept for launching power-harvesting kites autonomously by rotational start-up. In this design, a rotating pole with horizontal arm tows around the wings. Starting from standstill and close to the pole, the wings are progressively sped up and brought further by extending the tether. Figure 7.2 depicts the *carousel*, the experimental setup of the Highwind to test out rotational start-up as described in Geebelen 2012. The wings used in the project are modified RC-aeroplanes. The carousel is equipped with cameras, line-angle sensors, gyroscopes and accelerometers to measure the location and orientation of the aeroplane.

The main goal of the Highwind carousel is not to demonstrate power-generation with kites, but rather to offer an experimental platform to validate advanced control schemes (e.g. MHE and NMPC in Gros 2013b) that might ultimately be used in large-scale kite-power facilities. This chapter deals with steady-state orbits on the carousel.

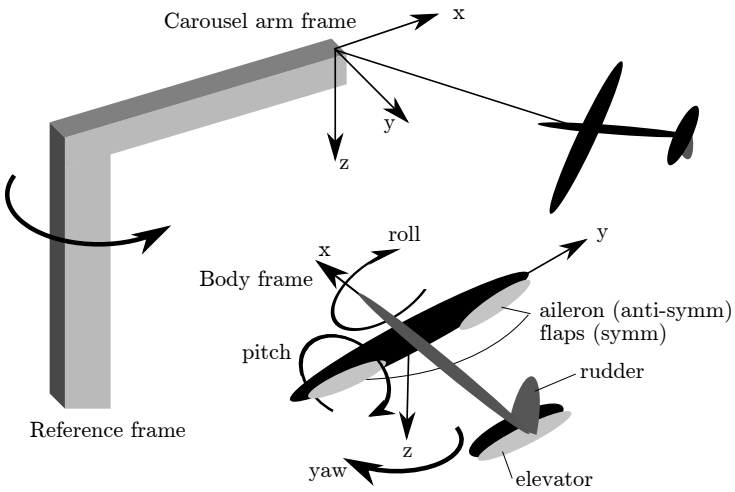


Figure 7.3: Stylised depiction of the carousel and aeroplane.

7.2 Model of the carousel

The carousel model is given as a `CasADi` script encoding a fully implicit DAE, using rigid-body mechanics and linear aerodynamics (Pamadi 2003) and derived with the Lagrange formalism in non-minimal coordinates (Flannery 2005; Gros 2013a) with the coordinates modified to North-East-Down (NED). Full parametrisation of the rotation matrix is employed as proposed in Gros 2012 to avoid singularities, reduce the complexity of the computational graph and make the mapping linear. The remotely rotatable control surfaces, as depicted in Figure 7.3, are the physical means to control the aeroplane. In the given carousel model, the angular positions of these surfaces are not used as inputs, rather their derivatives are. This is a trick to allow regularisation on control activity. For the purpose of steady-state, this is irrelevant and these pure integrator states are therefore removed in this context. Also, perfect speed control of the carousel is assumed, allowing to remove states corresponding to carousel dynamics.

This reduced model can be summarised as a fully implicit DAE with:

$$\begin{cases} g(x, \dot{x}, z, u) = 0 \\ C(x) = 0, \end{cases} \tag{7.1}$$

with:

states $x \in \mathbb{R}^{n=18}$:

`x`, `y`, `z`	aeroplane position in carousel arm frame [m]
`e11`,`e12`,`e13`, `e21`,`e22`,`e23`, `e31`,`e32`,`e33`	components of rotation matrix from carousel frame to aeroplane frame
`dx`, `dy`, `dz`	time derivatives of (`x`, `y`, `z`)
`w_x`, `w_y`, `w_z`	angular rates of aeroplane in its frame [rad/s],

controls $u \in \mathbb{R}^{m=6}$:

`aileron`, `rudder`, `elevator`, `flaps`	angular position of aerodynamic control surfaces [rad]
`r`	tether length [m]
`ddelta`	carousel turn speed [rad/s],

algebraic variables $z \in \mathbb{R}$:

`nu`	tether tension normalised by tether length [N/m],

invariants $C(x) \in \mathbb{R}^{r=8}$:

6 orthonormality constraints on $\mathtt{e}_{ij}$
consistency between (`x`, `y`, `z`) and `r`, and derivative of this,

and we augment this model with:

disturbances $w \in \mathbb{R}^{p=6}$:

`wind_x`, `wind_y`, `wind_z`	wind velocity in the world frame $[m/s]$, $\Sigma = 0.5^2$
`dist_aileron`, `dist_elevator`, `dist_rudder`, `dist_flaps`	biases on control surface positions $[rad]$, $\Sigma = 0.01^2$.

A property of this model is that the state derivatives $\dot{x}$ and algebraic variables z appear linearly in g. This allows us to obtain an explicit form for $\dot{x}$ from which z is eliminated:

$$\begin{bmatrix} \dot{x} \\ z \end{bmatrix} = -\begin{bmatrix} \frac{\partial g}{\partial \dot{x}} & \frac{\partial g}{\partial z} \end{bmatrix}^{-1} g(x, 0, 0, u) = \begin{bmatrix} f(x, u) \\ f_a(x, u) \end{bmatrix}. \tag{7.2}$$

In `CasADi`, this form can be obtained either by introducing a linear solve node or by performing the inversion symbolically. The latter was used here.

Quantity	*Betty* configuration	*Arianne* configuration
Angle of attack [deg]	$[-4.5, 8.5]$	$[-4.0, 8.0]$
Sideslip angle [deg]	$[-9, 9]$	$[-7.0, 7.0]$
Aeroplane position (`z`) [m]	$]-\infty, 1.5]$	$]-\infty, 1.0]$
Carousel motor torque [N·m]	$[-20, 20]$	-
Control surfaces angle [rad]	$[-0.20, 0.20]$	-
Lift coefficient [-]	$[-0.15, 1.5]$	$[-1.5, 1.5]$
Winch motor torque [N·m]	$[-78, 78]$	-
Tether tension [N]	$[0, 600]$	$[0, 100]$
Airspeed. [m/s]	$[10, 65]$	$[10, 50]$

Table 7.1: Table of operational bounds

The model presented here corresponds to the *Betty* configuration (outdoors carousel). The configuration of the indoors carousel is referred to as *Arianne*, lacks `rudder` and `flaps` controls and has different geometric and aerodynamic parameters.

The operational bounds defining our operational set $\mathbb{X}$ are, for the two configurations given in Table 7.1.

7.3 Obtaining steady-states and exploring system linear stability

Consider the continuous-time dynamic system:

$$\dot{x} = f(x, u), \tag{7.3}$$

with states $x \in \mathbb{R}^n$, control inputs $u \in \mathbb{R}^m$ and f continuously differentiable. When $\frac{\partial f}{\partial x}$ is invertible, the condition $f(x, u) = 0$ implicitly defines a locally unique steady-state $x^\star = S(u)$ as a function of control inputs. The mapping S from controls to steady-state solutions has a restricted domain; there might be no steady-state solution corresponding to a given control input. Assuming the region of interest lies in this domain, a simple root-finding Newton iteration scheme on $F(x) = f(x, u)$ can be used to obtain a steady-state:

$$x_{k+1} = x_k - \left[\frac{\partial F}{\partial x}(x_k)\right]^{-1} F(x_k). \tag{7.4}$$

A Newton process to solve an implicit function is an elementary building block in CasADi's matrix valued graphs.

As we carry out a homotopy of control inputs u, starting each numerical process with the solution of a neighbouring control input will keep the process in one domain of attraction, allowing to trace out one particular S.

Performing a similar process for the carousel model of Equation (7.1) on $F(x) = [f(x,u)^\top, C(x)^\top]^\top$ would fail. Both $f(x,u)$ and $C(x)$ describe the same dynamics. The resulting $\frac{\partial F}{\partial x}$ would not be invertible, being of dimension $n \times (n+r)$ and of rank n. There are two techniques to project away the redundant equations according to Sternberg 2012a.

The first technique makes use of the nullspace $Z = \text{null}(J) \in \mathbb{R}^{n\times(n-r)}$ of the Jacobian of invariance constraints $J = \frac{\partial C}{\partial x}(x) \in \mathbb{R}^{r\times n}$:

$$F(x) = \begin{bmatrix} Z^\top f(x,u) \\ C(x) \end{bmatrix} = 0. \tag{7.5}$$

There is functionality in `CasADi` to obtain a null-space in a matrix-valued expression graph. In an NLP context, a readily available alternative is to add Z as decision variables and formulate the definition of null-space as a constraint:

$$\begin{cases} JZ = 0 \\ Z^\top Z = I. \end{cases} \tag{7.6}$$

Note that null-space bases are not unique and a regularisation term on Z might be added to the NLP objective to aid convergence.

The second technique uses projection and results in:

$$F(x) = f(x,u) - J^\dagger C = 0, \tag{7.7}$$

with $J^\dagger = J^\top(JJ^\top)^{-1}$ the Moore-Penrose pseudoinverse of the invariance constraint Jacobian J. This condition simplifies into the steady-state condition $f(x,u) = 0$ if $C = 0$ holds. That the latter should hold becomes obvious by left multiplying Equation (7.7) with J:

$$\begin{aligned} Jf(x,u) &= C \quad \text{since } JJ^\dagger = I, \\ \dot{C} &= C \quad \text{since } \tfrac{\partial C}{\partial x}\tfrac{dx}{dt} = \tfrac{dC}{dt}, \\ 0 &= C \quad \text{Time invariance is implied by the dynamic equations.} \end{aligned}$$

A `Python`-based tool was created to explore the properties of steady-state solutions. It allows the user to sweep over two coordinates u_x and u_y of control space, while keeping the remaining control inputs at a fixed value. For each steady-state point, arbitrary properties can be calculated and displayed as a contour overlay. The process starts from one given steady-state point and spirals

outwards, each numerical process making use of the previous successful solve in closest proximity. Notably, as highlighted in Algorithm 7, Equation (7.5) is used, and for each numerical process, Z is fixed to the null-space as obtained in post-processing of the nearest solution.

Data: Sweeping points $\mathbb{S} = \{(u_x^0, u_y^0), (u_x^1, u_y^1), \ldots, (u_x^N, u_y^N)\}$
Data: A mutable associative data-structure D mapping from elements of $\mathbb{S}$ to arbitrary data
Input: Steady-state solution for (u_x^0, u_y^0) in D
for (u_x, u_y) *in* $\mathbb{S}$ *in order spiralling around* (u_x^0, u_y^0) **do**
  $(x_0, Z) \leftarrow$ Retrieve data from nearest visited point in D ;
  $x^\star \leftarrow$ Solve Equation (7.5) by Newton iteration with initial guess x_0;
  $Z \leftarrow \text{null}(\frac{\partial C}{\partial x}(x^\star))$;
  Store $(x^\star, Z, \text{analysis of } x^\star)$ at location (u_x, u_y) in D ;

Algorithm 7: Efficiently sweeping over steady-state solutions

The tool populates $\mathbb{S}$ with a linearly spaced rectangular created from bounds supplied by the user. After a sweep of Algorithm 7, plots are created from desired steady-state properties in the collected data in structure D. A background process is activated that decimates the mesh and re-runs the algorithm in a loop. This way, the user quickly sees rough plots when selecting regions of control space to explore, and will see gradual refinements appearing when idling. The result is fast visual feedback to action by the user resulting in a responsive user interface. A screenshot of the software is provided in Figure 7.4. Clicking anywhere in the sweep space at the left will reveal more information of the corresponding steady-states: eigenvalues in a polar plot at the right and an animation of eigenmodes through a third-party plotting tool (not depicted).

A notable property that is highlighted with the tool in the following figures is the stability metric:

$$s = \alpha(Z^\top \frac{\partial f}{\partial x} Z), \tag{7.8}$$

with $\alpha(\bullet)$ the spectral abscissa of Chapter 3, and Z projecting away the zero eigenvalues that arise from the system invariants.

In Figure 7.5, we observe that a rectangle of operational feasibility exists. The operational bounds are simply formed by sideslip on the horizontal axis, and angle-of-attack on the vertical axis. It makes intuitive sense to operate the system at a setpoint in the left-upper corner of this rectangle where it is most stable, but not too close to the edge, to be able to operate the system safely. The point $\texttt{aileron} = -0.15, \texttt{elevator} = -0.20$ is used further on.

Our main point of interest lies in the interplay of stability and feasibility as the tether is rolled out quasi-statically. From Figure 7.6, we observe that we should

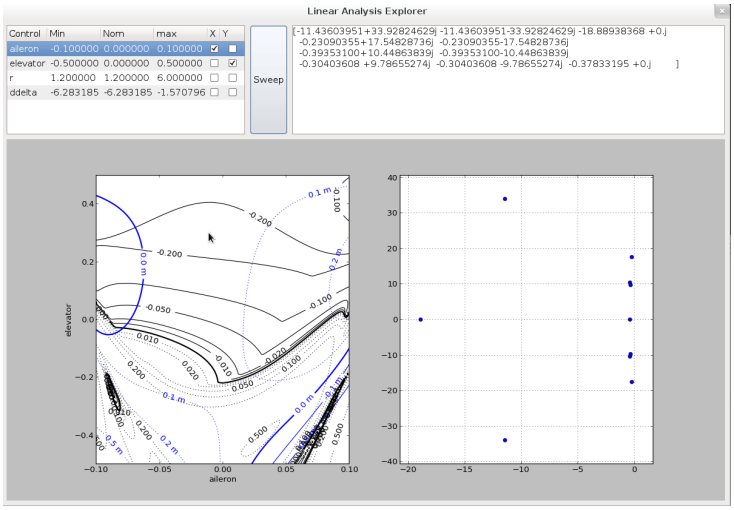


Figure 7.4: Screenshot of the steady-state exploration tool.

increase the `elevator` along with `r` to maintain operation feasibility.

In Figure 7.7a, it appears there is an unlimited region of stability that is impossible to attain with `elevator` alone, given the operational constraints. In Figure 7.7b, we learn that authority in `rudder` can strongly affect stability.

In conclusion, the stability space of the system is rich in features and inspecting slices of it to obtain a safe start-up trajectory is a difficult task.

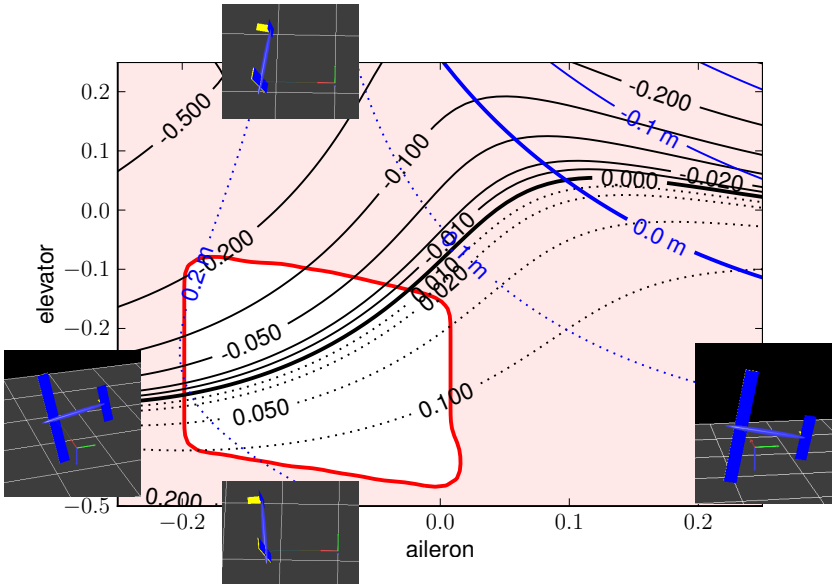
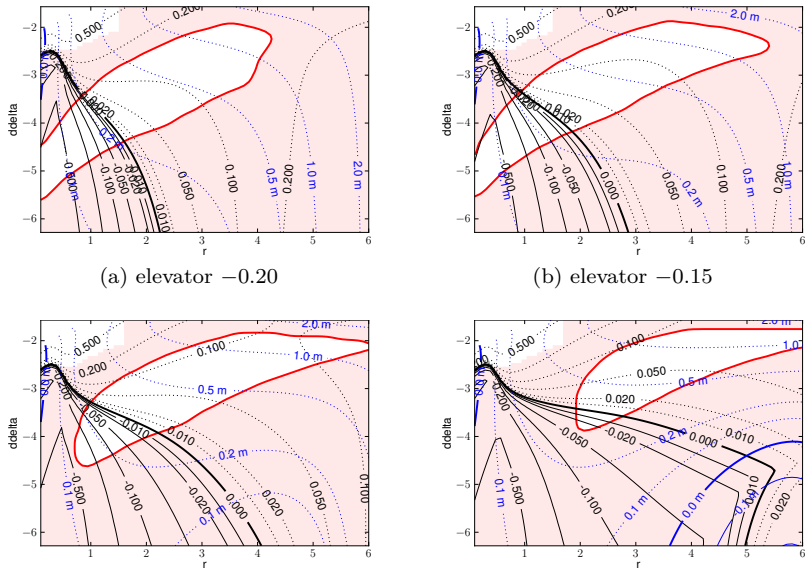


Figure 7.5: Steady-state plot for a sweep along (`aileron`, `elevator`) with `r` = 1m, `ddelta` = −4rad/s, `rudder` = 0, `flaps` = 0. Every point corresponds to a steady-state. Stability s is visible as thin black contours, full for stable and dotted for unstable. Height z is visible as blue contours, full for above carousel arm (negative in North-East-Down frame) and dotted for below. In the shaded region, operational bounds are violated (smoothened for visual cleanliness). Superimposed on the edges of the graph are depictions of aeroplane configurations that hold near these edges.



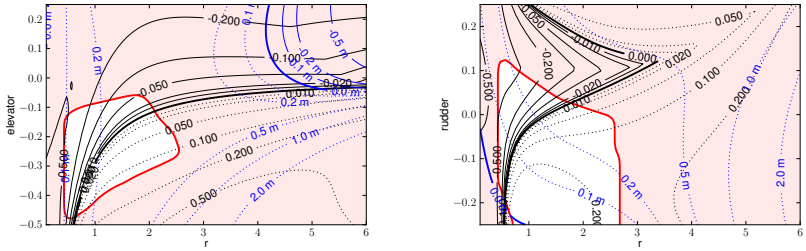
(a) elevator −0.20

(b) elevator −0.15

(c) elevator −0.10

(d) elevator −0.05

Figure 7.6: Steady-state plots for sweeps along (`r`, `ddelta`) with varying `elevator`. From below, the feasibility region is bounded by tether tension constraints. From above, the limit is minimal airspeed and height above the ground. To the right, the angle-of-attack becomes too small (pitching down too much)



(a) Sweeping with elevator

(b) Sweeping with rudder

Figure 7.7: Steady-state plots with increasing `r` on the x-axis. Stable region is at the left hand side.

7.4 Joint design of stochastically safe setpoints and controllers

This section is directly based on a publication at IFAC: Gillis 2014b. It starts from the observation that the carousel amounts to a substantial monetary investment for the Highwind group, and that care must be taken to operate the device. Choosing an unfortunate setpoint for control or having a control scheme or state observer that diverges may result in irreparable damage to the physical system.

As we explored linear stability in the previous section, it became clear that a) we are considering both stability and feasibility to determine safe setpoints, a trade-off that is not easy to make intuitively, and b) we should combine authority in multiple inputs to find a path to bring us up safely to long tether lengths, something which is difficult to do by interpreting two-dimensional plots. Stochastic safety is our answer to the former difficulty, and the methods presented in this section are an elegant – albeit approximate – solution to the latter. First, we give a formal definition to *safety*. Next, we propose a method to find steady-states that are optimally safe and lastly we demonstrate how incorporation of output feedback increases safety.

7.4.1 Marginal and stochastic safety

Neglecting modelling error and external disturbances for a moment, we define the system of Equation (7.3) to have an *asymptotically safe setpoint* $(x^\star, u^\star)$ if it holds that:

1. Control bounds are respected: $u^\star \in \mathbb{U} \subset \mathbf{R}^m$.
2. $x^\star$ is in the interior of the operational set $\mathbb{X}$.
3. The point is a stationary point, i.e. $f(x^\star, u^\star) = 0$ holds.
4. The undisturbed system is asymptotically stable at this point: $\alpha(\frac{\partial f}{\partial x}) < 0$, where α is the spectral abscissa of Chapter 3.

We will further assume that the operational set is defined by means of a collection of scalar operational bounds:

$$\mathbb{X} = \{x | h_i(x) \leq 0\}. \tag{7.9}$$

Furthermore, we will assume that the linearised system dynamics matrix $A = \frac{\partial f}{\partial x}$ is invertible on $\mathbb{X} \times \mathbb{U}$ such that for a given u, the steady-state equation

$f(x, u) = 0$ implicitly defines a locally unique steady-state function $x = S(u)$ with a particular domain. We restrict our analysis to one such S. Also, we assume further that the linearised system is controllable.

It follows from the above conditions that we can always construct an open region in state space around a safe setpoint $(x^\star, u^\star)$ which is invariant under forward propagation of system dynamics and which lies entirely inside the operational set. Next, we will extend the notion of this safe region to a stochastically excited system.

Allowing for disturbances $w \in \mathbb{R}^p$, we consider the system:

$$\dot{x} = f(x, u, w). \tag{7.10}$$

For a given fixed control input $u^\star$ and a stationary stochastic realisation of w, the disturbed system will never reach the true steady-state $x^\star = S(u^\star)$, but will remain in a region close to it if the system is asymptotically stable and the disturbances are sufficiently small. The deviation $\hat{x}$ of the state x from $x^\star$ will then behave as a stochastic process as well.

We define the point $(x^\star, u^\star)$ to be *stochastically safe* with confidence level $0 < \overline{p} \leq 1$ for the disturbed system if it holds that:

1. The point is *asymptotically safe* for the corresponding undisturbed system.
2. For each operational bound $h_i(x) \leq 0$, the probability of the stochastic realisation of $h_i(x)$ satisfying its bound is at least $\overline{p}$.

7.4.2 Formulation

If we assume zero-mean Gaussian white noise with covariance Σ_w as realisation of w, then the covariance of the deviation of states w.r.t. to a safe steady-state point is given in linear approximation as the (continuous) algebraic Lyapunov equation as elaborated on in Houska 2007; Zhou 1996. Denoting this covariance as $\Sigma_{\hat{x}} \equiv P$, we have:

$$\underbrace{AP + PA^\top}_{\text{sink}} + \underbrace{D\Sigma_w D^\top}_{\text{source}} = 0, \tag{7.11}$$

with $A = \frac{\partial f}{\partial x}(x, u, 0)$, $D = \frac{\partial f}{\partial w}(x, u, 0)$ and P a positive-definite symmetric matrix. This equation captures how the uncertainty injected through the source term (disturbances) finds a dynamic equilibrium with the sink term (strictly dissipative system dynamics). The equation is exact if f is linear in x and w.

In linear approximation of $h_i(x)$, we have that its (scalar) covariance is given by:

$$\Sigma_{h_i(x)} = \frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top. \tag{7.12}$$

Under these approximations, we can guarantee that realisations of the perturbed state around the safe setpoint are within the operational set with safety margin $0 < \gamma$, by imposing the following bounds:

$$h_i(x^\star) + \gamma\sqrt{\Sigma_{h_i}(x^\star)} \leq 0, \tag{7.13}$$

with γ bearing a relationship with the confidence level through the cumulative normal distribution function Φ:

$$\Phi(\gamma) = 1 - \overline{p}/2. \tag{7.14}$$

We propose to maximise the safety margin γ simultaneously with the search for a stochastically safe setpoint, yielding the *optimal stochastically safe* setpoint, in an optimal control problem formulation:

$$\underset{x,u,\gamma,P}{\text{minimise}} \quad -\gamma \tag{7.15a}$$

$$\begin{aligned}
\text{subject to} \quad & f(x,u,0) && = 0 && \text{(7.15b)}\\
& u && \in \mathbb{U} && \text{(7.15c)}\\
& h_i + \gamma\sqrt{\frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top} && \leq 0 && \text{(7.15d)}\\
& AP + PA^\top + D\Sigma_w D^\top && = 0. && \text{(7.15e)}
\end{aligned}$$

The idea of using the safety margin from the Lyapunov framework in the objective of an OCP was first mentioned in Houska 2007 and extended on for the multi-objective case in Logist 2011a. No formulations in literature could be found for the present case: steady-state design with a system with invariants.

For an illustrative summary of Method 7.15, we refer to Figure 7.8. Here, we have an illustration of state space for a given control input $u^\star$, with operational bounds $h_i = 0$ depicted in solid lines, defining the operational set. The dot near its center is the stationary point $x^\star = S(u^\star)$. The dotted lines represent the approximate operational bounds, linearised in $x^\star$. The inner ellipsoid is

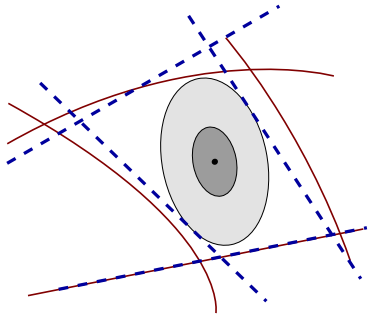


Figure 7.8: State-space, state covariance and bounds

a representation of covariance P, with the eigenvectors defining the principal axes, and eigenvalues the squares of axis lengths. The quantity $\sqrt{\frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top}$ corresponds to the maximum reach of the ellipsoid in the direction towards the linearised operational bound h_i. The outer ellipsoid is a scaled up version with factor γ. Note that two constraints on Equation (7.15d) are active here and none are violated; we can say that the point $(x^\star, u^\star)$ is stochastically safe with safety margin γ. The goal of Method (7.15) is to scout the design space u for the setpoint with the largest safety margin γ. The other decision variables can be eliminated in principle.

The existence of an open-loop stochastically safe setpoint as defined in Method (7.15) is not guaranteed. And when it exists, the corresponding safety margin might be insufficient. To diminish these problems, one can improve stability by embedding a controller into the dynamics. As an example, we propose to introduce simple output feedback control in the system with coefficients as new the decision variables. By adding additional degrees of freedom to the optimisation problem we can only perform better or as good as the original method.

Consider the linearisation of System (7.10) around a candidate steady-state point $(x^\star, u^\star)$:

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + B\hat{u} + Dw \\ \hat{y} &= Y\hat{x} + v, \end{aligned} \tag{7.16}$$

with $B = \frac{\partial f}{\partial u}(x^\star, u^\star, 0)$, $Y = \frac{\partial y}{\partial x}(x^\star)$, $y(x) \in \mathbb{R}^s$ an observation function, and v a stochastic variable corresponding to measurement noise.

In the previous section, we had $\hat{u} \equiv 0$. Here, we feed $\hat{y}$ back to $\hat{u}$ with a linear feedback matrix $K \in \mathbb{R}^{m \times s}$:

$$\hat{u} = K(Y\hat{x} + v). \tag{7.17}$$

Adding this feedback into the dynamics of Method (7.15) leads to a method for jointly designing setpoint and control for optimal stochastic safety:

$$\begin{aligned}
&\underset{x,u,\gamma,P,K}{\text{minimise}} && -\gamma && && \text{(7.18a)}\\
&\text{subject to} && f(x,u,0) && = 0 && \text{(7.18b)}\\
& && q_j + \gamma\sqrt{\frac{\partial q_j}{\partial u} KYPY^\top K^\top \frac{\partial q_j}{\partial u}^\top} && \leq 0 && \text{(7.18c)}\\
& && h_i + \gamma\sqrt{\frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top} && \leq 0 && \text{(7.18d)}\\
& && (A+BKY)P && && \\
& && \quad + P(A+BKY)^\top && = 0 && \text{(7.18e)}\\
& && \quad + D\Sigma_w D^\top + BK\Sigma_v K^\top B^\top && &&
\end{aligned}$$

where we have parametrised set $\mathbb{U}$ by functions q_j as we did earlier for $\mathbb{X}$. Note that the introduction of output feedback on the system linearised around steady-state has direct influence on the state covariance P only, and on the safety margin as a direct consequence. The potential of output feedback to alter also the steady-state point – the *joint* design aspect – arises indirectly through the optimiser.

7.4.3 Implementation

Elimination of covariance

It was observed in numerical experiments that, when started with a known asymptotically safe point as initial guess, both an implementation with an interior-point method (`IPOPT`, Wächter 2006a) and a sequential quadratic program (SQP) method (`WORHP`, Büskens 2012), showed better global convergence behaviour when P was eliminated as decision variable. Details of these equations follow in Section 7.5.

Indeed, Equation (7.15e) is linear in P and hence one can write the vector of its entries $\mathcal{P} = \text{vec}^\top(P)$ as the solution of a linear system $\mathcal{A}\mathcal{P} = \mathcal{B}$ using

$$\begin{aligned} \mathcal{A} = &\quad A \otimes I_n + I_n \otimes A \\ \mathcal{B} = &\quad \text{vec}^\top\left(D\Sigma_w D^\top\right), \end{aligned} \tag{7.19}$$

with $\otimes$ the Kronecker-product. The size of this linear system can be reduced from n^2 to $n(n+1)/2$ by taking symmetry of P into account. We omit a closed form expression for this reduced system here, but note that it is easily obtained with `CasADi`. We further note that there exist dedicated algorithms with better complexity to solve the Lyapunov equation (Benner 2013) cf. Section 5.2.3.

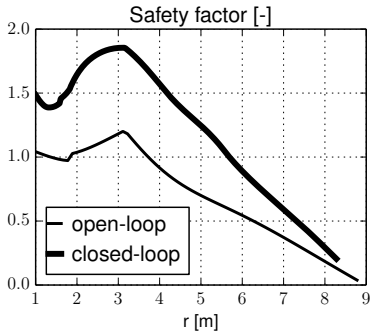
7.4.4 Results

Before, manual inspection of slices of the asymptotically safe set led us to believe that we should not expect asymptotically safe points further than at $\mathtt{r} = 4\text{m}$.

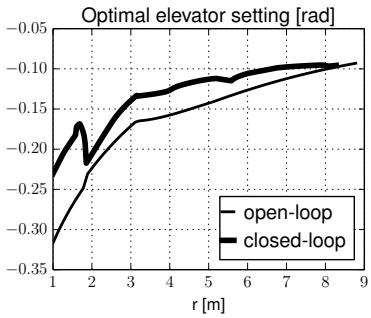
In this section, we sweep over `r` while optimising for stochastic safety over the remaining controls. In Methods (7.15) and (7.18), we simply restrict all `r` coordinates in U to a single value and use hot-starting to efficiently carry out the sweep. For the closed-loop methods, we choose $(\mathtt{y}, \mathtt{z})$ as observations and $(\mathtt{aileron}, \mathtt{rudder})$ as feedback controls. Measurement noise Σ_v was discarded in this simulation.

Figure 7.9 presents the results for this application. Figure 7.9a shows we found asymptotically safe points well beyond the tether lengths we expected (up to 8m). Using linear output feedback, we increase the safety margin but do not get significantly farther out. Since the problem is non-convex, it might be that solutions are missed that allow for longer tether lengths. For each tether length, we have in effect found the largest ellipsoid scaled up from state covariance with factor γ that still fits in the linearised operational set, as depicted in Figure 7.8. In Figure 7.9b, `elevator` is steadily increased; a correct prediction from the previous section. In Figure 7.9d, the controller becomes more aggressive with increasing `r`.

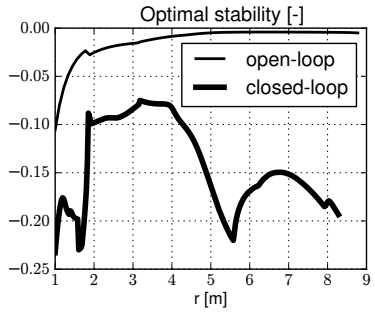
It is remarkable that, using the linear feedback, the maximum tether length cannot be extended with respect to the open-loop case. In Figure 7.10a, it can be observed that the Lyapunov matrix P increases at the end, despite the increasingly dissipative dynamics as in Figure 7.9c. This increase is explained by the steep increase of input noise $D\Sigma_w D^\top$. The increase of P is gentle, and cannot explain why the safety margin drops to zero as in Figure 7.9a. Another explanation might come from a runaway in the y-coordinate, which would hint



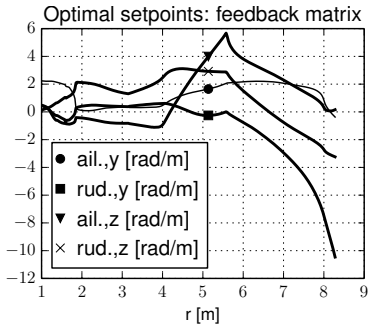
(a) Plot of the safety margin γ.



(b) Plot of the optimal elevator setting.



(c) Plot of stability s of Equation (7.8). The existence of γ implies stability.

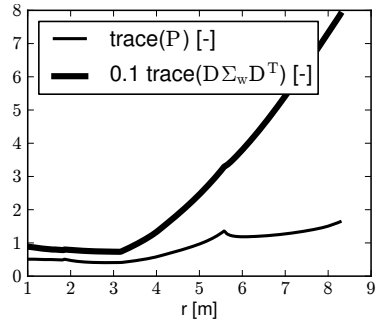


(d) Plot of the four entries of the optimal K.

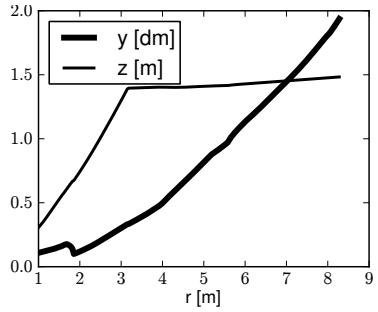
Figure 7.9: Results of the search for optimal stochastic safety for a sweep along `r`.

at the anticipated breakdown of rotational towing as predicted in Gillis 2012. However, judging from Figure 7.10b, there is no such breakdown occurring.

There is only one explanation left: the operational set becomes empty for large tether lengths. A plot of the multipliers of robustified bounds in Figure 7.11 can help to explore this route. Three important conflicting constraints are active near the end: torque limit of the carousel (`motor_torque`), negative of the height above the carousel arm (`z`) and side-slip angle (`beta_deg`). As the system avoids crashing into the ground, the aeroplane's nose is pointed upwards, but this action increases the drag which cannot be overcome due to the limitation in carousel torque. Note that the contraction of the operational set might be a local phenomenon.



(a) Plot of the magnitudes of covariances.



(b) Plot of the aeroplane coordinates in the carousel frame.

Figure 7.10: Explaining a cap on tether length for the feedback case?

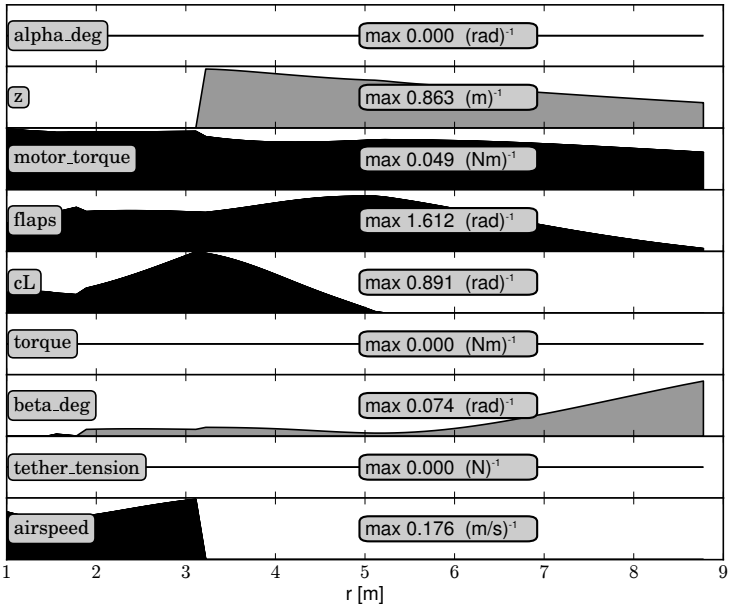


Figure 7.11: Constraint multipliers plot for system without feedback. The width of each band scales with the magnitude of the multipliers for one particular robustified path-constraint (lower-end of safety-margin in black, upper-end in grey). Zero width corresponds to non-active constraints.

7.5 Exploring alternative formulations

In this section, several optimisation runs using alternative formulations of Method (7.15) are examined in detail. Each run is denoted by a 7-letter identification code, each letter of which corresponds to one particular scenario setting, as explained in Table 7.2.

Code	Variation	Description
`A`, `B`	Problem	`A` corresponds to the Arianne aeroplane model, `B` corresponds to the Betty aeroplane model, featuring additional yaw and flaps control surfaces.
`J`,`Z`,`R`	Invariants	`J` corresponds to the J pseudo-inverse method, `Z` to the null-space approach, and `R` to the null-space approach, with additional regularisation in the objective $1 \times 10^{-3}\|\|\text{vec}(Z - \bar{Z})\|\|_2^2$.
`P`,`E`	Elimination	`P` corresponds to keeping P as decision variables, while `E` corresponds to inserting a linear solve node in the graph to eliminate them from the decision variables. In the `J`-Invariant case, also $J^\dagger$ is eliminated.
`N`,`L`	Lifting	`N` corresponds to no lifting (keeping $P - V(P + AP + PA^\top + D\Sigma_w D^\top)V = 0$ as constraint), while `L` corresponds to lifting this to $\begin{cases} P - V(P+Q)V = 0 \\ AP + PA^\top + D\Sigma_w D^\top = Q \end{cases}$ with Q an additional decision variable, which is eliminated together with P in the `E`-elimination case.
`M`,`X`	Expansion	`M` corresponds to keeping equations as a matrix graph in `CasADi`, while `X` corresponds expanding to scalar graph as much as possible.
`W`,`I`,`Q`	Solver	`W` corresponds to `WORHP`solver (state-of-the-art SQP), `I` corresponds to `IPOPT`solver (state-of-the-art interior point), `Q` corresponds to a simple textbook SQP method.
`F`,`T`	Initial	`F` corresponds to a feasible initial guess, `T` corresponds to an initial guess perturbed by 10%

Table 7.2: Identification code for different scenarios

Different combinations of Elimination, Invariant and Lifting give rise to different decision variables. The table below lists these differences. All formulations share a common part $\alpha = [x \in \mathbb{R}^{18}, u \in \mathbb{R}^4, \gamma \in \mathbb{R}]$.

	P	E
Z	α, $Z \in \mathbb{R}^{18\times 10}$, $P \in \mathbb{S}^{18}$ (, $Q \in \mathbb{S}^{18}$ if L-Lifting)	α, $Z \in \mathbb{R}^{18\times 10}$
J	α, $J^\dagger \in \mathbb{R}^{18\times 10}$, $P \in \mathbb{S}^{18}$ (, $Q \in \mathbb{S}^{18}$ if L-Lifting)	α

These combinations also result in different sets of constraints. All sets share a common part $\beta = [h_i + \gamma\sqrt{\frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top} \leq \bar{h}_i,\ \underline{h}_i \leq h_i - \gamma\sqrt{\frac{\partial h_i}{\partial x} P \frac{\partial h_i}{\partial x}^\top}, r = \bar{r}]$, where h_i lists all scalar constraints that make up the operational set (cf. Table 7.1).

	P	E
Z	$Z^\top f = 0$ $C = 0$ β $Z^\top Z = I$ $JZ = 0$ $ZPJ^\top = 0$ $JPJ^\top = 0$ $Z^\top QZ = 0$ $AP+PA^\top+D\Sigma_w D^\top=Q$ substitute in the above when N-Lifting	$Z^\top f = 0$ $C = 0$ β $Z^\top Z = I$ $JZ = 0$
J	$f - J^\dagger C = 0$ β $JJ^\top J^\dagger = J$ $P - V(P+Q)V = 0$ $AP+PA^\top+D\Sigma_w D^\top=Q$ substitute in the above when N-Lifting	$f - J^\dagger C = 0$ β

Figure 7.12 visualises the sparsity patterns arising the above formulations. The following remarks can be made about these patterns:

- Both the J-Invariant and Z-Invariant cases feature dense sub-blocks of size $\text{tril}(n) \times \text{tril}(n)$ and $\text{tril}(n - n_i) \times \text{tril}(n)$, respectively. For a small number of invariants, the blocks are of order $O(n^2) \times O(n^2)$ and hence the complexity of linear algebra for this problem is $O(n^6)$. In this context of the continuous-time algebraic Lyapunov, one might hope the blocks to be sparse due to the system matrix A to be sparse and due to the structure of $\mathcal{A}$ in Equation (7.19). This sparsity is destroyed by the dense nature of the invariant-treatment projections. Indeed, further research

could include obtaining a better complexity by finding a sparser null-space Z or sparser pseudo-inverse $J^\dagger$ such as with a 1-norm quality rather than the Moore-Penrose 2-norm quality as in Dokmanic 2013; Gotsman 2008. Alternatively, a dedicated Lyapunov solver could be embedded in the expression graph, similar as was done for the discrete-time periodic case in Chapter 5.

- Some blocks have a semi-circle-like sparsity-pattern. This is a direct result of using only the lower triangular part for symmetric decision variables and constraints.
- The blocks that disappear going from the `P`-Elimination to the `E`-Elimination case just end up being presented to the linear solve node introduced in the graph. The factorisation time associated with them remains present.

The result of the running all these scenarios is summarised graphically in the remainder of this section. Figure 7.13 shows convergence behaviour. Quite a few runs did not end successfully; due to stagnating convergence, excess time or excess memory use.

Some striking conclusions can be drawn from this graph:

- The `J`-Invariants case is much more successful than the `Z` or `R` cases. Since the sole purpose of the regularisation of `R` was to improve convergence behaviour over `Z` and this improvement is obviously lacking, `R`-Invariants case is rejected from further illustrations and comments.
- Eliminating the Lyapunov equation (`E`-Elimination case) appears to be an excellent idea. In combination with `J`-Invariants, success is guaranteed, even when using a textbook SQP method.
- The `WORHP` solver appears to be very successful for this type of problems, when `E`-Elimination is used. In case no elimination is performed, `IPOPT` performs better, but on an otherwise limited set of scenarios.

In summary, we identified `E`-Elimination + `J`-Invariant as the desirable numerically robust formulation. It should be noted that we have not investigated the numerical conditioning of the underlying KKT-system here. Corresponding to the notes of Section 5.6, it can be expected that the conditioning is worse in the `E`-Elimination case.

Figure 7.14 highlights runtimes for different formulations. The observations that can be made from this representations are along expectations:

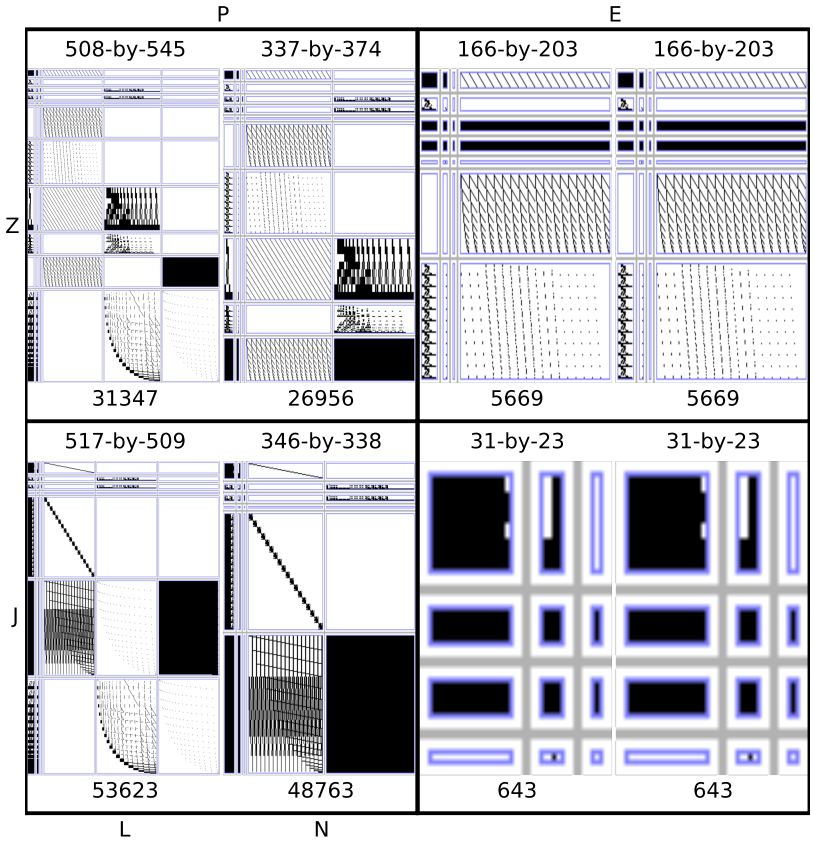


Figure 7.12: Constraint Jacobian sparsity patterns for different runs.

- Going from `P`-Elimination to `E`-Elimination amounts to transferring runtime from the solver to the constraint Jacobian. We can hence infer that the biggest contribution from both is caused by matrix factorisation.
- For the `P`-Elimination case, `IPOPT` (`I`-Solver) clearly performs much faster than `WORHP` (`W`-Solver), while they are on par for the `E`-Elimination case. This can be explained by the fact that an interior point method performs linearisation and linear algebra on the iteration level, while an SQP method linearises in each iteration and then iterates a QP to convergence, using linear algebra in an inner loop.
- Expansion (`X`-Expansion) is clearly beneficial to runtime.

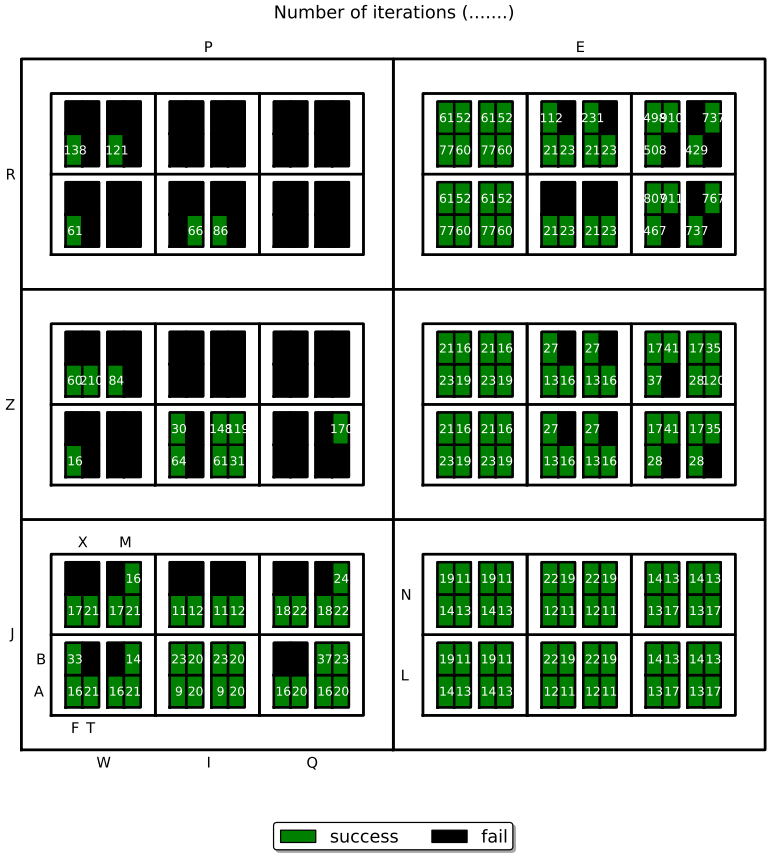


Figure 7.13: Convergence behaviour for different runs. For successful runs, the number of iterations is shown.

- Lifting (`L`-Lifting) is beneficial to runtime in the `E`-Elimination case. Lifting leads to less nodes in the expanded graph of the constraints, and hence faster evaluation of the constraints and their Jacobian. The beneficial effect exists for the `P`-Elimination too, but this is not visually apparent.
- The fastest runtimes are attained without elimination (`P`-Elimination) and with `J`-Invariant when using `IPOPT` (`I`-Solver). In the category that we identified as numerically robust before, `E`-Elimination + `J`-Invariant, combined expansion and lifting yields the best results.

The combination of `L`-Lifting + `X`-Expansion gives an order of magnitude speedup

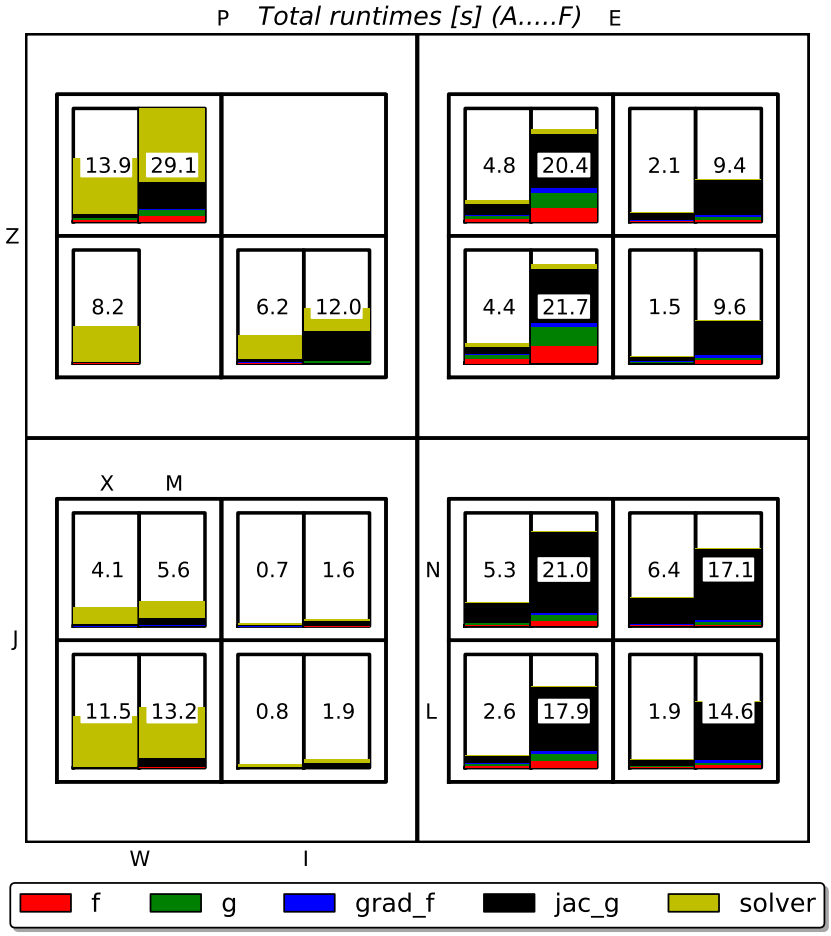


Figure 7.14: Runtime breakdown for different runs. The surface areas of opaque blocks correspond to runtimes of different parts of the run. The number corresponds to the total runtime.

for the preferred E-Elimination + J-Invariant formulation from above. Even then, it is not the absolute fastest of all scenarios.

Next, in Figure 7.15, the overhead cost, i.e. the cost of initialising the problem is highlighted.

From this figure, the following observations can be made:

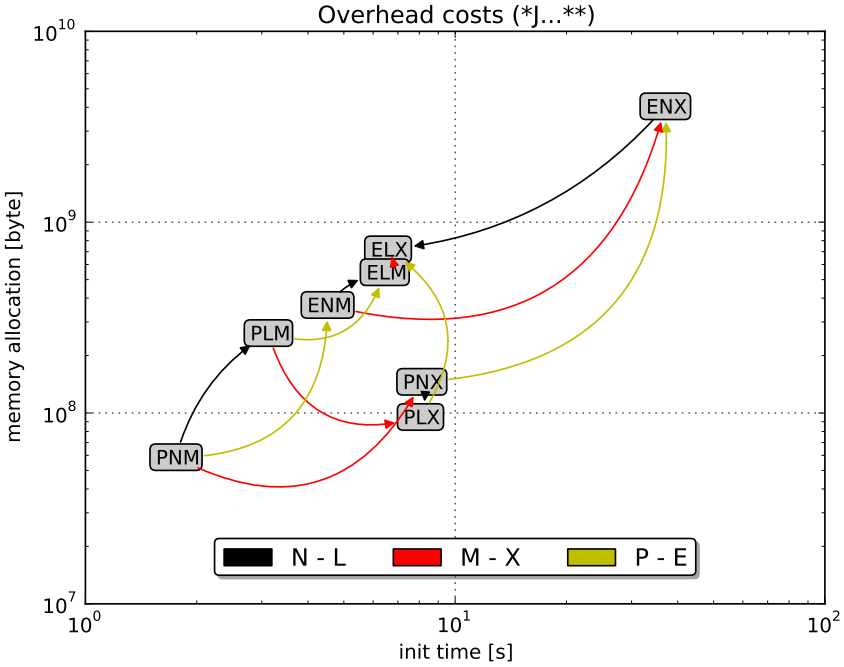


Figure 7.15: Overhead costs for different runs. Scenario codes corresponding to ∗ in the title are very much clustered in the plotted space and hence are represented by a single entity.

- All formulations require a large amount of memory. It is likely that memory will be a bottleneck for large applications.
- Introducing `E`-Elimination (P-E arrows), which was empirically shown to be beneficial for convergence, adds about an order of magnitude to overhead costs.
- The speedup of `L`-Lifting + `X`-Expansion from above does increase overhead, but not by an order of magnitude.

In summary, the speedups in runtime visible from Figure 7.14 do not come for free (mild overhead cost). Also, numerical robustness from Figure 7.13 does not come for free (large overhead).

In conclusion, we remark a) that a formulation with a pseudo-inverse and elimination of P is desirable for numerical robustness, b) that the runtime of such formulation can be sped up a lot with lifting and `CasADi`-specific graph

expansion, and c) that these runtime improving choices mildly deteriorate the overhead costs.

Conclusion

In this chapter we explored an application of the Lyapunov framework for a steady-state system. A carousel system (Geebelen 2010) for the launch of power harvesting tethered aeroplanes was considered. This system, intended for testing advanced control strategies, swings around a controllable aircraft connected with a tether of controllable length. The goal of this chapter was to identify, for a series of increasing tether lengths, setpoints to operate the device both open-loop and closed-loop. These set-points should both be stable and be far away from operational constraints that may destroy the system.

In the first part of the chapter, the multi-dimensional space of tentative setpoints was manually explored using two-dimensional slices in search of a trade-off between stability and feasibility.

In the second part of the chapter, the Lyapunov framework was used to bring stability into the same units as feasibility: under white Gaussian disturbance, the *algebraic* Lyapunov equation gives the state-covariance of a set-point, with which the probability of violating an operational bound can be expressed. By minimising this probability in an NLP, optimally *safe* setpoints could be automatically selected for the carousel, together with a suitable linear-feedback controller to decrease the probability of violation even further. This work was contributed to IFAC in (Gillis 2014b).

The idea of using a stochastic safety margin in the objective is not new (Logist 2011a), but has not been applied to a system with invariants.

The third part of the chapter provided an empirical comparison of numerical properties for a family of related formulations. The most striking observation from this section is that eliminating the Lyapunov matrix from the decision variables greatly aided convergence of the NLP.

Chapter 8

Robust periodic control of quadcopter flight

Flying machines have long fascinated humanity. While quite some baby boomers and generations there-after have enjoyed piloting radio-controlled flying machines as a pastime, these *unmanned aerial vehicles* or drones have only recently led to wide-spread civil and military applications.

In particular the quadcopter, a multi-rotor flying platform capable of hovering, is quickly claiming a place in the public space, following years of popularity amongst academic robotics groups as demonstration platform. Researchers typically fly them in a controlled indoor environment where a collection of cameras provides an accurate estimation of position and attitude, with which control laws can be applied. By combining highly agile gravity-defying motion with fully autonomous operation, quadcopters are suitable to showcase control algorithms while captivating the imagination of layman audiences. Notably the group at ETH Zurich has a history of spectacular demonstrations (D'Andrea 2014; Hehn 2013; Ritz 2012).

In this chapter, a quadcopter simulation model is used to showcase the methods presented in this thesis. While originally conceived as a mere benchmark for performance of `CasADi` in the second year of the research project, it has highlighted fundamental issues with the methods used previously for solving robustification problems. The quest to make this application work has ultimately driven the algorithmic innovations in this thesis.

Section 8.1 describes a robustification scenario we set out to solve, Section 8.2 details the modelling of the quadcopter system, Sections 8.3 and 8.4 describe methods and results for the nominal and robustified optimal control problems, respectively. A concluding section follows at the end.

8.1 Problem statement

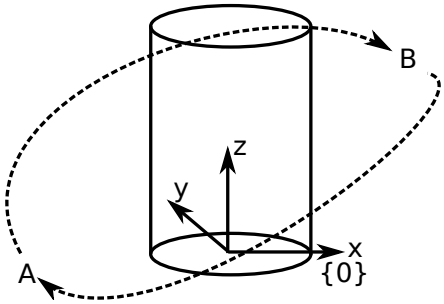


Figure 8.1: Quadcopter scenario

Referring to Figure 8.1, the quadcopter is to be flown around a cylindrical ($r = 1$m) obstacle positioned vertically at the center of the inertial frame of reference. Two waypoints $p_A = (-2, 0, 2)$ and $p_B = (2, 0, 1)$ are defined in the coordinates of the inertial frame, i.e. $\{1\}$. The nominal task of the quadcopter is to visit A and B in a *time-optimal* periodic trajectory that does not intersect with the obstacle. The solution of this *nominal optimal control* task consists of a feed-forward control trajectory as well as a corresponding state trajectory.

The main challenge is to *robustify* the obstacle collision constraints w.r.t. disturbance forces and moments acting on the quadcopter and measurement noise arising from the state-observer. Assuming a perfect state-observer with additive noise, a time-variant *linear feedback* controller is to be added as a degree of freedom in the robustified optimal control problem.

8.2 Modelling

This section documents the mathematical model created for the quadcopter system.

In Subsection 8.2.1, the dynamic equations for the quadcopter system are derived using a Newton approach. This approach is chosen over a Lagrangian approach, as the Newton approach requires more physical insight. The latter approach is to be preferred in a non-didactic setting, as it is generally less prone to mistakes and more straightforward to automate.

Subsection 8.2.2 details what forces and torques enter the dynamic equations. Subsection 8.2.3 lists all the parameters used in the model.

8.2.1 Dynamic equations

Two kinematic frames are employed for modelling the quadcopter. An inertial frame $\{0\}$, and a body frame $\{1\}$ attached at the center of mass of the quadcopter, with the z-axis pointing upwards.

The quadcopter body possesses linear momentum caused by the motion of its center of mass with respect to the inertial frame. Newton's law requires that its derivative in an inertial frame equals the force exerted upon it.

One can obtain a set of scalar equations out of this vector relation by performing projections. In this case, we choose to project onto the unit vectors of frame $\{0\}$, introducing velocity coordinates (v_x, v_y, v_z) expressed in $\{0\}$, quaternion coordinates (q_0,q_1,q_2,q_3) to parametrize the rotation from $\{1\}$ to $\{0\}$, and symbols for the forces expressed in $\{1\}$ to obtain 3 differential equations:

$$\underbrace{\begin{bmatrix} q_3^2+q_0^2-q_1^2-q_2^2 & 2q_0q_1-2q_3q_2 & 2q_0q_2+2q_3q_1 \\ 2q_0q_1+2q_3q_2 & q_3^2-q_0^2+q_1^2-q_2^2 & 2q_1q_2-2q_3q_0 \\ 2q_0q_2-2q_3q_1 & 2q_1q_2+2q_3q_0 & q_3^2-q_0^2-q_1^2+q_2^2 \end{bmatrix}}_{R\text{: transformation from } \{1\} \text{ to } \{0\}} \underbrace{\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}}_{F} = m \underbrace{\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix}}_{\dot{v}} \tag{8.1}$$

The dynamic equations for the linear part of the quadcopter motion are completed by introducing coordinates $(x,\ y,\ z)$ for the position expressed

in $\{0\}$:

$$\underbrace{\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}}_{\dot{\mathrm{r}}} = \underbrace{\begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix}}_{v} \tag{8.2}$$

The total angular momentum $\mathcal{H}$ of the quadcopter with respect to the inertial frame is given by the momentum for the system with stationary rotors, plus contributions of the rotor movements:

$$\mathcal{H} = \mathrm{I}\Omega + \sum_{i\in\text{rotors}} I^i\Omega^i, \tag{8.3}$$

in which I is the inertia tensor, and in which the rotors are modelled as discs.

The rotor spin axes are assumed to be aligned with the body frame, and the rotors are spun at a rate r_i:

$$\Omega^i = \begin{bmatrix}0\\ 0\\ s_i r_i\end{bmatrix}, \tag{8.4}$$

with s_i the spin direction (either 1 or -1).

Euler's equation dictates that the derivative of angular impulse of a body in an inertial frame equals the torque exerted upon that body. As the inertia tensor of a body is typically stationary when observed in the body frame, it is customary to transform the frame of differentiation to the body frame.

Introducing body coordinates for angular motion (ω_x, ω_y, ω_z) and projecting in the body frame, we obtain the Euler-like equations for the rotational motion of the system:

$$\begin{aligned}\overbrace{\begin{bmatrix}T_x\\ T_y\\ T_z\end{bmatrix}}^{\mathcal{T}} &= \overbrace{\begin{bmatrix}I_x\dot{\omega}_x\\ I_y\dot{\omega}_y\\ I_z\dot{\omega}_z + \sum I_z^i s_i\dot{r}_i\end{bmatrix}}^{\mathrm{I}\,\dot{\Omega} + \sum I^i\,\Omega^i}\\ &+ \underbrace{\begin{bmatrix}0 & -(\omega_z) & \omega_y\\ \omega_z & 0 & -(\omega_x)\\ -(\omega_y) & \omega_x & 0\end{bmatrix}}_{\tilde{\Omega}} \underbrace{\begin{bmatrix}I_x\omega_x\\ I_y\omega_y\\ I_z\omega_z + \sum I_z^i s_i r_i\end{bmatrix}}_{\mathrm{I}\,\Omega + \sum I^i\,\Omega^i}.\end{aligned} \tag{8.5}$$

The dynamic equations of the quadcopter base are completed with a kinematic relation between the orientation quaternions and the angular rates:

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_3 & -q_2 & q_1 \\ q_2 & q_3 & -q_0 \\ -q_1 & q_0 & q_3 \\ -q_0 & -q_1 & -q_2 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{8.6}$$

Each rotor is mounted on a shaft that passively transmits torques in the local x and y directions. In the z-direction, the torque goes through a motor, which delivers work on the rotor. This driving torque τ^i_{motor}, as well as the aerodynamic reaction torque τ^i_{aero} and the rotor speed r_i are defined positive if they drive the propeller in such a way as to generate upward thrust.

Angular motion of the rotor is governed by a balance between rotor inertia, driving torque and aerodynamic torque:

$$I^i_z(\dot{r}_i + s_i\dot{\omega}_z) = \tau^i_{\text{motor}} - \tau^i_{\text{aero}}, \quad i = 1, \ldots, n_{\text{r}} \tag{8.7}$$

In conclusion, we derived a model with $n = 13+n_{\text{r}}$ implicit differential equations:

$$g(\xi, \dot{\xi}, u, p) = 0, \tag{8.8}$$

with $13 + n_{\text{r}}$ differential states:

$$\xi = [\overbrace{x, y, z}^{p}, \overbrace{v_x, v_y, v_z}^{v}, \overbrace{q_0, q_1, q_2, q_3}^{q}, \overbrace{\omega_x, \omega_y, \omega_z}^{\omega}, \overbrace{r_1, \ldots, r_{n_{\text{r}}}}^{r}], \tag{8.9}$$

position; velocity; quaternions; angular rate; rotor speed

and n_{r} controls:

$$u = [\tau^i_{\text{motor}}]. \tag{8.10}$$

One invariant, quaternion norm preservation, is present due to the choice of quaternions for parametrising orientation:

$$C(\xi) = \sum_i q_i^2 - 1 = 0. \tag{8.11}$$

Since only the quaternion states enter the invariant in this particular system, it is possible to write the invariant alternatively as:

$$C'(q) = \sum_i q_i^2 - 1 = 0. \tag{8.12}$$

8.2.2 Forces and torques

The unspecified forces and torques in the previous equations are expressed in the body frame, which is convenient for the aerodynamic contributions.

Each rotor is modelled as giving an independent contribution, assumed undisturbed by each other, by the presence of the quadcopter support platform, and by the presence of airflow at the location of operation. Corresponding to Bristeau 2009, each rotor i has the effect of generating a force perpendicular to its disk and upwards, emanating from its center:

$$F^i_{\text{aero}} = \rho c_i R_i^3 r_i^2 C_{\text{L}} \left(\frac{\alpha_0}{3} - \frac{v_i^{\perp}}{2R_i r_i} \right), \tag{8.13}$$

with ρ the air density, c the chord length of the propeller, R its radius, C_{L} its lift coefficient, α_0 the zero-lift angle of attack and $v_i^{\perp}$ the perpendicular component of velocity due to quadcopter motion observed at the center of rotor, which is given by a simple kinematic relation:

$$v_i^{\perp} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} (Rv + \Omega \times \mathrm{r}), \tag{8.14}$$

with r the location of the rotor i on the quadcopter platform.

Each rotor further experiences a torque perpendicular to its disk arising from propeller form drag, induced drag, and a component of lift:

$$\tau^i_{\text{aero}} = \rho c_i R_i^4 r_i^2 \left(\frac{C_{\text{D}}}{4} + C_{\text{D,ind}} \alpha_0^2 \left(\frac{\alpha_0}{4} - \frac{2v_i^{\perp}}{3r_i R_i} \right) - C_{\text{L}} v_i^{\perp} \frac{\frac{\alpha_0}{3} - \frac{v_i^{\perp}}{2r_i R_i}}{r_i R_i} \right). \tag{8.15}$$

The total force acting on the quadcopter platform arises from gravity and rotor thrust:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = R \begin{bmatrix} 0 \\ 0 \\ -gm \end{bmatrix} + \sum \begin{bmatrix} 0 \\ 0 \\ F^i_{\text{aero}} \end{bmatrix}. \tag{8.16}$$

The total torque acting on the quadcopter platform arises from aerodynamic contributions:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum -s_i \tau^i_{\text{aero}} \end{bmatrix} + \sum \mathrm{r} \times \begin{bmatrix} 0 \\ 0 \\ F^i_{\text{aero}} \end{bmatrix}, \tag{8.17}$$

where we note that it is the *external* aerodynamic torque that acts on the complete quadcopter system. The motor torque is *internal* to this system.

8.2.3 Configuration

We consider a quadcopter configuration where half of the rotors have nominal spin directions s_i opposite of the other half, as this cancels out angular momentum contributions at nominal operation (limits undesirable gyroscopic effects) and reduces the magnitude of vorticity induced on the surrounding air (an unmodelled physical effect).

A configuration with $n_\mathrm{r} = 4$ rotors is chosen. The following parameters are unique to each rotor:

Quantity	$i = 0$	$i = 1$	$i = 2$	$i = 3$
r	$\begin{bmatrix} L \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ L \\ 0 \end{bmatrix}$	$\begin{bmatrix} -L \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -L \\ 0 \end{bmatrix}$
s_i	1	-1	1	-1

The remainder of rotor parameters are shared among all rotors:

c	Propeller chord length	[m]	0.01
R	rotor radius	[m]	0.127
C_L	rotor lift coefficient	[-]	6
α_0	zero-lift angle of attack	[rad]	0.15
C_D	rotor drag coefficient	[-]	0.02
$C_\mathrm{D,ind}$	rotor induced drag coefficient	[-]	0.05
m	propeller mass	[kg]	0.01
I_max	characteristic inertia (mR^2)	[kg · m^2]	161×10^{-6}
I_ref	reference inertia ($I_\mathrm{max}/5$)	[kg · m^2]	32.3×10^{-6}
I_x, I_y	rotor inertia ($I_\mathrm{ref}/2$)	[kg · m^2]	16.1×10^{-6}
I_z	rotor inertia (I_ref)	[kg · m^2]	32.3×10^{-6}

The quadcopter platform parameters are as follows:

m	platform total mass	[kg]	0.5
g	gravitational acceleration	[m/s^2]	9.81
ρ	air density	[kg/m^3]	1.225
L	platform characteristic length	[m]	0.25
I_max	characteristic inertia (mL^2)	[kg · m^2]	31.3×10^{-3}
I_ref	reference inertia ($I_\mathrm{max}/5$)	[kg · m^2]	6.25×10^{-3}
I_x, I_y	platform inertia ($I_\mathrm{ref}/2 + \sum I_x^i$)	[kg · m^2]	3.19×10^{-3}
I_z	platform inertia ($I_\mathrm{ref} + \sum I_z^i$)	[kg · m^2]	6.38×10^{-3}

It should be noted that these parameters, specifically the ones involving aerodynamics, can have uncertain values. In the Lyapunov framework, we can treat those uncertainties as extra white Gaussian disturbances acting on the system dynamics. However, the present chapter does not explore this further; the only three sources of disturbance considered here will be summarised in Figure 8.10.

8.3 System analysis and nominal optimal control

Since the final robustification problem formulation is non-convex, we highlight in this section a homotopy of increasingly more complex scenarios.

Starting with a steady-state analysis, we proceed over a tracking OCP and arrive at a time-optimal OCP. In addition, a time-variant feedback control is constructed to make the closed-loop trajectory stable.

The next section, Section 8.4, makes use of these results.

Each subsection uses a different treatment of the system invariants. These differences are not essential but are retained for the purpose of illustration.

8.3.1 Hovering

The condition of hovering, i.e. stationary flight in which generated thrust balances out gravity, is easily formulated as a root-finding problem:

$$g(\xi, 0, u, p) = 0 \tag{8.18}$$

with

$$\xi = [\overbrace{0,0,1}^{\text{position}}, \overbrace{0,0,0}^{\text{velocity}}, \overbrace{0,0,0,1}^{\text{quaternions}}, \overbrace{0,0,0}^{\text{angular rate}}, \overbrace{r,r,r,r}^{\text{rotor speed}}],$$

$$u = [\overbrace{\tau,\tau,\tau,\tau}^{\text{motor torques}}],$$

and r and τ respectively the unknown hovering rotor speed and torques.

Although this system of equations is overdetermined, a consistent solution is easily obtained by solving the following unconstrained NLP:

$$\underset{r,\tau}{\text{minimise}} \quad ||g(\xi(r), 0, u(\tau), p)||_2^2. \tag{8.19}$$

A numerical solution with `IPOPT` leads to the sought nominal values $r^\star = 403.6\,\mathrm{rad/s}$ and $\tau^\star = 2.617 \times 10^{-3}\,\mathrm{N \cdot m}$.

Based on these results, we define scaling matrices for states and controls:

$$S_x \in \mathbb{R}^{n\times n} = \mathrm{diag}(\overbrace{1\ldots1}^{13\text{ repetitions}}, 500, 500, 500, 500) \tag{8.20}$$

$$S_u \in \mathbb{R}^{m\times m} = \mathrm{diag}(0.005, 0.005, 0.005, 0.005), \tag{8.21}$$

and scaled reference values:

$$\tilde{r} = [\ \overbrace{r^\star/500}^{n_\mathrm{r}\text{ repetitions}}\ ]^\top \tag{8.22}$$

$$\tilde{\tau} = [\ \overbrace{\tau^\star/0.005}^{n_\mathrm{r}\text{ repetitions}}\ ]^\top. \tag{8.23}$$

To convey a physical interpretation of the quadcopter model, we report in Figure 8.3 on a simple open-loop flight scenario that starts from hovering and has a step increase of torque for the first rotor by 20%. Figure 8.2 is provided as a diagram to support the following analysis of asymptotic behaviour of the system.

The first rotor, positioned at $(L, 0, 0)$, accelerates immediately (first-order increase of rotation speed) due to the added torque. The resulting increased lift causes a tilting motion of the quadcopter platform along the local $-y$ axis (second-order) and accelerates the platform upwards (second-order). Since the tilting causes a contribution of lift in the inertial $-x$ axis, the platform accelerates in that direction (fourth-order).

Due to reaction on the platform by the motor, the platform accelerates around local $-z$ axis. This reaction is smaller than the rotor speed acceleration, but still of first-order. Since the platform accelerates, so must the remaining rotors to balance Equation (8.7), also a small but first-order change.

Two non-linear effects can be observed in the asymptotic behaviour. A fourth-order acceleration around the $+x$ axis is caused by the $\omega_y\omega_z$ term in the x coordinate in Equation (8.5). A sixth-order acceleration along the $+y$ axis is caused by a component of the lift in this direction. This component is the result of an angle-product that follows from rotating a frame around a local $-z$ axis and subsequently over a moved $-y$ axis.

After a while, the platform tilt angle becomes so large that gravity cannot be balanced by the (increased) thrust. The quadcopter drops down.

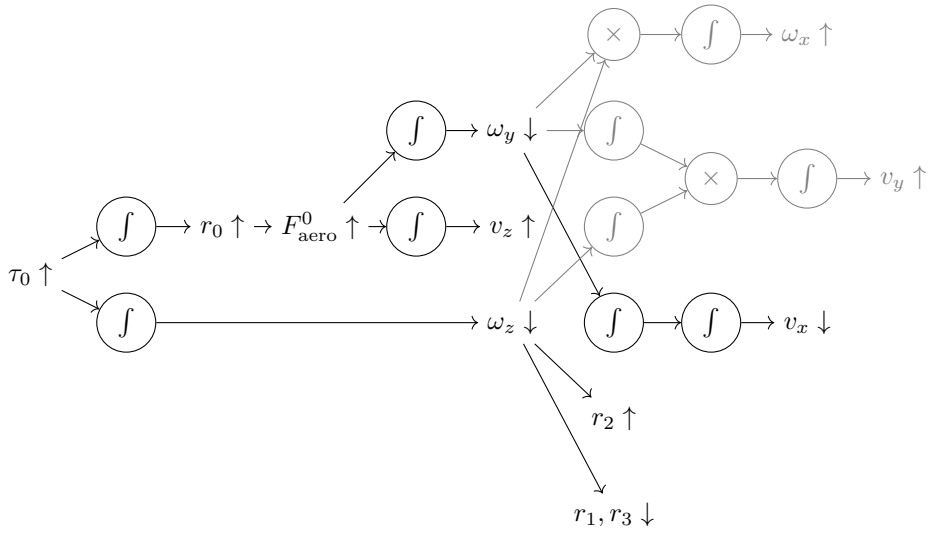
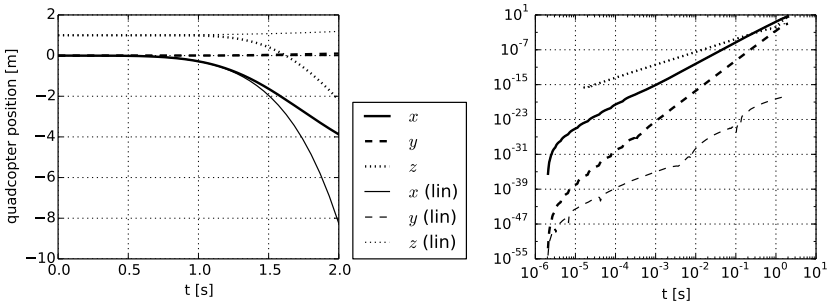
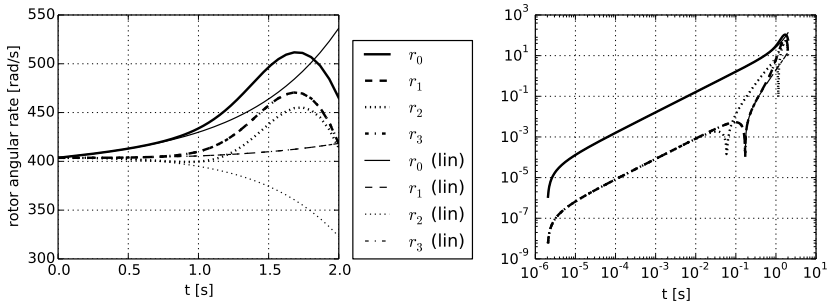


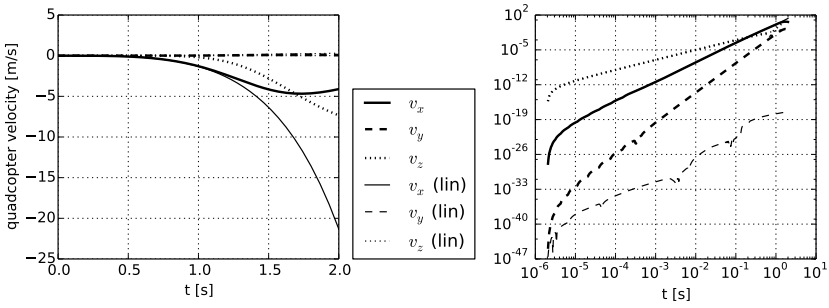
Figure 8.2: Diagram of asymptotic system step. The shaded region symbolises nonlinear effects.



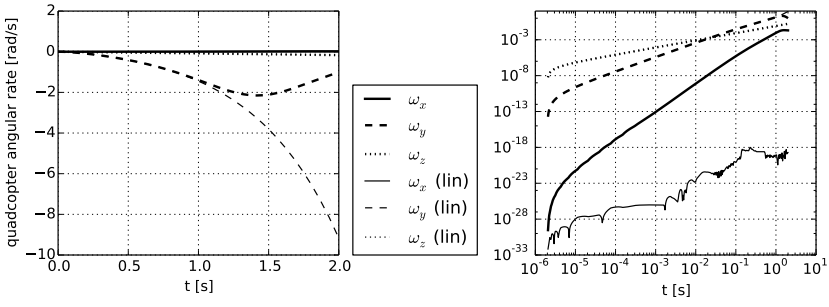
(a) Time evolution of quadcopter position



(b) Time evolution of rotor speeds



(c) Time evolution of quadcopter linear velocity



(d) Time evolution of quadcopter angular velocity

Figure 8.3: System response for a step in torque, both fully non-linear and linearised around the hovering configuration. The right panes show size of deviation from nominal hovering.

8.3.2 Tracking

Next, we formulate and solve a periodic optimal control problem with an objective tracking a reference trajectory that includes the waypoints and avoids the obstacle:

$$p_{\text{ref}}(t) = \left(-2\cos\beta(t), 2\sin\beta(t), 1 + \frac{\cos\beta(t)+1}{2}\right)^{\top}, \tag{8.24}$$

with $\beta(t) = \frac{2\pi t}{T_f}$ and $T_f = 10.0\,\text{s}$.

The system dynamics are discretised over $N = 20$ intervals using a collocation scheme with a Radau polynomial of degree $d = 3$:

$$L_i(\tau) = \prod_{\substack{\text{r}=0 \\ \text{r}\neq\text{i}}}^{d} \frac{\tau - \tau_i}{\tau_i - \tau_r}, \tag{8.25}$$

where the subscript τ symbols denote the entries of the polynomial roots list, i.e. $[0, 0.155051, 0.644949, 1]$ in this case.

The decision variable vector associated with state ξ at control interval $k \in [0, \ldots, N-1]$ and collocation point $j \in [0, \ldots, d]$ is denoted as a capital with double subscript X_{ij}. To denote the extraction of a few states from this vector, an extra index is used with the symbols of Equation (8.9) .

We formulate the tracking optimal control problem directly in discrete time as a direct collocation problem:

$$
\begin{aligned}
&\underset{X_{\bullet\bullet},U_\bullet,Z}{\text{minimise}} && \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,p} - p_{\text{ref}}(t_k)||_2^2 && \text{Tracking objective}\\
& && && \text{Regularise (penalise) ...}\\
& && + \epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||U_k - \tilde{u}||_2^2 && \text{control excursions}\\
& && + \epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,r} - \tilde{r}||_2^2 && \text{rotor speed excursions}\\
& && + \epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,\omega}||_2^2 && \text{rotation of body}\\
& && + \epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,q} - (0,0,0,1)||_2^2 && \text{deviation from upright pose}
\end{aligned}
$$

$$
\begin{aligned}
\text{s.t.}\quad & 0 = g\left(t_{kj}; S_x X_{kj}, S_x \dot{X}_{kj}, S_u U_k, p\right) && \begin{cases} k = 0 \dots \tilde{N} \\ j = 1 \dots d \end{cases} && \text{Collocation}\\
& 0 = X_{(k+1)0} - \textstyle\sum_{i=0}^{d} X_{ki} L_i(1) && k = 0 \dots N-2 && \text{Coupling}\\
& 0 = X_{00,(p,v,\omega,r)} - X_{\tilde{N}d,(p,v,\omega,r)} && && \text{Periodicity}\\
& 0 = C'(X_{00,q}) && && \text{Invariant}\\
& 0 = \tfrac{\partial C'}{\partial \xi}(X_{00,q}) Z && && \text{Null-space}\\
& 0 = Z^\top Z - I_3 && && \text{Orthonorm.}\\
& 0 = Z^\top \left(X_{00,q} - X_{\tilde{N}d,q}\right) && && \text{Quat. period.}
\end{aligned}
$$

$$
\begin{aligned}
-3 &\leq X_{k0,p} \leq 3 && k = 0 \dots \tilde{N} && \text{Confine to cube}\\
0 &\leq U_k \leq S_u^{-1} 2\tau^\star, && k = 0 \dots \tilde{N} && \text{Peak torque 200\%}
\end{aligned}
\tag{8.26}
$$

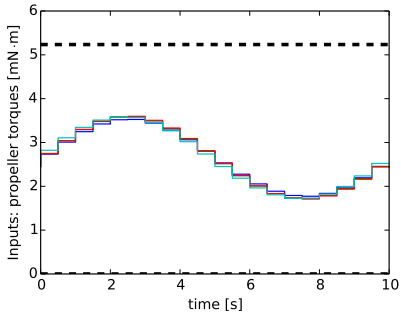
with $\tilde{N} = N - 1$, $t_{kj} = \frac{T_f}{N}(\tau_j + k)$, and $\dot{X}_{kj} = \sum_{i=0}^{d} X_{ki} \frac{dL_i}{d\tau}(\tau_j)$.

Note that the tracking objective introduces a natural phase preference for the trajectory. It is therefore not needed to fix the phase by introducing an additional scalar constraint. The NLP is initialised with:

$$
\begin{aligned}
X_{k\bullet}^0 &= S_x^{-1}[\overbrace{p_{\text{ref}}(t_{k0})}^{\text{position}}, \overbrace{0,0,0}^{\text{velocity}}, \overbrace{0,0,0,1}^{\text{quaternions}}, \overbrace{0,0,0}^{\text{angular rate}}, \overbrace{r^\star, r^\star, r^\star, r^\star}^{\text{rotor speed}}],\\
U_\bullet^0 &= S_u^{-1}[\tau^\star, \tau^\star, \tau^\star, \tau^\star],\\
Z^0 &= \text{(consistent with defining equations)}
\end{aligned}
$$

The NLP is solved to convergence with `IPOPT` using an exact Hessian, with the `metis`-enabled `ma57` linear solver and tolerance 1×10^{-9}. Figure 8.4 shows the solution trajectory. The main mode of the control trajectory is a coherent sinusoidal motion of all rotor torques around steady-state to create the up-down

motion. A second mode involves separate motion for each rotor to trace the circular path. Because the time period T_f is rather large, this mode is rather subtle and scarcely discernible on the plot. Note that control bounds are not active.



(a) Control trajectory. Control bounds are highlighted as dotted lines

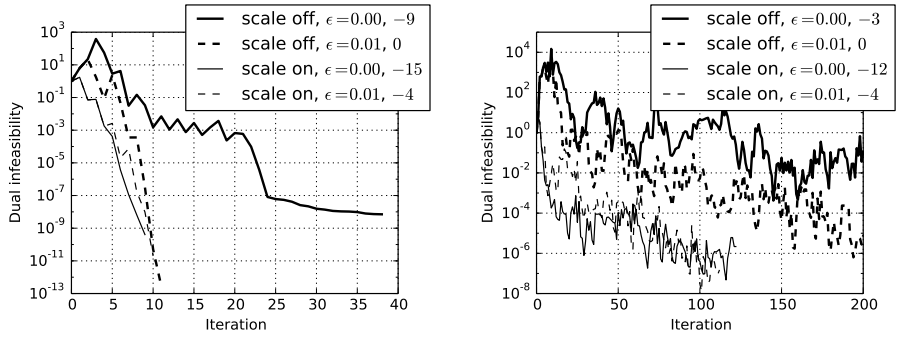
Number of variables	1454
Number of constraints	1372
Number of nonzeros in constraint Jacobian	15076
Solution time	0.34 s

(b) Problem statistics

Figure 8.4: Converged tracking problem

By nature of the system, the influence of the control inputs on the tracking objective is very indirect. Indeed, in the analysis of the preceding system near the hovering state, the influence is at least of third order. It is expected that the convergence is aided by adding a guidance to the NLP solver in the form of regularisation of excursions away from the nominal position. For the nominal solution, $\epsilon = 0.01$ is used. Also, scaling of decision variables (S_x, S_u) is expected to make a difference. Figure 8.5 shows how in this numerical case, scaling is beneficial and regularisation less clearly so.

The order of decision variables is important to obtain a quasi-banded constraint Jacobian structure. Both X and U, and collocation and coupling constraints have interleaved ordering, such that each discretisation interval gives rise to a separate block in the constraint Jacobian, as apparent from Figure 8.6. Each discretisation block itself is composed of n rows for the coupling constraints and $(d+1)$-by-$(d+1)$ cells of n-by-n system sensitivity matrices for the collocation constraints. Note that the collocation blocks are quite sparse owing to the particular sparse system dynamics at hand and to the fact that the DAE-formulation is suitable to maintain this sparsity. In general, the collocation blocks tend to be dense and this puts a practical limit on the degree of collocation. In that situation, it may be beneficial to decrease the degree (and correspondingly increase the number of discretisation intervals to maintain integration accuracy).



(a) Exact Hessian

(b) Limited-memory BFGS Hessian

Figure 8.5: Convergence for the tracking problems in different configurations. The tracking objective optimal value is shown as base-10 exponent in the legend.

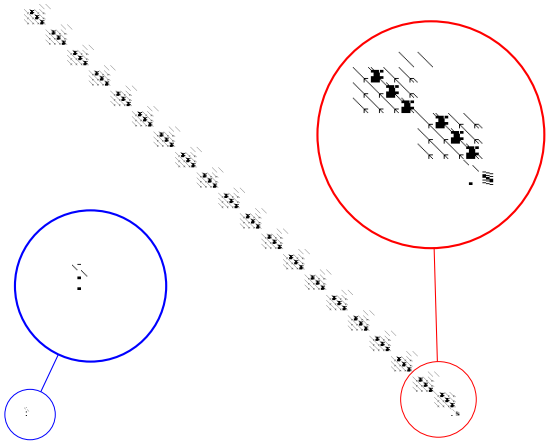


Figure 8.6: Constraint Jacobian structure

8.3.3 Time-optimal control

The next step in the homotopy is a time-optimal periodic formulation. The key component in such a formulation is time-scaling. In this particular case, since the time-of-visit for the waypoints is a degree of freedom, we use a time-scaling λ with two values, switching after half the control intervals have passed:

$$\lambda(k) = \begin{cases} \frac{T_w}{N/2} & 0 \leq k < N/2 \\ \frac{T-T_w}{N/2}, & N/2 \leq k < N \end{cases} \tag{8.27}$$

with T_w denoting the transit time of the waypoint B, while T denotes the total period.

We formulate the time-optimal optimal control problem directly in discrete time as follows:

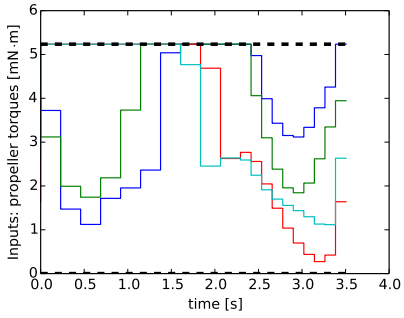
$$\begin{aligned}
&\underset{X_{\bullet\bullet}, U_{\bullet\bullet}, Z, T, T_w}{\text{minimise}} && T && && \text{Time optimality} \\
& && && && \text{Regularise (penalise) ...} \\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||U_k - \tilde{u}||_2^2 && && \text{control excursions} \\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,r} - \tilde{r}||_2^2 && && \text{rotor speed excursions} \\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,\omega}||_2^2 && && \text{rotation of body} \\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,q} - (0,0,0,1)||_2^2 && && \text{deviation from upright pose} \\
\text{s.t.} \quad & 0 = g\left(t_{kj}; S_x X_{kj}, \tfrac{S_x}{\lambda(k)} \dot{X}_{kj}, S_u U_k, p\right) && && \begin{cases} k = 0 \dots \tilde{N} \\ j = 1 \dots d \end{cases} && \text{Collocation} \\
& 0 = X_{(k+1)0} - \textstyle\sum_{i=0}^{d} X_{ki} L_i(1) && && k = 0 \dots N-2 && \text{Coupling} \\
& 0 = Z^\top (X_{00} - X_{\tilde{N}d}) && && && \text{Periodicity} \\
& 0 = C(X_{00}) && && && \text{Invariant} \\
& 0 = \tfrac{\partial C}{\partial \xi}(X_{00}) Z && && && \text{Null-space} \\
& 0 = Z^\top Z - I_{n-1} && && && \text{Orthonorm.} \\
& 0 = S_x X_{00,p} - p_A && && && \text{Waypoint A} \\
& 0 = S_x X_{\frac{N}{2}0,p} - p_B && && && \text{Waypoint B} \\
& 0 \leq h(X_{k0}) && && k = 0 \dots \tilde{N} && \text{Obstacle} \\
& 0 \leq U_k \leq S_u^{-1} 2\tau^\star && && k = 0 \dots \tilde{N} && \text{Peak torque 200\%} \\
& 0 \leq T_w \leq T \leq T_f && && && \text{Chronology} \\
& -4 \leq X_{k0,p} \leq 4 && && k = 0 \dots \tilde{N} && \text{Confine to cube}
\end{aligned} \tag{8.28}$$

with $t_{kj} = \lambda(k)\tau_j + \sum_{p=0}^{k-1} \lambda(p)$ and with h the obstacle constraint:

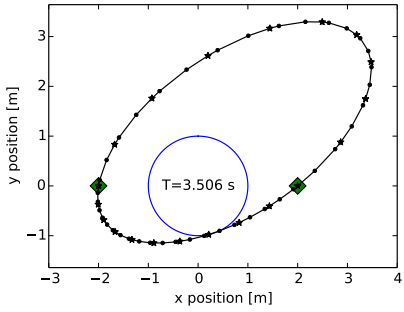
$$h(\xi) = \sqrt{x^2 + y^2} - 1 \geq 0. \tag{8.29}$$

Figure 8.8a shows that the structure of the NLP is very similar to the previous version, but has extra dense columns due to time scaling. The linear solver we use, `metis`-enabled `ma57`, can efficiently deal with this structure. For other linear solvers, it might be beneficial to localise the time-scaling: introduce time-scale decision variables on each control interval, and have coupling constraints. This would eliminate the dense column. Note that a large block is present near the bottom that was not present in the previous section. This block arises from the null-space normalisation constraint. It was present in both NLPs, but in this case, the more general $C(\xi)$ was used instead of $C'(q)$.

Figure 8.7 shows an impression of the optimal trajectory. Note how the control inputs are saturated quite often, a feature that is to be expected from time-optimal control.



(a) Control trajectory.

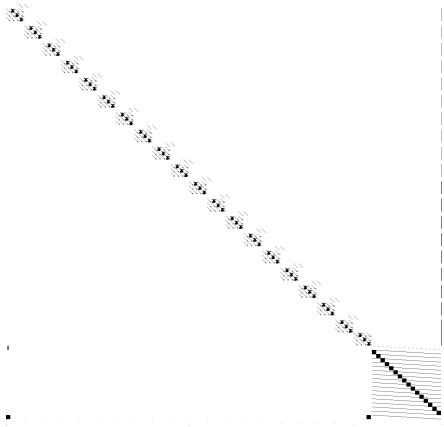


(b) Top down trajectory view. Stars mark the control intervals, dots mark the collocation points. Two diamond shapes mark the waypoints. The motion is clock-wise around the circular obstacle.

Figure 8.7: Trajectories of the converged time-optimal problem

The optimised trajectory is highly unstable. Indeed, when integrating the system for one period with the found feed-forward controls with high precision, the initial and final states deviate substantially. The collocation scheme is not of high accuracy and the mismatch is enlarged by the unstable nature of the system. The sensitivity of the final state w.r.t. the initial state for the

accurate integration is an approximation for the monodromy matrix. Some of its eigenvalue magnitudes are much higher than one. These and other statistics are enlisted in Figure 8.8b.



(a) Constraint Jacobian

Number of variables	1716
Number of constraints	1659
Number of nonzeros in constraint Jacobian	25884
Solution time	4.14 s
Optimal time	3.506 s
Unstable Floquet multipliers	2.4×10^{1}, 1.3×10^{3}, 1.6×10^{4}

(b) Statistics

Figure 8.8: NLP structure and statistics

8.3.4 Linear quadratic regulator

This step works towards having feedback-control on top of the feed-forward optimal control trajectory. The goal at this stage is to just find a series of full state feedback matrices $K_\bullet$ that stabilise the system along the trajectory of the previous system.

First, we aim to find a discrete linearisation $(A_\bullet, B_\bullet)$ or rather the scaled linearisation $(\tilde{A}_\bullet, \tilde{B}_\bullet)$ for the periodic system along the optimal trajectory:

$$\tilde{x}_{(k+1) \bmod N} = \tilde{A}_k \tilde{x}_k + \tilde{B}_k \tilde{u}_k, \quad k = 0 \dots N-1, \tag{8.30}$$

with $\tilde{x}_k$ and $\tilde{u}_k$ *scaled* linearised states and controls.

Note that a formulation as implicit linear system $E_k \tilde{x}_{(k+1) \bmod N} = A_k \tilde{x}_k + B_k \tilde{u}_k$, which can potentially avoid an $n-$by$-n$ system inversion, has little benefits in this case. Even though the continuous system is implicit, the implicit nature of the integration scheme leads to a linearisation that entails the inversion of a $dn-$by$-dn$ system.

In a general OCP framework like `CasADi`, the linearisation $(\tilde{A}_\bullet, \tilde{B}_\bullet)$ is in general trivially obtained by querying the Jacobian of the integration function. In this

particular case, we have manually constructed a collocation scheme and we will derive the sensitivities manually as well:

$$\tilde{A}_k = \frac{\partial X_{(k+1)0}}{\partial X_{k0}}, \quad \tilde{B}_k = \frac{\partial X_{(k+1)0}}{\partial U_{k0}}. \tag{8.31}$$

Using the coupling constraints, we have:

$$\tilde{A}_k = \sum_{j=0}^{d} L_j(1) \frac{\partial X_{kj}}{\partial X_{k0}}, \tag{8.32}$$

with $\frac{\partial X_{k0}}{\partial X_{k0}} = I_n$ and, using implicit function theorem, $\frac{\partial X_{kj}}{\partial X_{k0}}$ equal to rows $(j-1)n$ through $(jn-1)$ of $\left[G_k^X\right]^{-1} G_k^0$ for $j = 1, \ldots, d$.

Here, $G_k^X \in \mathbb{R}^{dn \times dn}$ is composed of the following blocks with block index (i, j) and $i, j = 1, \ldots, d$:

$$\frac{dg_{ki}}{dX_{kj}} = \frac{\partial g_{ki}}{\partial \xi} \frac{d\left[S_x X_{ki}\right]}{dX_{kj}} + \frac{\partial g_{ki}}{\partial \dot{\xi}} \frac{d\left[\frac{1}{\lambda(k)} S_x \sum_l X_{kl} \frac{dL_l}{d\tau}(\tau_i)\right]}{dX_{kj}} \tag{8.33}$$

$$= \left[\frac{\partial g_{ki}}{\partial \xi} \delta_{ij} + \frac{\partial g_{ki}}{\partial \dot{\xi}} \frac{1}{\lambda(k)} \frac{dL_j}{d\tau}(\tau_i)\right] S_x, \tag{8.34}$$

and $G_k^0 \in \mathbb{R}^{dn \times n}$ is composed of a block column matrix with block index $i \in [1, d]$:

$$\frac{dg_{ki}}{d\left[X_{k0}\right]} = \frac{\partial g_{ki}}{\partial \dot{\xi}} S_x \frac{1}{\lambda(k)} \frac{dL_0}{d\tau}(\tau_i). \tag{8.35}$$

The derivation for $\tilde{B}_k$ uses the same matrix G_k^X, and its inverse is right multiplied with a block column matrix with entries $\frac{dg_{ki}}{d[U_k]} = \frac{\partial g_{ki}}{\partial u} S_u$.

Next, we project away the invariant subspace of the system $(\tilde{A}_\bullet, \tilde{B}_\bullet)$ to obtain a reduced system:

$$\hat{A}_k = Z_{k+1 \bmod N}^\top \tilde{A}_k Z_k \tag{8.36}$$

$$\hat{B}_k = Z_{k+1 \bmod N}^\top \tilde{B}_k, \tag{8.37}$$

with $Z_k = \text{null}(\frac{\partial C}{\partial \xi}(\xi_k))$.

In this reduced space, we formulate a discrete periodic LQR:

$$\begin{array}{ll} \underset{\hat{x}_\bullet, \hat{u}_\bullet, \hat{K}_\bullet}{\text{minimise}} & \sum_{k=0}^{N-1} \hat{x}_k^\top Q \hat{x}_k + \hat{u}_k^\top R \hat{u}_k \\ \text{s.t.} & \tilde{x}_{k+1 \bmod N} = \hat{A}_k \hat{x}_k + \hat{B}_k \hat{u}_k \quad k = 0, \ldots, N-1 \\ & \hat{u}_k = -\hat{K}_k \hat{x}_k. \qquad\qquad\quad k = 0, \ldots, N-1 \end{array} \tag{8.38}$$

Since the system is already scaled, we simply choose weights Q and R identity. The solution of this optimal control problem can be easily obtained by the following numerical routine shown as Algorithm 8.

$P \leftarrow I_{n-1}$;
$P_0 \leftarrow \mathbf{0}^{(n-1)\times(n-1)}$;
while $||P - P_0|| \geq \epsilon$ **do**
  $P_0 \leftarrow P$;
  for $k = (N-1), \ldots, 0$ **do**
    $\hat{K}_k \leftarrow \left(R + \hat{B}_k^\top P \hat{B}_k\right)^{-1} \hat{B}_k^\top P \hat{A}_k$;
    $P \leftarrow Q + \hat{A}_k^\top P_k \hat{A}_k - \hat{A}_k^\top P_k \hat{B}_k \hat{K}_k$;
end

Algorithm 8: Solution of discrete periodic LQR

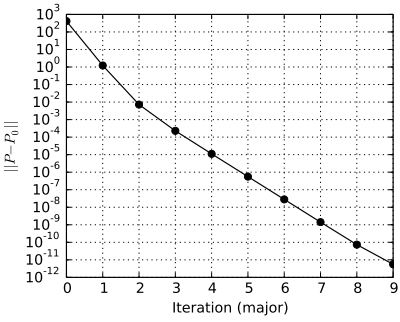
The procedure leads to linear, but fast convergence as shown in Figure 8.9a. Figure 8.9b shows the monodromy matrix characteristics for the nominal system, the reduced system and the system with LQR feedback. Note that the step from nominal to reduced system removes one multiplier of magnitude 1. The figure confirms further that a set of feedback gains $\hat{K}_\bullet$ was obtained that stabilises the periodic linearised system, since all closed-loop Floquet multipliers are stable.

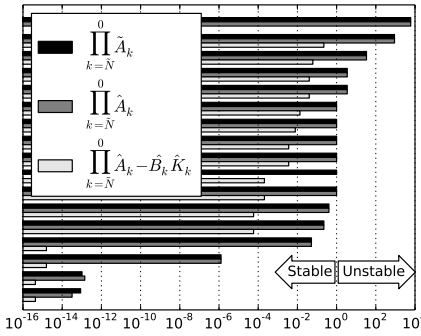
The resulting reduced space feedback matrices can be transformed to scaled full-space with:

$$\tilde{K}_k = \hat{K}_k Z_k^\top, \tag{8.39}$$

and to unscaled form with:

$$K_k = S_u \tilde{K}_k S_x^{-1}. \tag{8.40}$$





(a) Iterates of the algorithm. For each major iteration, the maximum of absolute values of $P - P0$ entries is shown.

(b) Depiction of Floquet multiplier magnitudes

Figure 8.9: Results of Algorithm 8.

8.4 Robustified optimal control

In this section, the time-optimal control problem is robustified using the Lyapunov framework. The crucial part of the formulation is the use of the DPLESolver algorithm for two main reasons. First, it eliminates P as decision variables, which was found to be a good idea in Chapter 7. Second, it drastically reduces the time-and memory footprint of the numerical solution.

The pseudo-inverse method of avoiding LICQ (Sternberg 2012a) for the covariance periodicity constraint can be cast in a form that the algorithm accepts:

$$\tilde{A}'_{\tilde{N}} = V\tilde{A}_{\tilde{N}} \tag{8.41}$$

$$\tilde{Q}'_{\tilde{N}} = V\tilde{Q}_{\tilde{N}}V^{\top}, \tag{8.42}$$

with $V = I_n - \frac{\partial C}{\partial \xi}(X_{k0})C(X_{k0})$. The effect of this transformation is to project the 1-eigenvalues in $\prod_{k=\tilde{N}}^{0} \tilde{A}_k - \tilde{B}_k\tilde{K}_k$ to zero in $\prod_{k=\tilde{N}}^{0} \tilde{A}'_k - \tilde{B}_k\tilde{K}_k$. The convergence of this approach will be given in Figure 8.14. However there are two related problems with this approach. First, for the chosen collocation scheme with its limited accuracy, the projection is not perfect. Rather than yielding a zero with machine precision, an eigenvalue with magnitude 1×10^{-6} is introduced, distorting other genuine eigenvalues of magnitude 1×10^{-5} in the process. Second, the external periodic Schur decomposition part of the DPLESolver algorithm uses a cutoff magnitude of eigenvalues for detection and treatment of singular modes. As the projection becomes more accurate by increasing the degree of the collocation polynomial, numerical issues arise as the eigenvalue approaches but never quite reaches this cut-off value.

Repairing these numerical issues is beyond the scope of this thesis, especially since two working alternative approaches exist that do not exhibit these issues.

One alternative is to introduce ad-hoc artificial dynamics (Gros 2012) that stabilise the norm constraint violation mode, e.g. by adding $-q(q^{\top}q - 1)$ to the right hand side of Equation (8.6). This addition does not influence the dynamics in the non-violation modes.

Proof. Write the augment quaternion dynamics as:

$$\dot{q} = E(q) - q(q^{\top}q - 1). \tag{8.43}$$

Work towards a first order Taylor expansion around a quaternion value $\bar{q}$ that satisfies the norm constraint $\bar{q}^\top \bar{q} - 1$:

$$\frac{d(\bar{q} + \delta q)}{dt} = E(\bar{q} + \delta q) - 2\bar{q}\delta q^\top \bar{q}, \tag{8.44}$$

and, noting that $\frac{d\bar{q}}{dq} = E(\bar{q})$:

$$\dot{\delta q} = \frac{\partial E}{\partial q}\delta q - 2\bar{q}\delta q^\top \bar{q}. \tag{8.45}$$

First, for perturbations δq compatible with the norm constraint, it holds that $2\delta q^\top \bar{q} = 0$, and hence the linearised augmented dynamics is unaltered for compatible modes.

Second, for perturbations that violate this constraint ($\delta q = \bar{q}\delta\alpha$ with α a scalar), the dynamics reads:

$$\bar{q}\dot{\delta\alpha} = \frac{\partial E}{\partial q}\bar{q}\delta\alpha - 2\bar{q}\delta\alpha. \tag{8.46}$$

Pre-multiplying with $\bar{q}^\top$ leads to:

$$\dot{\delta\alpha} = -2\delta\alpha, \tag{8.47}$$

from which the $\bar{q}^\top \frac{\partial E}{\partial q}\bar{q}$-contribution vanished since $\frac{\partial E}{\partial q}$ is skew-symmetric.

The violating mode $\delta\alpha$ is clearly stabilised by the proposed addition.

□

Our preferred approach is instead to work with the reduced linearised system, since it is easy to generalise and has the potential to reduce the dimensions of the Lyapunov solver:

$$\hat{P}_\bullet = \text{DPLESolver}(\hat{A}_\bullet - \hat{B}_\bullet \hat{K}_\bullet, \hat{Q}_\bullet). \tag{8.48}$$

Transforming this to non-reduced space:

$$\tilde{P}_k = Z_k \hat{P}_k Z_k^\top, \tag{8.49}$$

and scaling, we arrive at an expression for the covariance of the linearised original system:

$$P_k = S_x \tilde{P}_k S_x. \tag{8.50}$$

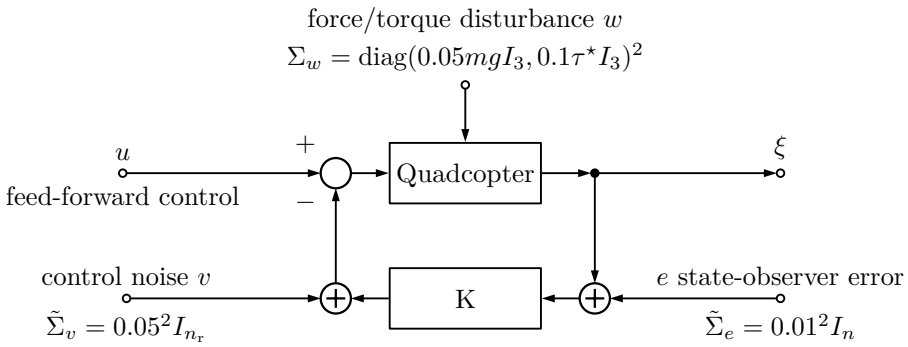


Figure 8.10: System diagram with noise inputs

From which we construct a margin to robustify the scalar path constraint $h(\xi)$:

$$h_k^{\mathrm{r}} = \gamma_h \sqrt{\frac{\partial h}{\partial \xi}(\xi_k) S_x Z_k \hat{P}_k (\hat{A}_\bullet - \hat{B}_\bullet \hat{K}_\bullet, \hat{Q}_\bullet) Z_k^\top S_x \frac{\partial h}{\partial \xi}^\top (\xi_k)}. \tag{8.51}$$

Similarly, we introduce robustifying margins for the control bounds:

$$\tau_k^{\mathrm{r}} = \gamma_\tau \sqrt{\hat{K}_k \hat{P}_k (\hat{A}_\bullet - \hat{B}_\bullet \hat{K}_\bullet, \hat{Q}_\bullet) \hat{K}_k^\top}. \tag{8.52}$$

For the noise entering the system, we choose a combination of different sources:

$$\hat{Q}_k = \hat{B}_k \tilde{\Sigma}_v \hat{B}_k^\top + \hat{D}_k \Sigma_w \hat{D}_k^\top + \hat{B}_k \hat{K}_k Z_k^\top \tilde{\Sigma}_e Z_k \hat{K}_k^\top \hat{B}_k^\top, \tag{8.53}$$

where $\hat{D}_k$ stems from a sensitivity of the system state w.r.t. to disturbance forces and torques acting on the platform, obtained by analogy to the derivation of $\hat{B}_k$ in Section 8.3.4. Figure 8.10 shows the system diagram that corresponds to this choice of disturbances, along with values for the noise covariances.

The proposed final robust time-periodic formulation reads:

$$
\begin{aligned}
&\underset{X_{\bullet\bullet}, U_\bullet, \hat{K}_\bullet, T, T_w}{\text{minimise}} && T && \text{Time optimality}\\
& && && \text{Regularise (penalise) ...}\\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||U_k - \tilde{u}||_2^2 && \text{control excursions}\\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,r} - \tilde{r}||_2^2 && \text{rotor speed excursions}\\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,\omega}||_2^2 && \text{rotation of body}\\
& && +\epsilon \textstyle\sum_{k=0}^{\tilde{N}} ||X_{k0,q} - (0,0,0,1)||_2^2 && \text{deviation from upright pose}\\
& && +1 \times 10^{-3} \textstyle\sum_{k=0}^{\tilde{N}} ||\hat{K}_k - \hat{K}_k^{LQR}||_2^2 && \text{deviation from LQR}\\
&\text{s.t.} && 0 = g\left(t_{kj}; S_x X_{kj}, \tfrac{S_x}{\lambda(k)} \dot{X}_{kj}, S_u U_k\right) && \begin{cases} k = 0 \dots \tilde{N} \\ j = 1 \dots d \end{cases} && \text{Collocation}\\
& && 0 = X_{(k+1)0} - \textstyle\sum_{i=0}^{d} X_{ki} L_i(1) && k = 0 \dots N-2 && \text{Coupling}\\
& && 0 = X_{00} - X_{\tilde{N}d} + \tfrac{\partial C}{\partial \xi}(X_{00})^\dagger C(X_{00}) && && \text{Periodicity}\\
& && 0 = S_x X_{00,p} - p_A && && \text{Waypoint A}\\
& && 0 = S_x X_{\frac{N}{2}0,p} - p_B && && \text{Waypoint B}\\
& && 0 \;\; \leq h(X_{k0}) - h_k^{\mathrm{r}} && k = 0 \dots \tilde{N} && \text{Robust obstacle}\\
& && \quad U_k + \tau_k^{\mathrm{r}} \leq S_u^{-1} 2\tau^\star && k = 0 \dots \tilde{N} && \text{Robust max. torque}\\
& && \quad U_k - \tau_k^{\mathrm{r}} \geq 0 && k = 0 \dots \tilde{N} && \text{Robust min. torque}\\
& && 0 \;\; \leq T_w \leq T \leq T_f && && \text{Chronology}\\
& && -4 \leq X_{k0,p} \leq 4, && k = 0 \dots \tilde{N} && \text{Confine to cube}
\end{aligned}
\tag{8.54}
$$

with $\gamma_h = 1$, $\gamma_\tau = 0.15$ and $\epsilon = 1 \times 10^{-2}$. This formulation is not suitable for algorithms using an exact Hessian, as the Hessian would be fully dense.

In this formulation, we have eliminated $Z_\bullet$ to reduce the number of decision variables, by using a closed-form expression of a QR decomposition. Such expression in general is only piecewise differentiable, and hence violates the assumption of Lipschitz continuity underlying convergence proofs of common nonlinear problem solvers. In practice, the convergence of this form was not found to exhibit convergence problems when compared to a full-space approach with artificial stabilising dynamics (plot follows in Figure 8.14).

The constraint Jacobian has some dense rows, as can be seen in Figure 8.11, one for each robustified constraint at each discretisation step, forming an approximately dense block of order $N \times [(d+1)n + nn_{\mathrm{r}}]$. Combined with the dense column resulting from the time parameter, this Jacobian structure is

	natural	largest first	dynamic largest first	smallest last	incidence degree	random
explicit	180/849 48.01	905/49 52.21	201/844 53.58	210/49 50.88	902/49 50.94	180/848 59.41
explicit modified	936/2205 193.77	936/2204 230.26	936/2204 201.2	936/2204 215.04	936/2204 229.66	936/2204 205.57
implicit	1200/1 0.3	195/47 12.89	224/15 0.85	195/180 16.15	195/179 15.24	1200/1 0.47
implicit greedy	1200/1 0.28	960/925 39.52	1200/78 2.38	960/1104 53.98	960/1055 51.1	1200/1 0.51

Table 8.1: Comparison of ColPack colouring strategies for the constraint Jacobian. For each algorithm (vertical) and ordering (horizontal) combination, we report the number of forward/adjoint seeds and colouring computation time in seconds.

an excellent showcase for bidirectional colouring. Table 8.1 reports on the performance of several colouring strategies.

Incidentally, in the specific problem at hand, the DPLESolver node is by far the most expensive part of the computational graph. Given that the number of constraints that depend on this node is small, but the node itself depends on almost all variables, simply using only adjoint seeding will outperform conventional uni- or bidirectional colouring strategies. Colouring would only be beneficial if performance-metrics could be used as weights. Such weighting is at present not possible in off-the-shelf colouring packages like ColPack.

A first variation of the robust formulation, in which $\hat{K}$ is kept fixed at the values produced in the Section 8.3.4 ($\hat{K}^{LQR}$), convergences fast and cleanly as can be seen on the top row of Figure 8.12. As expected, in Figure 8.13, an optimal period T larger than that for the purely time-optimal problem of Section 8.3.3 can be seen. The second variation of the robust formulation has $\hat{K}$ as free decision variables. This extra degree of freedom allows the optimal period to shrink significantly, almost to the level of the unrobustified problem.

The convergence of the second variation is observed to be slower. Before regularisation on K was introduced in the formulation, its convergence was problematic, with little progress over several thousands of iterations. This led to the study of a simplified Lyapunov problem in Appendix A.3 exhibiting similar convergence difficulties. However, a promising matrix product scaling technique from that section with $\Lambda_k = \text{chol}(\hat{B}_k^\top \hat{B}_k)^{-T}$ did not noticeably improve convergence.

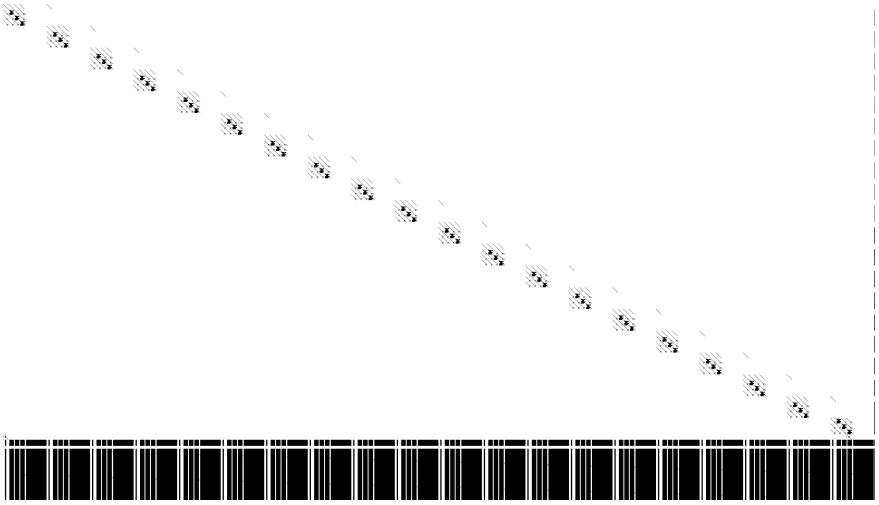


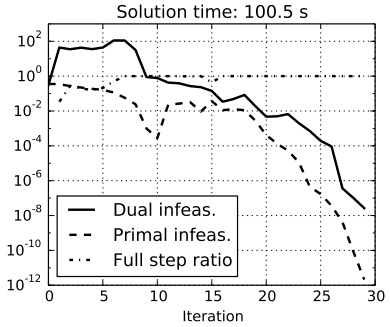
Figure 8.11: Constraint Jacobian structure: 1548 constraints, 2724 variables, 411859 non-zeros in Jacobian.

Comparing the `IPOPT` and `SNOPT` approach in Figure 8.12, one can see that convergence for `SNOPT` is cleaner and faster in terms of iterations. Since the number of QP iterations in each SQP step quickly reaches 1, one can expect the computational time per step to be similar for both solvers, and hence a faster total solution time for `SNOPT`. Still, because `IPOPT` is using a high-performant external linear solver (`ma57`, HSL 2011), and `SNOPT` an internal less performant solver, the expected trend is not visible.

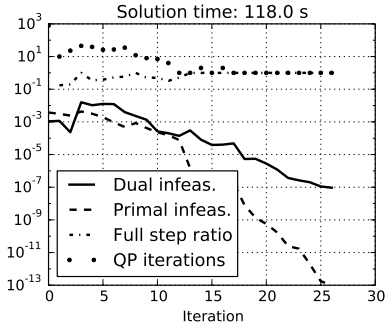
The feed-forward control inputs of the converged trajectories in Figure 8.12 are seen to keep clear from the bounds since a robustifying margin, given in Equation (8.52), is present. In the same plot, sections with active robustified control constraints are highlighted by triangle symbols.

The converged trajectory of the full robustified problem in Figure 8.13d has the interesting property that the uncertainty ellipsoids near the active bounds are squeezed in the directions normal to these constraints. The ability to change $\hat{K}$ is exploited by the optimiser to morph the initially quasi-spherical ellipsoids into an optimal shape. The physically meaningful noise inputs of Equation (8.53) (as opposed to a multiple of the unit matrix), together with the robustified control constraints, prevent the optimiser to simply choose $\hat{K}$ as to make $\hat{A} - \hat{B}\hat{K} = 0$ and ellipsoids shrink to points. The optimiser further exploits the fact that we only constrained on the control interval boundaries, artificially shrinking the effective margin by moving these boundaries. A finer sampling of constraints is

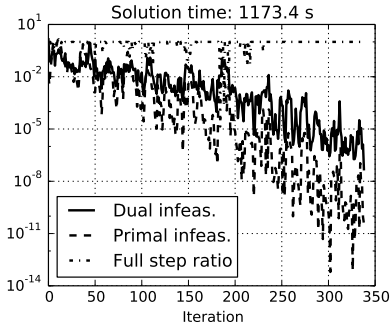
needed to mitigate this behaviour in a real-world application.



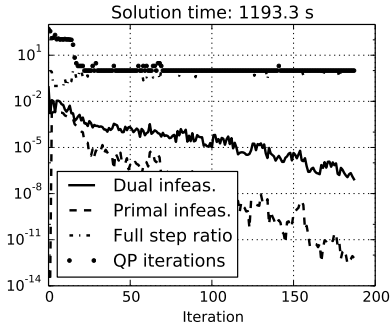
(a) `IPOPT`, with K fixed



(b) `SNOPT`, with K fixed



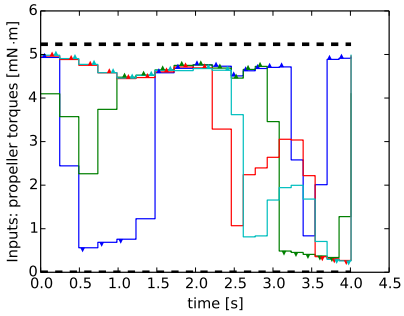
(c) `IPOPT`, with K free



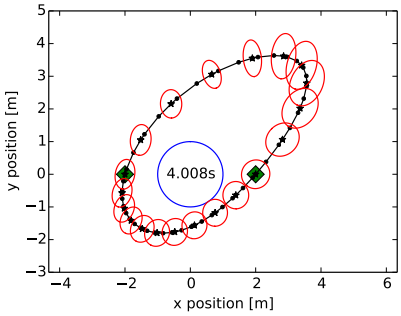
(d) `SNOPT`, with K free

Figure 8.12: Convergence of robustified problem for different solvers. Both solvers converge to the same optimum.

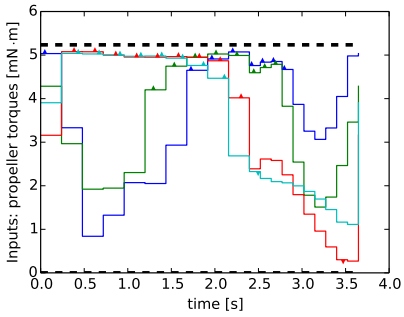
Figure 8.15 breaks down the computational cost of solving Formulation (8.54) with `CasADi`. The circles represent `CasADi` functions in three flavours: solid border for matrix-valued graph functions (`MXFunction`), dotted border for scalar-valued graph functions (`SXFunction`), and no border for other types of functions. Arrows between the circles depict the call dependency between the functions. At the center of each circle is a square with an area that is proportional to the total time spent in evaluating the corresponding function. This square is broken down into a treemap of components of the function algorithm, with each rectangular area scaling with the amount of time spent in the corresponding component. White patches correspond to calls made to other functions, and lightly shaded patches correspond to time spent in



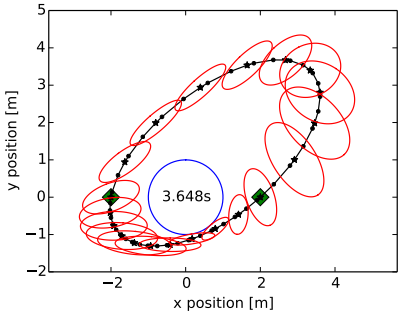
(a) Feed-forward control inputs, with K fixed



(b) Flight path, with K fixed



(c) Feed-forward control inputs, with K free



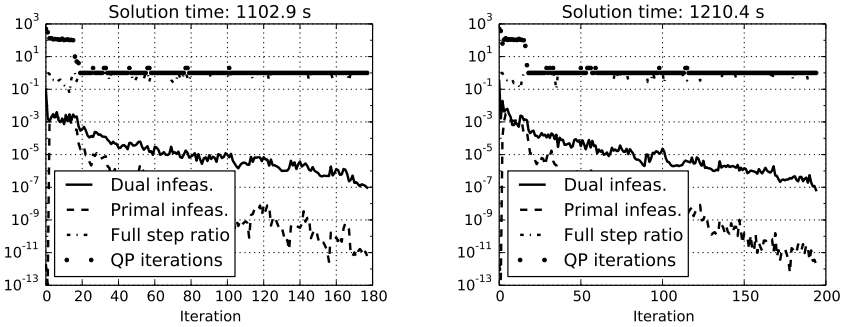
(d) Flight path, with K free

Figure 8.13: Converged trajectories of robustified problem

the algorithm that is not attributable identified components of the algorithm: overhead. Note that this overhead is still part of the evaluation costs, and not of the initialisation costs as was the case in Section 7.5. If one focuses on the shaded patches, one gets a visual impression of the total evaluation time and its attribution to various components of the full formulation.

We can make the following observations from a careful inspection of this diagram:

- For the NLP solver (see the circle labelled `nlp` at the bottom), the majority of time is spent in evaluating the constraint Jacobian (`jac_g`), and a rather small portion is spent in the overhead (on the order of 10 seconds). In this case, the overhead is the actual `IPOPT` main loop, which consists mainly of the sparse $LDL^\top$ factorisation and solution.



(a) Using stabilised dynamics

(b) Using pseudo-inverse projection (result is not accurate)

Figure 8.14: Convergence of alternative (full-space) formulations: `SNOPT`, K free

- The constraint Jacobian (`jac_g`) is a large matrix-valued graph function. Surprisingly, most time is spent in overhead of the `MX` virtual machine and in sparse matrix-matrix multiplication (`matmul`).
- The work that the constraint Jacobian function delegates consists roughly of two equal parts: a part related to computing adjoints of the collocation constraints and the linearised dynamics $\tilde{A}_\bullet, \tilde{B}_\bullet$ (`adj(resf)`), and a part related to the `slicot` embedded Lyapunov solver.
- The embedded Lyapunov solver (`slicot`) spends a large share of its evaluation time on overhead, in this case linear algebra cf. our earlier observations in Section 5.5.1.
- The total time spent in the periodic Schur decomposition (`schur`), and in solving the low-order discrete periodic Sylvester subproblems (`solve`) is small: both on the order of one second.
- The functions on the diagram that we have not discussed above have a negligible impact on total evaluation cost and hence are be omitted from the present discussion.

Lastly, we make a comparison with the computational diagram in Figure 8.16 of a *classic* problem formulation of the robust quadcopter flight: not using an embedded Lyapunov solver and propagating the covariance in continuous time with the same collocation integrator as the nominal states. To make the

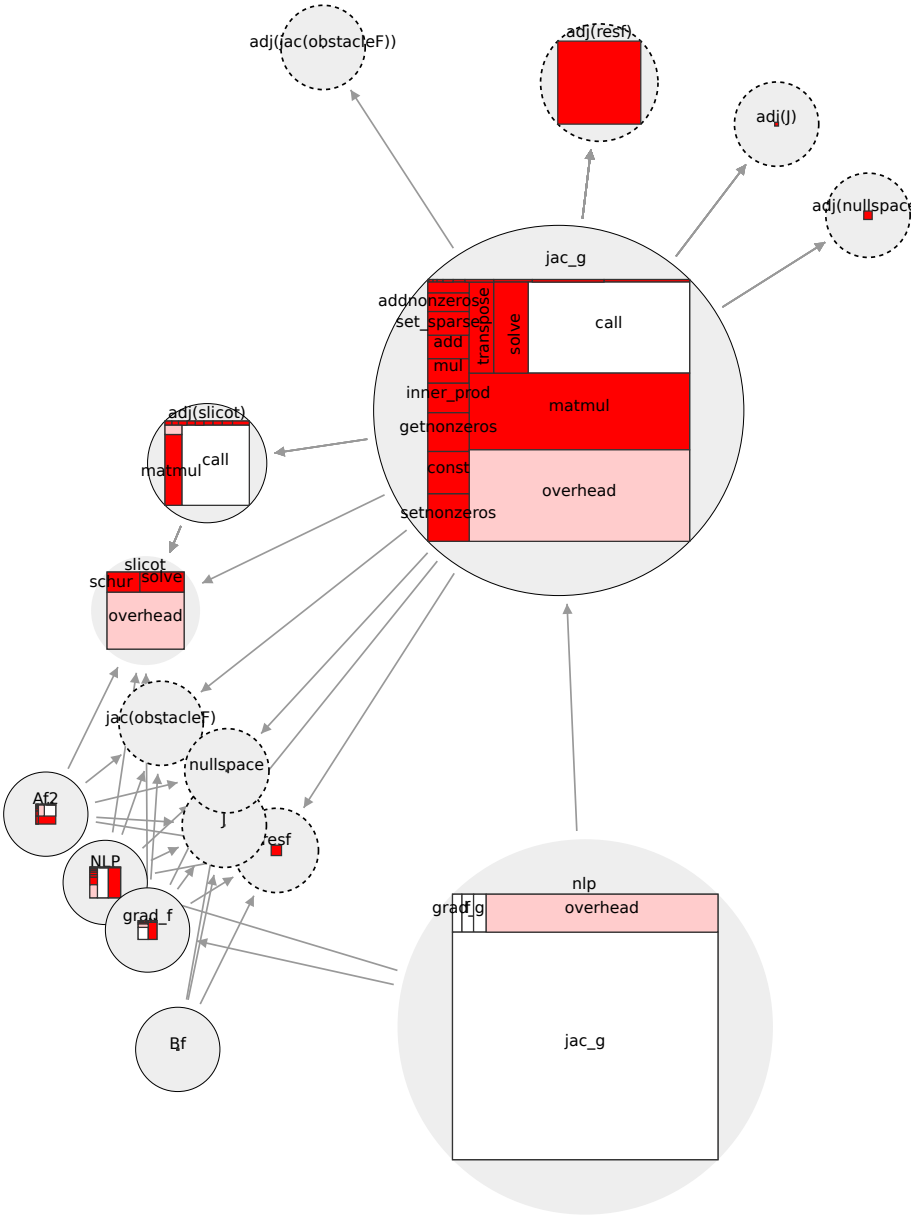


Figure 8.15: `CasADi` computational diagram of robust OCP problem with `IPOPT` and K fixed.

comparison meaningful, the NLP main loop was interrupted prematurely[1] after 29 steps i.e. the number of steps it took for the formulation of Figure 8.15 to converge.

We can make the following observations from a careful inspection of this diagram:

- The distribution of evaluation time is very simple in this case: the vast majority of time (on the order of 2000 seconds) is spent in the `IPOPT` main loop, which consists mainly of the sparse $LDL^\top$ factorisation and solution.
- A smaller fraction of evaluation time (around 200 seconds) is spent in evaluating the derivative of the collocation constraints (`fwd(resf)`)
- Comparing this diagram to the previous one, drawn on the same scale in the lower right corner, identifies the practical progress achieved with the formulations of this thesis: the time spent in the `IPOPT` main loop has been dramatically reduced. The computational bottleneck has shifted away from linear system factorisation towards overhead of the `MX` virtual machine and sparse matrix-matrix multiplication.
- It should be noted that the CasADi `MX` virtual machine has much improved over the course of this thesis in terms of both speed and memory requirements. Early attempts to use the classic problem formulation on the quadcopter had larger runtimes than reported here.

[1] In fact, in spite of major efforts, the quadcopter application was never successfully solved with a classic problem formulation.

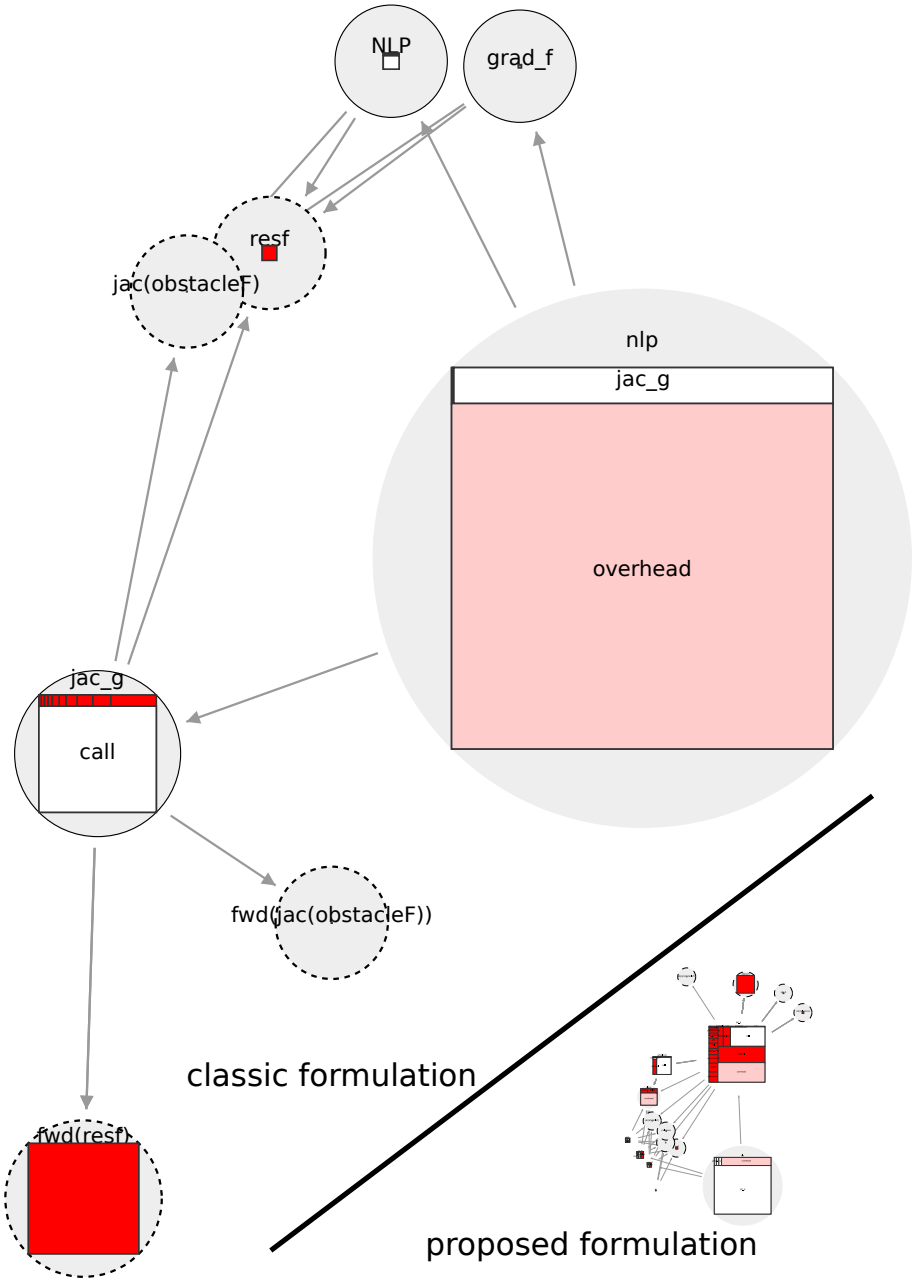


Figure 8.16: `CasADi` computational diagram of robust OCP problem with `IPOPT` and K fixed and without embedded Lyapunov solver. This figured is zoomed out by a factor of 4 in both dimensions compared to Figure 8.15.

Conclusion

In this chapter we demonstrated the main formulation of this thesis: robust periodic optimal control using discrete Lyapunov equations which are eliminated from the transcribed NLP and solved by efficient embedded Lyapunov solvers.

For a challenging 17-state nonlinear dynamic system with invariants, we demonstrated the concurrent design of a periodic trajectory, feed-forward and time-varying linear feedback control which are robust with respect to geometric path constraints under physically relevant disturbances.

Several elements contributed to the success of this process:

Challenge	Answer
DAE nature of model	Direct collocation method
Ill-scaling	Scaling of decision variables
Ill-conditioning of BFGS Hessian	Regularisation terms in objective
Non-convexity	Homotopy from simple to complex
Invariants in dynamics	Exploitation by covariance propagation on reduced space cf. Chapter 3
Convergence issues for robustification	Elimination of Lyapunov matrix as decision variables cf. Chapter 5 and Chapter 7
Large state space size	`slicot` solver of Chapter 5
Complex/non-standard problem formulation	`CasADi` environment cf. Chapter 6
Convergence issues with K free	Regularisation cf. Appendix A.3

In terms of physical interpretation of results, it was remarkable how setting the feedback matrix free allowed the optimiser to squeeze the uncertainty in the direction perpendicular to the obstacle and recover a lot of time-optimality that was lost when going from nominal flight-trajectory to the robustified one.

The incorporation of linear feedback into the robust optimal control formulation was demonstrated in Houska 2009, 2011b for a trivial 2-state system. The current chapter represents the first application to a non-trivial system, and the first application to make use of the efficient robust optimal control formulation. The embedded Lyapunov formulation was an improvement of at least an order of magnitude in terms of computation time per NLP step. Moreover, only the formulation with eliminated Lyapunov matrices lead to successful convergence of the NLP.

While the system model and noise inputs are physically inspired, the application

should be seen as an algorithmic and numerical *tour de force* rather than a real-world ready show-case for robust periodic control. Notably, the robustness factor γ_τ does not have a meaningful value, and we did not study how the optimised system behaves in a simulated environment with simulated disturbances.

Chapter 9

Conclusion and Outlook

This thesis started from the Lyapunov framework in continuous time for approximate robust periodic optimal control of nonlinear systems, which was recently proposed (Houska 2007) and demonstrated for low-dimensional (up to 5 state) systems (Houska 2007, 2009; Logist 2011a).

The aim of this thesis was to explore alternative formulations for this framework, as to extend its domain of application for large-dimensional systems, and make the technique practical.

9.1 Main conclusions and contributions

Part I: Formulations and Methods The existing Lyapunov framework as a mean to robustify optimal control problems was laid out in Chapter 3. For systems with invariants arising from modelling with non-minimal coordinates, a formulation was proposed that improves on the work by Sternberg 2012a in that it reduces the computational effort by shrinking the effective state space dimensions.

A classical approach to solving the robust OCP is to augment state space with the Lyapunov matrix entries and to use a direct method (Houska 2007, 2009; Logist 2011a) in which the periodic Lyapunov differential equations (PLDE) end up as dynamic constraints. Following the suggestion in Houska 2007, we dropped in Chapter 4 the positive definiteness constraint on P in order to conform to a generic NLP formulation. We remarked that this can only work if a positive definite initial guess is provided, and if the used integration scheme

guarantees preservation of positive-definiteness. One such simple scheme was created by applying the discrete variant of the Lyapunov framework directly on the *discrete* nominal OCP. The scheme was worked out for a direct collocation method, which has not been used to solve robust OCPs in literature so far, and demonstrated on a simple example, leading to a CDC contribution (Gillis 2013).

The novel approach of treating the covariance propagation for robust OCP in a discrete fashion, led to the proposal in Chapter 5 to eliminate P as decision variables, allowing for dedicated Lyapunov solvers from the literature to take over the responsibility from the NLP solver to make the discrete periodic Lyapunov equations (DPLE) consistent. A benchmarks study supported that for BFGS-type optimisation, this elimination technique has the potential to drastically reduce the amount of computational effort needed to solve robust OCPs with large-scale state dimensions, bringing complexity from $O(n^6N)$ to $O(n^3N^2)$ with n the number of states and N the horizon length.

The main contribution of this part is to formulate the robust OCP in such a way that the discrete periodic Lyapunov equations appear, and to eliminate these from the NLP by embedding well-known efficient DPLE solvers in the general optimal control setting as an infinitely-differentiable entity.

Part II: Software and Applications The second part of this thesis started with Chapter 6, providing an extensive introduction to the open-source framework CasADi, which can be seen as mixed outcome of the present thesis and the thesis of a colleague Andersson 2013. In particular, contributions to speed of initialisation (hierarchical sparsity recovery – Gillis 2014a) and speed of development (structured indexing) were highlighted. These contributed to making CasADi a modular and flexible framework for quick, yet highly efficient implementation of optimal control formulations, in which the extension for DPLE solvers for robust OCP fitted naturally.

Chapter 7 presented a major application of robust OCP in the domain of airborne wind-energy. It used the Lyapunov framework in a steady-state setting, with the algebraic Lyapunov equation, to find optimally safe regimes to operate the carousel launching device. Similar to work by Logist 2011a, a stochastic safety margin was used as an objective. The carousel application is the first for a steady-state operation and for a nonlinear system with invariants and resulted in an IFAC publication (Gillis 2014b). An extensive study of convergence was included in this chapter showing that formulations with P eliminated improve convergence. This supplements the conclusion from Part I that such formulation is beneficial for computational efficiency.

Chapter 8 presented the challenging quadcopter application involving the design

of a time-varying linear feedback controller within the Lyapunov framework. At the start of this doctorate, the application was intractable: overflooding the memory capability of a standard workstation (4 GB), and large computation times for the NLP steps (minutes), and with the NLP solver not successfully converging. At the end of this thesis, by virtue of advances in formulation, methods and solvers, the application is easily treatable. The Lyapunov framework is now deemed ready for engineering practice, especially since modular building blocks contributed to CasADi make formulation easy.

The main contribution of this part is the extension of an existing software package, and the demonstration of the Lyapunov framework on challenging applications.

9.2 Recommendations

The Lyapunov framework in general The target for the framework are applications in which periodic optimal control is currently employed in a nominal way for the purpose of system design, for trajectory generation or for the off-line design of time-varying controllers. The Lyapunov framework offers a relatively easy and intuitive extension to make path constraints or objective value approximately robust to the influence of stochastic disturbances by automatically computing safety margins. The approximation works when the nonlinearity is locally small, excluding notably hysteresis effects.

Much like economic MPC holds the promise for more profit by moving away from tracking criteria to more meaningful objectives, the Lyapunov framework can improve optimality: by operating closer to bounds if ad-hoc conservative safety margins are replaced by meaningful approximations, or by altering the objective. Consider the example of time-optimal control of race cars: instead of tracking a target in the geometric center of the road as in Verschueren 2014, one could simply have time as an objective and rely on the Lyapunov estimate to steer clear from corners and slip-force constraints.

Formulation with embedded Lyapunov solver The proposed formulation is an obvious candidate to try out when the classic approach fails on one of several accounts:

- when the NLP has difficulty converging. When the nominal periodic OCP is challengingly non-convex, the NLP solver may walk away from the nominal initial guess when adding the Lyapunov form and never find a

feasible solution. The embedded formulation has been observed to suffer less from this problem.

- when the NLP steps are slow due to a high number of states.
- when the workstation runs out of memory due to fill-in of the constraint Jacobian from the Lyapunov dynamic constraints.

The CasADi framework The framework is intended for users that know how optimisation and optimal control works. It is recommended for researchers that wish to debug formulations, explore new formulations, or have to deal with forms of optimal control problems that are not standard. Frameworks with a black-box interface may be more appropriate when there are no such needs.

9.3 Outlook

While CasADi is a valuable tool for a researcher, it is the optimal-control tools built on top of it (such as JModelica, dynobud) that provide a black-box optimal control problem solver for the convenience of practioning engineers. The results of this thesis remain to be integrated into such a high-level tool.

One route that has not been explored in this thesis is the use of a special type of NLP solver that handles convex constraints: sequential convex programming. For the case of small-scale systems with long horizon, for which the proposed elimination formulation is not beneficial, such approach may improve convergence with respect to the existing practice.

One formulation that has not been investigated, but fits in the presented framework, is the combined design of a state-estimator and a time-invariant controller.

As discussed in Section 6.4, there is a natural extension to the proposed hierarchical sparsity detection algorithm: support for bidirectional colouring. Similar to the speed-up for time-optimal control problems, this extension will decrease the complexity of initialisation time for robust time-optimal control with embedded Lyapunov solvers.

Also, the evaluation time of the Jacobian of problems with this structure may benefit from a bidirectional colouring approach. The practical benefit may be less visible, however, as some parts of the computational graph may dominate the total evaluation time, as discussed in Section 8.4. There is nevertheless room for improvement in this direction.

Appendix A

Appendix

A.1 Proof of covariance projection proposition

Proof of Proposition 3.1.1.

Proof. The set of points on the boundary of the tolerance region is given by:

$$\{sWx | x^\top x = 1\}.$$

The set of points obtained by projecting the boundary amounts to:

$$\{sv^\top Wx | x^\top x = 1\},$$

which has scalar dimension.

The supremum of this one-dimensional set can be obtained by:

$$\begin{aligned} \underset{x}{\text{minimise}} \quad & -s^2 \left(v^\top Wx\right)^2 \\ \text{subject to} \quad & xx^\top = 1. \end{aligned}$$

The objective can be rewritten as:

$$s^2 v^\top Wxx^\top W^\top v = s^2 w^\top xx^\top w,$$

with $w = W^\top v$. So we end up with the following problem:

$$\begin{aligned} \underset{x}{\text{minimise}} \quad & -s^2 w^\top x x^\top w \\ \text{subject to} \quad & x x^\top = 1. \end{aligned}$$

This problem is identified as a maximum eigenvalue problem of the symmetric rank-1 matrix $-s^2 w w^\top$:

$$\begin{aligned} \underset{x}{\text{minimise}} \quad & -x^\top s^2 w w^\top x \\ \text{subject to} \quad & x x^\top = 1, \end{aligned}$$

with the solution $-s^2 w^\top w$.

The extremum of the one-dimensional set resultant from projection the tolerance region hence has the following value:

$$s^2 w^\top w = s^2 v^\top W W^\top v = s^2 v^\top \Sigma v,$$

which proves the proposition. □

A.2 Derivation of the sensitivity matrix evolution equation

Consider the autonomous dynamic system $\dot{x} = f(x, p)$ with $p \in \mathbb{R}^m$. We define sensitivity w.r.t to the parameter p as:

$$S_p(t) \triangleq \frac{dx(t)}{dp} \tag{A.1}$$

We wish to derive an evolution equation for $S_p(t)$:

$$\dot{S}_p(t) = \frac{d\dot{x}(t)}{dp} = \frac{df(x,p)}{dp} = \frac{\partial f(x,p)}{\partial x}\frac{dx(t)}{dp} + \frac{\partial f(x,p)}{\partial p}, \tag{A.2}$$

which simplifies to:

$$\dot{S}_p(t) = \frac{\partial f(x,p)}{\partial x} S_p(t) + \frac{\partial f(x,p)}{\partial p}. \tag{A.3}$$

In this case, consider the dependence on p to be solely through initial conditions:

$$\dot{S}_p(0) = \frac{\partial x(0)}{\partial p} = I, \tag{A.4}$$

and $\frac{\partial f(x,p)}{\partial p} = 0$. This corresponds to Equation (2.5).

A.3 Convergence study for stability optimisation

In this section, we analyse the convergence behaviour of a simple NLP that identifies a feedback gain while optimising a Lyapunov metric. The purpose of this analysis is to partially explain bad convergence of the final robustifying problem in Chapter 8. It turns out that a condition number of a simple quantity determines the convergence behaviour for BFGS-class solution methods.

Given a fixed discrete system matrix $A \in \mathbb{R}^{n\times n}$ and a nonsingular full-state feedback control matrix $B \in \mathbb{R}^{n\times n}$, we construct an unconstrained NLP to find the feedback $K \in \mathbb{R}^{n\times n}$ that minimises the trace of the Lyapunov matrix under identity noise injection:

$$\begin{array}{ll} \underset{K}{\text{minimise}} & \operatorname{tr}(P) \\ \text{s.t.} & P = (A+BK)P(A+BK)^\top + I_n, \end{array} \tag{A.5}$$

where P is eliminated with a periodic Lyapunov solver with periodicity $N = 1$. The result is an unconstrained NLP with an objective that is nonlinear in the decision variable K. We will further also consider second order (quadratic) and fourth order (quartic) Taylor approximations to the objective.

The solution of the above NLP is trivially given by $P = I$ and $A + BK = 0$, i.e. perfect pole placement. The question is not *what* the solution is, but *how* BFGS-type methods converge to it and what combination of problem parameters A,B and K_0 predicts the ease of convergence.

Section A.3.1 attempts to establish a correlation between convergence behaviour and condition number of the Lagrange Hessian at the optimum. The link between condition number and problem parameters is laid out in Section A.3.2, while solutions for ill-conditioning are proposed and tested in Section A.3.3.

A.3.1 Convergence and condition number

The first part of the analysis consists of constructing a random set of problem instances, where we make sure that A and $A + BK_0$ correspond to stable discrete systems, i.e. poles in the interior of the unit circle. While solving these instances of Equation (A.5) to high precision (`tol`$=1 \times 10^{-12}$) with `IPOPT` using BFGS, we retain those instances where P is positive definite at every iteration. After this pruning step, all instances are solved with the simple `SQPMethod` to avoid `IPOPT` trickery to introduce artefacts in the data. The precision is again 1×10^{-12}, and a minimal step size of 1×10^{-10} is enforced. Finally, a reference solution with an exact Hessian approach provides us with the Lagrange Hessian $H^\star$ at the optimal solution.

From Figure A.1, it is clear that ill-conditioning of $H^\star$ correlates with bad convergence. We suspect that the BFGS-process has difficulty approximating the ill-conditioned Hessian. The BFGS approximation error is blown up by the conditioning, resulting in bogus search directions. Indeed, solving the same problem instances with an exact Hessian reveals no sensitivity to the condition number.

However, feeding the ill-conditioned quadratic approximation to a BFGS optimiser does not by itself impede fast convergence. In Figure A.2, we show the convergence of a particularly badly performing instance and that of its

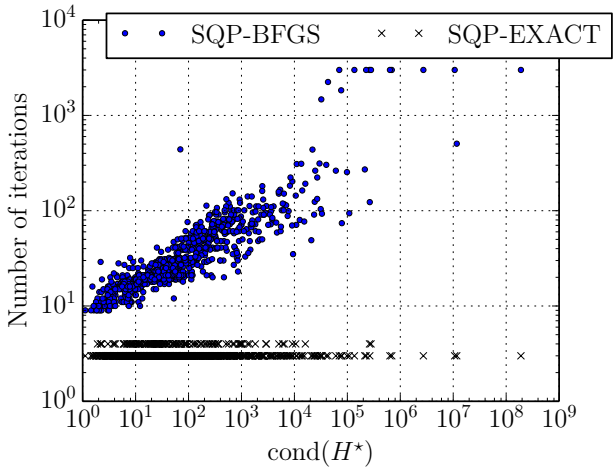


Figure A.1: Convergence properties for random problem instances.

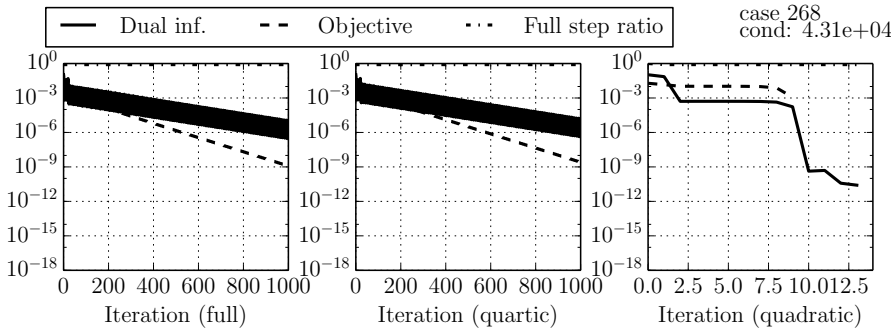


Figure A.2: Convergence of a typical ill-conditioned problem instance and its approximations

quadratic and quartic approximation (note that all cubic terms where found to be numerically zero). It is clear that the quartic terms have an influence on convergence. We simply conclude at this point that the presence of quartic terms has the potential to make a BFGS-method struggle to find the optimum of an ill-conditioned quadratic term. Note that this conclusion is vague on purpose, since some atypical cases can be found (Figure A.3).

A.3.2 Cause of ill-conditioning

The next part of the analysis is identifying which problem parameters affect the Hessian condition number.

Lemma A.3.1. *The Hessian at the optimal point is simply given by* $I_n \otimes 2B^\top B$.

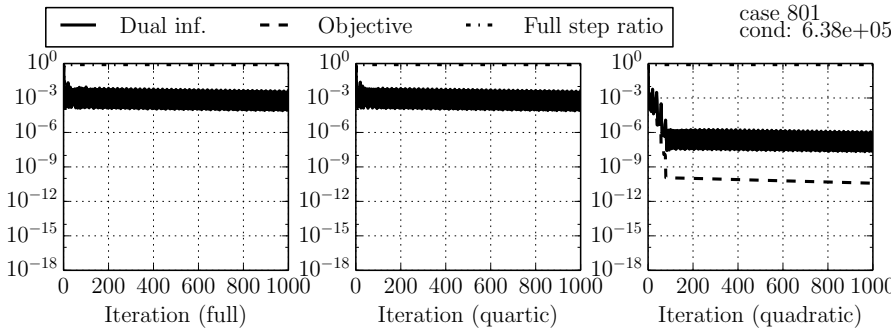
Proof. The discrete algebraic Lyapunov relation underlying Problem (A.5) reads:

$$P = \mathcal{A}P\mathcal{A}^\top + I, \tag{A.6}$$

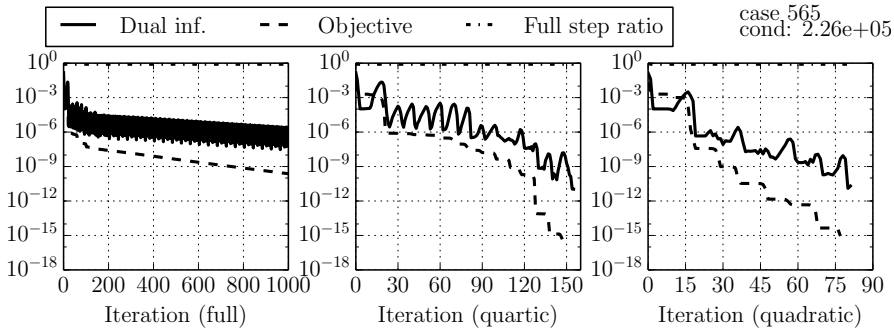
with $\mathcal{A} = A + BK$ which is to be evaluated at $\mathcal{A} = 0$.

Perform forward mode AD on this to obtain:

$$\dot{P} = \dot{\mathcal{A}}P\mathcal{A}^\top + \mathcal{A}\dot{P}\mathcal{A}^\top + \mathcal{A}P\dot{\mathcal{A}}^\top. \tag{A.7}$$



(a) A case of bad convergence even with a quadratic approximation



(b) A case where the quartic approximation is not sufficient to replicate bad convergence

Figure A.3: Convergence of some atypical instances.

Perform forward mode again:

$$\begin{aligned}\ddot{P} = &\cancel{\ddot{\mathcal{A}}P\mathcal{A}^\top} + \cancel{\dot{\mathcal{A}}\dot{P}\mathcal{A}^\top} + \dot{\mathcal{A}}P\dot{\mathcal{A}}^\top \\ &+ \cancel{\dot{\mathcal{A}}\dot{P}\mathcal{A}^\top} + \cancel{\mathcal{A}\ddot{P}\mathcal{A}^\top} + \cancel{\mathcal{A}\dot{P}\dot{\mathcal{A}}^\top} \\ &+ \dot{\mathcal{A}}P\dot{\mathcal{A}}^\top + \cancel{\mathcal{A}\dot{P}\dot{\mathcal{A}}^\top} + \cancel{\mathcal{A}P\ddot{\mathcal{A}}^\top},\end{aligned} \tag{A.8}$$

where we have struck out some terms that vanish due to $\mathcal{A} = 0$.

The resulting forward-over-forward AD mode, bearing in mind that $P = I$ and $\dot{\mathcal{A}} = B\dot{K}$, reads:

$$\ddot{P} = 2B\dot{K}\dot{K}^\top B^\top. \tag{A.9}$$

Correspondingly, the Hessian components are:

$$H^{\star}_{ij} = 2\mathrm{tr}\left(Be^{i}(e^{j})^{\top}B^{\top}\right), \tag{A.10}$$

where $e^{i} \in \mathbb{R}^{n\times m}$ is all-zero except for a one at the $i-$th flattened location. Using a column-major ordering, using Einstein notation, and with δ_{ij} the Kronecker delta function, we have:

$$e^{i}_{ab} = \delta_{a(i \bmod n)}\delta_{b\lfloor \frac{i}{n} \rfloor} \tag{A.11}$$

The Hessian components, in Einstein notation become:

$$H^{\star}_{ij} = 2\mathrm{tr}\left(B_{ab}\delta_{b(i \bmod n)}\delta_{c\lfloor \frac{i}{n} \rfloor}\delta_{d(j \bmod n)}\delta_{c\lfloor \frac{j}{n} \rfloor}B_{ed}\right) \tag{A.12}$$

$$= 2B_{ab}\delta_{b(i \bmod n)}\delta_{c\lfloor \frac{i}{n} \rfloor}\delta_{d(j \bmod n)}\delta_{c\lfloor \frac{j}{n} \rfloor}B_{ad} \tag{A.13}$$

$$= 2B_{a(i \bmod n)}\delta_{\lfloor \frac{i}{n} \rfloor \lfloor \frac{j}{n} \rfloor}B_{a(j \bmod n)} \tag{A.14}$$

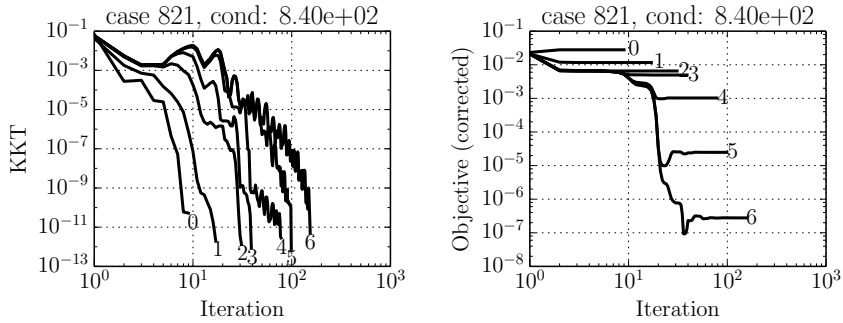
$$= 2(B^{\top}B)_{(i \bmod n)(j \bmod n)}\delta_{\lfloor \frac{i}{n} \rfloor \lfloor \frac{j}{n} \rfloor}, \tag{A.15}$$

which is equivalent to the expression that needed to be proven. □

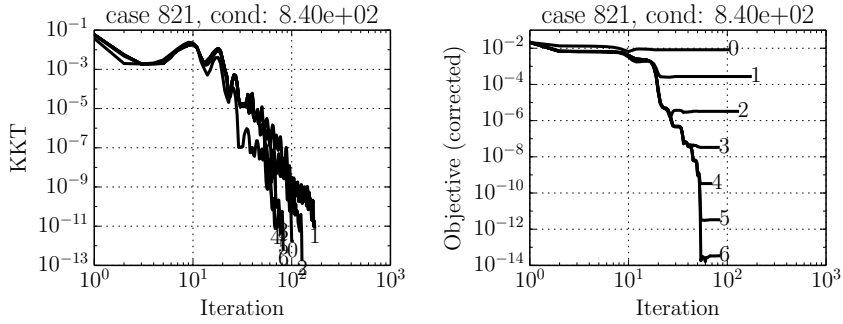
A.3.3 Treating ill-conditioning

Regularisation The most direct approach to avoid ill-conditioned Hessians is adding small artificial quadratic terms with positive curvature to the NLP objective, i.e. regularising the problem. The difficulty in finding a practical magnitude of regularisation is often to make a trade-off between loss of optimality and convergence behaviour. Figure A.4 shows the effect of two types of regularisation on both criteria. The first type clearly enhances speed of convergence towards an optimum. However, at no point during the iterations is the distance towards the true optimum smaller than for the unregularised iterations. In other words, in the current case study, one obtains a better result by cutting short the poorly advancing unregularised problem iterations than by adding any amount of regularisation.

Scaling Another approach to treat ill-conditioned Hessians is variable scaling. We consider two types of scaling. For scaling type 1, the feedback term has an element-wise product: $B\left[\Lambda \circ \tilde{K}\right]$. For Λ we chose the optimal K of the original



(a) Regularisation by adding $\alpha||K||_F^2$ to the objective.



(b) Regularisation by adding $\alpha BKK^\top B^\top$ to the noise injection term.

Figure A.4: The effect of regularisation. The lines in the graphs are annotated with a number β to denote a regularisation constant $\alpha = 10^{-\beta}$. The distance to the optimum can be measured in two ways: as the difference between the objective value without regularisation and the reference objective, and as a the Frobenius-norm between the current and reference optimal value of K. Only the former is shown, but the latter leads to a qualitatively similar graph.

problem, such that the optimal $\tilde{K}$ of the scaled problem consists of all ones. For scaling type 2, the feedback term has a matrix product: $B\Lambda\tilde{K}$. For Λ we chose B^{-1}. We would expect that scaling type 2 performs well, since it makes the condition number of the Hessian identically one. In Figure A.5, scaling type 2 can be seen to live up to this expectation. Surprisingly, scaling type 1 – which corresponds to the rule-of-thumb of scaling decision variables such that they have order of magnitude one – makes convergence worse than the original. It even jumps to solutions that violate the property of positive definiteness.

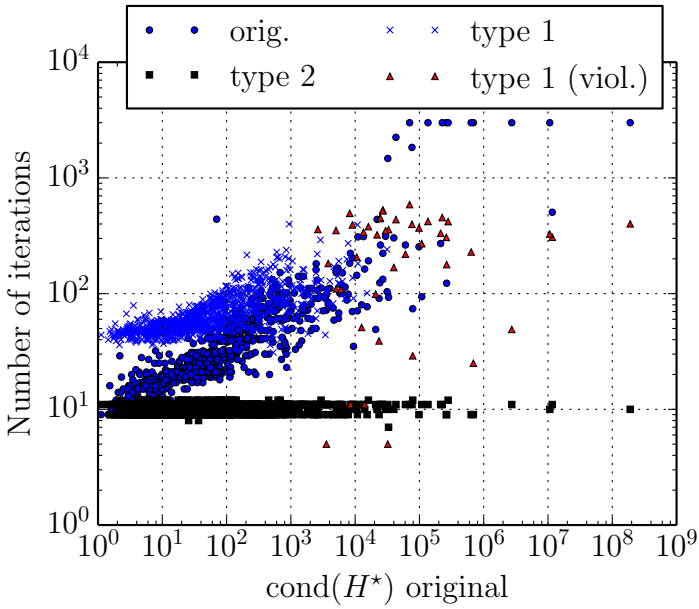


Figure A.5: The effect of scaling on convergence

Lifting A last approach consists of the lifting of BK:

$$\begin{array}{ll} \underset{K,C}{\text{minimise}} & \operatorname{tr}(P) \\ \text{s.t.} & P = (A+C)P(A+C)^\top + I_n \\ & C = BK \end{array} \tag{A.16}$$

Figure A.6 hints that lifting is slightly beneficial for very ill-conditioned problems, although the trend is not very clear.

A.3.4 Conclusion

This section studied stability optimisation with a degree of freedom in a linear feedback term. Such formulation may give rise to ill-conditioned Hessians. For the pole-cancelling case studied, the conditioning is determined solely by the system input sensitivity, and can best be treated by a matrix type of scaling. Unfortunately this treatment does not extend to the more general case of non-full-state feedback or a control space smaller than state space. Lifting the problem provides little benefits.

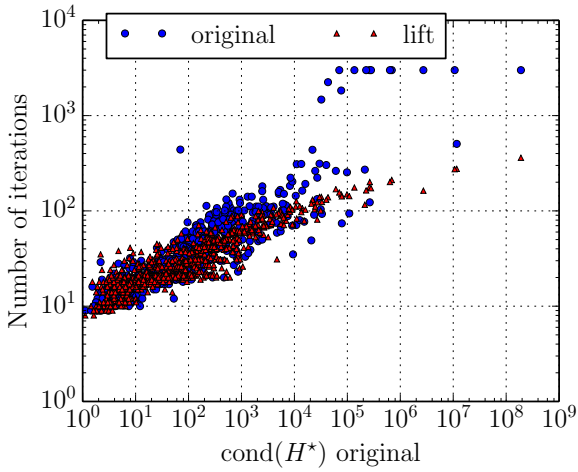


Figure A.6: The effect of lifting on convergence

Hence the message of this section is cautionary: regularisation is needed for good convergence of BFGS methods, but optimality may be worse than for the original problem during all iterations.

Bibliography

ACADO Toolkit (2009–2013). [Online; accessed 23-February-2009]. URL: `http://www.acadotoolkit.org` (cit. on pp. 41, 133, 139, 157).

Albersmeyer, J. and Diehl, M. (2010). "The Lifted Newton Method and its Application in Optimization". In: *SIAM Journal on Optimization* 20.3, pp. 1655–1684 (cit. on p. 40).

Amestoy, P., Davis, T., and Duff, I. (1996). "An Approximate Minimum Degree Ordering Algorithm". In: *SIAM Journal on Matrix Analysis and Applications* 17.4, pp. 886–905 (cit. on p. 31).

Amestoy, P. R., Duff, I. S., and L'Excellent, J.-Y. (2000). "Multifrontal parallel distributed symmetric and unsymmetric solvers". In: *Computer methods in applied mechanics and engineering* 184.2, pp. 501–520 (cit. on p. 32).

Anderson, E., C., Z. B., Bischof, Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Third. Philadelphia, PA: SIAM (cit. on p. 32).

Andersson, J., Åkesson, J., and Diehl, M. (2012). "CasADi – A symbolic package for automatic differentiation and optimal control". In: *Recent Advances in Algorithmic Differentiation*. Ed. by S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther. Lecture Notes in Computational Science and Engineering. Berlin: Springer (cit. on pp. 30, 35, 37, 41, 81, 132).

Andersson, J. (2013). "A General-Purpose Software Framework for Dynamic Optimization". PhD thesis. Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium: Arenberg Doctoral School, KU Leuven (cit. on pp. 41, 132, 157, 224).

Araya-Polo, M. and Laurent, H. (2005). "Certification of directional derivatives computed by automatic differentiation". In: *Proceedings of the 5th WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacuring Systems*. Cancun, Mexico (cit. on p. 92).

Athans, M. (1971). "The role and use of the stochastic linear-quadratic-Gaussian problem in control system design". In: *Automatic Control, IEEE Transactions on* 16.6, pp. 529–552 (cit. on p. 3).

Bank, R. E. and Douglas, C. C. (1993). "Sparse matrix multiplication package (SMMP)". English. In: *Advances in Computational Mathematics* 1.1, pp. 127–137 (cit. on p. 146).

Barraud, A. Y. (1977). "A numerical algorithm to solve $A^T X A - X = Q$". In: *Automatic Control, IEEE Transactions on* 22.5, pp. 883–885 (cit. on p. 97).

Bartholomew-Biggs, M., Brown, S., Christianson, B., and Dixo, L. (2000). "Automatic differentiation of algorithms". In: *Journal of Computational and Applied Mathematics* 124.1- 2. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations, pp. 171–190 (cit. on pp. 32, 135).

Beck, T. and Fischer, H. (1994). "The if-problem in automatic differentiation". In: *Journal of Computational and Applied Mathematics* 50.1–3, pp. 119–131 (cit. on p. 92).

Benner, P., Hossain, M.-S., and Stykel, T. (2014a). "Low-rank iterative methods for periodic projected Lyapunov equations and their application in model reduction of periodic descriptor systems". English. In: *Numerical Algorithms*, pp. 1–22 (cit. on p. 93).

Benner, P., Khoury, G. E., and Sadkane, M. (2014b). "On the squared Smith method for large-scale Stein equations". In: *Numerical Linear Algebra with Applications* 21.5, pp. 645–665 (cit. on p. 114).

Benner, P., Quintana-Orti, E. S., and Quintana-Orti, G. (2002). "Numerical solution of discrete stable linear matrix equations on multicomputers". In: *Parallel algorithms and applications* 17.2, pp. 127–146 (cit. on p. 97).

Benner, P. and Saak, J. (2013). "Numerical solution of large and sparse continuous time algebraic matrix Riccati and Lyapunov equations: a state of the art survey". In: *GAMM-Mitteilungen* 36.1, pp. 32–52 (cit. on p. 176).

Biegler, L. T. (2010). *Nonlinear Programming*. MOS-SIAM Series on Optimization. SIAM (cit. on p. 82).

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., (2002). "An updated set of basic linear algebra subprograms (BLAS)". In: *ACM Transactions on Mathematical Software* 28.2, pp. 135–151 (cit. on p. 32).

Bock, H. and Plitt, K. (1984). "A multiple shooting algorithm for direct solution of optimal control problems". In: *Proceedings 9th IFAC World Congress Budapest*. Pergamon Press, pp. 242–247 (cit. on pp. 40, 74).

Bojanczyk, A., Golub, G., and Van Dooren, P. (1992). "Periodic schur decomposition: algorithms and applications". In: *San Diego'92*. International Society for Optics and Photonics, pp. 31–42 (cit. on p. 95).

Bolzern, P. and Colaneri, P. (1988). "The periodic Lyapunov equation". In: *SIAM J. Matrix Anal. Appl.* 9(4), pp. 499–512 (cit. on pp. 3, 4, 7, 54, 70, 73, 74).

Bonnabel, S., Martin, P., Salaün, E., (2009). "Invariant Extended Kalman Filter: theory and application to a velocity-aided attitude estimation problem". In: *Proceedings of the 48th IEEE Conference on Decision and Control*, pp. 1297–1304 (cit. on pp. 62, 70).

Boyd, S., Ghaoui, L., Feron, E., and Balakrishnan, V. (1994). *Linear matrix inequalities in system and control theory*. Ed. by S. studies in applied mathematics. Vol. 14. SIAM (cit. on pp. 3, 24).

Bristeau, P., Martin, P., Salaün, E., and Petit, N. (2009). "The Role of Propeller Aerodynamics in the Model of a Quadrotor UAV". In: *Proceedings of the European Control Conference* (cit. on p. 192).

Broyden, C. (1969). "A new double-rank minimisation algorithm". In: *Notices of the American Mathematical Society*. Vol. 16. 4, p. 670 (cit. on p. 28).

Büskens, C. and Wassel, D. (2012). "Modeling and Optimization in Space Engineering". In: ed. by J. D. Pinter. Springer Verlag. Chap. The ESA NLP Solver WORHP (cit. on pp. 29, 81, 132, 175).

Byrd, R. and Schnabel, R. (1986). "Continuity of the null space basis and constrained optimization". English. In: *Mathematical Programming* 35.1, pp. 32–41 (cit. on p. 62).

Carnot, S. (1872). *Réflexions sur la puissance motrice du feu et sur les machines propres à développer cette puissance*. Vol. 1, pp. 393–457 (cit. on p. 159).

Charnes, A. and Cooper, W. (1959). "Chance Constrained Programming". In: *Management Science B* 6.1, pp. 73–78 (cit. on p. 2).

Chung, K. L. and Williams, R. J. (1990). *Introduction to stochastic integration*. Vol. 2. Springer (cit. on p. 55).

Coleman, T. and Moré, J. (1984). "Estimation of sparse Jacobian matrices and graph coloring problems". In: *SIAM Journal on Numerical Analysis* 20, pp. 187–209 (cit. on p. 62).

Cone, C. D. (1964). *A Mathematical Analysis of the Dynamic Soaring Flight of the Albatross: With Ecological Interpretations*. Virginia Institute of Marine Science Gloucester Point, Virgina (cit. on p. 160).

Curtis, A., Powell, M., and Reid, J. (1974). "On the estimation of sparse Jacobian matrices". In: *J. Inst. Math. Appl.* 13, pp. 117–119 (cit. on p. 33).

D'Andrea, R. (2014). *Flying Machine Arena*. URL: `http://flyingmachinearena.org/` (cit. on p. 187).

Dantzig, G. (1955). "Linear programming under uncertainty". In: *Management Science* 1, pp. 197–206 (cit. on p. 2).

Diamond, S., Chu, E., and Boyd, S. (2014). *CVXPY: A Python-Embedded Modeling Language for Convex Optimization, version 0.2.* `http://cvxpy.org/` (cit. on pp. 139, 157).

Dieci, L. and Eirola, T. (1994). "Positive definiteness in the numerical solution of Riccati differential equations". English. In: *Numerische Mathematik* 67.3, pp. 303–313 (cit. on pp. 72, 88).

Diehl, M. (2013). *Airborne Wind Energy.* Ed. by U. Ahrens, M. Diehl, and R. Schmehl. Springer. Chap. Airborne Wind Energy: Basic Concepts and Physical Foundations (cit. on p. 161).

Diehl, M., Bock, H., and Kostina, E. (2006). "An approximation technique for robust nonlinear optimization". In: *Mathematical Programming* 107, pp. 213–230 (cit. on pp. 3, 70).

Diehl, M., Ferreau, H. J., and Haverbeke, N. (2009a). "Nonlinear model predictive control". In: ed. by L. Magni, M. Raimondo, and F. Allgöwer. Vol. 384. Lecture Notes in Control and Information Sciences. Springer. Chap. Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pp. 391–417 (cit. on p. 30).

Diehl, M., Mombaur, K., and Noll, D. (2009b). "Stability Optimization of Hybrid Periodic Systems via a Smooth Criterion". In: *Automatic Control, IEEE Transactions on* 54.8, pp. 1875–1880 (cit. on p. 57).

Diwekar, U. (2008). "Optimization under uncertainty". In: *Introduction to Applied Optimization.* Springer, pp. 1–54 (cit. on p. 3).

Dokmanic, I., Kolundzija, M., and Vetterli, M. (2013). "Beyond Moore-Penrose: Sparse pseudoinverse". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, pp. 6526–6530 (cit. on p. 181).

Doležal, V. (1964). "The existence of a continuous basis of a certain linear subspace of E_r which depends on a parameter". In: *Časopis pro pěstování matematiky* 89.4, pp. 466–469 (cit. on p. 62).

Domahidi, A., Zgraggen, A., Zeilinger, M., Morari, M., and Jones, C. (2012). "Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control". In: *IEEE Conference on Decision and Control (CDC).* Maui, HI, USA, pp. 668–674 (cit. on p. 40).

Doyle, J. (1978). "Guaranteed margins for LQG regulators". In: *IEEE Transactions on Automatic Control* 23 (cit. on p. 3).

Duff, I. S. and Reid, J. K. (1983). "The Multifrontal Solution of Indefinite Sparse Symmetric Linear". In: *ACM Trans. Math. Softw.* 9.3, pp. 302–325 (cit. on p. 32).

Dullerud, G. and Paganini, F. (1999). *A Course in Robust Control Theory: A Convex Approach.* New York: Springer (cit. on p. 3).

Ferreau, H., Houska, B., Kraus, T., and Diehl, M. (2009). "Numerical Methods for Embedded Optimisation and their Implementation within the ACADO Toolkit". In: *7th Conference - Computer Methods and Systems (CMS'09)*. Ed. by W. M. R. Tadeusiewicz A. Ligeza and M. Szymkat. Krakow, Poland: Oprogramowanie Naukowo-Techniczne (cit. on p. 40).

Flannery, M. (2005). "The enigma of nonholonomic constraints". In: *American journal of physics* 73.3, pp. 265–272 (cit. on p. 163).

Fletcher, R. (1970). "A new approach to variable metric algorithms". In: *Computer J.* 13, pp. 317–322 (cit. on p. 28).

Floudas, C. and Stein, O. (2007). "The Adaptative Convexification Algorithm: a Feasible Point Method for Semi-Infinite Programming". In: *SIAM Journal on Optimization* 18.4, pp. 1187–1208 (cit. on p. 3).

Gahinet, P., Laub, A., Kenney, C., and Hewer, G. (1990). "Sensitivity of the stable discrete-time Lyapunov equation". In: *IEEE Transactions on Automatic Control* 35.11, pp. 1209–1217 (cit. on p. 125).

Gajić, Z. and Qureshi, M. T. J. (1995). *Lyapunov Matrix Equation in System Stability and Control*. Vol. 195. Mathematics in Science and Engineering. Elsevier (cit. on p. 47).

Gebremedhin, A. H., Manne, F., and Pothen, A. (2005). "What color is your Jacobian? Graph coloring for computing derivatives". In: *SIAM Review* 47, pp. 629–705 (cit. on pp. 33, 150).

Geebelen, K., Ahmad, H., Vukov, M., Gros, S., Swevers, J., and Diehl, M. (2012). "An experimental test set-up for launch/recovery of an Airborne Wind Energy (AWE) system". In: *Proceedings of the 2012 American Control Conference* (cit. on p. 162).

Geebelen, K. and Gillis, J. (2010). "Modelling and control of rotational start-up phase of tethered aeroplanes for wind energy harvesting". MA thesis. K.U.Leuven (cit. on pp. 162, 186).

Gertz, E. and Wright, S. (2003). "Object-Oriented Software for Quadratic Programming". In: *ACM Transactions on Mathematical Software* 29.1, pp. 58–81 (cit. on p. 30).

Ghaffari, A., Tomizuka, M., and Soltan, R. (2009). "The stability of limit cycles in nonlinear systems". English. In: *Nonlinear Dynamics* 56.3, pp. 269–275 (cit. on p. 26).

Giering, R. and Kaminski, T. (2006). "Automatic Sparsity Detection Implemented as a Source-to-Source Transformation". In: *Lecture Notes in Computer Science*. Vol. 3994. Springer Berlin Heidelberg, pp. 591–598 (cit. on p. 150).

Giles, M. (2008). "Collected matrix derivative results for forward and reverse mode algorithmic differentiation". In: *Advances in Automatic Differentiation*. Springer, pp. 35–44 (cit. on p. 35).

Gill, P., Murray, W., and Saunders, M. (2005). "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization". In: *SIAM Review* 47.1, pp. 99–131 (cit. on pp. 29, 132).

Gill, P., Kungurtsev, V., and Robinson, D. (2013). "A Regularized SQP Method Convergent to Second Order Optimal Points". In: *Submitted to SIAM Journal of Optimization* (cit. on p. 59).

Gill, P., Murray, W., Saunders, M., Michael, A., Stewart, G., and Wright, M. (1985). "Properties of a representation of a basis for the null space". English. In: *Mathematical Programming* 33.2, pp. 172–186 (cit. on p. 62).

Gill, P., Murray, W., and Wright, M. (1981). *Practical optimization*. London: Academic Press (cit. on p. 31).

Gillis, J. (2013a). *A simplified optimisation API for CasADi*. URL: `http://optoy.casadi.org` (cit. on p. 140).

Gillis, J. (2013b). *Tutorial on structured indexing in CasADi*. URL: `http://docs.casadi.org/v2.0.0/tutorials/tools/structure.pdf` (cit. on p. 144).

Gillis, J. (2014). *A tutorial on CasADi 2.0*. URL: `https://www.youtube.com/watch?v=ANicKS8gKdM` (cit. on p. 134).

Gillis, J. and Diehl, M. (2013). "A Positive Definiteness Preserving Discretization Method for nonlinear Lyapunov Differential Equations". In: *Proceedings of the 52nd IEEE Conference on Decision and Control* (cit. on pp. 8, 71, 87, 224).

Gillis, J. and Diehl, M. (2014a). "Hierarchical seeding for efficient sparsity pattern recovery in automatic differentiation". In: *CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing* (cit. on pp. 9, 150, 157, 224).

Gillis, J., Goos, J., Geebelen, K., Swevers, J., and Diehl, M. (2012). "Optimal periodic control of power harvesting tethered airplanes: how to fly fast without wind and without propellor?" In: *Proceedings of the American Control Conference*. Montreal, Canada (cit. on p. 177).

Gillis, J., Horn, G., and Diehl, M. (2014b). "Joint design of stochastically safe setpoints and controllers for nonlinear constrained systems by means of optimization". In: *Proceedings of the 19th IFAC World Congress*. World Congress of the International Federation of Automatic Control. Capetown, South Africa (cit. on pp. 9, 171, 186, 224).

Goldfarb, D. (1970). "A family of variable metric methods derived by variational means". In: *Maths. Comp.* 17, pp. 739–764 (cit. on p. 28).

Golub, G. H. and Pereyra, V. (1973). "The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate". In: *SIAM Journal on numerical analysis* 10.2, pp. 413–432 (cit. on p. 62).

Golub, G., Nash, S., and Van Loan, C. (1979). "A Hessenberg-Schur method for the problem AX+ XB= C". In: *Automatic Control, IEEE Transactions on* 24.6, pp. 909–913 (cit. on p. 97).

Gotsman, C. and Toledo, S. (2008). "On the computation of null spaces of sparse rectangular matrices". In: *SIAM Journal on Matrix Analysis and Applications* 30.2, pp. 445–463 (cit. on p. 181).

Griewank, A. (1989). "On Automatic Differentiation". In: *Mathematical Programming: Recent Developments and Applications*. Kluwer Academic Publishers, Dordrecht, Boston, London (cit. on p. 32).

Griewank, A., Juedes, D., Mitev, H., Utke, J., Vogel, O., and Walther, A. (1999). *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++*. Tech. rep. Updated version of the paper published in *ACM Trans. Math. Software* 22, 1996, 131–167. Technical University of Dresden, Institute of Scientific Computing and Institute of Geometry (cit. on pp. 36, 133, 157).

Gros, S., Zanon, M., and Diehl, M. (2013a). "A Relaxation Strategy for the Optimization of Airborne Wind Energy Systems". In: *European Control Conference* (cit. on p. 163).

Gros, S., Zanon, M., and Diehl, M. (2013b). "Control of Airborne Wind Energy Systems Based on Nonlinear Model Predictive Control & Moving Horizon Estimation". In: *European Control Conference* (cit. on p. 162).

Gros, S., Zanon, M., Vukov, M., and Diehl, M. (2012). "Nonlinear MPC and MHE for Mechanical Multi-Body Systems with Application to Fast Tethered Airplanes". In: *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands* (cit. on pp. 163, 208).

Gupta, A., Joshi, M., and Kumar, V. (2001). "WSMP: A high-performance shared-and distributed-memory parallel sparse linear equation solver". In: *IBM Research Division RC* 22038 (cit. on p. 32).

Hairer, E., Nørsett, S., and Wanner, G. (1993). *Solving Ordinary Differential Equations I*. 2nd. Springer Series in Computational Mathematics. Berlin: Springer (cit. on p. 20).

Hamermesh, M. (1989). *Group theory and its application to physical problems*. New York, NY: Dover (cit. on p. 18).

Hammarling, S. (1982). "Numerical Solution of the Stable, Non-negative Definite Lyapunov Equation Lyapunov Equation". In: *IMA Journal of Numerical Analysis* 2.3, pp. 303–323 (cit. on p. 97).

Hehn, M. and D'Andrea, R. (2013). "An iterative learning scheme for high performance, periodic quadrocopter trajectories". In: *Control Conference (ECC), 2013 European*. IEEE, pp. 1799–1804 (cit. on p. 187).

Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., and Woodward, C. (2005). "SUNDIALS: Suite of Nonlinear and Differential/Al-

gebraic Equation Solvers". In: *ACM Transactions on Mathematical Software* 31, pp. 363–396 (cit. on p. 79).

Holm, D. D. (2008). *Geometric Mechanics, Part I: Dynamics and Symmetry*. Imperial College Press (cit. on p. 18).

Horn, R. A. and Johnson, C. R., eds. (1986). *Matrix Analysis*. New York, NY, USA: Cambridge University Press (cit. on pp. 61, 124).

Houska, B. (2007). "Robustness and Stability Optimization of Open-Loop Controlled Power Generating Kites". MA thesis. University of Heidelberg (cit. on pp. 43, 53, 55, 70, 87, 172, 173, 223).

Houska, B. (2011). "Robust Optimization of Dynamic Systems". (ISBN: 978-94-6018-394-2). PhD thesis. Katholieke Universiteit Leuven (cit. on pp. 3, 5).

Houska, B. and Diehl, M. (2007). "Optimal Control for Power Generating Kites". In: *Proc. 9th European Control Conference*. (CD-ROM). Kos, Greece, pp. 3560–3567 (cit. on pp. 3, 4, 7, 70, 223).

Houska, B. and Diehl, M. (2009). "Robust nonlinear optimal control of dynamic systems with affine uncertainties". In: *Proceedings of the 48th Conference on Decision and Control*. Shanghai, China, pp. 2274–2279 (cit. on pp. 220, 223).

Houska, B. and Diehl, M. (2010). "Robustness and Stability Optimization of Power Generating Kite Systems in a Periodic Pumping Mode". In: *Proceedings of the IEEE Multi - Conference on Systems and Control*. Yokohama, Japan, pp. 2172–2177 (cit. on pp. 73, 74, 88).

Houska, B. and Diehl, M. (2011a). "Nonlinear Robust Optimization via Sequential Convex Bilevel Programming". In: *Mathmatical Programming*. (submitted) (cit. on p. 3).

Houska, B. and Diehl, M. (2011b). "Robust design of linear control laws for constrained nonlinear dynamic systems". In: *Proc. of the 18th IFAC World Congress*. Milan, Italy (cit. on pp. 3, 7, 70, 220).

Houska, B., Villanueva, M., and Chachuat, B. (2013). "A validated integration algorithm for nonlinear ODEs using Taylor models and ellipsoidal calculus". In: *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, pp. 484–489 (cit. on p. 3).

HSL (2011). *A collection of Fortran codes for large scale scientific computation*. URL: `http://www.hsl.rl.ac.uk` (cit. on pp. 32, 81, 213).

Hurtado, J. E. and Sinclair, A. J. (2011). "Lagrangian mechanics of overparameterized systems". In: *Nonlinear Dynamics* 66.1-2, pp. 201–212 (cit. on p. 64).

Ibens, K. and Peeters, K. (2009). "Modellering en controle van verankerde vliegtuigen". MA thesis. KULeuven (cit. on p. 162).

IBM Corp. (2009). *IBM ILOG CPLEX V12.1, User's Manual for CPLEX* (cit. on p. 30).

Ipsen, I. (2009). *Numerical Matrix Analysis: Linear Systems and Least Squares*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (cit. on p. 124).

JModelica.org. [Online; accessed 10-October-2013]. URL: `http://www.jmodelica.org/` (cit. on p. 41).

Jones, D. E. H. (1982). *The inventions of Daedalus: A compendium of plausible schemes*. Vol. 26. 4. WH Freeman (cit. on p. 17).

Jones, E., Oliphant, T., Peterson, P., (2001–2014). *SciPy: Open source scientific tools for Python*. URL: `http://www.scipy.org/` (cit. on p. 146).

Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F. (1995). "A new approach for filtering nonlinear systems". In: *American Control Conference, Proceedings of the 1995*. Vol. 3. IEEE, pp. 1628–1632 (cit. on p. 47).

Julier, S. and Uhlmann, J. (2004). "Unscented filtering and nonlinear estimation". In: *Proceedings of the IEEE* 92.3, pp. 401–422 (cit. on p. 47).

Kalman, R. (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82, pp. 35–45 (cit. on p. 3).

Kalman, R. (1963). "Lyapunov functions for the problem of Lur'e in automatic control". In: *Proceedings of the National Academy of Sciences USA* 49, pp. 201–205 (cit. on p. 4).

Karypis, G. and Kumar, V. (2009). "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0". In: (cit. on p. 144).

Karypis, G. and Kumar, V. (1998). "A fast and high quality multilevel scheme for partitioning irregular graphs". In: *SIAM Journal on scientific Computing* 20.1, pp. 359–392 (cit. on p. 31).

Laub, A. J. (2004). *Matrix Analysis For Scientists And Engineers*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (cit. on p. 124).

Leine, R. I. (2010). "The historical development of classical stability concepts: Lagrange, Poisson and Lyapunov stability". In: *Nonlinear Dynamics* 59.1-2, pp. 173–182 (cit. on p. 22).

Leineweber, D., Schäfer, A., Bock, H., and Schlöder, J. (2003). "An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization. Part II: Software Aspects and Applications". In: *Computers and Chemical Engineering* 27, pp. 167–174 (cit. on p. 41).

Lewis, A. D. (2007). "Is it worth learning differential geometric methods for modeling and control of mechanical systems?" In: *Robotica* 25.06, pp. 765–777 (cit. on p. 18).

Li, T., Weng, C.-Y., Chu, E., and Lin, W.-W. (2012). *Solving large-scale Stein and Lyapunov equations by doubling*. Tech. rep. NCTS Preprints in Mathematics, 2012-5-005. National Tsing Hua University, Hsinchu, Taiwan (cit. on pp. 111, 114).

Li, T., Weng, P.-Y., Chu, E.-w., and Lin, W.-W. (2013). "Large-scale Stein and Lyapunov equations, Smith method, and applications". English. In: *Numerical Algorithms* 63.4, pp. 727–752 (cit. on p. 114).

Löfberg, J. (2004). "YALMIP: A Toolbox for Modeling and Optimization in MATLAB". In: *Proceedings of the CACSD Conference*. Taipei, Taiwan (cit. on pp. 139, 157).

Logist, F., Houska, B., Diehl, M., and Impe, J. V. (2011a). "Robust multi-objective optimal control of uncertain biochemical processes". In: *Chemical Engineering Science* 66. (accepted), pp. 4670–4682 (cit. on pp. 173, 186, 223, 224).

Logist, F., Houska, B., Diehl, M., and Impe, J. V. (2011b). *Robust optimal control of a biochemical reactor with multiple objectives*. European Symposium on Computer Aided Process Engineering (ESCAPE). Chalkidiki, Greece (cit. on p. 88).

Lyapunov, M. (1907). "Probléme general de la stabilité du mouvement". In: *Ann. Fac. Sci. Toulouse Math.* 5(9), pp. 203–474 (cit. on p. 23).

Magnusson, F. (2012). "Collocation Methods in JModelica.org". MA thesis. Department of Automatic Control, Lund University, Sweden (cit. on pp. 41, 75).

Maplesoft (2004). *Maple*. 9.0. Waterloo: Maple Inc. (cit. on p. 134).

McAndrew, A. (1999). "MuPad". In: *Linux J.* 1999.63es (cit. on p. 134).

Miller, L. and Wagner, H. (1965). "Chance-constrained programming with joint constraints". In: *Operations Research* 13, pp. 930–945 (cit. on p. 2).

Mombaur, K., Bock, H., Schlöder, J., and Longman, R. W. (2005). "Open-loop stable solution of periodic optimal control problems in robotics". In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik* 85(7), pp. 499–515 (cit. on p. 4).

Nagy, Z. and Braatz, R. (2004). "Open-loop and closed-loop robust optimal control of batch processes using distributional and worst-case analysis". In: *Journal of Process Control* 14, pp. 411–422 (cit. on p. 3).

Nagy, Z. and Braatz, R. (2007). "Distributional uncertainty analysis using power series and polynomial chaos expansions". In: *Journal of Process Control* 17, pp. 229–240 (cit. on p. 70).

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. Springer (cit. on p. 26).

Noether, E. (1918). "Invariante Variationsprobleme. Nachr. D. König. Gesellsch. D. Wiss. Zu Göttingen, Math-phys. Klasse 1918: 235-257". In: *English Reprint: physics/0503066, http://dx. doi. org/10.1080/00411457108231446* (cit. on p. 57).

Pamadi, B. (2003). *Performance, Stability, Dynamics, and Control of Airplanes*. American Institute of Aeronautics and Astronautics, Inc. (cit. on p. 163).

Parlett, B. and Reinsch, C. (1969). "Balancing a matrix for calculation of eigenvalues and eigenvectors". English. In: *Numerische Mathematik* 13.4, pp. 293–304 (cit. on p. 97).

Pipeleers, G., Demeulenaere, B., Swevers, J., and Vandenberghe, L. (2009). "Extended LMI characterizations for stability and performance of linear systems". In: *Systems & Control Letters* 58.7, pp. 510–518 (cit. on p. 3).

Pittelkau, M. E. (1993). "Optimal periodic control for spacecraft pointing and attitude determination". In: *Journal of Guidance, Control, and Dynamics* 16.6, pp. 1078–1084 (cit. on p. 90).

Prèkopa, A. (1970). *On probabilistic constrained programming*. Proceedings of the Princeton Symposium on Mathematical Programming, Princeton University Press, pp. 113–138 (cit. on p. 2).

qpDUNES Website (2009). `http://mathopt.de/qpdunes` (cit. on p. 40).

Ritz, R., Mueller, M., and D'Andrea, R. (2012). "Cooperative Quadrocopter Ball Throwing and Catching". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4972–4978 (cit. on p. 187).

Roberts, J. D. (1980). "Linear model reduction and solution of the algebraic Riccati equation by use of the sign function". In: *International Journal of Control* 32.4, pp. 677–687 (cit. on p. 97).

Romanenko, A. and Castro, J. (2004). "The unscented filter as an alternative to the EKF for nonlinear state estimation: a simulation case study". In: *Computers and Chemical Engineering*, pp. 347–355 (cit. on p. 47).

Sadkane, M. (2012). "A low-rank Krylov squared Smith method for large-scale discrete-time Lyapunov equations". In: *Linear Algebra and its Applications* 436.8. Special Issue dedicated to Danny Sorensen's 65th birthday, pp. 2807–2827 (cit. on p. 114).

Sargent, R. and Sullivan, G. (1978). "The development of an efficient optimal control package". In: *Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2*. Ed. by J. Stoer. Heidelberg: Springer (cit. on p. 39).

Shanno, D. (1970). "Conditioning of quasi-Newton methods for function minimization". In: *Maths. Comp.* 24, pp. 647–656 (cit. on p. 28).

Smith, R. (1968). "Matrix Equation $XA + BX = C$". In: *SIAM Journal on Applied Mathematics* 16.1, pp. 198–201 (cit. on p. 111).

Sreedhar, J. and Van Dooren, P. (1994). "Periodic Schur form and some matrix equations". In: *Mathematical Research* 77, pp. 339–339 (cit. on p. 95).

Stein, O. (2003). *Bilevel Strategies in Semi Infinite Optimization*. Kluwer, Boston (cit. on p. 3).

Sternberg, J., Gros, S., Houska, B., and Diehl, M. (2012a). "Approximate Robust Optimal Control of Periodic Systems with Invariants and High-Index Differential Algebraic Systems". In: *In Proceedings of the 7th IFAC Symposium on Robust Control Design*, pp. 678–683 (cit. on pp. 7, 59–61, 70, 166, 208, 223).

Sternberg, J., Houska, B., and Diehl, M. (2012b). "A Structure Exploiting Algorithm for Approximate Robust Optimal Control with Application to Power Generating Kites". In: *In Proceedings of the American Control Conference* (cit. on p. 129).

Sternberg, J., Houska, B., Logist, F., Telen, D., Van Impe, J., and Diehl, M. (2012c). "A Toolkit for Efficient Computation of Sensitivities in Approximate Robust Optimal Control Problems". In: *In Proceedings of the 7th IFAC Symposium on Robust Control Design*, pp. 183–188 (cit. on pp. 129, 130).

Sternberg, J., Goit, J., Gros, S., Meyers, J., and Diehl, M. (2012d). "Robust and Stable Periodic Flight of Power Generating Kite Systems in a Turbulent Wind Flow Field". In: *In Proceedings of the 15th IFAC Workshop on Control Applications of Optimization* (cit. on p. 88).

Sundials (2009). *Sundials - SUite of Nonlinear and DIfferential/ALgebraic equation Solvers (web page and software)*. https://computation.llnl.gov/-casc/sundials (cit. on p. 22).

SymPy Development Team (2014). *SymPy: Python library for symbolic mathematics*. URL: `http://www.sympy.org` (cit. on p. 134).

TAPENADE (2013). [Online; accessed 4-June-2013]. URL: `http://www-sop.inria.fr/tropics/` (visited on 06/04/2013) (cit. on p. 36).

Teschl, G. (2012). *Ordinary differential equations and dynamical systems*. Vol. 140. American Mathematical Soc. (cit. on p. 22).

Thompson, J. M. T. and Stewart, H. B. (2002). "Nonlinear dynamics and chaos". In: (cit. on p. 26).

Tippett, M., Cohn, S., Todling, R., and Marchesin, D. (2000). "Conditioning of the Stable, Discrete-Time Lyapunov Operator". In: *SIAM Journal on Matrix Analysis and Applications* 22.1, pp. 56–65 (cit. on pp. 113, 125, 126).

Tuisku, P., Pernu, T. K., and Annila, A. (2009). "In the light of time". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science* 465.2104, pp. 1173–1198 (cit. on p. 159).

Van der Merwe, R. and Wan, E. (2001). "The Square-Root Unscented Kalman Filter for State and Parameter-Estimation". In: *International Conference*

on Acoustics, Speech, and Signal Processing. Salt Lake City, Utah, (cit. on p. 47).

Van Huffel, S., Sima, V., Varga, A., Hammarling, S., and Delebecque, F. (2004). "High-performance numerical software for control". In: *IEEE Control Systems Magazine* 24.1, pp. 60–76 (cit. on pp. 95, 106).

Van Rossum, G. and Drake, F. (2001). *Python Reference Manual, PythonLabs, Virginia, USA* (cit. on p. 132).

Vanbiervliet, J., Vandereycken, B., Michiels, W., Vandewalle, S., and Diehl, M. (2009). "The Smoothed Spectral Abscissa for Robust Stability Optimization". In: *SIAM Journal on Optimization* 20.1, pp. 156–171 (cit. on p. 57).

Varga, A. (1997). "Periodic Lyapunov equations: some applications and new algorithms". In: *International Journal of Control* 67.1, pp. 69–88 (cit. on pp. 105, 125, 130).

Varga, A. and Pieters, S. (1998). "Gradient-Based Approach to Solve Optimal Periodic Output Feedback Control Problems". In: *Automatica* 34.4, pp. 477–481 (cit. on p. 90).

Verschueren, R. (2014). *Design and implementation of a time-optimal controller for model race cars* (cit. on p. 225).

Wächter, A. and Biegler, L. (2006a). "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming* 106.1, pp. 25–57 (cit. on pp. 132, 175).

Wächter, A. and Biegler, L. T. (2006b). "Line Search Filter Methods for Nonlinear Programming: Motivation and Global Convergence". In: *SIAM Journal on Optimization* 16, pp. 1–31 (cit. on p. 30).

Walck, C. (2007). *Handbook on statistical distributions for experimentalists*. Internal Report SUF-PFY/96-01. University of Stockholm (cit. on p. 46).

Wedin, P. A. (1973). "Perturbation theory for pseudo-inverses". English. In: *BIT Numerical Mathematics* 13.2, pp. 217–232 (cit. on p. 62).

Willems, J. (1972). "Dissipative dynamical systems part I: General theory". English. In: *Archive for Rational Mechanics and Analysis* 45.5, pp. 321–351 (cit. on p. 24).

Wirsching, L., Bock, H. G., and Diehl, M. (2006). "Fast NMPC of a chain of masses connected by springs". In: *Proceedings of the IEEE International Conference on Control Applications, Munich*, pp. 591–596 (cit. on p. 116).

Zhou, K., Doyle, J., and Glover, K. (1996). *Robust and optimal control*. Englewood Cliffs, NJ: Prentice Hall (cit. on p. 172).

Curriculum Vitae

Name Joris (Karel Clothilde) Gillis

Birth coordinates July 17, 1987 @ Duffel, Belgium

Education

September 1999 – June 2005 Latin–Mathematics at Sint-Gummaruscollege, Lier, Belgium.

October 2005 – June 2008 Bachelor engineering sciences (major Mechanical, minor Electrical) at KU Leuven, Belgium.

October 2008 – June 2010 Master mechanical engineering (speciality thermotechnical) at KU Leuven, Belgium.
Thesis: *Modelling and control of rotational start-up phase of tethered aeroplanes for wind energy harvesting*

Scientific Work

July 2006 – August 2006 Holiday job at Physics department KU Leuven, Belgium. *Adapting an interferometer's path length by introduction and active steering of an LCD display with Matlab and Labview.*

July 2010 – September 2010 Trainee-ship at Joby Energy in Santa Cruz, California, U.S.A. *Machine vision system for controlling kites in the sky.*

October 2010 – February 2015 PhD student at the KU Leuven, Belgium under supervision of Moritz Diehl, Eric Van den Bulck and Jan Swevers.

List of publications

Papers at international conferences, published in full in proceedings

Gillis, J. and Diehl, M. (2013). "A Positive Definiteness Preserving Discretization Method for nonlinear Lyapunov Differential Equations". In: *Proceedings of the 52nd IEEE Conference on Decision and Control*.

Gillis, J., Goos, J., Geebelen, K., Swevers, J., and Diehl, M. (2012). "Optimal periodic control of power harvesting tethered airplanes: how to fly fast without wind and without propellor?" In: *Proceedings of the American Control Conference*. Montreal, Canada.

Gillis, J., Horn, G., and Diehl, M. (2014). "Joint design of stochastically safe setpoints and controllers for nonlinear constrained systems by means of optimization". In: *Proceedings of the 19th IFAC World Congress*. World Congress of the International Federation of Automatic Control. Capetown, South Africa.

Conference abstracts

Gillis, J. and Diehl, M. (2014). "Hierarchical seeding for efficient sparsity pattern recovery in automatic differentiation". In: *CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing*.

Gillis, J., Geebelen, K., Sternberg-Kaletta, J., Gros, S., Houska, B., and Diehl, M. (2012). "Lyapunov based design of robust linear-feedback for time-optimal periodic quadcopter motions". In: *Proc. Benelux Meeting on Systems and Control 2012*.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering Science
Department of Electrical Engineering
Afdeling ESAT - STADIUS, Stadius Centrum voor Dynamische Systemen, Signaalverwerking en Gegevensanalyse
Kasteelpark Arenberg 10
B-3001 Leuven