Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2016

An Optimal Control Toolbox for MATLAB® Based on CasADi

Viktor Leek

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

An Optimal Control Toolbox for MATLAB® Based on CasADi

Viktor Leek

LiTH-ISY-EX–16/4986–SE

Supervisor: **Vaheed Nezhadali**
ISY, Linköping University
Björn Johansson
Scania CV AB
Henrik Svärd
Scania CV AB

Examiner: **Lars Eriksson**
ISY, Linköping University

Division of Vehicular Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Sammanfattning

Många ingenjörsproblem formuleras naturligt som optimalstyrningsproblem. Det kan röra sig om att förflytta sig mellan två punkter på snabbast möjliga sätt, eller att sätta en satellit i omloppsbana med minsta möjliga energiåtgång. Många optimalstyrningsproblem är dock för svåra för att lösas analytiskt och kräver därför användandet av numeriska metoder. De numeriska metoder som vunnit störst spridning är de så kallade direkta metoderna. Det finns dock en stor nackdel med dessa. Om problemet inte är konvext så är lösningen som erhålls är inte garanterat globalt optimal, det vill säga den absolut bästa, istället är den garanterat lokalt optimal, det vill säga den bästa i sin närhet. För att kompensera för detta bör problemet lösas flera gånger, under olika betingelser, för att på så sätt undersöka om lösningen är en bra kandidat till globalt optimum.

CasADi är en mjukvara speciellt utformad för dynamisk optimering. Den har vunnit stor spridning under de senaste åren tack vare att den tillhandahåller alla nödvändiga byggstenarna för dynamisk optimering. Detta har givit enskilda ingenjörer och forskare möjligheten att på egen hand formulera och lösa optimalstyrningsproblem numeriskt, detta kräver dock goda teoretiska kunskap om de nödvändiga numeriska metoderna.

Fördelen med en toolbox som löser generella optimalstyrningsproblem är att de underliggande numeriska metoderna har testats och visats fungera på optimalstyrningsproblem med kända lösningar. Detta medför att användaren inte behöver uttömmande kännedom om de numeriska metoderna, utan kan fokusera på att formulera och lösa optimalstyrningsproblem.

Huvudbidraget i denna uppsats är en optimalstyrnings-toolbox för MATLAB baserad på CasADi. Toolboxen kräver inte expertkunskaper inom numeriska metoder för optimal styrning, men tillhandahåller ändå ett gränssnitt som möjliggör komplexa problemformuleringar.

Toolboxen implementerar två direkta metoder, *direct multiple shooting* och *direct collocation*. Det möjliggör en problemformulering med många frihetsgrader. Den viktigaste funktionalitet är att den direkta metoden kan bytas utan att problemformuleringen behöver ändras, vilket medför att användaren på ett enkelt sätt kan ändra betingelserna för sitt problem.

I uppsatsen beskrivs hur de två implementerade direkta metoderna fungerar och de designval som gjorts. Det beskrivs även vad som återstår att testa och utvärdera i toolboxen, samt de problem som använts som referens under utvecklingsarbetet.

Abstract

Many engineering problems are naturally posed as optimal control problems. It may involve moving between two points in the fastest possible way, or to put a satellite into orbit with minimum energy consumption. Many optimal control problems are too difficult to be solved analytically and therefore require the use of numerical methods. The numerical methods that are the most widespread are the so-called direct methods. However, there is one major drawback with these. If the problem is non-convex, the solution is not guaranteed globally optimal, that is, the absolute best, instead it is guaranteed locally optimal, that is the best in its vicinity. To compensate for this, the problem should be solved several times, under different conditions, in order to investigate whether the solution is a good candidate for the global optimum.

CasADi is a software specifically designed for dynamic optimization. It has gained wide spread in recent years because it provides all the necessary building blocks for dynamic optimization. This has given individual engineers and scientists the ability to independently formulate and solve all sorts of optimal control problems. However, this requires good theoretical knowledge of the necessary numerical methods.

The advantage of a toolbox, which solves general optimal control problems, is that the underlying numerical methods have been tested and shown to function on optimal control problems with known solutions. This means that the user does not need exhaustive knowledge of the numerical methods involved, but can focus on formulating and solving optimal control problems.

The main contribution of this thesis is an optimal control toolbox for MATLAB based on CasADi. The toolbox does not require expert knowledge of the numerical methods, but provides an alternative lower level abstraction that allows for more complex problem formulations.

The toolbox implements two direct methods, *direct multiple shooting* and *direct collocation*. This allows a problem formulation with many degrees of freedom. The most important property of the toolbox is that the discretization can be changed, without the problem formulation needing to be altered. This way the user can easily change the conditions for his/her problem.

The thesis describes how the two implemented direct methods work, and the design choices made. It also describes what remains to test and evaluate, and the problems that have been used as a reference during the development process.

Acknowledgments

This thesis was supported by Linköping University and Scania. I thank my colleagues from the Vehicular Devision at Linköping University and the Powertrain Control System Pre-development group at Scania who provided insight and expertise that greatly assisted the work. I would like to express a special thanks to my supervisors Björn Johansson, Veheed Nezhadali and Henrik Svärd for great support and lots of good ideas, and my examiner, Lars Eriksson who came up with the idea for the thesis.

I would also like to show my gratitude to Joel Andersson and Joris Gillis for creating and continuing to develop CasADi, without it, the toolbox would have never been possible.

Finally, I want to thank my parents, family and friends for lots of love and fun. Without you, I would not have endured many years of study.

Linköping, August 2016
Viktor Leek

Contents

Notation **xi**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Purpose . . . 2
1.3 Questions . . . 2
1.4 Delimitations . . . 3

2 Theory **5**
2.1 Optimal Control . . . 5
2.1.1 Basic problem formulation . . . 5
2.2 Numerical Solution to Optimal Control Problems . . . 6
2.2.1 A Brief Overview . . . 6
2.2.2 Direct Methods . . . 7
2.2.3 Integrating the Objective Function . . . 16
2.2.4 Mesh Refinement . . . 16
2.2.5 Scaling . . . 17
2.3 CasADi . . . 17
2.4 Optimal Control Software . . . 17

3 Method **19**
3.1 Introduction . . . 19
3.2 The Studied Optimal Control Problems . . . 19
3.2.1 The Bryson-Denham Problem . . . 19
3.2.2 An Optimal Control Benchmark: Transient Optimization of A Diesel-Electric Powertrain . . . 22
3.2.3 Optimal Control of a Diesel-Electric Powertrain During an Up-Shift . . . 25

4 Implementation **31**
4.1 Entering Optimal Control Problems . . . 31
4.2 Representing the Model . . . 33
4.3 Discretization Methods . . . 34

4.3.1 Multiple Shooting 34
4.3.2 Direct Collocation 36
4.3.3 Providing the Initial Guess 38
4.3.4 Control Continuity 38
4.4 Solving Optimal Control Problems 38
4.4.1 Solving the reference problems 38
4.4.2 Complete Code Example 45

5 Design Choices **47**
5.1 Entering Optimal Control Problems 48
5.2 Discretization Methods 48
5.3 Providing an Initial Guess 49
5.4 Control Continuity 49
5.5 Solving Optimal Control Problems 49

6 Discussion **51**
6.1 Results 51
6.2 Method 51

7 Conclusions **53**
7.1 Future Work 54

Bibliography **55**

Notation

Some sets

Notation	Meaning
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$\mathbb{C}$	The set of complex numbers
$\mathbb{Z}$	The mark of Zorro

Abbreviations

Abbreviation	Meaning
OCP	Optimal control problem
NLP	Nonlinear program
PMP	Pontryagin's maximum principle
MVEM	Mean value engine model

1 Introduction

Optimal control is one of the best ways to gain insights into the control of dynamic systems. It may involve finding the minimum energy way of keeping a satellite in orbit, the optimal drug scheduling for chemotherapy, or choosing the best torque converter for a wheel-loader. In all but the simplest cases, the solution to optimal control problems requires the use of computers. Implementing the numerical methods needed is not trivial, but rigorously made, the implementation can be used for solving general optimal control problems. This thesis is about the design and implementation of an optimal control toolbox for Matlab which aims to make numerical optimal control available to users in industry and academia, making it possible for them to solve real-world optimal control problems with little or no expertise within the field of numerical optimal control.

The thesis has been conducted as a cooperation between Linköping University and Scania. Work was conducted during the spring semester of 2016 and presented in August the same year. Work was located to both Linköping University and Scania in Södertälje.

1.1 Motivation

Solving real-world optimal control problems is a difficult task. A model needs to be provided and it needs to have certain properties [17]. For instance be smooth in the region of interest, quantitatively accurate and possess good extrapolation properties since the optimal solution is generally found at some extreme. Typical for real-world optimal control problems is that the dynamics are nonlinear and the constraints governing the allowed behaviour of the system complex.

Finding a good objective function is not trivial. Unilateral objective functions such as minimum time tend to produce solutions with undesirable properties such as jerk [13]. This due to that the solver only takes one aspect into account

when calculating the solution, quality aspects such as smoothness are not considered unless explicitly stated. A good objective function is therefore often a compromise between different properties that takes time and methodical effort to make.

Real-world optimal control problems often become large and non-convex. This means that the solver can only guarantee a local optimum, which is dependent on the starting point of the optimization (the inital guess). It it therefore a good idea to try a number of different starting points in order to assess the solution. It is also a good idea to investigate how the discretization method affects the optimization, this can be done by changing discretization method completely, or the settings of the current method.

CasADi is software specifically tailored for dynamic optimization and has gained wide spread since its release in 2012. In itself, CasADi is not a solver for general optimal control problems, but it provide the necessary building blocks for formulating and solving general optimal control problems. This has enabled engineers and researchers to formulate and solve more complex optimal control problems than was previously possible using standard optimal control software.

The benefits of using a toolbox based on CasADi, instead of tailoring the code for each problem, are several. For instance, trouble shooting CasADi-based code is tricky, because it is hard to distinguish between faults in the problem formulation, from bugs in the discretization method. A toolbox can provide discretization methods that have been proven to work on problems with known solutions, which makes troubleshooting easier. It can also provide a good separation of problem formulation and discretization method, which makes changing solver method and settings easier. The drawback of a toolbox is that it is hard to make it as flexible as custom written code.

This thesis is about the design and implementation of an optimal control toolbox, based on CasADi, that aims to be flexible enough to solve real-world optimal control problems, while being simple enough to be used by non-experts in the field of numerical optimal control.

1.2 Purpose

The purpose has been to make numerical optimal control accessible to researchers and engineers in academia and industry. This was contributed to by designing a MATLAB toolbox, based on CasADi[1], for solving optimal control problems.

1.3 Questions

To guide the design work, two main questions and several sub-questions was identified.

1. Which methods are available for solving optimal control problems numerically?
   (a) What are their different properties?

2. How should the optimal control toolbox be designed?
   (a) How should the optimization methods be implemented in the toolbox?
   (b) How should the model be represented in the toolbox?
   (c) How should the objective function be provided by the user?
   (d) How should the constraints be specified?
   (e) What should be the default behaviour of the toolbox?
   (f) What options should be supported?

1.4 Delimitations

Optimal control is a wide field that can be divided into two classes, continuous time optimal control and discrete time optimal control. The problems studied here are of continuous time, whose basic problem formulation is given in section 2.1.1. It would simply be too time consuming and disparate to cover both classes in a master's thesis.

The toolbox is only accessible via MATLAB. There was only time to implement the toolbox in one language and in cooperation with Scania the choice was made to choose MATLAB. Since many engineers and researchers already use MATLAB, it is hoped that this choice will benefit many of them.

How to model for optimal control is not included in the thesis. Primarily this is due to that it is a different topic than designing a toolbox. For the reader interested in vehicle powertrains and the optimal control of those, Martin Sivertsson's dissertation *Optimal Control of Electrified Powertrains* [15] and Jonas Asprion's dissertation *Optimal Control of Diesel Engines* [3] are good places to start at.

There are several ways to solve optimal control problems numerically. This toolbox uses only two, namely *direct multiple shooting* and *direct collocation*. The choice for these two was based on documented successful use [8] and important drawbacks of the other methods, briefly presented in chapter 2.

The method used to design the toolbox was to study and solve optimal control problems and through the teachings gradually come up with a design. Since time was a restriction there was only time to study a few problems. Those studied were chosen in cooperation with the supervisors. If they were sufficient remains to be seen in the first round of user testing.

2

Theory

2.1 Optimal Control

Optimal control is one of the most useful and systematic methods for controller design and evaluation [11]. Its strength lies in its systematic approach of attacking control problems and in the fact that many engineering challenges are naturally formulated as optimal control problems, for instance driving from Södertälje to Katrineholm, as fast as possible, without exceeding the speed limit. The benefit of an optimal control problem formulation is especially pronounced when it comes to multiple input multiple output systems, where traditional performance measures such as settling time, static gain and phase margin may be inadequate for describing the desired behaviour [12]. This section presents the basic optimal control problem (OCP) formulation, a brief overview of the different optimal control approaches and focuses on the numerical solution of OCPs using direct methods.

The problems studied are so called continuous time optimal control problems, but is for simplicity referred to as optimal control problems. Continuous time refers to that the system's dynamics are formulated in the continuous time domain.

2.1.1 Basic problem formulation

A basic OCP formulation is represented as follows:

$$
\begin{aligned}
\min_{u(t)} \quad & E(T, x(T), P) + \int_0^T L(t, x(t), u(t), P)\ \mathrm{d}t \\
\text{s.t.} \quad & x(0) = x_0, \\
& \dot{x}(t) = f(t, x(t), u(t), P), && t \in [0, T] \\
& h(t, x(t), u(t), P) \leq 0, && t \in [0, T] \\
& h_T(T, x(T), u, P) \leq 0,
\end{aligned}
\tag{2.1}
$$

where t represents the independent variable, x the state, u the control, P the free optimization parameter vector and T the terminal value of the independent variable. The sum $E(T, x(T), P) + \int_0^T L(t, x(t), u(t), P)\ \mathrm{d}t$ is known as the objective function and must be scalar valued [8]. $E(T, x(T), P)$ is the terminal state cost, also known as a Mayer term. $L(t, x(t), u(t), P)$ is the integral cost function and may be referred to as a Lagrange term. When the objective function consists of both a Mayer term and a Lagrange term the combination can be referred to as a Bolza objective. x_0 is the state initial value, $\dot{x}(t)$ the state derivative with respect to the independent variable and $f(t, x(t), u(t), P)$ the ordinary differential equation right hand side. $h(t, x(t), u(t), P)$ is the inequality path constraints and $h_T(T, x(t), u(t), P)$ the terminal inequality constraints.

2.2 Numerical Solution to Optimal Control Problems

Most real-world optimal control problems are too large and complex to be solved analytically and therefore require the use of numerical methods. This section gives a brief overview of the available methods. Most of the section is devoted to the direct methods which are essential for solving large and complex OCPs.

2.2.1 A Brief Overview

There are three classes of methods for solving optimal control problems [8]. There is continuous time dynamic programming, the indirect methods and the direct methods.

Continuous time dynamic programming is based on solving the Hamilton-Jacobi-Bellman equation [11]. Advantages of this approach is that it provides sufficient conditions for optimality and the optimal control is given in feedback form. The disadvantages are that the Hamilton-Jacobi-Bellman partial differential equation is hard to solve numerically, and that the so-called cost-to-go function is required to be continuously differentiable, which may not be the case. When discretizing the Hamilton-Jacobi-Bellman equation the problem can become impossible to solve due to its sheer size, which can occur even for a relatively small number of states [8]. This is what is known as Bellman's curse of dimensionality.

The indirect methods are based on Pontryagin's maximum principle (PMP) which uses necessary conditions for optimality to derive a boundary value prob-

lem (see [2] for a definition) which must be solved to find the optimal control [8]. The advantages are that the theory is well established, and it can in many cases be computationally efficient. The main disadvantages of PMP is that it is hard to find an initial guess for the so called adjoint variable, and that it is ill suited for inequality constraints due to problems in selecting the active constraint [5, 8].

The direct methods convert the OCP into a NLP by discretizing the independent variable and applying a numerical integration method to discretize the continuous time dynamics [5]. The NLP is then solved using an NLP-solver. The main advantage of this family of methods is that good NLP-solvers handles inequality constraints well. The main disadvantage is computational effort.

2.2.2 Direct Methods

This section introduces two of the direct methods, namely direct multiple shooting and direct collocation, both implemented in the toolbox. It is presented in a simple form, in order for the reader to easier develop a basic understanding of the methods.

Direct single shooting, another direct method, discretizes the continuous time dynamics by numerical integration of the state trajectory [8]. The integration is performed from the initial time instant to the end time in one sweep, an illustration is found in figure 2.1. The benefit of this is that it is not necessary to provide an initial guess. The disadvantage is poor performance when the underlying system is unstable. That is because the instability is inherited by the integration method, which may cause divergence. This makes it unsuitable for optimizing unstable systems, and has therefore not been considered for implementation. A method that solves the above mentioned problem is multiple shooting and is described in further detail below. Readers interested in learning more about direct single shooting are referred to [8, 9].

NLP

Since the direct methods transform the OCP into a NLP, the NLP-formulation used is presented in equation (2.2), where $\underline{x}$ represents the lower bound on the optimization variable x, and $\bar{x}$ the upper bound.

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & \underline{x} \leq x \leq \bar{x} \\ & h(x) \leq 0 \\ & g(x) = 0 \end{aligned} \tag{2.2}$$

Direct Multiple Shooting

Prior to the mathematical description an outline of direct multiple shooting is presented in order for the reader to easier understand the mathematics.

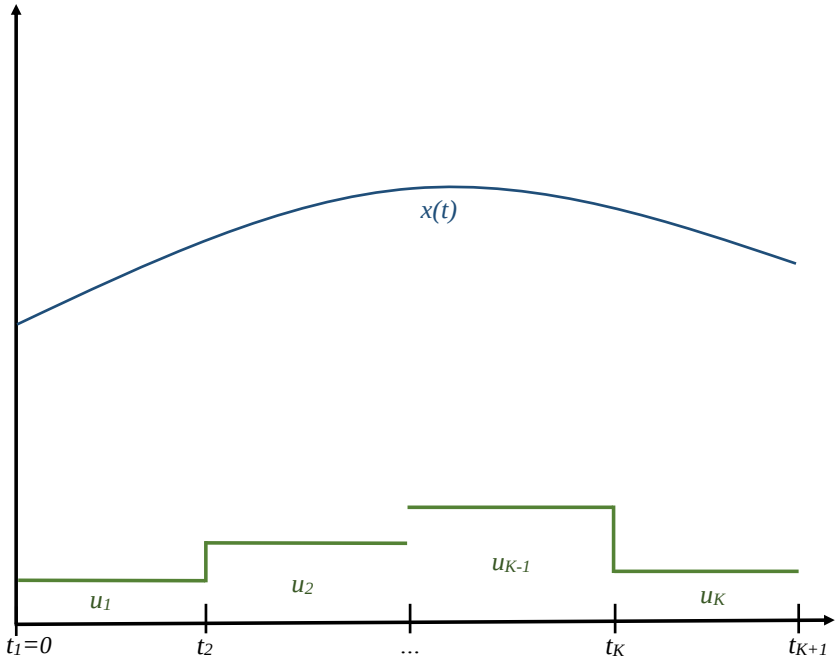

Figure 2.1: *Illustration of the direct single shooting method.*

As stated previously the basic idea of any direct method is to convert the OCP into a NLP. The conversion is performed by first discretizing the independent variable into a finite number of intervals. For convenience the independent variable is referred to as time. On each time interval the control signal is parametrized, typically as a constant, which makes the control signal piecewise constant over the entire time period. The continuous time dynamics is discretized separately on each segment using a numerical integration routine. Integrating the dynamics separately on each interval is what is meant by multiple shooting. The input to the system during the integration is the constant control signal value for that segment. Since the system is integrated separately on each time interval the overall state trajectory is not continuous. To correct this a constraint binding the trajectories together is introduced in the NLP-formulation. This way the original OCP has been transformed into a NLP. An illustration of the method is found in figure 2.2.

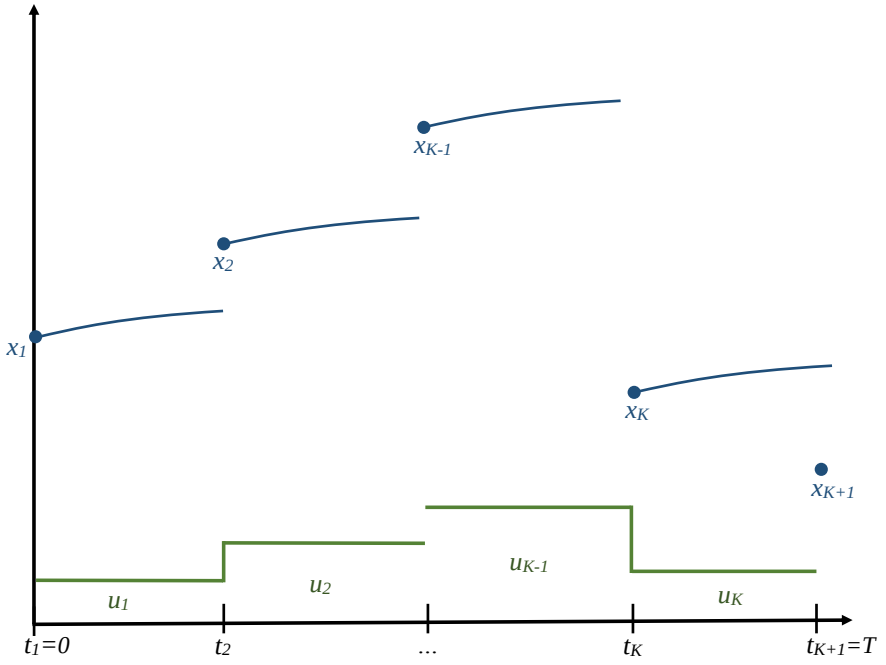

Figure 2.2: *Illustration of the direct multiple shooting method.*

Mathematically the procedure is described as follows. The time interval $t \in [0, T]$ is divided into K equidistant segments of length h according to

$$h = \frac{T}{K}, \tag{2.3}$$

where

$$\begin{aligned} t_1 &= 0 \\ t_{k+1} &= t_k + h \text{ for } k = 1, \ldots, K. \end{aligned} \tag{2.4}$$

On each time interval the control signal $u(t)$ is parametrized as

$$u(t) = u_k \text{ for } t \in [t_k,\ t_{k+1}] \text{ and } k = 1, \ldots, K. \tag{2.5}$$

The state trajectory is integrated numerically according to

$$x(t_{k+1}) = F(t_k, x_k, u_k, P) \approx \int_{t_k}^{t_{k+1}} f(t, x(t), u_k, P)\, \mathrm{d}t \tag{2.6}$$

where $F(t, x, u, P)$ is the numerical integration routine and x_k represents the state at $t = t_k$. The integral cost contribution on each segment is also calculated using

numerical integration

$$l(t_k, x_k, u_k, P) \approx \int_{t_k}^{t_{k+1}} L(t, x(t), u_k, P)\ \mathrm{d}t. \tag{2.7}$$

To force the state trajectory to be continuous the following constraint is formed:

$$x_{k+1} - F(t, x_k, u_k, P) = 0, \text{ for } k = 1, \ldots, K. \tag{2.8}$$

This yields the NLP-formulation of the OCP

$$\min_{\substack{x_1,\ldots,x_{K+1},\\ u_1,\ldots,u_K}} \quad \sum_{k=1}^{K} l(t_k, x_k, u_k, P) + E(T, x_{K+1}, P)$$

s.t.

$$\begin{aligned}
&\begin{bmatrix} \underline{x}_1 \\ \underline{u} \\ \vdots \\ \underline{x} \\ \underline{u} \\ \underline{x}_{K+1} \end{bmatrix} \leq \begin{bmatrix} x_1 \\ u_1 \\ \vdots \\ x_K \\ u_K \\ x_{K+1} \end{bmatrix} \leq \begin{bmatrix} \bar{x}_1 \\ \bar{u} \\ \vdots \\ \bar{x} \\ \bar{u} \\ \bar{x}_{K+1} \end{bmatrix} \\
&\begin{bmatrix} h(t_2, x_2, u_1, P) \\ \vdots \\ h(t_{K+1}, x_{K+1}, u_K, P) \\ h_T(T, x_{K+1}, u_K, P) \end{bmatrix} \leq 0. \\
&\begin{bmatrix} x_2 - F(t_1, x_1, u_1, P) \\ \vdots \\ x_{K+1} - F(t_K, x_K, u_K, P) \\ L(t_2, x_2, u_2, P) - l(t_1, x_1, u_1, P) \\ \vdots \\ L(t_K, x_K, u_K, P) - l(t_{K-1}, x_{K-1}, u_{K-1}, P) \end{bmatrix} = 0.
\end{aligned} \tag{2.9}$$

A common choice of integration method is a fixed step explicit Runge-Kutta method. However there are other choices, for instance can a variable step length solver be used. The use of such solvers is one of the major advantages of direct multiple shooting [9].

A more general way to parametrize the control signal is by using polynomials with constant polynomial coefficients in each interval, however a piecewise constant control signal, as described here, is the most widespread [8] and therefore used in the description.

Direct Collocation

Direct collocation is based on an idea almost identical to that of direct multiple shooting. The important difference is that the numerical integration methods

is integrated into the NLP. The integration method used is an implicit Runge-Kutta method based on collocation [2]. The basic idea behind this method is to approximate the state trajectory using polynomial interpolation. Starting with a number of predetermined points, called the collocation points, the interpolating polynomial is differentiated with respect to time. At the collocation points the differentiated polynomial is equaled to the state dynamics. The polyonomial can then be integrated and the state trajectory integrated. The benefit of this integration method is the possibility to get the highest possible order of any Runge-Kutta method, the possibility of having *stiff decay* (a property of the integration method that enables it to capture the quantitative behaviour of the system despite it being stiff [2]), and that the resulting NLP is very sparse [9, 6]. The major drawback of direct collocation is that an adaptive, error controller integration method can not be used [9].

The following mathematical description is to a large degree adapted from [1], the reason is that the implementation is based on that source. Having the same notation will hopefully facilitate for readers familiar with that source or are interested in reading it.

Direct collocation is described as follows (it might be a good idea to start by looking at figure 2.3, because this might get confusing). To begin with, the time interval $t \in [0,\ T]$ is discretized into K, equidistant, segments called control intervals according to:

$$h = \frac{T}{K}, \tag{2.10}$$

where

$$\begin{aligned} &t_{1,0,0} = 0, \\ &t_{k+1,0,0} = t_{k,0,0} + h, \quad \text{for } k = 1, \ldots, K. \end{aligned} \tag{2.11}$$

On each control interval the control signal is approximated as piecewise constant:

$$u(t) = u_k \text{ for } t \in [t_{k,0,0},\ t_{k+1,0,0}]. \tag{2.12}$$

Each control interval is further divided into N segments called collocation intervals, of size h_k:

$$h_k = \frac{h}{N}, \tag{2.13}$$

where

$$\begin{aligned} t_{k,i,0} = t_{k,0,0} + i\, h_k, \quad &\text{for } k = 1, \ldots, K, \\ &i = 0, \ldots, N. \end{aligned} \tag{2.14}$$

On each collocation interval $d + 1$, predetermined points are defined according to:

$$\begin{aligned} t_{k,i,j} = t_{k,i,0} + \tau_j\, h_k \quad &\text{for } k = 1, \ldots, K, \\ &i = 0, \ldots, N - 1, \\ &j = 0, \ldots, d. \end{aligned} \tag{2.15}$$

where $\tau_0 = 0$ and $\tau_{1,\ldots,d}$ are the collocation points. How they are calculated is described in [6]. An illustration of the procedure of discretizing the independent variable is found in figure 2.3.

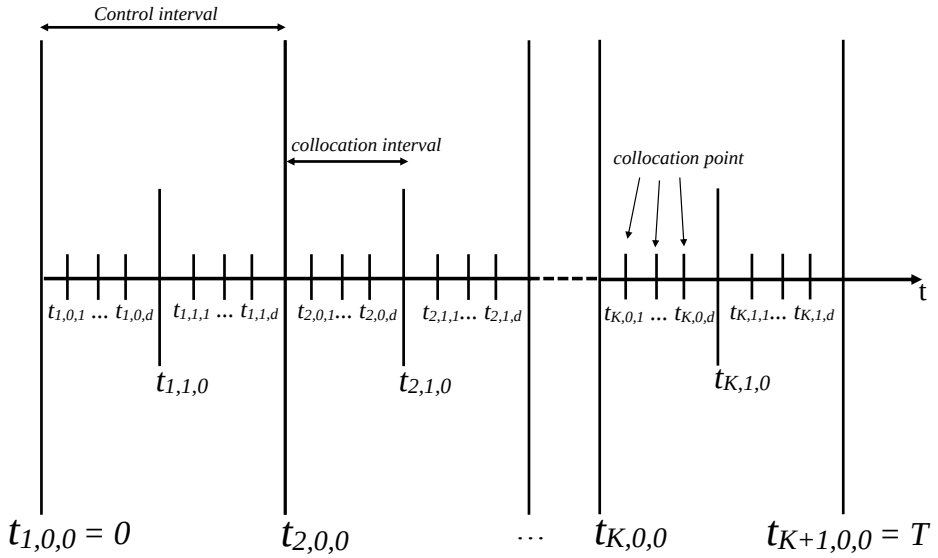

Figure 2.3: *Discretization of the independent variable using direct collocation.*

On each collocation interval the state trajectory is approximated using an interpolating polynomial written on Lagrange-form according to:

$$\begin{aligned} t &= t_{k,i,0} + h_k \tau, \quad t \in [t_{k,i,0},\ t_{k,i+1,0}] \\ x(t) &\approx \sum_{j=0}^{d} L_j(\tau)\, x_{k,i,j}, \end{aligned} \tag{2.16}$$

where

$$L_j(\tau) = \prod_{r=0, r\neq j}^{d} \frac{\tau - \tau_r}{\tau_j - \tau_r}, \tag{2.17}$$

and $x_{k,i,j}$ are the polynomial coefficients. The benefit of writing the polynomial on Lagrange form is that the bounds on the coefficients are the same as those for the state trajectory [6]. The trajectory polynomial is then differentiated and

forced to equal the dynamics at the collocation points according to:

$$f(t_{k,i,j}, x_{k,i,j}, u_k, P) = \frac{1}{h_k} \sum_{r=0}^{d} \frac{\partial L_r}{\partial \tau}(\tau_j)\, x_{k,i,r} \text{ for } \begin{matrix} k = 1, \ldots, K, \\ i = 0, \ldots, N-1, \\ j = 1, \ldots, d, \end{matrix} \tag{2.18}$$

To get the state value at the end of the interval the interpolating polynomial is simply evaluated at that point according to:

$$x^+_{k,i} = \sum_{r=0}^{d} L_r(1) x_{k,i,r} \text{ for } \begin{matrix} k = 1, \ldots, K, \\ i = 0, \ldots, N-1, \end{matrix} \tag{2.19}$$

where

$$x^+_{k,i} = \begin{cases} x_{k+1,0} & \text{if } i = N-1 \\ x_{k,i+1} & \text{otherwise.} \end{cases} \tag{2.20}$$

An illustration of the state interpolation is found in figure 2.4. The Lagrange term also needs to be integrated, this is performed by quadrature:

$$\int_0^T L(t, x(t), u(t), P)\, \mathrm{d}t \approx \sum_{k=1}^{K} \sum_{i=0}^{N-1} h_k \sum_{r=0}^{d} \int_0^1 L_r(\tau)\, \mathrm{d}\tau\, L(t_{k,i,r}, x_{k,i,r}, u_k, P). \tag{2.21}$$

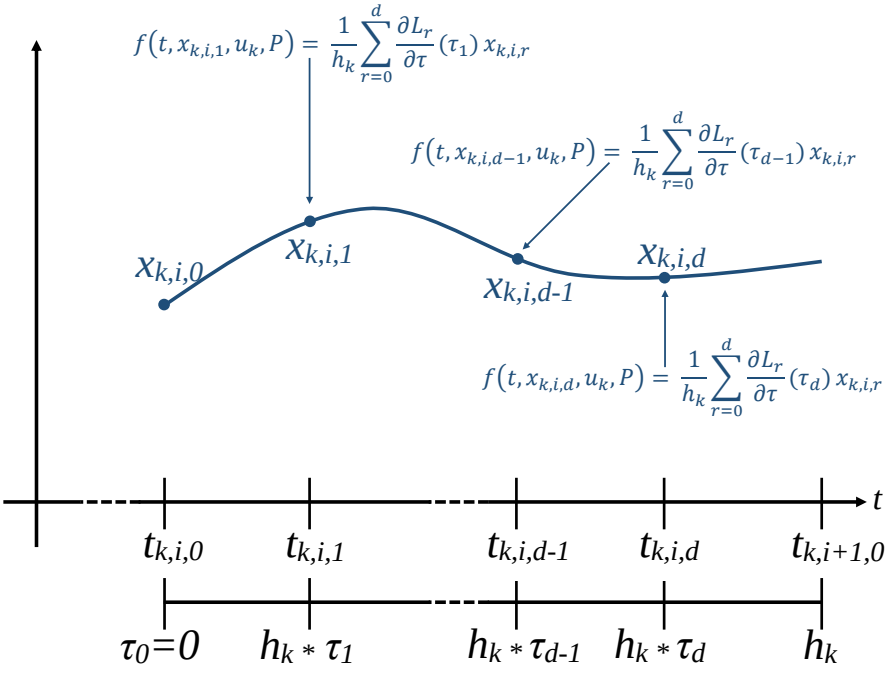

Figure 2.4: *Representation of the state trajectory.*

Together this creates the NLP

$$\min_{\substack{x_{1,0,0},\dots,x_{K,N-1,d},x_{K+1,0,0}\\ u_1,\dots,u_K}} \quad \sum_{k=1}^{K}\sum_{i=0}^{N-1} h_k \sum_{r=0}^{d} B_r L(t_{k,i,r}, x_{k,i,r}, u_k, P) + E(T, x_{K+1,0,0}, P)$$

s.t.

$$\begin{bmatrix} \underline{x}_1 \\ \underline{x} \\ \vdots \\ \underline{x} \\ \underline{u} \\ \vdots \\ \underline{x} \\ \vdots \\ \underline{x} \\ \underline{u} \\ \underline{x}_{K+1} \end{bmatrix} \le \begin{bmatrix} x_{1,0,0} \\ x_{1,0,1} \\ \vdots \\ x_{1,N-1,d} \\ u_1 \\ \vdots \\ x_{K,0,0} \\ \vdots \\ x_{K,N-1,d} \\ u_K \\ x_{K+1,0,0} \end{bmatrix} \le \begin{bmatrix} \bar{x}_1 \\ \bar{x} \\ \vdots \\ \bar{x} \\ \bar{u} \\ \vdots \\ \bar{x} \\ \vdots \\ \bar{x} \\ \bar{u} \\ \bar{x}_{K+1} \end{bmatrix}$$

$$\begin{bmatrix} h(t_{1,0,1}, x_{1,0,1}, u_1, P) \\ \vdots \\ h(t_{1,N-1,d}, x_{1,N-1,d}, u_1, P) \\ h(t_{2,0,0}, x_{2,0,0}, u_1, P) \\ h(t_{2,0,1}, x_{2,0,1}, u_2, P) \\ h(t_{2,N-1,d}, x_{2,N-1,d}, u_2, P) \\ \vdots \\ h(t_{K,0,0}, x_{K,0,0}, u_{K-1}, P) \\ h(t_{K,0,1}, x_{K,0,1}, u_K, P) \\ \vdots \\ h(t_{K,N-1,d}, x_{K,N-1,d}, u_K, P) \\ h(t_{K+1,0,0}, x_{K+1,0,0}, u_K, P) \\ h_T(T, x_{K+1,0,0}, P) \end{bmatrix} \le 0,$$

$$\begin{bmatrix} f(t_{k,i,1}, x_{k,i,1}, u_k, P) - \frac{1}{h_k}\sum_{r=0}^{d} \frac{\partial L_r}{\partial \tau}(\tau_1) x_{k,i,r} \\ \vdots \\ f(t_{k,i,d}, x_{k,i,d}, u_k, P) - \frac{1}{h_k}\sum_{r=0}^{d} \frac{\partial L_r}{\partial \tau}(\tau_d) x_{k,i,r} \\ x_{k,i}^{+} - \sum_{r=0}^{d} L_r(1) x_{k,i,r} \end{bmatrix} = 0,$$

$$\text{for } k = 1,\dots,K,\ \ i = 0,\dots,N-1. \tag{2.22}$$

Collocation points

The choice of collocation points influences the behaviour of the integration method [2]. Two sets of points are more common than the other, the Legendre points and the Radau points, their values up to order five is given in table 2.1 and included for completeness [6]. Both are high order methods. The Legendre points have an order of $2d$ and Radau $2d-1$. If the system is stiff the Radau points are preferable because they provide stiff decay.

Points d	Legendre points	Radau points
1	0.500000	1.000000
2	0.211325 0.788675	0.333333 1.000000
3	0.112702 0.500000 0.887298	0.155051 0.644949 1.000000
4	0.069432 0.330009 0.669991 0.930568	0.088588 0.409467 0.787659 1.000000
5	0.046910 0.230765 0.500000 0.769235 0.953090	0.057104 0.276843 0.583590 0.860240 1.000000

Table 2.1: *Collocation points up to order five. Source: table 10.1 in [6].*

2.2.3 Integrating the Objective Function

Every differentiable Mayer term can be reformulated into a Lagrange term, the same is valid for the other way around [5]. This means that they are mathematically equivalent. Despite this the Lagrange term is numerically preferable, partly this is because the Mayer formulation results in a larger NLP [3], and partly because it is not as accurate. Since the Mayer term needs to be included as an extra state in the OCP formulation this increases the size of the NLP, which also means that the discretization error is introduced twice [5].

2.2.4 Mesh Refinement

Mesh refinement is a technique for controlling the discretization error [5]. It resembles in many ways the error control of adaptive step length integration methods. By estimating how large the discretization error is the algorithm decides whether to make the mesh finer, or if it is sufficient as it is. This way the problem is solved with a finer and finer grid until an acceptable accuracy is reached.

Mesh refinement as a method of error control is not implemented in the toolbox, but the idea of solving the OCP with a finer and finer grid is supported. By solving the OCP multiple times, starting with a coarse grid, and use the previous solution as an initial guess, it is sometimes possible to achieve faster convergence than trying to solve a finely resolved OCP with a poor initial guess from the start.

2.2.5 Scaling

Proper scaling is very important when it comes to solving NLPs [4]. Despite this there is no scaling method that guarantees a proper scaling of the NLP, mainly due to the fact that there is no definition of what proper scaling is. For OCPs it is preferable to scale the OCP and hope that it results in a properly scaled NLP. However, some rules of thumb can be applied. Choose the scaling factor X so that the state variable take on values ranging from zero to one, and shift the scaled variable using r_x according to:

$$x(t) = X\tilde{x}(t) + r_x. \tag{2.23}$$

The same technique is applied to the control signal, using the scaling factor U and shifting term r_u:

$$u(t) = U\tilde{u}(t) + r_u. \tag{2.24}$$

The dynamics are then scaled according to:

$$\dot{\tilde{x}}(t) = f(t, x(t), u(t), P)/X. \tag{2.25}$$

2.3 CasADi

CasADi [1] provides the key technology for making the optimal control toolbox possible. CasADi is an open-source software for dynamic optimization. It is not a general purpose OCP solver, but has been designed to provide the necessary building blocks for solving OCPs. Key features of CasADi are algorithmic differentiation, interfaces to NLP and ODE solvers, and functions tailored for implementing direct methods. How CasADi works is a dissertation of its own (namely [1]) and is therefore covered in no further detail, however it is noticed that CasADi made the toolbox possible.

2.4 Optimal Control Software

There are a bunche of optimal control software on the market, some are open source and some are proprietary, in table 2.2 a table of some of them are presented.

Name	Discretization method	NLP-solver	Accessible via
This toolbox	Direct multiple shooting and direct collocation	all CasADi supported solvers	MATLAB
PROPT	Pseudospectral	KNITRO, SNOPT, CONOPT, CPLEX	MATLAB
MUSCOD-II	Multiple shooting	SQP-method	C, FORTRAN, gPROMS
ACADO	Multiple shooting	SQP-method	C++, MATLAB
DIDO	Pseudospectral	-	MATLAB

Table 2.2: *Table of Optimal Control Software*

3
Method

3.1 Introduction

The method for designing the toolbox was to read theory regarding the numerical solution of OCPs, then solve optimal control problems using CasADi to gain practical experience, and finally to combine theory with experience to come up with a design. Three main problems have been studied during the work. One simple problem with multiple formulations and an analytical solution, one problem with nonlinear dynamics and time variant constraints, and one complex multiple phase problem were the dynamics changes with the phase.

3.2 The Studied Optimal Control Problems

This section presents the three main problems that was studied during the design process.

3.2.1 The Bryson-Denham Problem

The Bryson-Denham problem is a classical minimum energy optimal control problem and is presented in [7, p. 122]. The formulation presented in [7] is

the following:

$$\begin{aligned}
\min_{a(t)} \quad & \frac{1}{2}\int_0^1 a(t)^2 \,\mathrm{d}t \\
\text{s.t.} \quad & \\
& \dot{v}(t) = a(t), \\
& \dot{x}(t) = v(t), \\
& v(0) = -v(1) = 1, \\
& x(0) = x(1) = 0, \\
& x(t) \leq l.
\end{aligned} \tag{3.1}$$

The analytical solution to the problem for $0 < l \leq 1/6$ is:

$$\begin{aligned}
a(t) &= \begin{cases} -\frac{2}{3l}\left(1 - \frac{t}{3l}\right), & 0 \leq t \leq 3l \\ 0, & 3l \leq t \leq 1 - 3l \\ -\frac{2}{3l}\left(1 - \frac{1-t}{3l}\right), & 1 - 3l \leq t \leq 1 \end{cases} \\
v(t) &= \begin{cases} \left(1 - \frac{t}{3l}\right)^2, & 0 \leq t \leq 3l \\ 0, & 3l \leq t \leq 1 - 3l \\ -\left(1 - \frac{1-t}{3l}\right)^2, & 1 - 3l \leq t \leq 1 \end{cases} \\
x(t) &= \begin{cases} l\left(1 - \left(1 - \frac{t}{3l}\right)^3\right), & 0 \leq t \leq 3l \\ l, & 3l \leq t \leq 1 - 3l \\ -l\left(1 - \left(1 - \frac{1-t}{3l}\right)^3\right), & 1 - 3l \leq t \leq 1. \end{cases}
\end{aligned} \tag{3.2}$$

For $l = \frac{1}{9}$ the analytical solution is illustrated in figure 3.1.

The Bryson-Denham problem can easily be augmented to capture different types of problem formulations. Below follows how the problem can be augmented to include free end time, a Mayer formulation (in contrast to the Lagrange formulation presented above) and a two phase formulation.

Free end time

To augment (3.1) to include free end time, the end time is, instead of being a fixed value, changed into a variable whose value is determined in the optimization. For

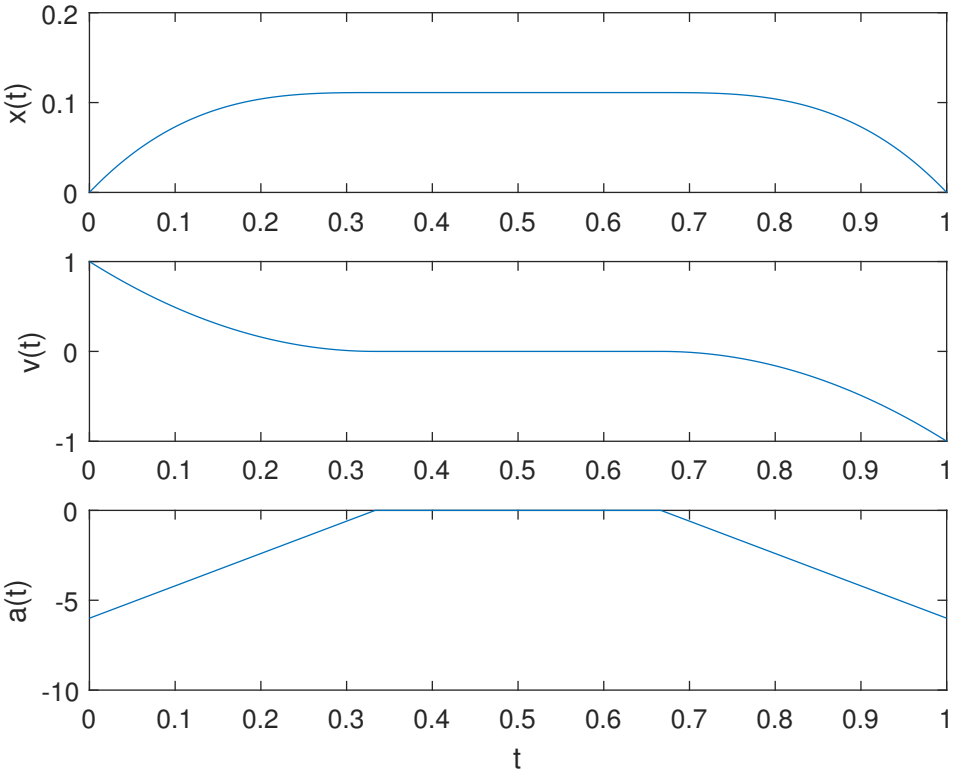

Figure 3.1: *Analytical solution to problem* (3.1) *for* $l = \frac{1}{9}$.

(3.1) the new optimization variable T is introduced according to:

$$\begin{aligned} \min_{a(t)} \quad & \frac{1}{2}\int_0^T a(t)^2 \,\mathrm{d}t \\ \text{s.t.} \quad & \\ & \dot{v}(t) = a(t), \\ & \dot{x}(t) = v(t), \\ & v(0) = -v(T) = 1, \\ & x(0) = x(T) = 0, \\ & x(t) \leq l. \end{aligned} \tag{3.3}$$

Mayer formulation

Problem (3.3) can also be reformulated as a Mayer formulation instead of the Lagrange formulation presented in (3.1):

$$\begin{aligned} \min_{a(t)} \quad & z(T) \\ \text{s.t.} \quad & \\ & \dot{z}(t) = a(t)^2/2 \\ & \dot{v}(t) = a(t), \\ & \dot{x}(t) = v(t), \\ & v(0) = -v(T) = 1, \\ & x(0) = x(T) = 0, \\ & x(t) \leq l. \end{aligned} \tag{3.4}$$

Two-phase problem

Problem (3.3) can also be converted into a two phase OCP:

$$\min_{a(t)} \quad \frac{1}{2}\int_0^{T_1} a(t)^2 \,\mathrm{d}t + \frac{1}{2}\int_{T_1}^{T_2} a(t)^2 \,\mathrm{d}t$$

s.t.

$$\begin{aligned} \dot{v}(t) &= a(t), \\ \dot{x}(t) &= v(t), \\ x(t) &\leq l. \\ v(0) &= 1, \\ x(0) &= 0, \\ v(T_1) &= 0, \\ x(T_1) &= l, \\ v(T_2) &= -1, \\ x(T_2) &= 0. \\ 2T_1 - T_2 &= 0 \end{aligned} \tag{3.5}$$

3.2.2 An Optimal Control Benchmark: Transient Optimization of A Diesel-Electric Powertrain

The following problem is presented in [16] and is designed as a benchmark problem for optimal control software [16]. For that reason the problem has been included as one of the reference problems, but also for its nonlinear dynamics and time varying constraints. The nomenclature is taken from the original paper [16], apart from a few changes, and is presented in table 3.1 and 3.2.

Symbol	Description	Unit
$\dot{m}$	Mass flow	kg/s
p	Pressure	Pa
t_f	End time	s
u_f	Injected fuel	mg/cycle
u_{wg}	Wastegate position	-, [0, 1]
BSR	Blade speed ratio	-
E	Energy	J
F	Force	N
J	Inertia	kg $\cdot$ m^2
M	Torque	Nm
P	Power	W
T	Temperature	K
λ	Air-fuel equivalence ratio	-
ϕ	Fuel-air equivalence ratio	-
ρ	Density	kg/m^3
ω	Rotational speed	rad/s

Table 3.1: *Optimal control benchmark symbol description*

The problem (3.9) consists of optimizing the transient response of a diesel-electric powertrain when an increase in the generator electrical power is required. The model is a four state, three control, mean value engine model (MVEM) paired with an electrical generator (the combination of the internal combustion engine and electrical generator is referred to as genset) of a medium-duty electrified powertrain. The dynamics are nonlinear and the constraints, as included here, are time varying.

For the reader not familiar with engine modeling and control, a MVEM is a control-oriented model designed to study the fuel- and air-path of the engine [10]. The individual cylinders are not modeled, instead the mean value over a cycle is used, and thereof its name. By a Diesel-electric powertrain a configuration having both an internal combustion engine and electric motor/generator is meant, colloquially referred to as a hybrid.

The four states $x(t)$ are, engine speed $\omega_{ice}(t)$, intake manifold pressure (the pressure in the volume prior to the combustion chamber) $p_{im}(t)$, exhaust manifold pressure (pressure in the volume after the combustion chamber, before the turbocharger) $p_{em}(t)$, and the turbocharger speed $\omega_t(t)$:

$$x(t) = \Big[\omega_{ice}(t), \quad p_{im}(t), \quad p_{em}(t), \quad \omega_t(t)\Big]^T \tag{3.6}$$

The three control signals $u(t)$ are, injected amount of fuel $u_f(t)$, wastegate position $u_{wg}(t)$ and generator power $P_{gen}(t)$:

$$u(t) = \Big[u_f(t), \quad u_{wg}(t), \quad P_{gen}(t)\Big]^T \tag{3.7}$$

Index	Description
a	air
c	compressor
$c, surge$	compressor surge limit
em	exhaust manifold
f	fuel
gen	generator electrical
$genset$	combustion engine and generator
ice	Internal combustion engine
im	Intake manifold
max	maximum
$mech$	generator mechanical
min	minimum
t	turbine
tm	turbine mechanical
tc	turbocharger
wg	wastegate

Table 3.2: *Optimal control benchmark index description*

The differential equation describing the dynamics is

$$\dot{x}(t) = f(x(t), u(t)) = \begin{bmatrix} \frac{P_{ice}(t) - P_{mech}(t)}{J_{genset}\omega_{ice}(t)} \\ \frac{R_a T_{im}}{V_{im}}(\dot{m}_c(t) - \dot{m}_{ac}(t)) \\ \frac{R_e T_{em}(t)}{V_{em}}(\dot{m}_{ac}(t) + \dot{m}_f(t) - \dot{m}_t(t) - \dot{m}_{wg}(t)) \\ \frac{P_t(t)\eta_{tm} - P_c(t)}{J_{tc}\omega_{tc}(t)} \end{bmatrix}. \tag{3.8}$$

The optimal control problem is formulated either as minimum fuel or minimum

time according to the following:

$$\min_{u(t)} \quad \int_0^{t_f} \dot{m}_f(t)\,\mathrm{d}t \text{ or } \min_{u(t)} t_f$$

s.t.

$$\begin{aligned}
x(0) &= x_0 \\
\dot{x}(t) &= f(x(t), u(t)) \\
x_{min} \leq x(t) &\leq x_{max} \\
u_{min} \leq u(t) &\leq u_{max} \\
0 \leq P_{gen}(t) &\leq 100 \text{ kW} \\
P_{ice}(x(t), u(t)) &\leq P_{ice,max}(x(t), u(t)) \\
\Pi_c &\leq \Pi_{c,surge} \\
BSR_{min} \leq BSR(x(t), u(t)) &\leq BSR_{max} \\
0 \leq \phi(x(t), u(t)) &\leq 1/\lambda_{min} \\
P_{gen}(t_f) &= 100 \text{ kW} \\
E_{gen}(t_f) &= 100 \text{ kJ} \\
\dot{x}(t_f) &= 0.
\end{aligned} \tag{3.9}$$

The optimal solution to problem (3.9) is accompanied with the paper and has been calculated using PROPT [18] and ACADO [14]. The minimum fuel solution is illustrated in figure 3.2.

3.2.3 Optimal Control of a Diesel-Electric Powertrain During an Up-Shift

This OCP, presented in [13], consists of finding the optimal control of an electrified, heavy-duty powertrain, during an up-shift. The model consists of the same genset as in (3.8) and a flexible driveline model, where the flexibilities have been lumped into a single flexibility in the driveshaft.

For simplicity the nomenclature is the same as that of the previously presented OCP (3.9) (see table 3.1 och 3.2 for nomenclature), which is the same nomenclature used in [13]. Indices specific to this problem is presented in table 3.3.

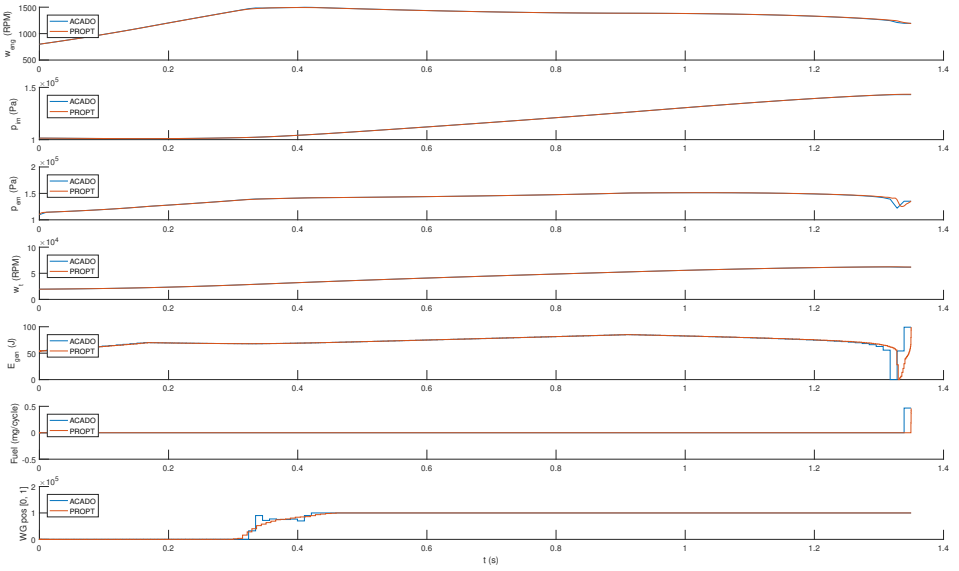

Figure 3.2: *Minimum fuel solution to problem* (3.9).

Index	Description
ds	driveshaft
tr	transmission
w	wheel
D	drag
g	gear
fd	final drive

Table 3.3: *Index specific to the up-shift OCP* (3.15)

The complete model is not presented, it would simply consume too much space, instead enough to form the OCP is presented. For the complete model the reader is referred to [17] for the genset model, and [13] for the phase specific models.

In essence, a gear shift using an automated manual transmission can be broken down into three steps [13]. First, reduce transmission torque and disengage the present gear. Two, prepare the new gear and match rotational speeds of the input and output shafts of the gearbox. Three, accelerate to previous vehicle speed. During the first step the genset is connected to the wheels via the transmission and driveline. During the second phase the wheels roll freely. During the third step the genset and wheels are connected again, but this time the transmission inertia and gear ratio is different. In order to capture this in an OCP, the problem is divided into three phases corresponding to the three steps described above. In [13] the three phases are named *torque phase*, *synchronisation phase* and *inertia*

phase, the same designation is used here.

To account for the complete powertrain, not just the genset, the genset (3.8) is extended with three new states. Transmission speed $\omega_{tr}(t)$, driveshaft windup angle $\theta_{ds}(t)$ and wheel speed $\omega_w(t)$:

$$x(t) = \left[\omega_{ice}(t), \quad p_{im}(t), \quad p_{em}(t), \quad \omega_t(t), \quad \omega_{tr}(t), \quad \theta_{ds}(t), \quad \omega_w(t)\right]^T. \tag{3.10}$$

Since the dynamics of the genset do not change with the phase it is denoted $f(x(t), u(t))$, and calculated according to (3.8). For the torque phase, the dynamics are calculated as follows.

$$f_1(x(t), u(t)) = \begin{bmatrix} f(x(t), u(t)) \\ \dot{\omega}_{ice}(t)/\gamma_{g1} \\ \omega_{ice}(t)/(\gamma_{g1}\gamma_{fd}) - \omega_w(t) \\ (M_{ds}(t) - 0.5\rho_a c_D A \omega_w(t)^2 r_w^3 - F_r r_w)/(J_w + m_v r_w^2) \end{bmatrix} \tag{3.11}$$

where $M_{ds}(t) = k_{ds}\theta_{ds}(t) + b_{ds}(\omega_{ice}(t)/(\gamma_{g1}\gamma_{fd}) - \omega_w(t))$. For the synchronisation phase the dynamics are calculated according to:

$$f_2(x(t), u(t)) = \begin{bmatrix} f(x(t), u(t)) \\ (-b_{tr}\omega_{tr}(t) - M_{ds}/\gamma_{fd})/J_{tr2} \\ \omega_{tr}(t)/\gamma_{fd} - \omega_w(t) \\ (M_{ds}(t) - 0.5\rho_a c_D A \omega_w(t)^2 r_w^3 - F_r r_w)/(J_w + m_v r_w^2) \end{bmatrix} \tag{3.12}$$

where $M_{ds}(t) = k_{ds}\theta_{ds}(t) + b_{ds}(\omega_{tr}(t)/\gamma_{fd} - \omega_w(t))$. For the inertia phase the dynamics are calculated in the same way as in the torque phase (3.11), with the difference that a new gear ratio γ_{g2} is used, and the inertia the engine experiences changes with the new gear according to:

$$f_3(x(t), u(t)) = \begin{bmatrix} f(x(t), u(t)) \\ \dot{\omega}_{ice}(t)/\gamma_{g2} \\ \omega_{ice}(t)/(\gamma_{g2}\gamma_{fd}) - \omega_w(t) \\ (M_{ds}(t) - 0.5\rho_a c_D A \omega_w(t)^2 r_w^3 - F_r r_w)/(J_w + m_v r_w^2) \end{bmatrix}, \tag{3.13}$$

where $M_{ds}(t) = k_{ds}\theta_{ds}(t) + b_{ds}(\omega_{ice}(t)/(\gamma_{g2}\gamma_{fd}) - \omega_w(t))$.
For objective function, minimum *jerk* in the transmission speed was chosen [13]. For this problem it is defined as the squared time derivative of the transmission acceleration according to (3.14).

$$\begin{aligned} jerk &= \dot{\alpha}_{tr}(t)^2 \\ \alpha_{tr}(t) &= \dot{\omega}_{tr}(t) \end{aligned} \tag{3.14}$$

To complete the OCP formulation the boundary conditions needs to be determined. The initial value is determined from a desired vehicle speed and calculated backwards through the driveline, which results in the corresponding rotational speeds of wheels, transmission and engine. Furthermore, the dynamics are required to be zero at $t = 0$.

For the boundary between the torque and synchronisation phase the genset torque is required to be zero according to: $M_{genset}(t_1) = M_{ice}(t_1) + P_{mech}(t_1)/\omega_{ice}(t_1) = 0$. This enables the present gear to be disengaged.

For the boundary between the synchronisation och inertia phase the same condition that was applied to the previous boundary is added: $M_{genset}(t_2) = M_{ice}(t_2) + P_{mech}(t_2)/\omega_{ice}(t_2) = 0$. An additional constraint that forces the engine to match the speed of the prepared gear is also added, in order to enable the new gear.

For the terminal conditions, the optimization is required to stop when the model reaches stationary conditions, the wheel speed is the same as the beginning of the up-shift, and the generator energy reaches the same value as the beginning of the up-shift. The OCP, including the boundary conditions, is presented in (3.15). The solution is illustrated in figure 3.3 and 3.4.

$$\min_{x,u} \quad \int_0^{t_1} \dot{\alpha}_{tr}(t)^2 \,\mathrm{d}t + \int_{t_1}^{t_2} \dot{\alpha}_{tr}(t)^2 \,\mathrm{d}t + \int_{t_2}^{t_3} \dot{\alpha}_{tr}(t)^2 \,\mathrm{d}t$$

s.t.

$$\begin{aligned}
u_{min} &\leq u(t) \leq u_{max} \\
x_{min} &\leq x(t) \leq x_{max} \\
P_{ice}(x(t), u(t)) &\leq P_{ice,max}(x(t), u(t)) \\
\Pi_c(x(t), u(t)) &\leq \Pi_{c,surge} \\
BSR_{min} &\leq BSR(x(t), u(t)) \leq BSR_{max} \\
0 &\leq \phi(x(t), u(t)) \leq 1/\lambda_{min} \\
\dot{x}(t) &= \begin{cases} f_1(x(t), u(t)), & 0 \leq t \leq t_1 \\ f_2(x(t), u(t)), & t_1 \leq t \leq t_2 \\ f_3(x(t), u(t)), & t_2 \leq t \leq t_3 \end{cases} \\
\omega_{ice}(0) &= \omega_{ice,0} \\
\omega_{tr}(0) &= \omega_{ice,0} \,/\, \gamma_{g1} \\
\omega_w(0) &= \omega_{ice,0} \,/\, (\gamma_{g1}\gamma_{fd}) \\
E_{gen}(0) &= 0 \\
\dot{x}(0) &= 0 \\
M_{genset}(t_1) &= M_{genset}(t_2) = 0 \\
\omega_{ice}(t_2) &= \omega_{tr}(t_2)\,\gamma_{g2} \\
\omega_w(t_3) &= \omega_w(0) \\
P_{gen}(t_3) &= 0 \\
E_{gen}(t_3) &= 0 \\
\dot{x}(t_3) &= 0.
\end{aligned} \tag{3.15}$$

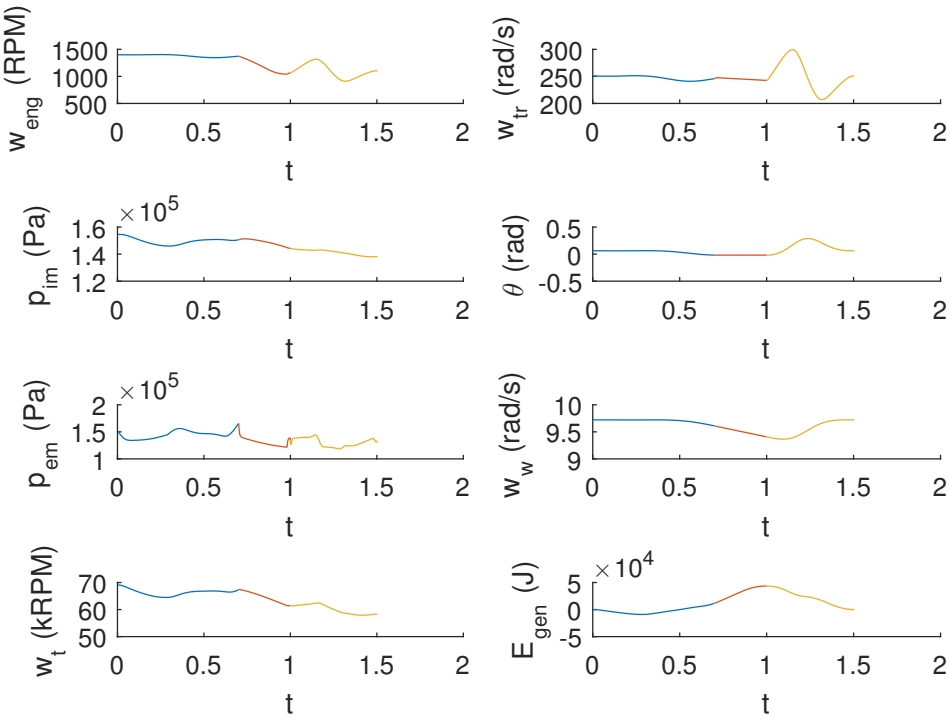

Figure 3.3: *Optimal state trajectory for problem* (3.15)*. The blue color shows the state trajectory in the torque phase, the red shows the trajectory in the synchronization phase, and the yellow the trajectory in the inertia phase.*

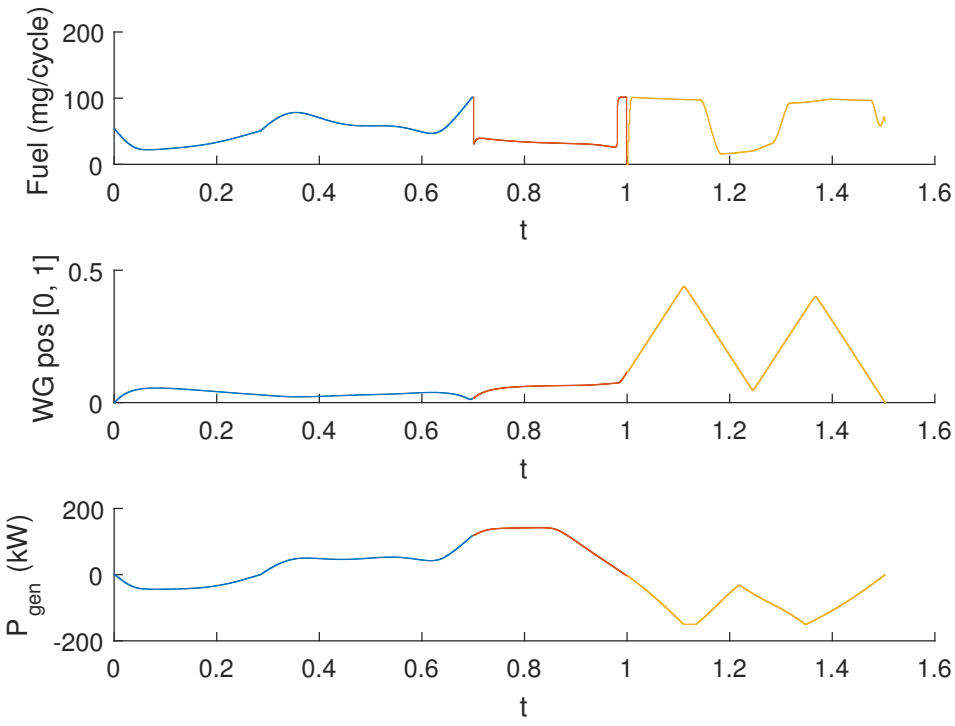

Figure 3.4: *Optimal control signal for problem* (3.15)*. The blue color shows the control signal in the torque phase, the red shows the control signal in the synchronization phase, and the yellow the control signal in the inertia phase.*

4

Implementation

4.1 Entering Optimal Control Problems

An OCP is represented using a data type called `ocp_ocp`. It consist of four parts: Phases, bounds on the free optimization variables, symbolic variables and properties.

The phases are, part from the free optimization parameters P, what form the OCP. A phase is represented using the data type `ocp_phase`. It is defined in a way similar as the basic OCP (2.1), with the difference that an `ocp_phase` is formulated in a slightly more general way. The mathematical description of how the `ocp_phase` is defined is presented in (4.1).

$$
\begin{aligned}
\min_{x(t),\,u(t)} \quad & E(T, x(T), u(T), P) + \int_0^T L(t, x(t), u(t), P)\ \mathrm{d}t \\
\text{s.t.} \quad & \dot{x}(t) = f(t, x(t), u(t), P) \\
& \underline{T} \leq T \leq \bar{T} \\
& \underline{x} \ \leq \ x \ \leq \bar{x} \\
& \underline{x}_0 \leq x(0) \leq \bar{x}_0 \\
& \underline{x}_T \leq x(T) \leq \bar{x}_T \\
& \underline{u} \ \leq \ u \ \leq \bar{u} \\
& g_i(0, x(0), u(0), P) = 0 \\
& g_f(T, x(T), u(T), P) = 0 \\
& h(t, x(t), u(t), P) \leq 0 \\
& h_i(0, x(0), u(0), P) \leq 0 \\
& h_f(T, x(T), u(T), P) \leq 0
\end{aligned}
\tag{4.1}
$$

The free optimization parameter vector P is not included in `ocp_phase`, instead it is entered into the `ocp_ocp` where it is subjected to the constraint $\underline{P} \leq P \leq \bar{P}$.

The symbolic variables (t, x, u and P) are used to create the mathematical expressions, for instance the Lagrange term $L(t, x, u, P)$ or dynamics $f(t, x, u, P)$.

The properties are information regarding the size of the problem. This include number of states, number of controls, number of free optimization parameters and number of phases.

In order to illustrate how an OCP is entered into the toolbox a small example where (3.3) is entered is presented below.

```
function ocp = bryson_denham_single_phase_ocp()

% OCP properties
nx = 2;
nu = 1;
np = 0;
n_phase = 1;

% Symbolic variables
[t, x, u, p] = ocp_var('t', nx, nu, np);

% Phase description
xi = [0; 1];        % Inital value
xdot = [x(2); u];   % Dynamics
xf = [0; -1];       % Terminal value

phase_desc = ...
    {'L', u^2, ...          % Lagrange term
     'f', xdot, ...         % Dynamics
     'T_lb', 1, ...         % End time lower bound
     'T_ub', 1, ...         % End time upper bound
     'x_lb', [0; -1], ...   % State lower bound
     'x_ub', [1/9; 1], ... % State upper bound
     'xi_lb', xi, ...       % Inital state lower bound
     'xi_ub', xi, ...       % Inital state upper bound
     'xf_lb', xf, ...       % Terminal state lower bound
     'xf_ub', xf, ...       % Terminal state upper bound
     'u_lb', -inf, ...      % Control lower bound
     'u_ub', inf};          % Control upper bound

% Create phase
p1 = ocp_phase(phase_desc);

% OCP formulation
ocp_desc={'t', t, ... % Symbolic variables
          'x', x, ...
          'u', u, ...
```

```
          'p', p, ...
          'nx', nx, ... % Problem size
          'nu', nu, ...
          'np', np, ...
          'n_phase', n_phase};

% Create OCP
ocp = ocp_ocp(ocp_desc, p1);

end
```

4.2 Representing the Model

The models that can be entered are limited to ordinary differential equations (ODEs) and required to be entered on the format $\dot{x}(t) = f(t, x, u, P)$. It is up to the user to design how to implement the model equations, but certain ways of doing it are more beneficial than others.

To facilitate the simulation of the model it should be implemented in a separate function. That function should return the state derivative, but also any constraints and other calculations of interest to the OCP formulation. To use MATLABs ODE-solvers the model function needs simply be wrapped into a function that conforms to the ODE-Solvers standard.

To clarify how a model can be implemented the torque phase dynamics (3.11) implementation is presented:

```
function [dX, h, c] = torque_phase_dynamics(X, U, param)

% Indices
[we, pim, pem, wtc, wtr, flex, ww, e] = enum(8);
[umf, uwg, pgen] = enum(3);

% Variables
w_e = X(we);
theta = X(flex);
w_w = X(ww);
P_gen = U(pgen);

% Air and drag resistnace
F_a = 0.5 * param.rho_air * param.c_d * ...
   param.A_f * w_w^2 * param.r_w^2;

% Gear ratio
i_tot = param.i_t(1) * param.i_f;

% Drive shaft torque
T_ds = param.k_ds*theta + param.c_ds * (w_e/i_tot - w_w);
```

```
% Change intertia depending on phase
param.J_genset = param.J_phase(1);

% MVEM dynamics and constratins
[dX, h, c] = MVEM(X, U, T_ds/i_tot, param);

% Dynamics not part of MVEM
dw_tr = dX(we)/param.i_t(1);
dtheta = (w_e/i_tot - w_w);
dw_w = (T_ds - F_a*param.r_w - ...
   param.F_r*param.r_w)/param.J_veh;
dE = P_gen;

% RHS
dX = [dX; dw_tr; dtheta; dw_w; dE];

% Aux calcs
c.dtheta = dtheta;
c.dw_tr = dw_tr;
c.T_ds = T_ds;

end
```

4.3 Discretization Methods

The toolbox provides two ways of discretizing OCPs, either using direct multiple shooting or direct collocation. The implementation follows the theory presented in theory chapter. However, certain things are worth pointing out. Therefore follows a description of the two implementations starting with multiple shooting.

4.3.1 Multiple Shooting

The integration method used is a fixed step, fourth order, explicit Runge-Kutta method. It discretizes the state and integral cost separately in every control interval. Two settings are controllable: the number of control intervals, K, and the number of Runge-Kutta steps, M, per control interval. If the problem contains multiple phases it is possible to choose a different number of control intervals and Runge-Kutta steps for each phase.

The inequality constraints are calculated in the following way:

$$h_k = h(t_k, x_k, u_{k-1}, P) \text{ for } k = 2, \ldots, K+1 \tag{4.2}$$

This means that the inequality constraints are not imposed at $t = 0$.

The NLP variable vector resulting from the discretization is found in (4.3), where $\mathbb{V}$ is the number of phases and the elevated index represents the phase,

not an exponent. The indices to the control and state variables in the NLP vector are stored and returned when a call to the discretization function has been made.

$$\begin{bmatrix} T_1 \\ \vdots \\ T_{\mathbb{V}} \\ P \\ x_1^1 \\ u_1^1 \\ \vdots \\ x_{K(1)}^1 \\ u_{K(1)}^1 \\ x_{K(1)+1}^1 \\ \vdots \\ x_1^{\mathbb{V}} \\ u_1^{\mathbb{V}} \\ \vdots \\ x_{K(\mathbb{V})}^{\mathbb{V}} \\ u_{K(\mathbb{V})}^{\mathbb{V}} \\ x_{K(\mathbb{V})+1}^{\mathbb{V}} \end{bmatrix} \tag{4.3}$$

To give an example of the syntax, the following code where problem (3.3) is solved using direct multiple shooting, is provided below. Note, the function `bryson_denham_single_phase_ocp()` was presented in a previous example.

```
% Formulate OCP
ocp = bryson_denham_single_phase_ocp();

% Initial guess: zeros
v0 = struct;
v0.T = 2;
v0.p = 0;
v0.x = @(t) interp1([0, v0.T], [0 0; 0 0], t, 'linear', 0);
v0.u = @(t) interp1([0, v0.T], [0, 0], t, 'linear', 0);

% Discretize and solve NLP
K = 100;  % Control intervals in each phase
opts = {'rk_steps', 10}; % RK steps per control interval
ocp_sol = ocp_solve(ocp, v0, 'shooting', K, opts);
```

4.3.2 Direct Collocation

The order of the implicit Runge-Kutta method depends on the choice of collocation points, and the number of collocation points, both up to the user to decide. The implementation has two modes, `normal` and `rigid`. The difference between the modes is in the treatment of the inequality constraints. In `normal` mode the inequality constraints are only enforced at the end of the control intervals, this is the same treatment that is made in the direct multiple shooting case. In contrast to direct multiple shooting, direct collocation also defines the state points in between the control intervals $x_{k,i,j}$ (provided that more than one collocation point is used), this makes it possible to enforce the inequality constraints on these points as well. In the toolbox that is done by choosing the `rigid` mode. A mathematical description of how the inequality constraints are calculated is shown in (4.4).

$$h_{k,i,j} = \begin{cases} 0 & \text{if } k = 1,\ i = 0,\ j = 0 \\ h(t_{k,0,0}, x_{k,0,0}, u_{k-1}, P) & \text{if } k \neq 1,\ i = 0,\ j = 0 \\ h(t_{k,i,j}, x_{k,i,j}, u_k, P) & \text{if } k \geq 1,\ i > 0,\ j > 0 \\ h(t_{K+1,0,0}, x_{K+1,0,0}, u_K, P) & \text{if } k = K+1 \end{cases} \tag{4.4}$$

$$\text{for } k = 1, \ldots, K+1 \quad i = 0, \ldots, N-1 \quad j = 0, \ldots, d.$$

The NLP variable vector that the direct collocation implementation creates is found in (4.5), where V is the number of phases.

$$\begin{bmatrix} T_1 \\ \vdots \\ T_V \\ P \\ x^1_{1,0,0} \\ x^1_{1,0,1} \\ \vdots \\ x^1_{1,N-1,d} \\ u^1_1 \\ \vdots \\ x^1_{K(1),0,0} \\ \vdots \\ x^1_{K(1),N(1)-1,d} \\ u^1_{K(1)} \\ x^1_{K(1)+1,0,0} \\ \vdots \\ x^{\mathbb{V}}_{1,0,0} \\ x^{\mathbb{V}}_{1,0,1} \\ \vdots \\ x^{\mathbb{V}}_{1,N-1,d} \\ u^{\mathbb{V}}_1 \\ \vdots \\ x^{\mathbb{V}}_{K(V),0,0} \\ \vdots \\ x^{\mathbb{V}}_{K(V),N(V)-1,d} \\ u^{\mathbb{V}}_{K(V)} \\ x^{\mathbb{V}}_{K(V)+1,0,0} \end{bmatrix} \tag{4.5}$$

Only indices to the control variables and state variables at control boundaries, that is $x_{1,0,0}$ to $x_{K+1,0,0}$, are returned when direct collocation is used. This is the case even if `rigid` mode is used.

4.3.3 Providing the Initial Guess

The initial guess needs to be handed to the discretization methods in the form of a struct with four entries: End time guess, free parameter guess, state guess and control guess. The state and control entries should be functions that take the independent variable as input and returns the guess at the input value. Examples have been provided above, in the description of multiple shooting and direct collocation. After every call to `ocp_solve()` the solution is also returned in the form of an initial guess struct.

4.3.4 Control Continuity

When dealing with multiple phase OCPs it needs to be decided how the control signal should be treated when it passes the phase boundary. For both the direct multiple shooting and the direct collocation implementation the control signal is not continuous across the boundary as a default. That is, the first control value in the next phase is not required to equal the last control value of the previous phase. If a continuous control signal is desired the function `control_continuity()` can be used or such constraints can be added manually to the NLP.

4.4 Solving Optimal Control Problems

The NLP resulting from the discretization process is solved usign an NLP-solver. Since the toolbox is built on CasADi it supports all NLP-solvers that are supported in CasADi, however IPOPT [20] is the default solver. How to set options using CasADi syntax and solve problem (3.3) (the example from the direct multiple shooting and direct collocation description) is presented in an example below.

```
% NLP options
nlp_opts = struct;
nlp_opts.ipopt.max_iter = 1000;
nlp_opts.ipopt.acceptable_iter = 500;
nlp_opts.ipopt.acceptable_tol = 1e-6;
nlp_opts.ipopt.tol = 1e-7;

% Solve ocp with user defined NLP options
ocp_opts = {'nlp_opts', nlp_opts};
ocp_sol = ocp_solve(ocp, v0, 'shooting', K, opts);
```

4.4.1 Solving the reference problems

This section contains the solution to the reference problems using the toolbox.

Bryson-Denham problem

The analytical solution to the Bryson-Denham problem (3.1) is given in equation (3.2). The analytical solution is plotted together with the numerical solution, found using the toolbox, in figure 4.1.

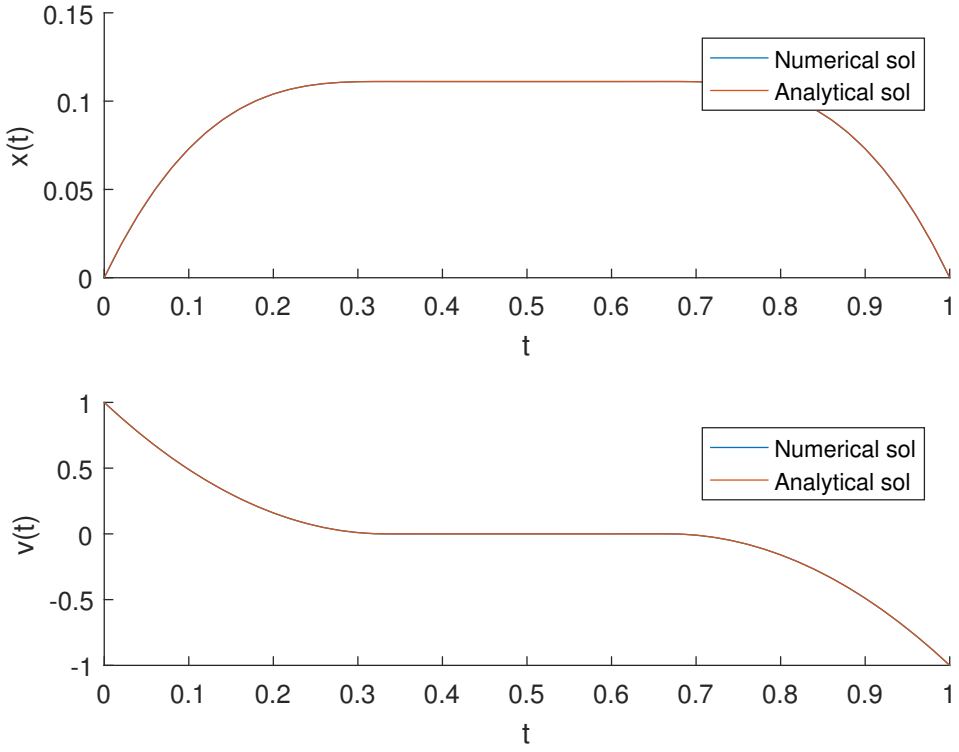

Figure 4.1: *Comparison of the state trajectories obtained when solving the Bryson-Denham problem* (3.1) *analytically and numerically. The numerical solution was obtained using the direct collocation with 50 control intervals.*

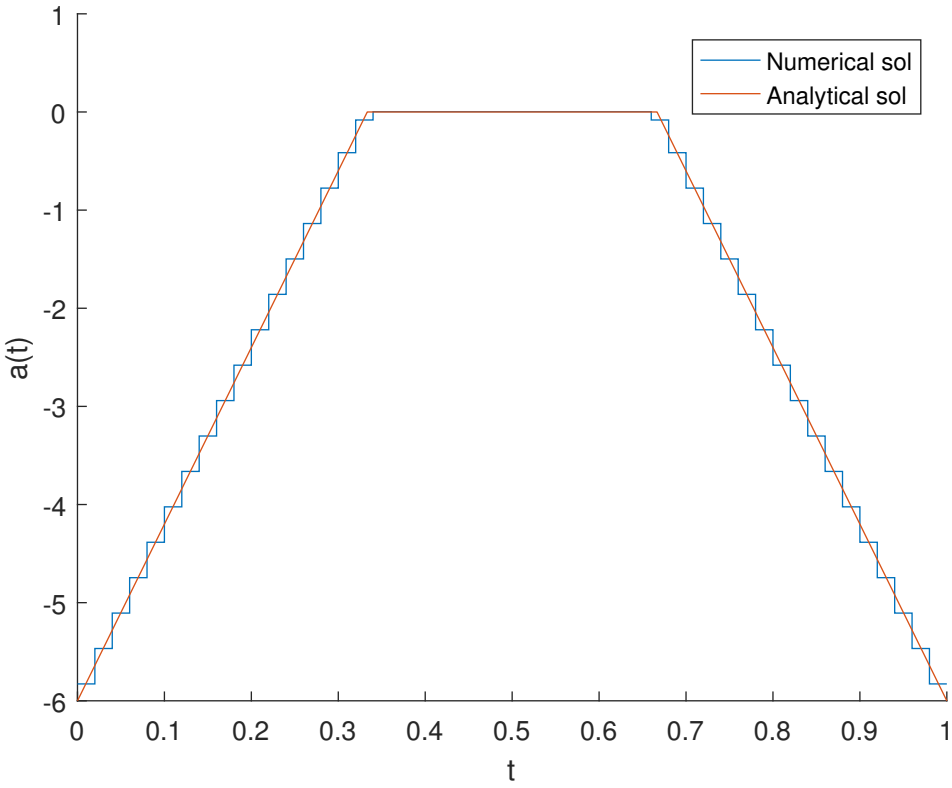

Figure 4.2: *Comparison of the control signals obtained when solving the Bryson-Denham problem* (3.1) *analytically and numerically using the toolbox. The numerical solution was obtained using the direct collocation with 50 control intervals.*

An Optimal Control Benchmark: Transient Optimization of A Diesel-Electric Powertrain

For the optimal control benchmark problem, a comparison of the minimum fuel solution is plotted in figure 4.3 and 4.4

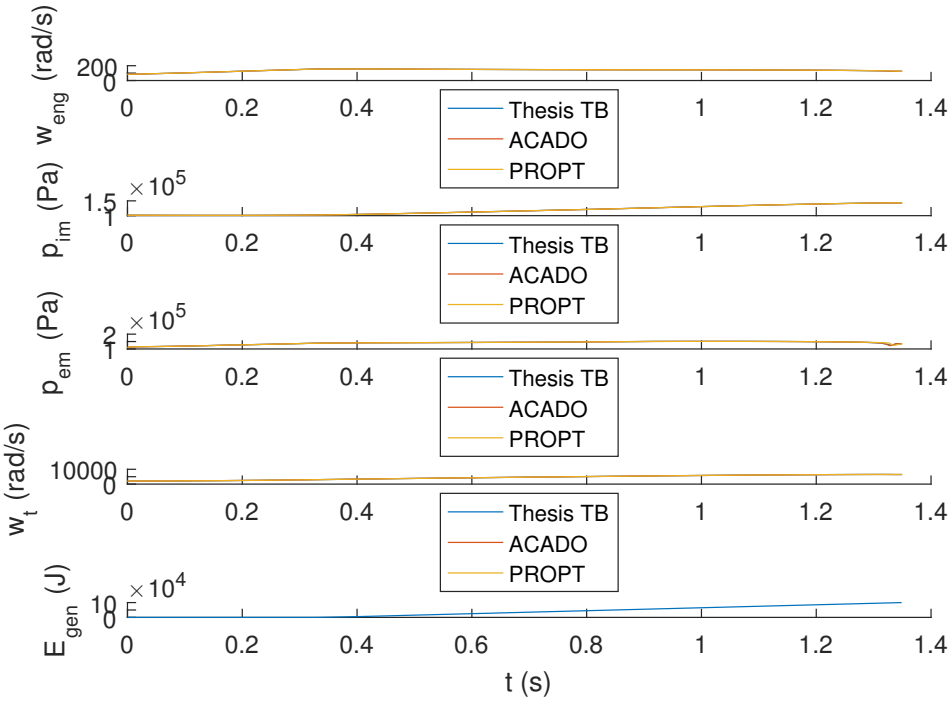

Figure 4.3: *Comparison of the state trajectories obtained when solving the optimal control benchmark problem* (3.9) *for minimum fuel. The numerical solution using this toolbox was obtained using the direct collocation with 100 control intervals.*

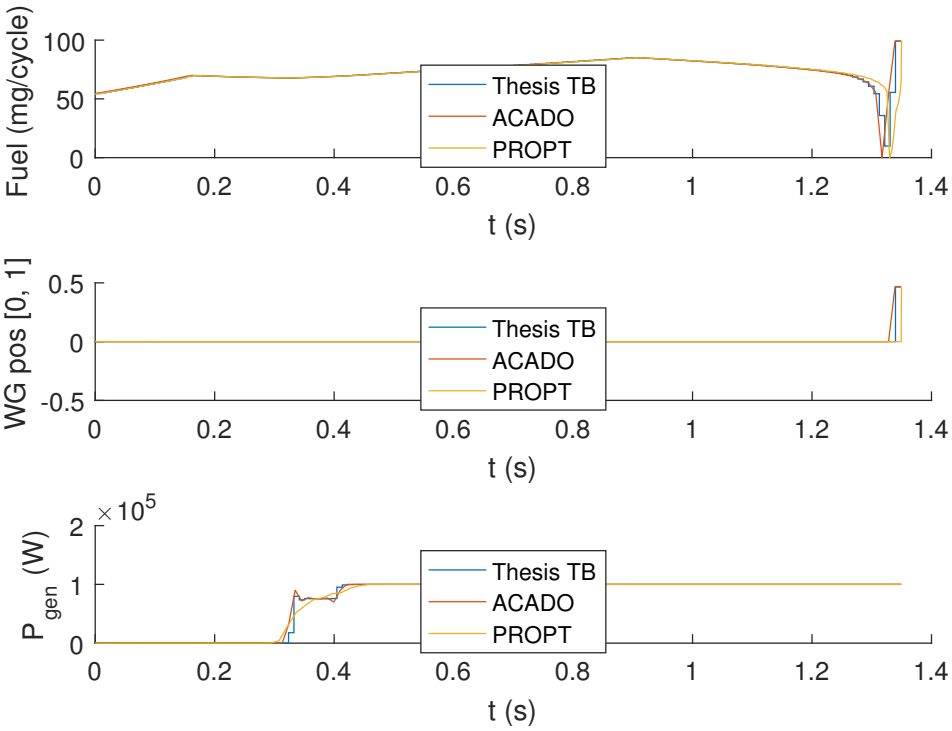

Figure 4.4: *Comparison of the control signals obtained when solving the optimal control benchmark problem* (3.9) *for minimum fuel. The numerical solution using this toolbox was obtained using the direct collocation with 100 control intervals.*

Optimal Control of a Diesel-Electric Powertrain During an Up-Shift

For the up-shift problem, a comparison of the minimum jerk solution is plotted in figure 4.5 and 4.6. Notice that the solutions are not the same. This is because the objective function is not formulated in such a way that the solution is unique. From the transmission speed and down to the wheels the solutions are the same, which shows that the optimization worked properly. That they differ in controls and some states is due to the problem formulation. An objective function comprised of both jerk and fuel consumption would have solved the problem.

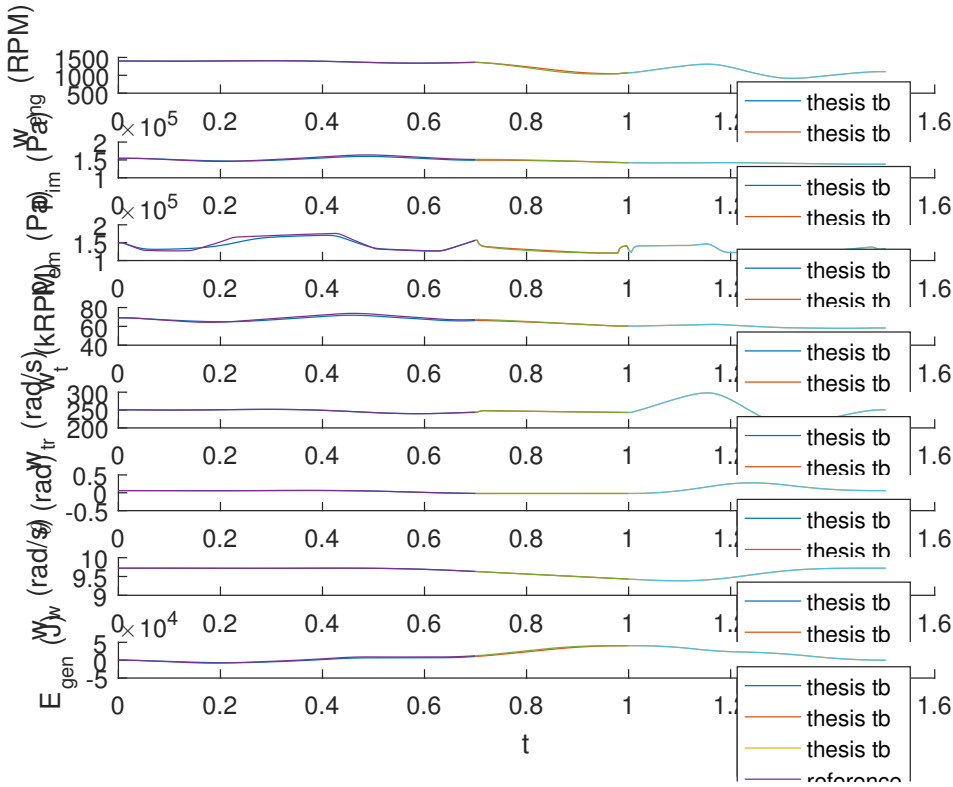

Figure 4.5: *Comparison of the state trajectories obtained when solving the up-shift problem* (3.15) *for minimum jerk. The numerical solution using this toolbox was obtained using the direct collocation with 100 control intervals in each phase. The blue, red and yellow colors belong to the toolbox solution, and purple, green and turquoise to the reference solution. Blue and purple represents the torque phase, green and red the synchronization phase, and turquoise and yellow the inertia phase.*

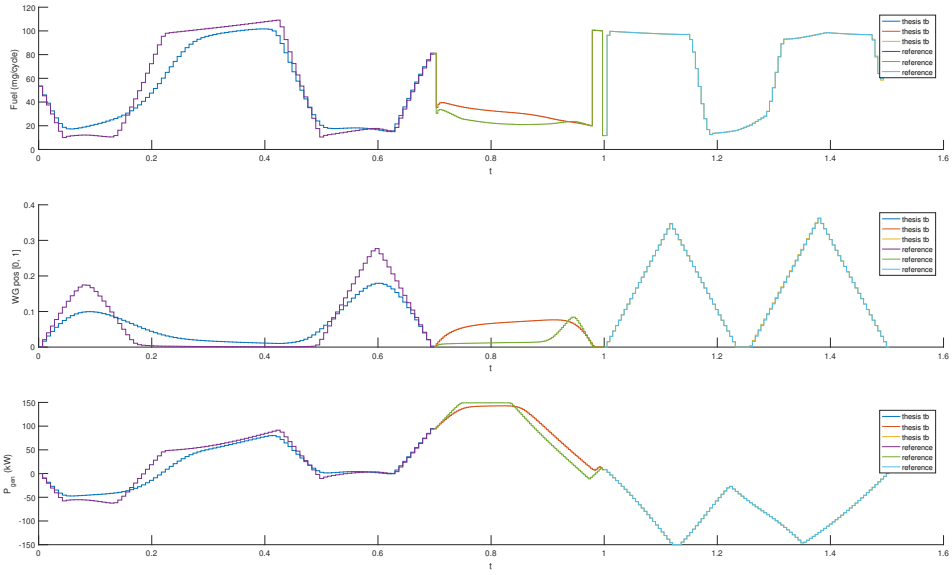

Figure 4.6: *Comparison of the control signals obtained when solving the upshift problem* (3.15) *for minimum jerk. The numerical solution using this toolbox was obtained using the direct collocation with 100 control intervals in each phase. The blue, red and yellow colors belong to the toolbox solution, and purple, green and turquoise to the reference solution. Blue and purple represents the torque phase, green and red the synchronization phase, and turquoise and yellow the inertia phase.*

4.4.2 Complete Code Example

Here follows the entire solution to problem (3.3).

```
% OCP properties
nx = 2;
nu = 1;
np = 0;
n_phase = 1;

% Symbolic variables
[t, x, u, p] = ocp_var('t', nx, nu, np);

% Phase description
xi = [0; 1];        % Initial value
xdot = [x(2); u];   % Dynamics. x(2) = a
xf = [0; -1];       % Terminal value

phase_desc =
    {'L', u^2, ...          % Lagrange term
     'T_lb', 0.1, ...       % End time lower bound
     'T_ub', 5, ...         % End time upper bound
     'xi_lb', xi, ...       % Initial state lower bound
     'xi_ub', xi, ...       % Initial state upper bound
     'x_lb', [0, -1], ...   % State lower bound
     'x_ub', [1/9, 1], ... % State upper bound
     'xf_lb', xf, ...       % Terminal state lower bound
     'xf_ub', xf, ...       % Terminal state upper bound
     'u_lb', -inf, ...      % Control lower bound
     'u_ub', inf, ...       % Control upper bound
     'f', xdot};            % Dynamics

% Create phase
p1 = ocp_phase(phase_desc);

% OCP formulation
ocp_desc={'t', t, ... % Symbolic variables
          'x', x, ...
          'u', u, ...
          'p', p, ...
          'nx', nx, ... % Problem size
          'nu', nu, ...
          'np', np, ...
          'n_phase', n_phase};

% Create OCP
ocp = ocp_ocp(ocp_desc, p1);
```

```
% Initial guess: zeros
v0 = struct;
v0.T = 2;
v0.p = 0;
v0.x = @(t) interp1([0, v0.T], [0 0; 0 0], t, 'linear', 0);
v0.u = @(t) interp1([0, v0.T], [0, 0], t, 'linear', 0);

% Discretize and solve NLP
K = 100; % Control intervals in each phase

ocp_sol = ocp_solve(ocp, v0, 'collocation', K);

% Plot results
xlabels = {'x_1', 'x_2'};
ulabels = {'u'};
tlabel = 't';
plot_results(ocp_sol, tlabel, xlabels, ulabels);
```

5

Design Choices

The leading design choice was to provide two discretization methods. This facilitates performing a sensitivity analysis of the initial guess and discretization method, which should always be conducted in order to ensure that the solution is a good candidate for optimum. When performing a sensitivity analysis, changing discretization method seamlessly, without changing the problem implementation in any other way, is of great importance. Therefore, it was crucial to design this into the toolbox. The other core decision was to give access to the internal variables in the toolbox. This means giving access to the NLP variables and indexing them so that they can be manipulated in a meaningful way.

The key to seamlessly switch discretization method lies in separating problem formulation from discretization, but also in making the output from the discretization methods behave identically to functions that take the output from the discretization as input. The latter means that no other function need to require the information whether the problem was discretized using direct multiple shooting or direct collocation than the one that decides what discretization method to choose. By providing only one data type for representing OCPs, `ocp_ocp`, the discretization methods have the same information about the OCP. This separates the OCP from the discretization method. By making the return value structures from the discretization methods have the same entries, that are constructed in the same way, it is not possible to distinguish between the methods based on output. Thereby there is no need for any other function to have information about the discretization method. This together with separating OCP from discretization enables seamless switching of the discretization method.

5.1 Entering Optimal Control Problems

The phase definition (4.1) includes two boundary equality constraints. It would, mathematically, have been sufficient to provide two inequality constraints. Numerically that proved to be less efficient. Using the NLP-solver IPOPT it proved not as easily for the solver to identify the constraint as an equality unless explicitly formulated so. There are other constraints that could have been added. For instance, the dynamic optimization software *Optimica* [21] uses equality constraints subjected to the entire time horizon, and point constraints at arbitrary time points. This was not included since it was not necessary for solving the reference problems. If point constraints at arbitrary time points are desired the problem can be implemented as a multi-phase problem in the toolbox, or if user testing proves it necessary the toolbox can be extended to include it.

The symbolic variables are necessary to store in `ocp_ocp`. That is because they are needed during the discretization, when the symbolic functions are defined. In that process the constructor is given two arguments: the set of input variables and the output expressions. The connection between input and output is created through having the output expressions made up of the exact same variables as the input variables, and that is why the symbolic variables need to be stored.

5.2 Discretization Methods

For multi-phase problems it can be desirable to have an equidistant discretization, even if the phases are of different length. That is why it is possible to choose individual discretization setting for each of the phases.

The indices to the variable in the NLP vectors (4.3) and (4.5) are stored for two reasons. The first is that it makes it simpler to add new constraints using the variables. The second is that it makes it simpler to retrieve the OCP solution from the NLP solution, it is just a matter of looking up the indices in the NLP solution. It is not strictly necessary to store the indices to be able to retrieve the solution or manipulate the variables. The NLP variable vector is always constructed in the same way and it is therefore possible to calculate the indices, however it is not as convenient or as robust as storing the indices, for instance would a change in the structure of the NLP vector force a change in the calculations.

The reason for not imposing the inequality constraints at the first time instant, by default, is simple, it is impossible to affect the state at that time instant. Looking at only the first control interval $t = [t_0, t_1]$ the control signal $u(t) = u_0$ is constant. $u(t)$ drives the state $x(t)$ from $x(t_0)$ to $x(t_1)$. Clearly, $u(t)$ affects the end state ,$x(t_1)$, but not the initial value ,$x(t_0)$. It is simply impossible to apply a control signal u_0 that affects $x(t_0)$, and that is why the inequality constraints $h(t_0, x(t_0), u(t_0), P) \leq 0$ are not imposed by default.

Because of the possibility to enforce the inequality constraints on the intermediary points it was made possible using the `rigid` setting in direct collocation. It was not made the default behaviour because the number of inequality constraints

increases considerably. This means longer execution time, which can pose a problem for large OCPs, therefore it is reasonable to have the faster mode, `normal` as the default one.

The fourth order Runge-Kutta method implemented in the multiple shooting function was chosen because it is well recognized [2]. There are fixed-step explicit Runge-Kutta schemes of higher order, but they require more evaluations of the dynamics and therefore takes longer time to execute. It is therefore a compromise between computational effort and order, in this case the choice was made to go for the fourth order method. However, since the toolbox is open-source a user could easily change the method, to a method of his or her liking.

5.3 Providing an Initial Guess

The format for providing the initial guess was chosen for simplicity. Using a struct reduces the number of arguments that needs to be handed to the discretization methods. Noticable is that the control and state initial guess is provided as a function. The explanation is that it is the easiest format, that I found, to provide initial values at time points that are chosen by the discretization method.

5.4 Control Continuity

Control continuity across phase boundaries is not implemented as a default. For some problems having control continuity is the best solution, at least for some control signals, for other problems it might not. An alternative to continuity across the boundary is to limit the change in the control signal, although that is problem specific. There is no obvious solution how to handle this, the best solution is therefore to let the user decide based on the problem. Control discontinuity across the phase boundary was therefore the default choice.

5.5 Solving Optimal Control Problems

Since CasADi provides an excellent interface to many NLP-solvers it has not been necessary to make any design choices other than making IPOPT the default solver. This is mainly motivated by that it is distributed freely with CasADi. It is also motivated by that is an interior-point method and therefore handles very sparse problems well [6].

6 Discussion

6.1 Results

The result of the thesis is a toolbox for MATLAB, based on CasADi, for solving optimal control problems numerically. The toolbox have been proven to correctly solve OCPs on at least three different types of problems. The toolbox is open-source and will be distributed freely in order for others to harness the power of CasADi for optimal control purposes.

6.2 Method

The method for designing the toolbox was to read theory, solve a number of selected problems and then create the design. The method was found from studying the background of other toolbox creators such as those of PROPT and ACADO. They are all people, or teams of people, with much experience within the field of dynamic optimization [14, 19].

A drawback of this method is that it is hard to repeat the design of the toolbox. However, this form of repeatability is not of great importance, since it is repeatability of the optimal solution that is important. An optimal solution found using this toolbox must be repeatable using another optimal control software, under the same conditions. That is why the OCPs that was studied during the design work have known solutions.

During the design work a new version of the toolbox was made for every reference problem solved. The toolbox changed dramatically for every new problem and the final version is very good. Judging from this, it is impossible to design an optimal control toolbox without practical experience of actually solving problems. Although, the method should be critiqued from two different aspects: How the theory was deemed correct and how the reference problems were chosen.

The theory was deemed correct because it is described in multiple sources, for instance [5, 6, 8, 1]. From several, mostly independent, sources describing the same theory, in the same way, the conclusion was made that the theory is correct. To further strengthen this conclusion, a problem with an analytical solution was studied. The problem, the Bryson-Denham problem (3.1), have an analytical solution that was obtained using PMP. Since these results were repeated numerically using this implementation it strengths the conclusion.

To the second point of critique. The reference problems was chosen as a compromise between different aspects. The most important aspect was that the solutions had to be known and published. It is a good way to verify that the implementation works correctly. Another important aspect was the variety of the problems. By variety, problems with fixed/free end time, with/without inequality constraints, Lagrange/Mayer formulation, multiple phases, parameter estimation components, explicate dependence on the independent variable, etc, are meant. Ideally all types of problems should be analysed, however there was only time to study a few. There is also the aspect that at least one of the problems should reflect a real use case, a really hard problems of real-world importance. The three problems chosen are motivated below.

The Bryson-Denham problem (3.1) was chosen because it is a well known OCP and it has an analytical solution. It is also possible to formulate it in more than one way which makes it versatile in the design work. It is also a good problem to start with for a beginner.

The optimal control benchmark problem (3.9) was chosen because it is a fairly large OCP and the solution is readily available. The problem includes time varying inequality constraints, free end time and a derivative constraint which makes it interesting to study. The problem was designed as a reference problem for developers of optimal control software which meant that it also provides solutions to simplified versions of the problem, which was very handy. It was also accompanied with the model code which made the implementation less time consuming.

The up-shift problem (3.15) is a complex optimal control problem that qualifies as being of real-world importance and was constructed in a cooperation between Linköping University and Scania. It includes among other, complex constraints, multiple phases and free end time.

Two important classes of problems have not been studied in the design work. A problem were the dynamics depends explicitly on the independent variable and a parameter optimization problem. An example of the first type of problem is a down-shift of a heavy-duty truck in an uphill. The latter type could be some kind of component dimensioning or the optimization of regulator parameters for some application.

7

Conclusions

The following questions were identified in the beginning of the work:

1. What methods are available for solving optimal control problems numerically?
   (a) What are their different properties?
2. How should the optimal control toolbox be designed?
   (a) How should the optimization methods be implemented in the toolbox?
   (b) How should the model be represented in the toolbox?
   (c) How should the objective function be provided by the user?
   (d) How should the constraints be specified?
   (e) What should be the default behaviour of the toolbox?
   (f) What options should be supported?

The available methods for solving optimal control problems numerically was continuous time dynamic programming, Pontryagin's maximum principle and direct transcription. They were covered in the theory section along with their advantages and drawbacks.

How the toolbox design should be made was answered in the design choice chapter (chapter 5). The most important question was how to implement the direct transcription methods since it determines whether the entire toolbox is working or not.

7.1 Future Work

There are several interesting topics that could be investigated for future work. A natural extension would be to test the toolbox on new categories of problems, specifically those mentioned in the previous chapter, when the dynamics explicitly depends on the independent variable, and a parameter estimation problem. A comparative study of different optimal control software would also be of interest.

A slightly different track is to make a more in depth analysis of dynamic optimization, and the underlying numerical methods. Maybe even to go as far as analysing nonlinear optimization and convex optimization. This could for instance be used for designing real-time nonlinear model predictive control (MPC) software tailored to a specific application.

A third track is to dig deeper into how to formulate OCPs and how to utilize the results. Should the results be used in an MPC implementation or should some other control structure be used. If MPC, how to formulate the objective function, constraints and model?

Bibliography

[1] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013. Cited on pages 2, 11, 17, and 52.

[2] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998. Cited on pages 7, 11, 16, and 49.

[3] Jonas Asprion. *Optimal Control of Diesel Engines: Modeling, Numerical Methods and Applications*. PhD thesis, ETH Zürich, Sonneggstrasse 3, CH-8092 Zurich, Switzerland, 2013. Cited on pages 3 and 16.

[4] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM, first edition, 2001. Cited on page 17.

[5] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, second edition, 2010. Cited on pages 7, 16, and 52.

[6] Lorenz T. Biegler. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization., 2010. Cited on pages 11, 12, 16, 49, and 52.

[7] Arthur E. Bryson and Yu-Chi Ho. *Applied Optimal Control*. Taylor Francis Group, 1975. Cited on page 19.

[8] Moritz Diehl. Numerical optimal control, 2011. Cited on pages 3, 6, 7, 10, and 52.

[9] Moritz Diehl, Hans Georg Bock, Holger Diedam, and Pierre-Brice Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast Motions in Biomechanics and Robotics*, Heidelberg, Germany, 2005. <inria-00390435>. Cited on pages 7, 10, and 11.

[10] Lars Eriksson and Lars Nielsel. *Modeling and Control of Engines and Drivelines*. Wiley, 2014. Cited on page 23.

[11] Ulf Jönsson, Claes Trygger, and Petter Ögren. Optimal control: Lecture notes by ulf jönsson. Optimization and Systems Theory, Royal Institute of Technology, SE-100 44, Stockholm, Sweden, 2010. Cited on pages 5 and 6.

[12] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications., 2004. Cited on page 5.

[13] Vaheed Nezhadali and Lars Eriksson. Optimal control of a diesel-electric powertrain during an up-shift. In *SAE 2016 World Congress, paper number 2016-01-1237*, 2016. Cited on pages 1, 25, 26, and 27.

[14] Optimization in Engineering Center (OPTEC). ACADO, March 2016. http://acado.github.io. Cited on pages 25 and 51.

[15] Martin Sivertsson. *Optimal Control of Electrified Powertrains*. PhD thesis, Linköping University, SE–581 83, Linköping, Sweden, 2015. Cited on page 3.

[16] Martin Sivertsson and Lars Eriksson. An optimal control benchmark: Transient optimization of a diesel-electric powertrain. In *SIMS 2014 - 55th International Conference on Simulation and Modelling*, Aalborg, Denmark, 2014. Cited on page 22.

[17] Martin Sivertsson and Lars Eriksson. Modeling for optimal control: A validated diesel-electric powertrain model. In *SIMS 2014 - 55th International Conference on Simulation and Modelling*, Aalborg, Denmark, 2014. Cited on pages 1 and 26.

[18] Tomlab Optimization Inc. PROPT, March 2016. http://tomdyn.com. Cited on page 25.

[19] Tomlab Optimization Inc. TOMLAB, August 2016. http://tomopt.com/tomlab/company/creator.php. Cited on page 51.

[20] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):22–57, 2006. Cited on page 38.

[21] Johan Åkesson. *Languages and Tools for Optimization of LargeScale Systems*. PhD thesis, Lund University - Department of Automatic Control, Dep. of Automatic Control, Lund University, Box 118, SE-221 00 Lund, Sweden, 2007. Cited on page 48.