Diploma Thesis

On the implementation of a direct numerical method for optimal control of ordinary differential equations

submitted by

Markus Esenwein

April 9, 2011

Advisor:

PD Dr. Dirk Lebiedz

Albert-Ludwigs-Universität Freiburg
Fakultät für Mathematik und Physik

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Freiburg, März 2011

(Markus Esenwein)

Preface

First of all I want to thank

- Dirk Lebiedz for supervising my diploma thesis. I don't think it is usual to have such a great atmosphere and support like I had in his working group. I am very glad to be part of it.
- Dominik Skanda for always having answers to my questions, no matter how stupid they are. He is simply great... and implemented an entire integrator in a couple of weeks, unbelievable.
- Marc Fein for sharing the desk with me. He always knows something, not only about mathematics.
- Marcel Rehberg for MATLAB® instructions and all the other ways he helped me.
- Jochen Siehr for the help with LaTeX and all the valuable comments on formalities.
- Jonas Unger for being kind of pioneer concerning diploma theses in this working group. I have benefited greatly from his experiences.
- everyone who have proofread this thesis.
- my family for making my studies possible and supporting me in every thinkable way.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The act of determining control functions and state trajectories for a dynamical system to optimize an objective functional is called *optimal control.* Consider for example a system which describes a certain process and can be influenced by at least one control function, e.g. the movement of car where the speed can be regulated by pressing the accelerator pedal. Here, it could be desired to minimize the traveling time or the fuel consumption between two points in space.
The theory of optimal control is closely related to the theory of *calculus of variations.* Hence, mathematicians like Euler, Lagrange or Hamilton were occupied with this issue and still serve as eponyms for important functions, variables, etc. Especially with the works of Bellman and Pontryagin in the 20th century optimal control theory reached a new level. Their developments in combination with the advances of technology opened the door for a computer based handling and thus many approaches to tackle these problems with the help of numerical mathematics accrued. These numerical methods are indeed necessary, since optimization problems and especially problems of optimal control arise in many application fields where an analytical solution can hardly be obtained (e.g. because of occurring nonlinearities). Optimal control problems based on dynamical processes can be found for example in the context of physical, (bio)chemical, medical, biological or micro electronic tasks [13, 19, 27, 29]. Due to the rising complexity of such models there is a

demand for efficient methods to solve them, but these methods have to meet several requirements.
First of all it has to be taken into account that the mathematical models are often nonlinear, thus when focusing on those problems an optimization algorithm which can handle nonlinear problems is an essential part. Here, it is necessary to decide which kind of optimization method shall be used. The two traditional optimization search methods are the *direct-search* and the *gradient-based* method. Direct-search methods only use values of the objective function and the constraints to determine search directions whereas gradient-based methods additionally make use of the informations obtained from first or second-order derivatives of both objective and constraint functions. Thus, one advantage of a direct-search methods is that they are easy to apply on a variety of problems, since there is no need of computing derivatives. However, the great benefit of gradient-based methods is that they need less function evaluations and thus converge faster to an optimal solution. To reduce the effort of applying a gradient-based method on different problems, there are procedures which exploit derivation rules and automatically compute derivative values. Thus, it is not necessary to calculate them analytically.
A second question which can occur is whether to use an indirect or a direct approach for the optimization. The indirect approach first determines necessary optimality conditions analytically and discretizes the resulting system of equations whereas the direct approach starts instantly with a discretization of the whole system. So, simply stated, this is the question on which point the problem should be discretized.
Another challenge comes with the dynamics of a process. They are usually described by a system of differential equations, therefore a numerical integration method is indispensable. There is a wide range of algorithms for example Adams-Bashforth (explicit), Adams-Moulton (implicit) or Runge-Kutta (both implicit and explicit) methods. It is difficult to decide which is the best method since this depends closely on the problem itself.

1.2 Introducing DOT

DOT (**D**irect **O**ptimization **T**ool) is the implementation of an algorithm which rises to the challenges mentioned above and has been developed in this work. This code can be used to solve nonlinear problems of optimal control with respect to ordinary differential equations. As the name suggests, DOT uses a direct approach. That means that the problem is discretized on a time grid by approximating the state values and the values of the control function (e.g. piecewise constant parametrization). Thus a finite dimensional problem formulation results which can be solved numerically. The user decides how fine this discretization should be. The underlying optimization algorithm used in this work is an interior point method which is a gradient-based optimization method and assures, under certain conditions, global convergence. The needed derivatives are computed via automatic differentiation which guarantees highly accurate values. To handle systems of ordinary differential equations (ODE) we use a multiple shooting approach combined with a linear multi-step method known as Backward Differentiation Formula (BDF). Hence, the ODE system is solved separately on subintervals and an absolutely continuous solution is obtained by introducing additional equality constraints into the optimization problem.

1.3 Outline

Chapter 2

This thesis starts with a chapter about *automatic differentiation*, a technique to compute derivatives numerically. We consider functions which can be decomposed in *elemental functions* and we show how this can be used to obtain derivative values. In this context Taylor series get important, hence we give an overview of computing its coefficients and its propagation through a composition of functions. Based on the chain rule and with the help of Taylor series we present two modes of automatic differentiation, i.e. the *forward* and the *backward mode*, and give example calculations to make them more accessible.

Chapter 3

In this chapter we discuss how to obtain a numerical solution of ordinary differential equations. We start with a short introduction of *Runge-Kutta methods*, which are part of the so-called *one-step methods*. Afterwards the focus is on a different kind of numerical methods, *linear multi-step methods*, and theoretical foundations for both fixed and variable step sizes are presented. Statements about consistency, stability and convergence are given. In addition, we present in section 3.3 a *Backward Differentiation Formula*. We show the fundamentals of this method and which additional settings, e.g. concerning step sizes, characterize the integrator which is used in DOT.

Chapter 4

This chapter focuses on *optimization*. Gradient-based methods to solve optimization problems under equality and inequality constraints try to find solutions which satisfy the *Karush-Kuhn-Tucker conditions*, hence we present the required theory for those first order necessary conditions. We continue with a detailed explanation of an *interior point method* which is part of the optimization software package IPOPT. We point out in particular the related *barrier problem*, the *filter line search method* and its *convergence properties*.

Chapter 5

Theoretical and practical explanations of DOT are topics of chapter 5. We formulate *Pontryagin's minimum principle* for optimal control problems and mention the difference between *indirect* and *direct* methods. Afterwards we present in the main part of this chapter the discretization and parametrization techniques which are used in DOT. The chapter finishes with a detailed problem formulation.

Chapter 6

This chapter presents *application examples* solved with DOT. Starting with two simple examples concerning the time optimal and energy optimal movement of a car we go on with two more complex problems concerning biochemical processes. Here, we can see that DOT performs well even for higher dimensional and/or highly nonlinear problems. The examples in this chapter are part of a special problem class for which we present assumptions and *sufficient conditions* to ensure optimality.

Chapter 7

The last chapter summarizes this work and gives an outlook.

Appendix

A short user guide for DOT.

CHAPTER 2

Automatic differentiation

We deal with problems of the type

$$\min_{x \in \mathbb{R}^n} f(x) \tag{2.1}$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is continuously differentiable. It is reasonable to use a gradient-based optimization method since this suggests fast convergence properties. Assuming unconstrained problems (2.1) their optima can be found at points $x^* \in \mathbb{R}^n$ with

$$\frac{df(x^*)}{dx} = 0 \text{ and } \frac{d^2 f(x^*)}{dx^2} \neq 0.$$

The necessary conditions for constrained problems will be elaborated explicitly in a later chapter (KKT conditions in chapter 4). It is desirable to have numerical methods for computing derivative values, since otherwise we would have to differentiate every function analytically (i.e. by hand). The most common method to calculate derivatives numerically is known as the method of finite differences. Function values are used to estimate derivative values, e.g. by computing a differential quotient

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

with a small step $h > 0$.
We will present a version of a different and more accurate technique known as *Automatic Differentiation (AD)*. Here, the computational representation of a

function is used to obtain analytic values for the derivatives. A function $y = f(x)$ is decomposed in a sequence of elemental functions φ which derivatives can easily be computed. In accordance with the chain rule, this gives a quite simple way to get exact (at least to numerical accuracy) derivatives. There are two basic modes in which we will use AD, a *forward mode* and a *backward mode*. They mainly differ in the sequence of evaluating the terms given in the chain rule. Assuming

$$f(x) = g(h(x))$$

we have

$$\frac{df}{dx} = \frac{dg}{dh}\frac{dh}{dx}$$

by following the chain rule. While in the forward mode $\frac{dh}{dx}$ is evaluated first and $\frac{dg}{dh}$ afterwards, in the backward mode it is the other way around. We can simultaneously calculate derivatives of different order in any desired direction. This chapter starts with a few theoretical aspects about Taylor series in section 2.1 before details about forward and backward mode follow in sections 2.2 and 2.3. In this chapter we refer to the explanations which are given in [7] and [12], the tables illustrated here are a slight modification of those which can be found in [12].

Notation

The notation will be as follows,

x_i denotes a Taylor series in t,

$x_i^{(k)}$ the k-th coefficient of x_i and with this

x_i is also written as $x_i = \left[x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, \dots\right]$.

2.1 Taylor series propagation

We start with

$$y = f(x)$$

where we assume $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ to be a function which can be decomposed in a sequence of elemental functions φ_i (e.g. trigonometric functions, multiplication, . . .) whereby we get a computational graph, i.e.

$$y = f(x) = \varphi_1 \circ \varphi_2 \circ \dots \circ \varphi_p(x) \tag{2.2}$$

with $p \in \mathbb{N}$.

Assumption 2.1 (Elemental Differentiability)
All elemental functions φ_i are d times continuously differentiable on their open domains $\mathcal{D}_i$, that means $\varphi_i \in \mathcal{C}^d(\mathcal{D}_i)$ with $0 \leq d \leq \infty$.

Starting with the independent variable x and by successively passing the computational graph (2.2) we obtain intermediate variables depending on the arguments of φ_i on every step of the graph and finally the value of the final variable $y = f(x)$ we are interested in. We will handle all involved variables (and functions) as Taylor series (or its Taylor approximations).
For a given polynomial

$$x = x(t) = x^{(0)} + x^{(1)}t + x^{(2)}t^2 + \cdots + x^{(d)}t^d \in \mathbb{R}^n \tag{2.3}$$

and a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we consider $y = f(x)$ as a Taylor series and get

$$y = y(t) \equiv y^{(0)} + y^{(1)}t + y^{(2)}t^2 + \cdots + y^{(d)}t^d = f(x(t)) + \mathcal{O}(t^{d+1}) \in \mathbb{R}^m. \tag{2.4}$$

Definition 2.2 (Taylor coefficient functions)
Under Assumption 2.1 let for $k \leq d$

$$f_k(x^{(0)}, x^{(1)}, \ldots, x^{(k)}) = y^{(k)} \quad \textit{with} \quad f_k : \mathbb{R}^{n\times(k+1)} \to \mathbb{R}^m$$

denote the ***Taylor coefficient function*** *which is defined by the relations (2.3) and (2.4).*

Since every f_k is a composition of elemental functions φ_i it is necessary to propagate corresponding Taylor coefficients through all intermediate variables of this composition. We will denote intermediates with u, v or w. The propagation is done by following basic rules. For arithmetic operations these rules are an application of the polynomial arithmetic. Table 2.1 gives an overview.
For univariate operations it is more difficult to obtain the propagation rules. All elemental functions $\varphi(u)$ are solutions of linear ODEs, i.e. all $\varphi(u)$ satisfy an identity of the form

$$b(u)\varphi'(u) - a(u)\varphi(u) = c(u). \tag{2.5}$$

The coefficient functions $a(u), b(u)$ and $c(u)$ of course differ for different $\varphi(u)$, but they are always given in terms of arithmetic operations and univariate functions, which already can be differentiated. In other words, it is supposed that the Taylor coefficients $a^{(k)}, b^{(k)}$ and $c^{(k)}$ are determined by the coefficients $u^{(k)}$ of u.

$v =$	Recurrence for $k = 1 \dots d$
$u + cw$	$v^{(k)} = u^{(k)} + cw^{(k)}$
$u \cdot w$	$v^{(k)} = \sum_{j=0}^{k} u^{(j)} w^{(k-j)}$
$\frac{u}{w}$	$v_k = \frac{1}{w^{(0)}} \left[u^{(k)} - \sum_{j=0}^{k-1} v^{(j)} w^{(k-j)} \right]$
u^2	$v^{(k)} = \sum_{j=0}^{k} u^{(j)} u^{(k-j)}$
$\sqrt{u}$	$v_k = \frac{1}{2} v^{(0)} \left[u^{(k)} - \sum_{j=1}^{k-1} v^{(j)} v^{(k-j)} \right]$

Table 2.1: Taylor coefficient propagation through arithmetic operations

Start with the identity $v(t) = \varphi(u(t))$. Differentiating leads to

$$\frac{dv(t)}{dt} = \varphi'(u) \frac{du(t)}{dt}$$

and multiplying with $b(u(t))$ and using (2.5) gives

$$b(u(t)) \frac{dv(t)}{dt} = \left[c(u(t)) + a(u(t)) \cdot \varphi(u(t)) \right] \frac{du(t)}{dt} \tag{2.6}$$

which is a useful result, as we will see in the following. As a consequence of Assumption 2.1 $u(t)$ and $v(t)$ are at least d-times continuously differentiable, thus their derivatives can be written as

$$\frac{du(t)}{dt} = \tilde{u}^{(1)} + \tilde{u}^{(2)} t + \dots + \tilde{u}^{(d)} t^{d-1} + \mathcal{O}(t^d) \text{ with } \tilde{u}^{(j)} = j u^{(j)} \ \forall\, j > 0$$

and similarly

$$\frac{dv(t)}{dt} = \tilde{v}^{(1)} + \tilde{v}^{(2)} t + \dots + \tilde{v}^{(d)} t^{d-1} + \mathcal{O}(t^d) \text{ with } \tilde{v}^{(j)} = j v^{(j)} \ \forall\, j > 0.$$

These considerations are sufficient to formulate the following result, which helps to propagate Taylor series coefficients for univariate elementals. A list with examples for a few of those elemental functions is given in Table 2.2.

Proposition 2.3
For $b^{(0)} = b(u^{(0)}) \neq 0$ *we have*

$$\tilde{v}^{(k)} = \frac{1}{b^{(0)}} \left[\sum_{j=1}^{k} \big(c^{(k-j)} + e^{(k-j)}\big) \tilde{u}^{(j)} - \sum_{j=1}^{k-1} b^{(k-j)} \tilde{v}^{(j)} \right] \quad \textit{for} \quad k = 1, \dots, d$$

with

$$e^{(k)} = \sum_{j=0}^{k} a^{(j)} v^{(k-j)} \quad \textit{for} \quad k = 0, \dots, d-1.$$

Proof: To get this term, substitute the derivatives in (2.6) by the expansions given above and use again the identity $v(t) = \varphi(u(t))$. Now identifying coefficients and rearranging the term leads to the desired expression.

□

v	a	b	c	Recurrence for $k = 1 \dots d$
$\ln(u)$	0	u	1	$\tilde{v}^{(k)} = \frac{1}{u^{(0)}} \left[\tilde{u}^{(k)} - \sum_{j=1}^{k-1} u^{(k-j)} \tilde{v}^{(j)} \right]$
$\exp(u)$	1	1	0	$\tilde{v}^{(k)} = \sum_{j=1}^{k-1} v^{(k-j)} \tilde{u}^{(j)}$
u^r	r	u	0	$\tilde{v}^{(k)} = \frac{1}{u^{(0)}} \left[r \sum_{j=1}^{k} v^{(k-j)} \tilde{u}^{(j)} - \sum_{j=1}^{k-1} u^{(k-j)} \tilde{v}^{(j)} \right]$
$\sin(u)$	0	1	$\cos(u)$	$\tilde{v}^{(k)} = \sum_{j=1}^{k} \tilde{u}^{(j)} c^{(k-j)}$
$\cos(u)$	0	-1	$\sin(u)$	$\tilde{v}^{(k)} = \sum_{j=1}^{k} - \tilde{u}^{(j)} c^{(k-j)}$

Table 2.2: Taylor coefficient propagation through univariate elementals

With this preliminary work we can now turn toward the two modes of automatic differentiation we want to treat within this chapter.

2.2 Forward mode

In the forward mode of AD we evaluate and carry forward a directional derivative of each intermediate variable in a given direction p. We will outline the method in a low dimensional case to keep the notation as clear as

possible. For further details see [7, 12, 23].
Considering the two-dimensional case $y = f(x_1, x_2)$ with x_1, x_2 being Taylor series in t. The Taylor approximation of this function gives

$$
\begin{aligned}
f(x_1(t), x_2(t)) = f(x_1(0), x_2(0)) + \left(f_{x_1} \frac{\partial x_1}{\partial t}\bigg|_{t=0} + f_{x_2} \frac{\partial x_2}{\partial t}\bigg|_{t=0} \right) t + \\
+ \Bigg(f_{x_1 x_1} \frac{\partial x_1}{\partial t} \frac{\partial x_1}{\partial t}\bigg|_{t=0} + f_{x_1} \frac{\partial^2 x_1}{\partial t^2}\bigg|_{t=0} + f_{x_1 x_2} \frac{\partial x_1}{\partial t} \frac{\partial x_2}{\partial t}\bigg|_{t=0} + \\
+ \quad f_{x_2 x_2} \frac{\partial x_2}{\partial t} \frac{\partial x_2}{\partial t}\bigg|_{t=0} + f_{x_2} \frac{\partial^2 x_2}{\partial t^2}\bigg|_{t=0} + f_{x_2 x_1} \frac{\partial x_2}{\partial t} \frac{\partial x_1}{\partial t}\bigg|_{t=0} \Bigg) \frac{t^2}{2} + \dots
\end{aligned}
$$

which is identically

$$
\begin{aligned}
f(x_1(t), x_2(t)) = f(x_1^{(0)}, x_2^{(0)}) + \left(f_{x_1} x_1^{(1)} + f_{x_2} x_2^{(1)} \right) t + \\
+ \Big(f_{x_1 x_1} x_1^{(1)} x_1^{(1)} + f_{x_1} x_1^{(2)} + f_{x_1 x_2} x_1^{(1)} x_2^{(1)} + \\
+ \quad f_{x_2 x_2} x_2^{(1)} x_2^{(1)} + f_{x_2} x_2^{(2)} + f_{x_2 x_1} x_2^{(1)} x_1^{(1)} \Big) \frac{t^2}{2} + \dots
\end{aligned}
$$

where f_{x_i} denotes the partial derivative $\frac{\partial f}{\partial x_i}$.
Obviously the Taylor coefficients of f essentially include informations about the derivatives with respect to the independent variables x_1, x_2. We compare the coefficients $y^{(k)}$ of y and $f^{(k)}$ of f which have to be the same. So it is clear that

$$
\begin{aligned}
y^{(0)} &= f(x_1^{(0)}, x_2^{(0)}) \\
y^{(1)} &= f_{x_1} x_1^{(1)} + f_{x_2} x_2^{(1)} \\
&\vdots
\end{aligned}
\tag{2.7}
$$

To see how the forward mode works we use the function

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

as an exemplary demonstration.
x_1, x_2 are the independent start variables and we write the decomposition of f as

$$
\begin{aligned}
u_1 &= x_1 \\
u_2 &= x_2 \\
u_3 &= u_1 \cdot u_2 \\
u_4 &= \sin(u_1) \\
u_5 &= u_3 + u_4
\end{aligned}
$$

with u_k being intermediate Taylor series and obviously $u_5 = y = f(x_1, x_2)$. By following naively the propagation rules we mentioned in the previous section these Taylor series are given by

$$\begin{aligned} u_1 &= \left[x_1^{(0)}, x_1^{(1)}, x_1^{(2)}, \dots \right] \\ u_2 &= \left[x_2^{(0)}, x_2^{(1)}, x_2^{(2)}, \dots \right] \\ u_3 &= \left[u_1^{(0)} u_2^{(0)}, u_1^{(0)} u_2^{(1)} + u_1^{(1)} u_2^{(0)}, \dots \right] \\ u_4 &= \left[\sin(u_1^{(0)}), u_1^{(1)} \cos(u_1^{(0)}), \dots \right] \\ u_5 &= \left[u_3^{(0)} + u_4^{(0)}, u_3^{(1)} + u_4^{(1)}, \dots \right] \end{aligned} \tag{2.8}$$

and by incorporating x_1 and x_2 in every step this results in

$$y = u_5 = \left[x_1^{(0)} x_2^{(0)} + \sin(x_1^{(0)}),\ x_1^{(0)} x_2^{(1)} + x_1^{(1)} x_2^{(0)} + x_1^{(1)} \cos(x_1^{(0)}),\ \dots \right]$$

for the last Taylor series y. We are free to set the Taylor series x_1 and x_2 the way we want to. These input variables are often called seeds. According to (2.7) it is clear that $x_1^{(0)}$ and $x_2^{(0)}$ are the points the function is evaluated at. Suppose we are interested in the first derivative of f in direction $\binom{1}{1}$, we see with (2.7) that we only have to set the seeds $x_1^{(1)} = 1$ and $x_2^{(1)} = 1$ and consequently we obtain

$$y^{(1)} = x_1^{(0)} + x_2^{(0)} + \cos(x_1^{(0)})$$

which is indeed the requested derivative. To get informations about derivatives of higher order or into a different direction all we have to do is setting the coefficients $x_1^{(k)}$ and $x_2^{(k)}$ accordingly.

2.3 Backward mode

As mentioned before, in the backward mode we start at the end of the decomposition of a function f and work it over from bottom to top. For reasons of simplification we introduce an additional notation for Taylor series coefficients. $[x]_k$ stands, as well as $x^{(k)}$ does, for the k-th coefficient of the Taylor series x.

Theorem 2.4
Let x be an univariate Taylor series in t, and let $y = f(x)$ for some smooth function f. Then it holds

$$\frac{\partial y^{(k+p)}}{\partial x^{(p)}} = \frac{\partial y^{(k)}}{\partial x^{(0)}} \quad \textit{and} \quad \frac{\partial y^{(k)}}{\partial x^{(k+p)}} = 0 \tag{2.9}$$

for all $k \geq 0, p > 0$.

Proof: Since f is smooth, y can be written as a Taylor series in t, i.e. $y = \sum_k y^{(k)} t^k$. Expand $f(x)$ as a power series in x, substitute $x = \sum_k x^{(k)} t^k$ and collect terms with same powers of t. Each $y^{(k)}$ is a function of $x^{(j)}$ for $j \leq k$, but $y^{(k)}$ does not itself depend on t, whereas y can be considered as a function dependent on all $x^{(k)}$ and t.
Considering the derivative of y with respect to $x^{(p)}$ the chain rule gives

$$\frac{\partial y}{\partial x^{(p)}} = f'(x) \frac{\partial x}{\partial x^{(p)}}$$

which since

$$\frac{\partial x}{\partial x^{(p)}} = t^p$$

and

$$\frac{\partial y}{\partial x^{(0)}} = f'(x) \frac{\partial x}{\partial x^{(0)}} = f'(x)$$

is

$$\frac{\partial y}{\partial x^{(p)}} = \frac{\partial y}{\partial x^{(0)}} t^p. \tag{2.10}$$

If we consider the two sides of this equation as Taylor series, corresponding coefficients have to be equal. In particular considering the coefficient of t^{k+p} that gives

$$\frac{\partial y^{(k+p)}}{\partial x^{(p)}} = \left[\frac{\partial y}{\partial x^{(p)}}\right]_{k+p} \overset{(2.10)}{=} \left[\frac{\partial y}{\partial x^{(0)}} t^p\right]_{k+p} = \left[\frac{\partial y}{\partial x^{(0)}}\right]_k = \frac{\partial y^{(k)}}{\partial x^{(0)}}$$

as asserted by the theorem. The second identity follows with a similar argument. □

This conclusion is important for the reverse accumulation with Taylor series coefficients. Let x and u be univariate Taylor series in t with $u = f(x)$ and f is a one-argument elemental function. Additionally, let y be a scalar valued variable with Taylor series coefficients $y^{(k)}$ which is dependent on u.
For the intermediate variable u we define the Taylor series $\bar{u}$ by

$$\bar{u} = \frac{\partial y}{\partial u} = \frac{\partial y}{\partial u^{(0)}}$$

and it follows

$$\bar{u}^{(k)} = \frac{\partial y^{(k)}}{\partial u^{(0)}} \overset{(2.9)}{=} \frac{\partial y^{(k+p)}}{\partial u^{(p)}} \text{ for } k, p \geq 0.$$

Thus

$$\frac{\partial y^{(p)}}{\partial x^{(0)}} = \sum_{k \leq p} \frac{\partial y^{(p)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial x^{(0)}} = \sum_{k \leq p} \frac{\partial y^{(p-k)}}{\partial u^{(0)}} \frac{\partial u^{(k)}}{\partial x^{(0)}} = \sum_{k \leq p} [\bar{u}]_{p-k} [f'(x)]_k = [\bar{u} \cdot f'(x)]_p$$

where $\bar{u} \cdot f'(x)$ denotes the convolution of $\bar{u}$ and $f'(x)$ since we considered Taylor series. According to the forward step $u = f(x)$ we associate the reverse accumulation step $\bar{x} = \bar{u} \cdot f'(x)$. A similar argument applies to the two-argument elemental functions g. With $u = g(x, y)$ we associate $\bar{x} = \bar{u} \cdot g_x(x, y)$ and $\bar{y} = \bar{u} \cdot g_y(x, y)$ where g_x and g_y denote the partial derivatives of g with respect to x and y (and again we consider g_x and g_y to be Taylor series). This holds for the elemental functions. Considering a function as a composition of elementals, we have to adapt these formulas since an independent variable x not necessarily has to be the argument of only one elemental function. Hence, we set all $\bar{x}$ initially zero and increment $\bar{x}$ by every term appearing during the computational graph, i.e. $\bar{x} = \bar{x} + \bar{u} \cdot f'(x)$ and of course in case of two argument functions $\bar{x} = \bar{x} + \bar{u} \cdot g_x(x, y)$ and $\bar{y} = \bar{y} + \bar{u} \cdot g_y(x, y)$.
Thus we can reverse accumulate the gradient ∇y as follows. Start by setting $\bar{y} = 1$ (i.e. set $\bar{y} = [1, 0, 0, \ldots]$) and go backwards through the composition. This means, we begin with the last operation, carry out the reverse accumulation steps in Taylor form and then go on in reverse order to the original sequence. Finally we gain $\bar{x} = \frac{\partial y}{\partial x}$ for each independent variable x.
Recall the example $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$ and the decomposition (2.8)

$$\begin{aligned}
u_1 &= \left[x_1^{(0)}, x_1^{(1)}, x_1^{(2)}, \ldots \right] \\
u_2 &= \left[x_2^{(0)}, x_2^{(1)}, x_2^{(2)}, \ldots \right] \\
u_3 &= \left[u_1^{(0)} u_2^{(0)}, u_1^{(0)} u_2^{(1)} + u_1^{(1)} u_2^{(0)}, \ldots \right] \\
u_4 &= \left[\sin(u_1^{(0)}), u_1^{(1)} \cos(u_1^{(0)}), \ldots \right] \\
u_5 &= \left[u_3^{(0)} + u_4^{(0)}, u_3^{(1)} + u_4^{(1)}, \ldots \right]
\end{aligned}$$

Here, again it is $y = u_5$ and thus we start with $\bar{y} = \bar{u}_5 = [1, 0, 0, \ldots]$ and for the reasons mentioned above we set the initial values of $\bar{u}_i = [0, 0, 0, \ldots]$

for $i \in \{1,2,3,4\}$. The algorithm works through the decomposition from bottom to top with

$$\begin{aligned}\bar{u}_4 = \bar{u}_5 \cdot \frac{\partial u_5}{\partial u_4} &= \Big[1,0,0,\dots\Big] \cdot \Big[1,0,0,\dots\Big] = \Big[1,0,0,\dots\Big]\\ \bar{u}_3 = \bar{u}_5 \cdot \frac{\partial u_5}{\partial u_3} &= \Big[1,0,0,\dots\Big] \cdot \Big[1,0,0,\dots\Big] = \Big[1,0,0,\dots\Big]\end{aligned}$$

followed by

$$\bar{u}_2 = \bar{u}_3 \cdot \frac{\partial u_3}{\partial u_2} = \Big[1,0,0,\dots\Big] \cdot \Big[u_1^{(0)}, u_1^{(1)}, u_1^{(2)}, \dots\Big] = \Big[u_1^{(0)}, u_1^{(1)}, u_1^{(2)}, \dots\Big]$$

and

$$\begin{aligned}\bar{u}_1 &= \bar{u}_4 \cdot \frac{\partial u_4}{\partial u_1} = \Big[\cos(u_1^{(0)}), -u_1^{(1)} \sin(u_1^{(0)}), \dots\Big]\\ \bar{u}_1 &= \bar{u}_1 + \bar{u}_3 \cdot \frac{\partial u_3}{\partial u_2} = \Big[\cos(u_1^{(0)}), -u_1^{(1)} \sin(u_1^{(0)}), \dots\Big] + \Big[u_2^{(0)}, u_2^{(1)}, u_2^{(2)}, \dots\Big] =\\ &= \Big[u_2^{(0)} + \cos(u_1^{(0)}), u_2^{(1)} - u_1^{(1)} \sin(u_1^{(0)}), \dots\Big]\end{aligned}$$

where we have to factor in that there are indeed two elemental functions in this composition where u_1 is an input argument. That is the reason why we summarize in the last line.
Rewriting this in terms of x instead of u (since $u_1 = x_1$ and $u_2 = x_2$) this results in

$$\begin{aligned}\bar{x}_1 &= \Big[x_2^{(0)} + \cos(x_1^{(0)}), x_2^{(1)} - x_1^{(1)} \sin(x_1^{(0)}), \dots\Big]\\ \bar{x}_2 &= \Big[x_1^{(0)}, x_1^{(1)}, x_1^{(2)}, \dots\Big].\end{aligned} \tag{2.11}$$

Let us assume we start with the forward mode in such a way that we obtain the first order derivative with respect to x_1, i.e. we set $x_1 = [x_1^{(0)}, 1, 0, \dots]$ and $x_2 = [x_2^{(0)}, 0, 0, \dots]$ and obtain y with $y^{(1)} = f_{x_1}$. Using these values to initiate the backward mode consequently yields derivatives of higher order, e.g.

$$\bar{x}_1^{(1)} = \frac{\partial y^{(1)}}{\partial x_1^{(0)}} = f_{x_1 x_1} \text{ and } \bar{x}_2^{(1)} = \frac{\partial y^{(1)}}{\partial x_2^{(0)}} = f_{x_1 x_2}$$

which can easily be computed for this example as $f_{x_1 x_1} = -\sin(x_1^{(0)})$ and $f_{x_1 x_2} = 1$ (evaluate (2.11) with the given values of x_1 and x_2).

CHAPTER 3

Numerical solution of ordinary differential equations

We refer to optimization problems

$$\begin{aligned} \min_{y \in \mathbb{R}^{n_y}} \quad & F(y) \\ \text{s.t.} \quad & \dot{y}(t) = f(y(t), t) \\ & y(t_0) = y_0 \end{aligned}$$

with $F : \mathbb{R}^{n_y} \longrightarrow \mathbb{R}, f : \mathbb{R}^{n_y} \times [t_0, t_f] \longrightarrow \mathbb{R}^{n_y}$ and $y_0 \in \mathbb{R}^{n_y}$.
Thus we have to solve the initial value problem (IVP)

$$\begin{aligned} \dot{y}(t) &= f(y(t), t) \\ y(t_0) &= y_0. \end{aligned} \tag{3.1}$$

There is a wide range of numerical integration methods and each of them entails certain advantages. An explicit method, for instance, requires much less effort to compute an integration step than an implicit one, but if the implicit method allows larger step sizes and assures the same accuracy this incident can be compensated. The right choice of an integration method is problem-specific. The books of Hairer et al. [14, 15] give a detailed overview of numerical methods to solve ordinary differential equations. This chapter is mainly based on those works and we outline a couple of those methods. After a short introduction of one-step methods, we focus on multi-step methods with both fixed and variable step sizes. In the latter section it follows a detailed explanation of the method which is used in DOT. We consider a finite set of time points $t_j \in [t_0, t_f]$, $j \in \mathbb{N}$ and use y_j to denote the approximated value of an exact solution $y(t_j)$.

3.1 One-step methods

Most commonly known one-step methods are the so-called Runge-Kutta methods (RKM). Data from the actual step n is used to estimate new values in step $n+1$ in the following way

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i$$

with $h := t_{n+1} - t_n$ denoting the step size and

$$k_i := f(y_n + h\sum_{j=1}^{s} a_{ij}k_j, t_n + hc_i) \qquad i = 1, \dots, s.$$

The number of stages $s \in \mathbb{N}$ and the coefficients $b_i, c_i, a_{ij} \in \mathbb{R}$ characterize a particular form of the RKM and are often listed in a so-called Butcher array as illustrated in figure 3.1.

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \dots & a_{1s} \\
c_2 & a_{21} & a_{22} & \dots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\
\hline
 & b_1 & b_2 & \dots & b_s
\end{array}$$

Figure 3.1: Butcher array for RKM

3.2 Linear multi-step methods

In this work we will concentrate on multi-step methods, i.e. not only data from the actual step but also from the steps before is used to obtain an approximation to the exact solution. We start with a theoretical overview of methods with fixed step sizes followed by an adaption to variable step sizes. Again, we use the notation y_j for an approximation of $y(t_j)$. If not explicitly given, proofs can be found in the work of Hairer et al. [14].

3.2.1 Fixed step size

Definition 3.1 (Linear multi-step method)
*A k-step **linear multi-step method** (LMM) with the given initial values $y_n, y_{n+1}, \dots, y_{n+k-1}$ and step size h to solve* (3.1) *is defined by*

$$\alpha_k y_{n+k} + \sum_{i=0}^{k-1} \alpha_i y_{n+i} = h \sum_{i=0}^{k} \beta_i f(y_{n+i}, t_{n+i}) \tag{3.2}$$

with $\alpha_i, \beta_i \in \mathbb{R}, \alpha_k \neq 0, |\alpha_0| + |\beta_0| > 0$ and $t_{n+j} = t_n + jh$.
*A method is called **explicit** if $\beta_k = 0$ and **implicit** otherwise.*

In order to rate the quality of a linear multi-step method the consistency order and stability properties are used.

Definition 3.2 (Consistency)
A LMM (3.2) *is **consistent of order p**, if*

$$L(y(t), h) := \sum_{i=0}^{k} \big(\alpha_i y(t+ih) - h\beta_i \dot{y}(t+ih)\big) = \mathcal{O}(h^{p+1}) \text{ for } h \to 0$$

holds for all $y(t) \in \mathcal{C}^{p+1}\big([t_0, t_f]\big)$.
*$L(y(t), h)$ is called **local (discretization) error**.*

The local discretization error propagates during the steps, hence we are interested in analyzing what happens for n increasing. Therefore, we define the following polynomials.

Definition 3.3 (Characteristic polynomials)
For a LMM (3.2) *we define the **characteristic polynomials***

$$\varrho(\zeta) := \alpha_k \zeta^k + \alpha_{k-1}\zeta^{k-1} + \dots + \alpha_0 \tag{3.3}$$
$$\sigma(\zeta) := \beta_k \zeta^k + \beta_{k-1}\zeta^{k-1} + \dots + \beta_0. \tag{3.4}$$

These polynomials contain informations we need to evaluate the stability of a linear multi-step method.

Definition 3.4 (0-Stability)
A LMM (3.2) *is called **(zero)stable** if the characteristic polynomial $\varrho(\zeta)$ satisfies the following conditions*

(i) The roots of $\varrho(\zeta)$ lie on or within the unit circle: $|\zeta| \leq 1$ for $\varrho(\zeta) = 0$

(ii) The roots on the unit circle are simple

Finally we want to make statements about convergence of those numerical methods, but for this purpose it should be ensured that (3.1) exhibits a unique solution. Hence, we make the following assumptions.

Assumption 3.5
In (3.1) *it holds:*

1. *f is continuous on $D := \big\{(y,t) \;\big|\; \|y(t) - y\| \le b,\ t \in [t_0, t_f]\big\}$*
2. *$\|f(y,t) - f(z,t)\| \le L\|y - z\|$ for $(y,t), (z,t) \in D$*

with $y(t)$ denoting the exact solution of (3.1)*, $b > 0$ and L is the so-called Lipschitz constant.*

Definition 3.6 (Convergence)
Under assumption 3.5 a LMM (3.2) *is called* ***convergent of order p*** *if for any IVP* (3.1) *with f sufficiently differentiable, there exists $h_0 > 0$ such that whenever the starting values satisfy*

$$\|y(t_0 + ih) - y_h(t_0 + ih)\| \le C_0 h^p \qquad \textit{for } h \le h_0, i = 0, \dots, k-1$$

it holds

$$\|y(t) - y_h(t)\| \le C h^p \qquad \textit{for } h \le h_0$$

where y_h is used to denote the numerically calculated solution.

Dahlquist showed in [10] that there is a correlation between the stability and consistency of a method and its convergence.

Theorem 3.7
A LMM (3.2) *is convergent of order p if and only if it is consistent of order p and zerostable.*

Proof: A detailed proof is also given by Hairer et al. in [14]. □

Besides zero-stability there are other concepts of stability, mainly based on the stability region. To illustrate what is meant by this we have to use the Dahlquist test function

$$\dot{y}(t) = \lambda y(t), \quad \lambda \in \mathbb{C}, \operatorname{Re}\lambda < 0. \tag{3.5}$$

Applying a linear multi-step method (3.2) on function (3.5) results in

$$(\alpha_k - h\lambda\beta_k)y_{n+k} + \cdots + (\alpha_0 - h\lambda\beta_0)y_n = 0 \tag{3.6}$$

and with setting $y_i = \zeta^i$ and dividing by ζ^n this motivates the next definition.

Definition 3.8 (Characteristic function)
*The **characteristic function** is defined as*

$$\Upsilon(\zeta; h\lambda) := \sum_{i=0}^{k} \big(\alpha_i - h\lambda\beta_i\big)\zeta^i = \varrho(\zeta) - h\lambda\sigma(\zeta)$$

An analytical solution of (3.5) can be given by $e^{\lambda t}$ and we note that this is a decreasing function (i.e. $|e^{\lambda s}| < |e^{\lambda t}|$ for $s > t$), since $\operatorname{Re}\lambda < 0$. In this case it is appropriate to define stability of a LMM if it holds $|y_{n+1}| \leq |y_n|$, since this is the behavior of the analytical solution. If we are considering the roots ζ_j of the characteristic function, they should satisfy $|\zeta_j| \leq 1$. Otherwise the next computed step would lead away from the exact solution.

Definition 3.9 (Stability region)
Let $\zeta_j(h\lambda)$, $j \in \mathbb{N}$ denote the roots of the function $\Upsilon(\zeta; h\lambda)$. The set $A \subset \mathbb{C}$ defined by

$$A := \Big\{ h\lambda \in \mathbb{C} \,\Big|\, |\zeta_j(h\lambda)| \leq 1 \textit{ for all roots}, \ |\zeta_j(h\lambda)| < 1 \textit{ for all multiple roots} \Big\}$$

*is then called **stability region**.*

Definition 3.10 (A-stability)
A LMM (3.2) *is said to be **A-stable** if its stability region contains the entire left half plane of* $\mathbb{C}$.

A slight weaker form of stability was first introduced by Widlund [34].

Definition 3.11 (A(α)-stability)
A LMM (3.2) *is said to be **A(α)-stable** with $0 < \alpha < \frac{\pi}{2}$ if its stability region contains the set $S := \{h\lambda \in \mathbb{C} \mid |\arg(-h\lambda)| < \alpha, h\lambda \neq 0\}$*

In this work we focus on a specific form of linear multi-step methods called Backward Differentiation Formulas (short BDF methods). These methods are suitable especially for stiff problems. There is no general definition for what is meant by stiff, but commonly this term is used for problems of type (3.1) if there are eigenvalues μ of the Jacobian $\frac{\partial f}{\partial y}$ with $\operatorname{Re}\mu < 0$ and $|\operatorname{Re}\mu| \gg 0$. Stiff problems often appear in context of chemical kinetics.

Definition 3.12 (BDF method)
*A k-step **BDF method** with given initial values $y_n, y_{n+1}, \ldots, y_{n+k-1}$ to solve* (3.1) *is defined by*

$$\sum_{i=0}^{k} \alpha_i y_{n+j} = hf(y_{n+k}, t_{n+k}) \tag{3.7}$$

According to definition 3.1 BDF methods are implicit, since $\beta_k = 1$. BDF methods are often called “methods of order k” instead of “k-step method”.

Theorem 3.13
BDF methods of order k are zero-stable for $k \leq 6$ and unstable for $k > 6$.

Proof: This is shown in [9]. A proof is also given in [14] where furthermore the factors α_i are listed for the stable BDF methods. □

BDF methods provide additional stability properties.

Theorem 3.14
BDF methods of order k are

A-stable for $k = 1, 2$

$A(\alpha)$-stable for $k = 3, \dots, 6$.

Proof: This is shown for example in [15]. The values of the corresponding α are listed below.

k	3	4	5	6
α	86.03°	73.35°	51.84°	17.84°

□

The stability regions of BDF methods are illustrated in figure 3.2. We consider the so-called root locus curve

$$h\lambda = \frac{\varrho(e^{\Theta i})}{\sigma(e^{\Theta i})}$$

and plot $h\lambda$ in the complex plane for $0 \leq \Theta \leq 2\pi$ (see e.g. [2]).

3.2.2 Variable step size

The first section deals with numerical integration algorithms on uniform time grids, but in practice numerical methods often use strategies to vary order and step size in adaption to the differential equation system. Thus we transfer the previous statements on a variable time grid, i.e. on variable step sizes h_j. We denote the time grid by $I_k = \{t_n, t_{n+1}, \dots, t_{n+k}\}$, hence the step sizes are given through $h_j = t_j - t_{j-1}$. It is also helpful to introduce $\omega_j = \frac{h_j}{h_{j-1}}$, the quotient of the step sizes.

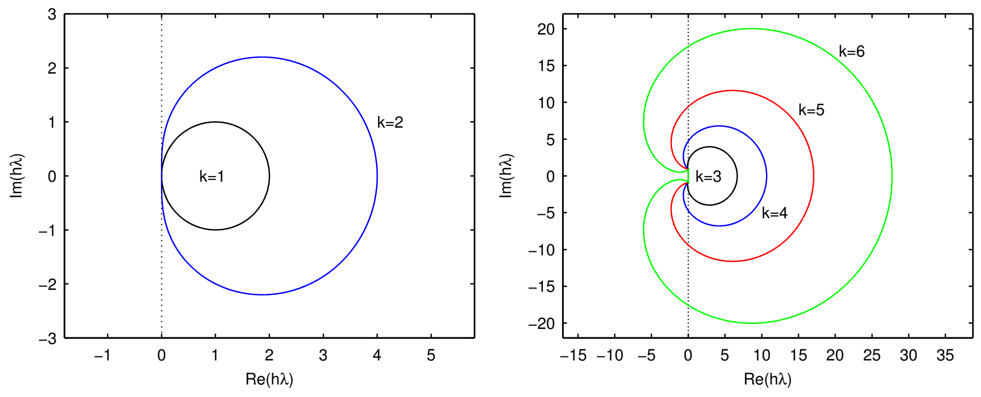

Figure 3.2: Stability regions of BDF methods up to order 6. Each method is stable for values of $h\lambda$ which are outside the given area.

Definition 3.15 (LMM on variable time grid)
*A k-step **linear multi-step method on a variable time grid** I_k with initial values $y_n, \dots, y_{n+k-1}$ to solve* (3.1) *is defined by*

$$y_{n+k} + \sum_{i=0}^{k-1} \alpha_{in} y_{n+i} = h_{n+k} \sum_{i=0}^{k} \beta_{in} f(y_{n+i}, t_{n+i}) \tag{3.8}$$

where $\alpha_{in} := \alpha_i(\omega_{n+2}, \dots, \omega_{n+k})$ and $\beta_{in} := \beta_i(\omega_{n+2}, \dots, \omega_{n+k})$ depend on the ratios ω_j.

Definition 3.16 (Consistency on variable time grid)
A linear multi-step method as in (3.8) *is **consistent of order p**, if*

$$P(t_{n+k}) + \sum_{i=0}^{k-1} \alpha_{in} P(t_{n+i}) = h_{n+k} \sum_{i=0}^{k} \beta_{in} \dot{P}(t_{n+i})$$

holds for all polynomials $P(t)$ with degree p or less and for all time grids I_k.

Theorem 3.17
Suppose a method as given in (3.8) *has the following characteristics:*

(i) *consistency order* $p \geq 0$

(ii) *the coefficients $\alpha_{in} = \alpha_i(\omega_{n+2}, \dots, \omega_{n+k})$ are continuous in a neighborhood of* $(1, \dots, 1)$

(iii) all roots of

$$\zeta^k + \sum_{i=0}^{k-1} \alpha_i(1,\ldots,1)\zeta^i = 0$$

lie in the open unit disc, with the exception of $\zeta_1 = 1$. That means the method is strongly stable on an equidistant time grid.

Then there exist $\omega, \Omega \in \mathbb{R}$ with $\omega < 1 < \Omega$ such that the method is stable if $\omega \leq \omega_j \leq \Omega$ for all j.

Proof: We refer to [14]. □

Similarly to the case with fixed step size we can find a correlation between stability and consistency and convergence, but more assumptions have to be made here.

Theorem 3.18
Let a method as given in (3.8) *fulfill the following conditions:*

(i) consistent of order p, stable and the coefficients α_{in}, β_{in} are bounded

(ii) the initial values satisfy $\|y(t_{n+i}) - y_{n+i}\| = \mathcal{O}(h^p), \quad i = 0,\ldots,k-1$

(iii) all step size ratios are bounded, i.e. $\omega_j \leq \Omega, \quad j = n+2,\ldots,n+k$

Then the method is convergent of order p. This means, there exists a $C \in \mathbb{R}$ for which

$$\|y(t_n) - y_n\| \leq C h_{max}^p$$

is satisfied if $t_n \in [t_0, t_f]$ and $h_{max} = \max_j h_j$.

Proof: See [14]. □

3.3 BDF method as implemented in DOT

In this section we present the fundamental ideas of the implementation of a variable step variable order backward differentiation formula. This implementation is based on the work by Byrne and Hindmarsh [5] with an application of Nordsieck arrays [24] and the step size strategy introduced by Calvo et al. in [6]. This method is used in DOT to solve ordinary differential equation systems. To simplify the notation in the next subsections, especially the indices, we consider the next step to be from $n-1$ to n, thus the previously computed values are indicated by $n-i$ with $i > 0$.

3.3.1 Prediction and correction

Given approximations $y_{n-1}, y_{n-2}, \dots, y_{n-k}$ we consider the interpolation polynomial $p_{n-1}(t)$ of degree k or less which satisfies the $k+1$ conditions

$$\begin{aligned} p_{n-1}(t_{n-j}) &= y_{n-j} \quad \forall j \in \{1, \dots, k\} \\ \dot{p}_{n-1}(t_{n-1}) &= \dot{y}(t_{n-1}) = f(y(t_{n-1}), t_{n-1}). \end{aligned} \tag{3.9}$$

We want to construct the polynomial p_n which satisfies

$$\begin{aligned} p_n(t_{n-j}) &= y_{n-j} \quad \forall j \in \{0, \dots, k\} \\ \dot{p}_n(t_n) &= \dot{y}(t_n) = f(y(t_n), t_n) \end{aligned} \tag{3.10}$$

and thus compute a value of y_n which makes this possible. By this we get a polynomial of degree k or less which has to satisfy $k+2$ conditions. We obtain a classical linear multi-step formulation as in definition 3.8 by a restatement of the conditions in (3.10) (e.g. see [5]). A first guess for approximations of y_n and $\dot{y}_n$ can be made by extrapolation of p_{n-1} to t_n. Let

$$y_{n,0} = p_{n-1}(t_n) \quad \text{and} \quad \dot{y}_{n,0} = \dot{p}_{n-1}(t_n)$$

denote those approximations. It should be clear that these estimations not necessarily have to be very accurate, i.e. the gap

$$\dot{y}_{n,0} - f(y_{n,0}, t_n) \tag{3.11}$$

may be large whereas for an exact solution it has to be zero. This gap is used as a residual and thus to correct $y_{n,0}$ and obtain $y_{n,1}$. This correction process is repeated iteratively until a sufficiently accurate solution which solves (3.10) is gained (see [5] for further details).
With y and f being vector valued functions, the ideas and strategies are adapted component-by-component but generally stay the same. But here, the Nordsieck arrays come into play. Instead of storing the vectors y_{n-1} and $\dot{y}_{n-1}$ in (3.9) we use the $n_y \times (k+1)$ matrix

$$z_{n-1} = \left[y_{n-1}, h y_{n-1}^{(1)}, \dots, \frac{h^k}{k!} y_{n-1}^{(k)} \right].$$

It should be pointed out that the derivatives used in here are the approximations gained by the polynomial

$$y_{n-1}^{(j)} = p_{n-1}^{(j)}(t_{n-1})$$

and $h = t_n - t_{n-1}$ is the step size of the next step. Although we operate with approximations, if the basic data is correct within an order of $\mathcal{O}(h^{k+1})$ we keep this correction order even for the derivatives (shown in [5]). According to the one dimensional case the task is to find z_n in dependency of z_{n-1}. We approximate z_n with the help of Pascal triangle matrices $A(k)$. These are lower triangular $(k+1)\times(k+1)$ matrices whose entries consist of the values in Pascal's triangle

$$A^{ij}(k) = \left\{ \begin{matrix} 0, & i<j \\ \binom{i}{j}, & i \geq j \end{matrix} \right\}, \quad i,j = 0,1,\ldots,k.$$

The first approximation for z_n is done by the prediction

$$z_{n,0} = z_{n-1}A(k) \tag{3.12}$$

We want to show a simple but fundamental relation between the first approximation $z_{n,0}$ and the exact solution z_n which will allow us to correct the entire prediction array (3.12) as soon as we obtain y_n. Consider we gained the correct y_n and hereby the polynomial p_n and the corrected Nordsieck array

$$z_n = \left[y_n, \ldots, \frac{h^k}{k!} y_n^{(k)} \right], \quad y_n^{(j)} = p_n^{(j)}(t_n). \tag{3.13}$$

We analyze the difference between the interpolation polynomials p_n and p_{n-1} which we describe by a new polynomial

$$\Delta_n(t) := p_n(t) - p_{n-1}(t).$$

For obvious reasons $\big((3.9)$ and $(3.10)\big)$ we obtain

$$\Delta_n(t_{n-j}) = 0, \quad j = 1,\ldots,k$$

thus the roots of this polynomial are t_{n-j} with $j = 1,\ldots,k$ and by using the Lagrange form for polynomials this gives us

$$\Delta_n(t) = \prod_{j=1}^{k} \frac{t - t_{n-j}}{t_n - t_{n-j}} e_n$$

with $e_n := p_n(t_n) - p_{n-1}(t_n)$.
We introduce new variables

$$x := \frac{t - t_n}{h}, \quad \xi_i := \frac{t_n - t_{n-i}}{h} \tag{3.14}$$

and the polynomial

$$\Lambda_n(x) := \prod_{j=1}^{k} \left(1 + \frac{x}{\xi_j}\right)$$

to get the result

$$\Delta_n(t) = \Delta_n(t_n + hx) = \Lambda_n(x)e_n. \tag{3.15}$$

By collecting terms in powers of x we write $\Lambda_n(x)$ as

$$\Lambda_k(x) = \sum_{j=0}^{k} l_j x^j$$

where l_j is given in the following way. With respect to (3.12),(3.13) and (3.15) we see that column j of $z_n - z_{n,0}$ is

$$\frac{h^j}{j!} p_n^{(j)}(t_n) - \frac{h^j}{j!} p_{n-1}^{(j)}(t_n) = \frac{h^j}{j!} \Delta_n^{(j)}(t_n) = \frac{1}{j!} \Lambda_n^{(j)}(0)e_n = l_j e_n$$

for $j = 0, \dots, k$.
Finally we define

$$l = [l_0, l_1, \dots, l_k]$$

a $1 \times (k+1)$ vector an get the relation

$$z_n = z_{n,0} + e_n l. \tag{3.16}$$

It can be shown that the algorithm given by (3.12) and (3.16) yields solutions y_n and thus z_n which are accurate to order k if $l_0 = 1$ and l_1 is bounded away from zero [5].

3.3.2 Step size strategy

Variable step sizes offer the opportunity to guarantee a desired accuracy in every step and thus influence the global error. To evaluate the actual step size a prescribed tolerance (TOL) and the discretization error (DE) are used. According to the notation in the previous sections a BDF method with variable step sizes is given in the form

$$\alpha_{0n} y_n = \sum_{j=1}^{k} \alpha_{jn} y_{n-j} - h_n \dot{y}_n$$

and therefore the discretization error in the n-th step is defined as

$$\mathrm{DE}_n := \sum_{j=0}^{k} \alpha_{jn} y(t_{n-j}) - h_n \dot{y}(t_n).$$

If $|\mathrm{DE}_n| > \mathrm{TOL}$ the step size will be rejected and a parameter θ is needed to define a new step size $h'_n = \theta h_n$ for which $|\mathrm{DE}'_n| \leq \mathrm{TOL}$ is fulfilled. A useful strategy has two goals. In case the actual step size is rejected, we want to compute a reliable θ with minimum effort to make sure that the new discretization error is approximately TOL. The second goal concerns the preparation of the next step. Predicting a factor α to have an optimal step size $h_{n+1} = \alpha h_n$ in the next step reduces the number of failed steps, the number of calculations and thus makes the code much more efficient.
The usual way (see e.g. [6]) is to set

$$\theta = \hat{\theta} \equiv \left(\frac{\mathrm{TOL}}{|\mathrm{DE}_n|} \right)^{\frac{1}{k+1}}$$

but it turns out that the choice of $\hat{\theta}$ for variable time grids is not as good as it is for uniform time grids. In fact, Calvo et al. mentioned in [6] that a better value for θ can be greater or smaller than $\hat{\theta}$ depending on increasing or decreasing step sizes. Defining

$$\mu := \frac{\mathrm{TOL}}{|\mathrm{DE}_n|}$$

it is recommended to use

$$\theta = \begin{cases} \sqrt{\mu^{\frac{1}{2}+\frac{1}{k+1}}} & \text{if } 0.05 \leq \mu \leq 1 \\ \nu \mu^{\frac{1}{k+1}} + (1-\nu)\mu^{\frac{1}{2}} & \text{if } \mu < 0.05 \end{cases}$$

as correction factor for a k-step BDF-method where

$$\nu := \left[\left(\frac{\prod_{j=2}^{k} \xi_j}{\sum_{j=1}^{k} \frac{1}{\xi_j} \prod_{j=2}^{k} (\xi_j - 1)} \right)^{\frac{1}{2}} \right] \mu^{\frac{(k-1)}{(2k+2)}}$$

and ξ_i as defined in (3.14).
The recommended term for α is

$$\alpha = \left(\frac{\sum_{j=1}^{k} \frac{1}{j}}{\sum_{j=1}^{k} \frac{1}{\xi_j}} \cdot \frac{\prod_{j=2}^{k} \xi_j}{k} \right)^{\frac{1}{(k+1)}} \mu^{\frac{1}{(k+1)}}$$

We refer to [6] for detailed informations about those terms. The integrator within the DOT package follows the suggestions of Calvo et al. and makes use of this step size strategy.

3.3.3 Sensitivity analysis

Instead of dealing with an IVP of type (3.1) we can consider

$$\begin{aligned} \dot{y}(t) &= f(y(t), p, t) \\ y(t_0) &= y_0 \end{aligned} \tag{3.17}$$

where $p \in \mathbb{R}^{n_p}$ is a vector consisting of parameters which affect the system. In many applications it is of interest how the system model reacts to a variation of those parameters, i.e. we are interested in the values $\frac{dy}{dp}$.
One way to obtain these values is computing finite differences. Here, we need a solution y of system (3.17) and a solution y_{per} of the perturbed system

$$\begin{aligned} \dot{y}_{\text{per}}(t) &= f(y(t), p + \epsilon q, t) \\ y_{\text{per}}(t_0) &= y_0 \end{aligned}$$

with $\epsilon > 0$ and $q \in \mathbb{R}^{n_p}$. The finite differences approach yields

$$\frac{dy(t)}{dp} = \frac{y_{\text{per}}(t) - y(t)}{\epsilon} + \mathcal{O}(\epsilon).$$

But in general, and especially if using an adaptive integrator, finite differences are not recommendable, since they suffer from inaccuracies. The values calculated by an adaptive method commonly do not exhibit continuous dependencies on the input. The settings of the integration scheme (e.g. step sizes) can differ significantly even for slight modifications and to guarantee a certain accuracy for the derivatives the accuracy of the computed solution has to be increased what in turn is computationally expensive.
The usual alternative is using the so-called variational differential equations. We denote the solution of (3.17) by $y(t; y_0, p)$ to point out the dependencies on y_0 and p.

Theorem 3.19
Let $k \in \mathbb{N}$ and $f \in \mathcal{C}^k(S, \mathbb{R}^{n_y})$ with $S = \mathbb{R}^{n_y} \times \mathbb{R}^{n_p} \times [t_0, t_f]$ and consider the derivatives of f on S are bounded. Then the solution $y(t; y_0, p)$ of (3.17) is

- *k-times continuously differentiable in y_0 and p*
- *$(k+1)$-times continuously differentiable in t*

and the derivatives of this solution with respect to y_0 or p are given as the solutions of the following variational differential equations:

$$\begin{aligned} \frac{d}{dt}\frac{\partial y(t; y_0, p)}{\partial y_0} &= f_y(y(t; y_0, p), p, t)\frac{\partial y(t; y_0, p)}{\partial y_0} \\ \frac{\partial y(t_0; y_0, p)}{\partial y_0} &= I \end{aligned}$$

and

$$\begin{aligned}\frac{d}{dt}\frac{\partial y(t;y_0,p)}{\partial p} &= f_y(y(t;y_0,p),p,t)\frac{\partial y(t;y_0,p)}{\partial p} + f_p(y(t;y_0,p),p,t)\\ \frac{\partial y(t_0;y_0,p)}{\partial p} &= 0\end{aligned}$$

Proof: See e.g. [8] □

Even when facing a nonlinear problem, the variational differential equations are always in linear form. The integration algorithm used in DOT includes a strategy which is based on the work of Albersmeyer and Bock [1] where they presented sophisticated principles of internal numerical differentiation in a forward and an adjoint mode. All needed values are calculated by differentiating the obtained integration scheme directly via automatic differentiation. To improve the robustness of the optimization approach the integrator is able to use the sensitivity information which is computed during the forward mode of automatic differentiation. Further explanations can be found in the upcoming PhD thesis of Dominik Skanda, who has implemented the whole BDF method and has kindly provided the integrator for this work.

CHAPTER 4

Numerical optimization

A general form of an optimization problem is given by

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & x \in S \end{aligned} \tag{4.1}$$

where $f : X \longrightarrow \mathbb{R}$ and $S \subset X$ for a topological space X.
Having this general formulation we can distinguish between the finite form, where $X = \mathbb{R}^n$, and the infinite form, where X is assumed to be $X = B$ for any Banach space B $\big($e.g. $B = \mathcal{C}([a, b], \mathbb{R})\big)$. In this chapter, we demonstrate details for the finite form since this is the one which is treated numerically. More details on the infinite formulation follow in the next chapter.
A general form of a finite-dimensional equality and inequality constrained optimization problem can be stated as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \\ & h(x) \leq 0 \end{aligned} \tag{4.2}$$

with $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^{n_g}$ and $h : \mathbb{R}^n \longrightarrow \mathbb{R}^{n_h}$ being at least twice continuously differentiable functions. We want to minimize the objective function f while maintaining both the equality constraints g and the inequality constraints h. Instead of searching for a maximum of f, we can search for a minimum of $-f$ which is equivalent.

4.1 Theoretical aspects

Definition 4.1 (Feasibility)
*The **feasible set** of* (4.2) *is defined by*

$$S := \{x \in \mathbb{R}^n \mid g(x) = 0, h(x) \leq 0\}$$

*and a point $x \in S$ is called **feasible**.*

Definition 4.2 (Optimality)
*A feasible point x_* is a **local minimum** of* (4.2) *if there exists a neighborhood $U_\epsilon(x_*)$, $\epsilon > 0$ such that it holds*

$$f(x_*) \leq f(x) \quad \forall x \in U_\epsilon(x_*) \cap S.$$

Definition 4.3 (Lagrangian function)
*The **Lagrangian function** $\mathcal{L}$ is defined by*

$$\mathcal{L}(x, \lambda, \mu) := f(x) + \lambda^\top g(x) + \mu^\top h(x)$$

with the so-called Lagrange multipliers $\lambda \in \mathbb{R}^{n_g}$ and $\mu \in \mathbb{R}^{n_h}$.

Definition 4.4 (Activity)
*Let x be a feasible point. An inequality constraint h_i is called **active**, if it holds $h_i(x) = 0$. We denote the **index set of active inequalities** in x by*

$$\mathcal{I}_h(x) := \{i \in \{1, \ldots, n_h\} \mid h_i(x) = 0\}.$$

Definition 4.5 (Regularity)
*A point x is called **regular**, if x is feasible and the gradients*

$$\{\nabla g_i(x)\}_{i=1}^{n_g} \cup \{\nabla h_j(x)\}_{j \in \mathcal{I}_h(x)}$$

are linear independent.

Theorem 4.6 (First order necessary conditions)
Let x_* be a regular point which is a local minimum of (4.2)*. Then there exist Lagrange multipliers $\lambda_* \in \mathbb{R}^{n_g}$ and $\mu_* \in \mathbb{R}^{n_h}$ such that the triple (x_*, λ_*, μ_*) satisfies the following conditions*

$$\begin{aligned} \nabla_x \mathcal{L}(x_*, \lambda_*, \mu_*) &= 0 \\ g(x_*) &= 0 \\ \mu_*^\top h(x_*) &= 0 \\ h(x_*) &\leq 0 \\ \mu_* &\geq 0 \end{aligned} \tag{4.3}$$

Proof: Nocedal and Wright give a proof in [23]. □

These conditions are also known as Karush-Kuhn-Tucker (KKT) conditions. Numerical methods for solving nonlinear optimization problems aim to obtain a point which fulfills this first order necessary conditions. Basically there are two numerical approaches. The first one is a so-called active-set strategy commonly used in context of sequential quadratic programming (see e.g. [23, Chapter 18]). In this work, we concentrate on the second approach, known as interior point method.

4.2 Interior point methods

Before we explain them in detail, we want to give a short sketch of the main idea of interior point methods. Interior point methods are iterative algorithms which solve a sequence of problems. The origin function is extended by a penalty term with a so-called penalty parameter. By this, the solutions, which are gained during the iteration process, are forced to remain strictly inside the feasible region. According to this, the optimal solution is reached by traversing the feasible region. The penalty parameter is slowly driven to zero, thus the solutions of the corresponding problems converge to the solution of the original problem. We present in this section an interior point method with a special focus on the implementation in the software package IPOPT by Wächter and Biegler [31, 32, 33].

Notations

The notation will be as follows,

$x^{(i)}$ stands for the i-th component of a vector $x \in \mathbb{R}^n$,

$X \in \mathbb{R}^{n \times n}$ denotes the matrix $X :=\text{diag}(x)$ for a vector x and

$e := [1, 1, \dots, 1]^\top \in \mathbb{R}^n$

We describe the method for a slight modification of (4.2), where we enlarge the system by introducing slack variables to rewrite the inequality constraints as equalities. In addition, to keep the notation as clear as possible, we only handle the special case with $x \geq 0$. In general, we would have to consider lower and upper bounds of x, $x_L \in [-\infty, \infty)^n$ and $x_U \in (-\infty, \infty]^n$ with $x_L^{(i)} \leq x_U^{(i)}$. Few changes are necessary to cover the general case, see [33].

Considering those rearrangements we reformulate (4.2) as

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{4.4a}$$

$$\text{s.t.} \quad g(x) = 0 \tag{4.4b}$$

$$x \geq 0 \tag{4.4c}$$

Interior points methods solve a sequence of problems

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \varphi_\mu(x) = f(x) - \mu\sum_{i=1}^{n} \ln(x^{(i)}) \\ \text{s.t.} \quad & g(x) = 0 \end{aligned} \tag{4.5}$$

for $\mu \downarrow 0$. This barrier problem (4.5) consists of the original objective function combined with a so-called penalty term. In fact, for $\mu = 0$ it is identically the objective function of (4.4). It is

$$\lim_{x^{(i)}\downarrow 0} \ln(x^{(i)}) = -\infty$$

thus $\varphi_\mu(x)$ in (4.5) increases exceedingly for $x^{(i)} \downarrow 0$. Consequently, it holds $x^{(i)} > 0$ $(i = 1, \ldots, n)$ for the computed solution of (4.5) as long as $\mu > \mu_\epsilon$ with $\mu_\epsilon \geq 0$.
The KKT conditions (4.3) of (4.5) are

$$\begin{aligned} \nabla\varphi_\mu(x) + \nabla g(x)\lambda &= 0 \\ g(x) &= 0 \end{aligned} \tag{4.6}$$

and here we encounter the following problem. In $\nabla\varphi_\mu(x)$ terms appear which include $\frac{1}{x^{(i)}}$ $\left(\text{e.g. } \frac{\partial\varphi_\mu(x)}{\partial x^{(1)}} = \frac{\partial f(x)}{\partial x^{(1)}} - \frac{\mu}{x^{(1)}}\right)$, thus the system (4.6) is not defined for solutions of (4.4) with an active bound $x^{(i)} = 0$ and consequently additional problems occur, e.g. when decreasing μ the radius of convergence of Newton's method applied to (4.6) converges to zero [30]. Introducing so-called dual variables

$$s^{(i)} := \frac{\mu}{x^{(i)}}$$

allows to formulate the conditions known as the primal-dual equations

$$\nabla f(x) + \nabla g(x)\lambda - s = 0 \tag{4.7a}$$

$$g(x) = 0 \tag{4.7b}$$

$$XSe - \mu e = 0 \tag{4.7c}$$

which are equivalent to (4.6).

Setting $\mu = 0$ under the requirements $x > 0$ and $s > 0$ these equations are equivalent to the KKT conditions (4.3) of the original problem (4.4).
In IPOPT, a primal-dual interior-point line-search filter method is implemented to solve problems of type (4.4) with the help of barrier problems (4.5). After fixing a value of $\mu = \mu_j$, an approximated solution of (4.5) is computed. As soon as this approximation satisfies certain conditions (which are given in section 4.2.2) the barrier parameter is decreased.
The computation of the solution for a barrier problem is done by an application of a damped Newton's method to the primal-dual equations (4.7). Given (x_k, λ_k, s_k) with $x_k, s_k > 0$ at iteration step k the following system, which is gained from the linearization of (4.7),

$$\begin{bmatrix} Q_k & A_k & -I \\ A_k^\top & 0 & 0 \\ S_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^s \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k\lambda_k - s_k \\ g(x_k) \\ X_k S_k e - \mu_j e \end{pmatrix} \tag{4.8}$$

is solved to obtain search directions d_k^x, d_k^λ and d_k^s. Here, $A_k := \nabla g(x_k)$ and Q_k is either the exact Hessian $H_k := \nabla^2_{xx}\mathcal{L}(x_k, \lambda_k, s_k)$ or an approximation to this Hessian. $\mathcal{L}$ denotes the Lagrangian function which is defined as

$$\mathcal{L}(x, \lambda, s) := f(x) + g(x)^\top \lambda - s^\top x.$$

Having these directions we need step sizes $\alpha_k, \alpha_k^s \in (0, 1]$ to set the next iterate

$$x_{k+1} := x_k + \alpha_k d_k^x \tag{4.9a}$$
$$\lambda_{k+1} := \lambda_k + \alpha_k d_k^\lambda \tag{4.9b}$$
$$s_{k+1} := s_k + \alpha_k^s d_k^s \tag{4.9c}$$

Details on how to determine these step sizes will be elaborated in section 4.2.3, first we want to focus on solving the non-symmetric system (4.8).

4.2.1 Solving the barrier problem

In IPOPT this system is solved in two steps. The first one is to solve the symmetric and smaller system

$$\begin{bmatrix} Q_k + X_k^{-1} S_k & A_k \\ A_k^\top & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_{\mu_j}(x_k) + A_k\lambda_k \\ g(x_k) \end{pmatrix} \tag{4.10}$$

In the second step the vector d_k^s is computed by

$$d_k^s = \mu_j X_k^{-1} e - s_k - X_k^{-1} S_k d_k^x. \tag{4.11}$$

This yields an equivalent solution, since only the last block row in (4.8) has been eliminated. We have to ensure the following properties of the matrix in (4.10) to guarantee certain decent properties for the filter line-search procedure. The top left block of this matrix – projected onto the null space of the constraint Jacobian $A_k^\top$ – has to be positive definite. Additionally problems occur if A_k does not have full rank, because singularities appear and a solution of (4.10) might not exist. Therefore in IPOPT a small modification of this matrix

$$\begin{bmatrix} Q_k + X_k^{-1} S_k + \delta_w I & A_k \\ A_k^\top & -\delta_c I \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_{\mu_j}(x_k) + A_k \lambda_k \\ g(x_k) \end{pmatrix} \tag{4.12}$$

with $\delta_w, \delta_c \geq 0$ is used.

4.2.2 Optimality error

Since we solve a sequence of problems (4.5), the algorithm needs a criterion to decide at which point

1. the barrier parameter μ can be decreased
2. the algorithm found a satisfying solution of the original problem and thus should finish.

This motivates the definition of an optimality error

$$E_\mu(x, \lambda, s) := \max \left\{ \frac{\|\nabla f(x) + \nabla g(x)\lambda - s\|}{\sigma_d}, \|g(x)\|, \frac{\|XSe - \mu e\|}{\sigma_c} \right\}$$

with $\|\cdot\| = \|\cdot\|_\infty$ the maximum norm and scaling parameters $\sigma_d, \sigma_c \geq 1$. By this we obtain a measure for the violation of the primal-dual equations (4.7). The scaling gets necessary if the multipliers λ or s become very large and this can occur even if the original problem (4.4) is well scaled. In IPOPT these scaling factors are set to

$$\sigma_d = \frac{\max\left\{\sigma_{\max}, \dfrac{\|\lambda\|_1 + \|s\|_1}{n_g + n}\right\}}{\sigma_{\max}} \quad \text{and} \quad \sigma_c = \frac{\max\left\{\sigma_{\max}, \dfrac{\|s\|_1}{n}\right\}}{\sigma_{\max}}$$

with constant $\sigma_{\max} \geq 1$.

The algorithm pursues a strategy proposed by Byrd, Liu and Nocedal in [4] which offers fast local convergence properties (section 4.2.4). Let $(\hat{x}_j, \hat{\lambda}_j, \hat{s}_j)$ denote the approximated solution of the barrier problem for fixed μ_j. The barrier parameter will be decreased, if

$$E_{\mu_j}(\hat{x}_j, \hat{\lambda}_j, \hat{s}_j) \leq \kappa_\epsilon \mu_j$$

for a constant $\kappa_\epsilon > 0$ is satisfied. The new parameter is set as

$$\mu_{j+1} = \max\left\{\frac{\epsilon_{\text{tol}}}{10}, \min\{\kappa_\mu \mu_j, \mu_j^{\theta_\mu}\}\right\} \tag{4.13}$$

with constants $\kappa_\mu \in (0,1)$, $\theta_\mu \in (1,2)$ and a selectable tolerance $\epsilon_{\text{tol}} > 0$. $E_0(x, \lambda, s)$ gives a measure for the optimality error of the original problem. Hence the algorithm will stop as soon as

$$E_0(x, \lambda, s) \leq \epsilon_{\text{tol}} \tag{4.14}$$

is satisfied.

4.2.3 Obtaining step sizes

We introduce a parameter

$$\tau_j := \max\left\{\tau_{\min}, 1 - \mu_j\right\} \tag{4.15}$$

with $\tau_{\min} \in (0,1)$ and formulate the fraction-to-the-boundary rule

$$\alpha_k^{\max} := \max\left\{\alpha \in (0,1] \;\middle|\; x_k + \alpha d_k^x \geq (1-\tau_j)x_k\right\} \tag{4.16a}$$

$$\alpha_k^{s} := \max\left\{\alpha \in (0,1] \;\middle|\; s_k + \alpha d_k^s \geq (1-\tau_j)s_k\right\} \tag{4.16b}$$

to treat the step sizes in (4.9). Hence we have a definition for the step size α_k^s used in (4.9c). To determine the second step size $\alpha_k \in (0, \alpha_k^{\max}]$ in (4.9a) and (4.9b), we make use of a backtracking line-search procedure on a sequence of trial step sizes

$$\beta_l = 2^{-l}\alpha_k^{\max} \text{ with } l = 0, 1, 2, \ldots .$$

We set $\alpha_k = \beta_{l_*}$ where l_* is the index of the first accepted trial step size β_l. The idea of the line-search filter method, first presented in [11], is essential. Recall (4.5), a β_l is going to be accepted, if it either yields a satisfying decrease of the objective function φ_{μ_j} *or* the constraint violation. With

$$x_k(\beta_l) := x_k + \beta_l d_k^x \tag{4.17}$$

and

$$\Theta(x_k(\beta_l)) := \big\|g(x_k(\beta_l))\big\|$$

as a measure for the constraint violation ($\|\cdot\|$ is a fixed vector norm) this criterion can be formulated as satisfying

$$\begin{aligned} \varphi_{\mu_j}(x_k(\beta_l)) &\leq \varphi_{\mu_j}(x_k) - \gamma_\varphi \Theta(x_k) && (4.18a)\\ \text{or} \quad \Theta(x_k(\beta_l)) &\leq (1-\gamma_\Theta)\Theta(x_k) && (4.18b) \end{aligned}$$

with $\gamma_\Theta, \gamma_\varphi \in (0,1)$ being constant factors.
However, these conditions alone do not guarantee global convergence of the algorithm since several problems can occur. We will work out details on convergence properties in the next section, but first we present three essential techniques which prevent the algorithm from providing incorrect solutions or being stalled.

Switching condition

Criterion (4.18) is changed when $\Theta(x_k) < \Theta_{\min}$ holds and the so-called switching condition

$$\nabla\varphi_{\mu_j}(x_k)^\top d_k^x < 0 \quad \text{and} \quad \beta_l[-\nabla\varphi_{\mu_j}(x_k)^\top d_k^x]^{s_\varphi} > \delta[\Theta(x_k)]^{s_\Theta} \tag{4.19}$$

with $\delta > 0, s_\Theta > 1$ and $s_\varphi \geq 1$ is satisfied for the current iterate. Condition (4.19) can be fulfilled for a feasible but non-optimal point $\bar{x}$ and to prevent the method, i.e. the solutions computed by the method, from converging to $\bar{x}$, a sufficient progress concerning the objective function has to be ensured. Here, this is done by replacing criterion (4.18) by the Armijo condition

$$\varphi_{\mu_j}(x_k(\beta_l)) \leq \varphi_{\mu_j}(x_k) + \eta_\varphi \beta_l \nabla\varphi_{\mu_j}(x_k)^\top d_k^x \tag{4.20}$$

where $\eta_\varphi \in (0, \frac{1}{2})$ is a constant.

Filter method

A second problem which can appear is the following. Recall that the step size is accepted if either (4.18a) or (4.18b) is satisfied. Assume two feasible points $\bar{x}_1$ and $\bar{x}_2$ with

$$\begin{aligned} \varphi_{\mu_j}(\bar{x}_1) < \varphi_{\mu_j}(\bar{x}_2) \text{ and}\\ \Theta(\bar{x}_2) < \Theta(\bar{x}_1). \end{aligned} \tag{4.21}$$

Hence, it is conceivable, that the algorithm yields solutions which alternate between $\bar{x}_1$ and $\bar{x}_2$. This incident is called cycling and to strictly avoid cycling the algorithm makes use of a so-called filter. We want to point out the main idea of a filter, for this purpose let (Θ_k, φ_k) denote the pair $\big(\Theta(x_k), \varphi_{\mu_j}(x_k)\big)$.

Definition 4.7 (Domination)
A pair (Θ_k, φ_k) ***dominates*** *another pair (Θ_l, φ_l) if it holds both $\Theta_k \leq \Theta_l$ and $\varphi_k \leq \varphi_l$.*

Definition 4.8 (Filter set)
A list of pairs such that no pair is dominated by any other is called ***filter****. The set which contains the filter and all pairs which are dominated is called the* ***filter set****.*

A graphical illustration is given in figure (4.1).

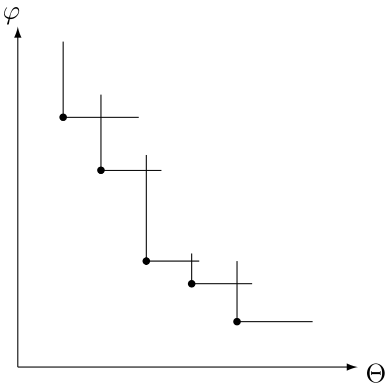

Figure 4.1: Illustration of a filter. We sketch filter points and domination barriers, i.e. any pair (Θ_k, φ_k) to the upper right of a filter point is dominated.

The IPOPT algorithm maintains for each iteration k a filter set

$$\mathcal{F}_k \subseteq \{(\Theta, \varphi) \in \mathbb{R}^2 \mid \Theta \geq 0\}.$$

This set contains those combinations of Θ and φ which are ineligible for a successful trial point in this iteration step. The filter is initialized to

$$\mathcal{F}_0 = \{(\Theta, \varphi) \in \mathbb{R}^2 \mid \Theta \geq \Theta^{\max}\} \tag{4.22}$$

an will be extended by the formula

$$\mathcal{F}_{k+1} = \mathcal{F}_k \cup \big\{(\Theta, \varphi) \in \mathbb{R}^2 \big| \Theta \geq (1 - \gamma_\Theta)\Theta(x_k) \text{ and } \varphi \geq \varphi_{\mu_j}(x_k) - \gamma_\varphi \Theta(x_k)\big\}$$

if (4.19) or (4.20) is not fulfilled for the accepted trial step size. Whenever

$$(\Theta(x_k(\beta_l)), \varphi_{\mu_j}(x_k(\beta_l))) \in \mathcal{F}_k$$

holds for a trial point $x_k(\beta_l)$, this point is not acceptable to the current filter and is rejected during the line search. By this, the algorithm cannot stuck between two points that decrease constraint violation and objective function rotationally (see (4.21)). The filter is reset to its initial definition (4.22) when the barrier parameter μ_j is decreased.

Restoration phase

Generally it is not guaranteed to find an appropriate step size which satisfies the filter criterion given above. For this reason the algorithm enters the feasibility restoration phase if the trial step size becomes smaller than $\alpha_k^{\min}$. Here, an iterative method is used to decrease the constraint violation until a new iterate x_{k+1} is found, which satisfies (4.18) and is accepted by the filter. According to [30] the value of $\alpha_k^{\min}$ is defined by

$$\alpha_k^{\min} := \begin{cases} \gamma_\alpha \cdot \min\left\{\gamma_\Theta, \frac{\gamma_\varphi \Theta(x_k)}{-\nabla\varphi_{\mu_j}(x_k)^\top d_k^x}, \frac{\delta[\Theta(x_k)]^{s_\Theta}}{[-\nabla\varphi_{\mu_j}(x_k)^\top d_k^x]^{s_\varphi}}\right\} & \text{if } \nabla\varphi_{\mu_j}(x_k)^\top d_k^x < 0 \\ \gamma_\alpha \cdot \gamma_\Theta & \text{otherwise} \end{cases}$$

with $\gamma_\alpha \in (0, 1]$.
This definition results from the following considerations. Assuming the actual step size β_l being large enough such that the switching condition (4.19) can be fulfilled for a $\alpha \leq \beta_l$. So we can see that for

$$\beta_l > \frac{\delta[\Theta(x_k)]^{s_\Theta}}{[-\nabla\varphi_{\mu_j}(x_k)^\top d_k^x]^{s_\varphi}}$$

there is the possibility of satisfying the Armijo condition (4.20) with a shorter step. If (4.19) does not hold for β_l (and all $\alpha < \beta_l$), the decision whether to enter the restoration phase depends on the linear approximations

$$\begin{aligned} \tilde{\varphi}_{\mu_j}(x_k + \nu d_k^x) &= \varphi_{\mu_j}(x_k) + \nu\nabla\varphi_{\mu_j}(x_k)^\top d_k^x \\ \tilde{\Theta}(x_k + \nu d_k^x) &= \Theta(x_k) - \nu\Theta(x_k) \end{aligned}$$

in the following way. Substituting those approximations into (4.18) leads to the conclusion that in case of $\nabla\varphi_{\mu_j}(x_k)^\top d_k^x < 0$ condition (4.18a) only is ensured if

$$\beta_l > \frac{\gamma_\varphi \Theta(x_k)}{-\nabla\varphi_{\mu_j}(x_k)^\top d_k^x}$$

and that (4.18b) only can be guaranteed if

$$\beta_l > \gamma_\Theta.$$

The restoration phase also helps to detect errors in the modeling of the problem. For example, if the given problem is infeasible because it is ill-posed, the algorithm is ultimately not able to generate sufficient progress in the line-search method and reverts to the restoration phase [33]. In this case, it holds

$$\Theta(x) > c_\Theta$$

for all feasible points x and $c_\Theta > 0$, since the constraints are always violated.

4.2.4 Global and local convergence

It is sufficient to ensure global convergence for a fixed value of μ_j to ensure global convergence of the overall algorithm. Wächter and Biegler showed both global and local convergence properties for problems of type

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \end{aligned} \tag{4.23}$$

in [32] and [31]. Problem (4.5) indeed is of that type.
The linearization of the KKT conditions for (4.23) at an iterate point x_k leads to

$$\begin{bmatrix} Q_k & A_k \\ A_k^\top & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ \tilde{\lambda}_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) \\ g(x_k) \end{pmatrix} \tag{4.24}$$

with $\tilde{\lambda}_k := d_k^\lambda - \lambda_k$. The notation remains the same as in (4.8), but the Lagrangian function has the form

$$\mathcal{L}(x, \lambda) = f(x) + g(x)^\top \lambda.$$

The assumptions which have to be made to ensure global convergence are stated in the following. We introduce the notation $[a, b] \subset \mathbb{R}^n$ with $a, b \in \mathbb{R}^n$ which is defined as the box $[a, b] := [a_1, b_1] \times \cdots \times [a_n, b_n]$.

Assumption 4.9
Consider that the algorithm described in this chapter generates a sequence $(x_k)_{k \in \mathbb{N}}$, where the restoration phase always terminates properly and the algorithm does not stop at a KKT point.

(A1) The functions f and g are differentiable on an open set $C \subseteq \mathbb{R}^n$, with $[x_k, x_k + d_k^x] \subset C$ and both their function values and first derivatives are bounded and Lipschitz-continuous on this set C.

(A2) The Hessian approximations Q_k in (4.24) are uniformly bounded

(A3) The matrices Q_k are uniformly positive definite projected onto the null space of the Jacobian $A_k^\top$

(A4) There exists a constant $c_A > 0$ so that the smallest singular value of A_k denoted by $\sigma_{\min}(A_k)$ holds $\sigma_{\min}(A_k) \geq c_A$

(A5) There exists a constant $c_\theta > 0$ so that the system (4.24) is not ill-conditioned or singular for $\theta(x_k) \leq c_\theta$

To be more precise, the assumptions *(A1)-(A4)* only have to hold for those iteration steps where the restoration phase has not been invoked because of an ill-conditioned or singular system (4.24). We refer to [32] for a detailed view on these assumptions.

Theorem 4.10 (Global convergence)
Suppose assumption 4.9 hold. Then all limit points of the sequence $(x_k)_{k\in\mathbb{N}}$ are feasible, and if $(x_k)_{k\in\mathbb{N}}$ is bounded, there exists a limit point x_* which fulfills the KKT-conditions for (4.23).

Proof: A proof is given in [32]. An elaboration of the algorithm can also be found there. □

We sketch now necessary assumptions for the local convergence analysis.

Assumption 4.11
Suppose the generated sequence $(x_k)_{k\in\mathbb{N}}$ of iterates converges to a local solution x_* of (4.23). *Additionally the following assumptions have to be made.*

(B1) The functions f and g are twice continuously differentiable in a neighborhood of x_*

(B2) There exists $\lambda_* \in \mathbb{R}^m$ so that the KKT conditions of (4.23) *are fulfilled for (x_*, λ_*)*

(B3) The Jacobian $A(x_*)^\top$ has full rank

(B4) The Hessian of the Lagrangian $H_* := \nabla^2_{xx}\mathcal{L}(x_*, \lambda_*)$ projected on the null space of $A(x_*)^\top$ is positive definite

(B5) Q_k is bounded and uniformly positive definite on the null space of $A_k^\top$

(B6) Q_k always fulfills $(H_k - Q_k)d_k^x = o(\|d_k^x\|)$

(B7) There exists a constant $c_\theta > 0$ so that the system (4.24) *is not ill-conditioned or singular for $\theta(x_k) \leq c_\theta$*

We have to enlarge these assumptions for the following reason. In [11], Fletcher and Leyffer considered the possibility of suffering from the so-called Maratos effect [23, Chapter 15.5]. This incident occurs when the filter line search method only allows small steps because otherwise both the objective value an the constraint violation will be increased. Since this could happen even close to a local solution, a poor local convergence behavior would be the consequence. The calculation of a second order correction step [31] is proposed. This correction step is computed whenever $x_k(\beta_0)$ (recall (4.17)) is rejected due to one of the following reasons (*i*) $(\Theta(x_k(\beta_0)), \varphi_{\mu_j}(x_k(\beta_0))) \in \mathcal{F}_k$

(*ii*) neither (4.18) nor (4.19) are satisfied

(*iii*) (4.19) is fulfilled but not (4.20).

Before decreasing the trial step size β_0 to β_1, it is first verified if the corrected step

$$x_{k+1} := x_k + d_k^x + d_k^{\text{soc}}$$

is accepted by the algorithm. We have to formulate

(B8) Q_k^{soc} *is bounded and uniformly positive definite on the null space of* $(A_k^{soc})^\top$

for the corresponding matrices Q_k^{soc} and A_k^{soc} which are used to obtain the second order correction step size d_k^{soc} (see [31] for details).

Theorem 4.12 (Local convergence)
Suppose assumption 4.11 hold. For $k \geq N$ *with* $N \in \mathbb{N}, N \gg 1$ *full steps of the form* $x_{k+1} = x_k + d_k^x$ *or* $x_{k+1} = x_k + d_k^x + d_k^{soc}$ *are taken and* x_k *converges to* x_* *superlinearly.*

Proof: The proof and detailed informations about the assumptions and the second order corrections can be found in [31]. □

CHAPTER 5

On the implementation of DOT

We want to discuss optimal control problems and the way we solve them in DOT. For the theory of optimal control, results of the functional analysis are needed due to the fact that functions between any desired Banach spaces have to be considered. Thus we have the infinite form of an optimization problem. In DOT, we tackle the infinite form by discretizing whereby we achieve a finite dimensional formulation which can be solved by the techniques described in the previous chapters. We want to give both theoretical and practical details on DOT but nevertheless, we do not want to disregard the infinite problem class completely. Hence, we begin with a short sketch of theoretical fundamentals of the optimal control theory and present Pontryagin's minimum principle which gives first order necessary conditions. Afterwards we spot the difference between direct and indirect methods, i.e. mainly the decision whether to make use of Pontryagin or not. Subsequently we give a step by step explanation on how the discretization, based on the infinite form, is done in DOT. We complete this chapter with an entire demonstration of the implemented problem formulation.

Notation

In this chapter we use the following notation

$y(t) \in \mathbb{R}^n$ denotes the state vector of a system at time t

$u(t) \in \mathbb{R}^m$ stands for the control vector at time t

$T_0 \in \mathbb{R}$ is the start and

$T_f \in \mathbb{R}$ the end time of the process

5.1 Theoretical basis of optimal control

We consider dynamics which can be described by a system of ordinary differential equations

$$\dot{y}(t) = f(y(t), u(t), t) \text{ with } t \in [T_0, T_f]. \tag{5.1}$$

Here, $f : \mathbb{R}^n \times \mathbb{R}^m \times [T_0, T_f] \longrightarrow \mathbb{R}^n$ is a continuous function and the derivatives $f_y(y, u, t)$ and $f_u(y, u, t)$ exist as matrices with continuous entries. The system is called autonomous if $\dot{y}(t) = f(y(t), u(t))$.

Definition 5.1
A pair (y, u) with

$y : [T_0, T_f] \longrightarrow \mathbb{R}^n$ *continuous and piecewise continuously differentiable*

$u : [T_0, T_f] \longrightarrow \mathbb{R}^m$ *piecewise continuous*

*is a **solution** of* (5.1)*, if*

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

holds for all $t \in [T_0, T_f]$ where u is continuous.

Besides the dynamics of the system, there are three other attributes which characterize an optimal control problem, in particular, the boundary conditions (state values $y(t)$ at $t = T_0$ and $t = T_f$), the admissible range U of the control function and the objective function F.

Definition 5.2 (Standard optimal control problem)
*A nonlinear **optimal control problem with standard boundary conditions** is defined by*

$$\begin{aligned} \min\ F(y, u) &:= \Phi(y(T_f)) + \int_{T_0}^{T_f} f_0(y(t), u(t), t)dt \\ \text{subject to} \quad \dot{y}(t) &= f(y(t), u(t), t) \text{ with } t \in [T_0, T_f] \\ y(T_0) &= y_0 \\ \psi(y(T_f)) &= 0 \\ u(t) &\in U \end{aligned} \tag{5.2}$$

with $y_0 \in \mathbb{R}^n, \Phi : \mathbb{R}^n \longrightarrow \mathbb{R}$ and $\psi : \mathbb{R}^n \longrightarrow \mathbb{R}^r$ $(0 \leq r \leq n)$ continuously differentiable and $f_0 : \mathbb{R}^n \times \mathbb{R}^m \times [T_0, T_f] \longrightarrow \mathbb{R}$ continuously partial differentiable with respect to y and u. The final time T_f can be fix or free and $U \subset \mathbb{R}^m$ is a nonempty, closed and convex set.

Definition 5.3 (Hamiltonian function)
According to (5.2) *we define the* ***Hamiltonian function***

$$\mathcal{H}(y, u, \lambda, t) := \lambda_0 f_0(y, u, t) + \lambda^\top f(y, u, t) \tag{5.3}$$

with $\lambda_0 \in \mathbb{R}$ and $\lambda \in \mathbb{R}^n$. The vector λ is called ***adjoint variable****.*

Theorem 5.4 (Pontryagin's minimum principle)
Let (y^*, u^*) and, if applicable, T_f^* be an optimal solution of (5.2)*. Then there exist $\lambda_0 \geq 0$, a continuous and piecewise continuous differentiable function $\lambda : [T_0, T_f] \longrightarrow \mathbb{R}^n$ and a vector $\nu \in \mathbb{R}^r$ with $(\lambda_0, \lambda, \nu) \neq 0 \;\forall\, t \in [T_0, T_f]$ such that the following conditions hold:*

1. *The* ***minimum condition*** *at all points $t \in [T_0, T_f]$ where u^* is continuous*
$$\mathcal{H}(y^*(t), u^*(t), \lambda(t), t) = \min_{u \in U} \mathcal{H}(y(t), u, \lambda(t), t)$$

2. *The* ***adjoint differential equation*** *at all points $t \in [T_0, T_f]$ where u^* is continuous*
$$\dot{\lambda}(t) = -\mathcal{H}_y(y^*(t), u^*(t), \lambda(t), t)$$

3. *The* ***transversality condition*** *at the final time T_f*
$$\lambda(T_f) = \lambda_0 \Phi_y(y^*(T_f), T_f) + \nu^\top \psi_y(y^*(T_f))$$

4. *In case of a free end time it holds for T_f^**
$$\mathcal{H}(y^*(T_f^*), u^*(T_f^*), \lambda(T_f^*), T_f^*) = 0$$

Proof: The first three statements are proved in [25] and the last one is shown in [17]. $\square$

5.2 Indirect and direct approach

There are basically two approaches, an indirect and a direct one, to handle optimal control problems. Indirect methods essentially make use of Pontryagin's minimum principle. The minimum condition presented in theorem 5.4

is used to express the control function u in terms of y and λ whereby u is eliminated as an variable. Additionally, the adjoint differential equation and the transversality condition are used to formulate a boundary value problem for $x = (y, \lambda) \in \mathbb{R}^{2n}$ which, in the simplest case, is characterized by

$$\dot{x}(t) = \begin{pmatrix} \theta_1(x_1, x_2, \dots, x_{2n}, t) \\ \theta_2(x_1, x_2, \dots, x_{2n}, t) \\ \vdots \\ \theta_{2n}(x_1, x_2, \dots, x_{2n}, t) \end{pmatrix}, \qquad \Psi(x(T_0), x(T_f)) = 0$$

with $\Psi : \mathbb{R}^{2n} \times \mathbb{R}^{2n} \longrightarrow \mathbb{R}^{2n}$ and $\theta_i : \mathbb{R}^{2n} \times [T_0, T_f] \longrightarrow \mathbb{R}$ $(i = 1, \dots, 2n)$.
To solve such boundary value problems so-called shooting methods (single or multiple shooting, see e.g. [28]) are used. The indirect method is often named as *first optimize then discretize* approach.
In contrast, the functions for the states *and* controls are discretized when using a direct method what yields a finite dimension optimal control problem. For this reason the direct method is also called *first discretize then optimize* approach. We give more details on this approach in the following subsections.
Indirect methods yield very accurate solutions, but to formulate the boundary value problem knowledge about the theory of optimal control is needed. Additionally, the formulation of the adjoint differential equation system can become very exhausting and to gain convergence of an indirect method usually good initial guesses for the adjoint variables are needed.
Direct methods need none of those assumptions. The adjoint differential equation system does not have to be formulated, but anyhow we gain values of the adjoint variables after obtaining a solution of the problem. We concentrate here on autonomous differential equations and thus skip the notation of t as an explicit argument at certain points. Every non-autonomous problem of type (5.2) can be transformed in an autonomous one by introducing a new state variable $y_{n+1}(t) = t$ with the corresponding constraints $\dot{y}_{n+1} = 1$ and $y_{n+1}(T_0) = T_0$.
We enlarge our problem class (as in section 3.3.3) by introducing a parameter vector p which potentially influences the system, formally we append p as an argument. Additionally, there will occur constraints g, together with a lower bound g^L and a upper bound g^U. These constraints are collected together in the finite-dimensional problem formulation which is given in (5.12).

5.2.1 Time discretization

The discretization starts with generating a time grid

$$T_0 < T_1 < T_2 < \ldots < T_M := T_f$$

with $M \in \mathbb{N}$ and considering subintervals $[T_j, T_{j+1}]$ for $j = 0, \ldots, M-1$. The grid points T_j are also called nodes. We define

$$\Delta t_j := T_{j+1} - T_j \qquad j = 0, \ldots, M-1 \tag{5.4}$$

as the length of a subinterval.
The time grid could be forced to be equidistant, i.e. $\Delta t_j = \Delta t_k$ $(j, k = 0, \ldots, M-1)$, but also time intervals with different length are possible. To get a more accurate solution, we can increase M, but consequently this increases the computational effort. If the end time T_f is free, the length of a subinterval Δt_j itself gets an optimization variable. In DOT all this options are handled by adding the constraint

$$g_{\text{time}}^L \leq g_{\text{time}}(p) \leq g_{\text{time}}^U \tag{5.5}$$

where the final time T_f and Δt_k $(k = 0, \ldots, M-1)$ are implemented as coefficients of the parameter vector p.

5.2.2 Control parametrization

Based on this time grid every control function gets piecewise represented on a subinterval by a polynomial of order k

$$u(t) = u_0 + u_1 t + u_2 t^2 + \ldots + u_k t^k =: \varphi_j(t, v_j) \qquad t \in [T_j, T_{j+1}] \tag{5.6}$$

with $v_j = [u_0, \ldots, u_k]$. The order k of this polynomial does not necessarily have to stay the same on all subintervals. By finding an optimal control, in fact, we try to find the optimal control coefficients parameterizing this control. Working on subintervals explicitly allows discontinuous solutions for the control functions on the whole interval. In section 5.2.4 we show how to guarantee an overall continuous solution.

5.2.3 Multiple shooting approach

These discretization techniques assist the multiple shooting approach of Bock and Plitt [3]. The system variables are considered on every node T_j for

$j = 0, \dots, M$. We use $y(t; s_j, v_j)$ to denote the solution of an initial value problem

$$\begin{aligned} \dot{y}(t) &= f(y(t), \varphi_j(t, v_j), p) \quad t \in [T_j, T_{j+1}] \\ y(T_j) &= s_j. \end{aligned}$$

The solution on the whole interval $[T_0, T_M]$ then has to be a continuous function pieced together by these sub-solutions,

$$\begin{aligned} y(t) &= y(t; s_j, v_j) \text{ for } t \in [T_j, T_{j+1}[, \;\; j = 0, \dots, m-1 \\ y(T_M) &= s_M. \end{aligned} \tag{5.7}$$

In order to guarantee continuity in (5.7) the following conditions have to be fulfilled

$$g_{\text{ms}} := \begin{pmatrix} y(T_1; s_0, v_0) - s_1 \\ y(T_2; s_1, v_1) - s_2 \\ \vdots \\ y(T_M; s_{M-1}, v_{M-1}) - s_M \end{pmatrix} = 0 \tag{5.8}$$

ignoring the explicit dependencies of the constraint function g_{ms}.
With this approach, we solve the differential equation system piecewise on subintervals which reduces the growth of error noticeable. In DOT, before the integration starts, there is a rescaling whereby the integration limits are set to 0 and 1. This is based on the following substitution. Remembering the definition of Δt_j in (5.4) any $t \in [T_j, T_{j+1}]$ can be written as

$$t = \sum_{i=0}^{j-1} \Delta t_i + \vartheta \Delta t_j \text{ with } \vartheta \in [0, 1]$$

which gives

$$\frac{dt}{d\vartheta} = \Delta t_j.$$

Therefore a substitution of

$$\int_{T_j}^{T_{j+1}} f(y(t), u(t), p) dt$$

is given by

$$\int_0^1 f\left(y\Big(\sum_{i=0}^{j-1}\Delta t_i+\vartheta\Delta t_j\Big),u\Big(\sum_{i=0}^{j-1}\Delta t_i+\vartheta\Delta t_j\Big),p\right)\Delta t_j d\vartheta.$$

This substitution allows to obtain time derivatives directly out of the integrator, since we have Δt_i as explicit arguments.

5.2.4 Constraints

As a further consequence of the time discretization we can formulate constraints among the resulting time grid. We use the start, end and interior nodes to separate those constraint in three parts. More precisely, for the constraints at the beginning we have

$$g_{\text{beg}}^L \le g_{\text{beg}}(y(T_0),u(T_0),p) \le g_{\text{beg}}^U \tag{5.9}$$

with the understanding of $u(T_0)$ as $\varphi_0(T_0,v_0)$. In a similar way we introduce at the end

$$g_{\text{end}}^L \le g_{\text{end}}(y(T_M),u(T_M),p) \le g_{\text{end}}^U \tag{5.10}$$

with the understanding of $u(T_M)$ as $\varphi_{M-1}(T_M,v_{M-1})$. For the interior nodes we also have

$$g_{\text{int}}^L \le g_{\text{int}}(y(t),u(t),p) \le g_{\text{int}}^U \tag{5.11}$$

with $t \in \{T_1,T_2,\dots,T_{M-1}\}$ as an acceptable description, but we want to point out details here. Consider a node T_j with $j \in \{1,\dots,M-1\}$ and recall (5.6). We have $\varphi_{j-1}(T_j,v_{j-1})$, the control function defined on the interval which ends at this node. But, since T_j is also starting point for the next interval, we have the related control function $\varphi_j(T_j,v_j)$ as well. This incident is used in DOT to get the possibility of forcing continuous solutions of the control functions. The required condition which has to be set on each interior node is

$$\varphi_{j-1}(T_j,v_{j-1})-\varphi_j(T_j,v_j)=0.$$

5.3 Finite dimensional formulation

The objective function F in (5.2) is given in a so-called *Bolza problem* formulation

$$F(y,u,p):=\Phi(y(T_f))+\int_{T_0}^{T_f} f_0(y(t),u(t),p)dt$$

with the Mayer-Term Φ and Lagrange-Term f_0. There are several other formulations possible in which a Bolza problem can be transformed, e.g.

$$\begin{aligned} \textit{Lagrange problem} \qquad & F(y,u,p) = \int_{T_0}^{T_f} f_0(y(t),u(t),p)dt \\ \textit{Mayer problem} \qquad & F(y,u,p) = \Phi(y(T_f)). \end{aligned}$$

In the implementation of DOT the latter formulation is used. The transformation from a Bolza problem to a Mayer problem works in the following way. Starting with a vector $y = [y_1(t), \dots, y_n(t)]^\top$ we introduce a new state variable

$$y_{n+1}(t) := \int_{T_0}^{T_f} f_0(y(t),u(t),p)dt$$

and thus y_{n+1} satisfies the IVP

$$\dot{y}_{n+1}(t) := f_0(y(t),u(t),p), \qquad y_{n+1}(T_0) = 0.$$

We obtain a $n+1$ dimensional Mayer problem formulation with

$$F(y,u,p) = \Phi(y(T_f)) + y_{n+1}(T_f).$$

Together with the explanations given in the previous sections, we achieve a finite dimensional nonlinear optimization problem out of the infinite dimensional optimal control problem (5.2). The entire formulation as implemented in DOT is given by

$$\boxed{\begin{aligned} \min\ & F(y,p) \\ \text{s.t. } & \dot{y}(t) = f(y(t),u(t),p) \\ & 0 = g_{ms}(y(t),u(t),p) && t \in \{T_1, T_2, \dots, T_M\} \\ & g_{\text{beg}}^L \le g_{\text{beg}}(y(T_0),u(T_0),p) \le g_{\text{beg}}^U \\ & g_{\text{int}}^L \le g_{\text{int}}(y(t),u(t),p) \le g_{\text{int}}^U && t \in \{T_1, T_2, \dots, T_{M-1}\} \\ & g_{\text{end}}^L \le g_{\text{end}}(y(T_M),u(T_M),p) \le g_{\text{end}}^U \\ & g_{\text{time}}^L \le g_{\text{time}}(p) \le g_{\text{time}}^U \end{aligned}} \tag{5.12}$$

CHAPTER 6

Application on problems with linear control

In this chapter we focus on problems with control function u occurring in linear form. First, we want to adapt and extend the definitions given in section 5.1 which makes it possible to state sufficient conditions for optimality (and corresponding assumptions). Afterwards we present four examples which have been solved with DOT.

6.1 Problems with linear control

By using the Mayer problem formulation (as in section 5.3) we state the optimal control problem (5.2) as

$$\begin{aligned} \min \ F(y,u) &= \Phi(y(T_f)) \\ \text{subject to} \quad \dot{y}(t) &= f_1(y(t),t) + f_2(y(t),t)u(t) \qquad t \in [T_0, T_f] \\ y(T_0) &= y_0 \\ \psi(y(T_f)) &= 0 \\ u(t) &\in U \end{aligned} \tag{6.1}$$

with twice continuous differentiable functions $f_1 : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^n$, $f_2 : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^{n\times m}, \Phi : \mathbb{R}^n \longrightarrow \mathbb{R}$ and $\psi : \mathbb{R}^n \longrightarrow \mathbb{R}^r$ with $0 \leq r \leq n$. The admissible control set $U \subset \mathbb{R}^m$ is supposed to be the cube

$$U := \{u \in \mathbb{R}^m \mid u_i^{\min} \leq u_i \leq u_i^{\max}, i = 1, \ldots, m\}.$$

The Hamiltonian function (5.3) corresponding to (6.1) is

$$\mathcal{H}(y,u,\lambda,t) := \lambda^\top \Big(f_1(y,t) + f_2(y,t)u(t) \Big).$$

Definition 6.1 (Switching function)
For an optimal control problem as in (6.1)*, the* ***switching function*** *is defined as*

$$\sigma(y,\lambda,t) := \mathcal{H}_u(y,u,\lambda,t) = \lambda^\top f_2(y,t) \in \mathbb{R}^{1\times m}.$$

We will shorten the notation via

$$\sigma(t) := \sigma(y(t),\lambda(t),t) = (\sigma_1(t),\dots,\sigma_m(t)).$$

The minimum condition in theorem 5.4 can be stated as

$$\sigma(t)u^*(t) := \min_{u\in U} \sigma(t)u \quad \forall\, t \in [T_0,T_f]$$

and allows to determine the structure of the optimal control function directly through the values of the switching function. We get

$$u_i^*(t) = \begin{cases} u_i^{\min} & \text{if } \sigma_i(t) > 0 \\ \text{arbitrary in } U & \text{if } \sigma_i(t) = 0 \\ u_i^{\max} & \text{if } \sigma_i(t) < 0 \end{cases}$$

as result.

Definition 6.2 (Bang-bang and singular control)
Let $[T_a,T_b] \subset [T_0,T_f]$ *and* $T_b > T_a$

1. $u_i(t)$ *is called* ***bang-bang*** *in* $[T_a,T_b]$*, if* $\sigma_i(t)$ *only has isolated zeros for* $t \in [T_a,T_b]$*. The zeros of* $\sigma_i(t)$ *are called* ***switching points*** *and it holds* $u_i(t) \in \{u_i^{\min}, u_i^{\max}\}$.
2. $u_i(t)$ *is called* ***singular*** *in* $[T_a,T_b]$*, if it holds* $\sigma_i(t) \equiv 0$ *for* $t \in [T_a,T_b]$.

Numerical methods yield solutions which satisfy first order necessary conditions, but to verify optimality there is a need of sufficient conditions. In the following, we present second order sufficient conditions for bang-bang controls. These results refer to the work of by Maurer et al. [22]. We have to start with two assumptions.

Assumption 6.3
Let $(\bar{y},\bar{u})$ *be an admissible pair, i.e. the conditions of* (6.1) *are fulfilled. We assume* $\bar{T} = \{\bar{T}_1, \bar{T}_2, \dots, \bar{T}_{M-1}\}$ *to be a finite set of switching points with*

(i) $T_0 =: \bar{T}_0 < \bar{T}_1 < \bar{T}_2 < \cdots < \bar{T}_{M-1} < \bar{T}_M := T_f$,

(ii) *$\bar{u}$ is continuous on $[T_0, T_f] \backslash \bar{T}$ and $\bar{u}_i(t) \in \{u_i^{\min}, u_i^{\max}\}$ for $i = 1, \ldots, m$,*

(iii) *an uniquely determined index $j = j(k)$ such that $\sigma_{j(k)}(\bar{T}_k) = 0$ and $\sigma_i(\bar{T}_k) \neq 0$ with $i \in \{1, \ldots, m\} \setminus \{j(k)\}$ for every $k = 1, \ldots, M-1$.*

Assumption 6.3 allows to state the control $\bar{u}$ as

$$\bar{u}(t) \equiv \bar{u}^k \quad \text{for } \bar{T}_{k-1} < t < \bar{T}_k \tag{6.2}$$

with vectors $\bar{u}^k \in \mathbb{R}^m$ and $k = 1, \ldots, M$.
The second assumption we need is known as strict bang-bang Legendre condition or strict bang-bang property (cf. [22]).

Assumption 6.4
We suppose that it holds

$$-\dot{\sigma}_j(\bar{T}_k)(\bar{u}_j^{k+1} - \bar{u}_j^k) > 0$$

for $k = 1, \ldots, M-1$ and $j = j(k)$ is the unique index in assumption 6.3(iii).

We introduce the vector

$$x := (T_1, T_2, \ldots, T_{M-1}, T_M) \in \mathbb{R}^M. \tag{6.3}$$

For any x we denote the corresponding bang-bang control by $u(t, x)$ with

$$u(t, x) \equiv u^k \quad \text{for } T_{k-1} < t < T_k,\ k = 1, \ldots, M$$

and $u^k \in \mathbb{R}^m$. Consequently, we can consider the differential equation

$$\begin{aligned} \dot{y}(t) &= f_1(y(t), t) + f_2(y(t), t)u^k \quad \text{for } T_{k-1} < t < T_k \\ y(T_0) &= y_0 \end{aligned}$$

and to simplify the notation we will denote the absolutely continuous solution of this initial value problem by

$$y(t, x; y_0) := y(t, T_1, \ldots, T_{M-1}; y_0).$$

We formulate the switching point optimization problem

$$\begin{aligned} \min_{x \in \mathbb{R}^M} \quad & G(x) := \Phi(y(T_f, x; y_0)) \\ \text{subject to} \quad & \Psi(x) := \psi(y(T_f, x; y_0)) = 0 \end{aligned} \tag{6.4}$$

with the Lagrange function

$$\mathcal{L}(x, \lambda) := G(x) + \lambda^\top \Psi(x)$$

and $\lambda \in \mathbb{R}^r$.

Theorem 6.5 (Second order sufficient conditions)
Let $(\bar{y}, \bar{u})$ be an admissible pair for (6.1) *and $\bar{u}$ is a bang-bang control which fulfills assumptions 6.3 and 6.4. The pair $(\bar{y}, \bar{u})$ is a strict strong minimum, if it holds*

(i) $\mathcal{L}_x(x, \lambda) = G_x(x) + \lambda^\top \Psi_x(x) = 0$

(ii) $\operatorname{rank}\Psi_x(x) = r$

and for all $v \in \mathbb{R}^M \setminus \{0\}$ with $\Psi_x(x)v = 0$ it is

(iii) $v^\top \mathcal{L}_{xx}(x, \lambda)v > 0$

for the corresponding optimization vector x as in (6.3) *and the resulting formulation* (6.4).

Proof: We refer to [22]. □

This formulation of second order sufficient conditions (SSC) is not yet suitable for numerical verification. For instance in DOT, we consider the length of an interval between two points (recall (5.4)) as optimization variable. Thus we define

$$\xi_j := T_j - T_{j-1} \quad \text{for } j = 1, \dots, M$$

and use the following optimization vector

$$\tilde{x} := (\xi_1, \xi_2, \dots, \xi_M)$$

which relation to x can be given by linear transformations $\tilde{x} = Qx$ and $x = Q^{-1}\tilde{x}$ with

$$Q = \begin{pmatrix} 1 & 0 & \dots & 0 \\ -1 & 1 & \ddots & 0 \\ & \ddots & \ddots & \vdots \\ 0 & & -1 & 1 \end{pmatrix} \text{ and } Q^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 1 & \dots & 1 & 1 \end{pmatrix}.$$

Hence (6.4) is equivalent to

$$\begin{aligned} \min_{\tilde{x} \in \mathbb{R}^M} \quad & \tilde{G}(\tilde{x}) := \Phi(y(T_f, \tilde{x}; y_0)) \\ \text{subject to} \quad & \tilde{\Psi}(\tilde{x}) := \psi(y(T_f, \tilde{x}; y_0)) = 0 \end{aligned} \tag{6.5}$$

with $T_f = \sum_{k=1}^{M} \xi_k$ and the Lagrange function is given by

$$\tilde{\mathcal{L}}(\tilde{x}, \lambda) := \tilde{G}(\tilde{x}) + \lambda^\top \tilde{\Psi}(\tilde{x}).$$

Using this notation a condition equivalent to 6.5(iii) can be given by

$$N^* \tilde{\mathcal{L}}_{\tilde{x}\tilde{x}}(\tilde{x}, \lambda) N > 0 \tag{6.6}$$

with N being the matrix which columns span the kernel of $\tilde{\Psi}_{\tilde{x}}$ [22].

6.2 Applications

In the following sections, we present four examples which have been solved with DOT. We start with two standard examples in context of optimal control which serve as performance tests. Subsequently we solve problems which are more sophisticated. Section 6.2.3 treats a model for circadian oscillations and in section 6.2.4 we show calculations for a calcium oscillation process. The plots illustrated here have been generated with MATLAB®.

6.2.1 Time optimal car

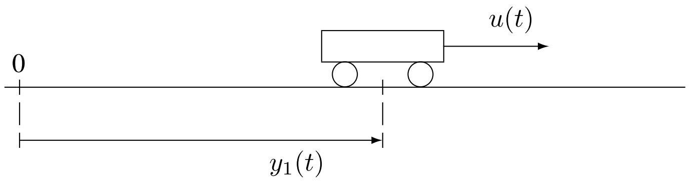

Figure 6.1: Model for the movement of a car

We describe the state of a car by

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$$

where $y_1(t)$ is the location and $y_2(t)$ is the velocity of the car at a certain time point t. The control function in this system concerns the acceleration of the car (see figure 6.1). We neglect incidents like friction and formulate the system dynamics as

$$\begin{aligned} \dot{y}_1(t) &= y_2(t) \\ \dot{y}_2(t) &= u(t) \end{aligned}$$

by following the laws of classical mechanics. The optimization criterion is stated as

$$\min \; T_f$$

with the free end time T_f.

The boundary conditions are

$$y(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad y(T_f) = \begin{pmatrix} 300 \\ 0 \end{pmatrix}$$

and the restriction of the control function is given via $u \in [-2, 1]$.
Hence the problem formulation as in (6.1) is

$$\begin{aligned} & \min T_f \\ \text{s.t.} \quad & \dot{y}(t) = \begin{pmatrix} y_2(t) \\ u(t) \end{pmatrix} \qquad t \in [0, T_f] \\ & y(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \psi(y(T_f)) = & \begin{pmatrix} 300 - y_1(T_f) \\ y_2(T_f) \end{pmatrix} \\ & u(t) \in [-2, 1] \end{aligned}$$

See figures 6.2 and 6.3 for the computed solutions and table 6.1 for informations about the numerical settings and results.

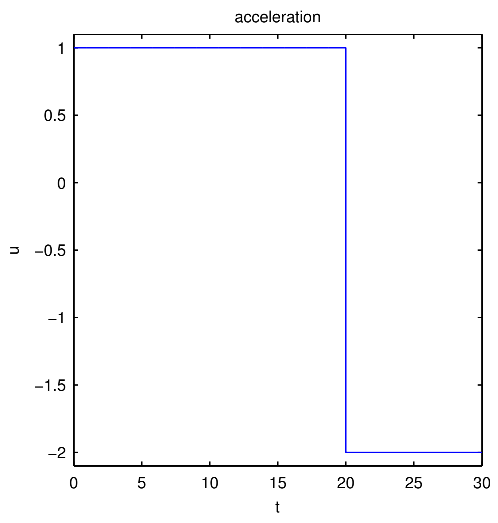

Figure 6.2: Bang-bang control for example 6.2.1

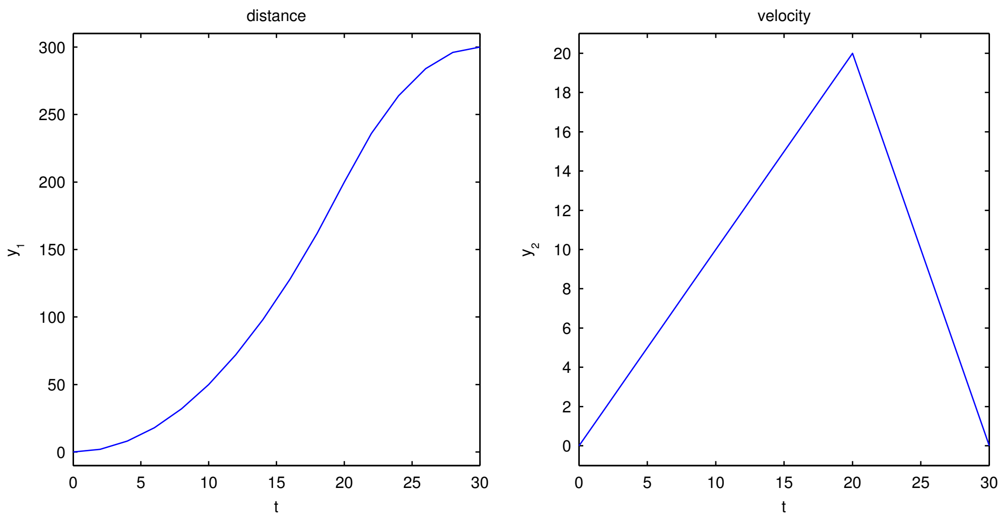

Figure 6.3: System state trajectories for example 6.2.1

Number of multiple shooting nodes	16
Control parametrization (piecewise, see section 5.2.2)	constant
Tolerance integrator (TOL in section 3.3.2)	10^{-8}
Number of iterations needed by IPOPT (section 4.2)	41
Computing time	0.552 s

Table 6.1: Numerical settings and results for example 6.2.1

6.2.2 Energy optimal car

We consider the same model as in 6.2.1 but this time we fix the time scale on 40 time units and use the optimization criterion

$$\min \int_0^{40} u(\tau)^2 d\tau$$

which is a physical interpretation of the energy which is spent in this system. We consider the boundary conditions

$$y(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad y(40) = \begin{pmatrix} 300 \\ 0 \end{pmatrix} \qquad u(t) \in [-3, 3].$$

This yields the problem formulation

$$\begin{aligned} &\min \int_0^{40} u(\tau)^2 d\tau \\ \text{s.t.} \quad \dot{y}(t) &= \begin{pmatrix} y_2(t) \\ u(t) \end{pmatrix} \qquad t \in [0, 40] \\ y(0) &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \psi(y(40)) &= \begin{pmatrix} 300 - y_1(40) \\ y_2(40) \end{pmatrix} \\ u(t) &\in [-3, 3] \end{aligned}$$

Results are shown in figures 6.4 and 6.5 and in table 6.2.

Number of multiple shooting nodes	21
Control parametrization (piecewise, see section 5.2.2)	linear
Tolerance integrator (TOL in section 3.3.2)	10^{-8}
Number of iterations needed by IPOPT (section 4.2)	21
Computing time	0.716 s

Table 6.2: Numerical settings and results for example 6.2.2

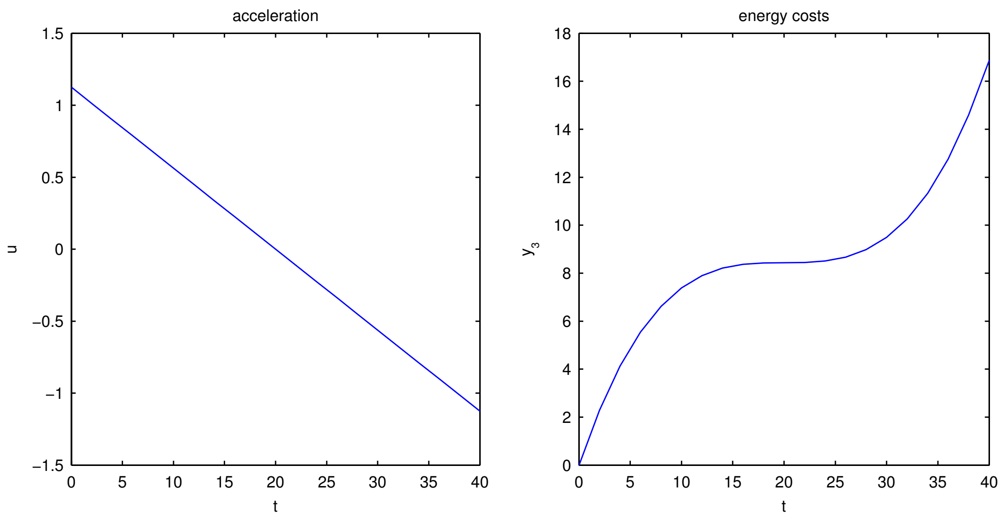

Figure 6.4: Control and cost function for example 6.2.2

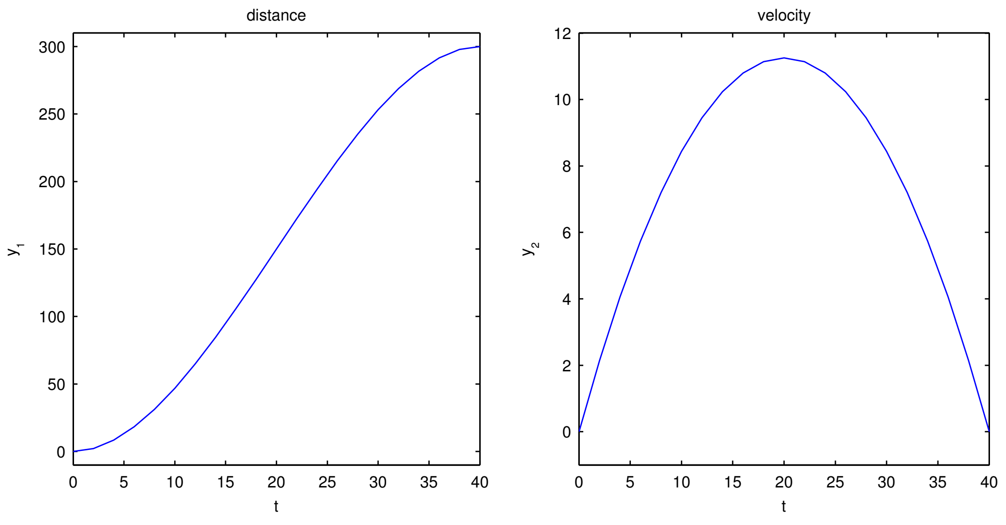

Figure 6.5: System state trajectories for example 6.2.2

6.2.3 Circadian rhythm

The following optimal control problem is taken from references [21, 26]. It describes a circadian rhythm which is based on negative auto-regulation of gene expression of the two proteins *per* and *tim*. Figure 6.6 is taken from [26] and gives an overview of this process.

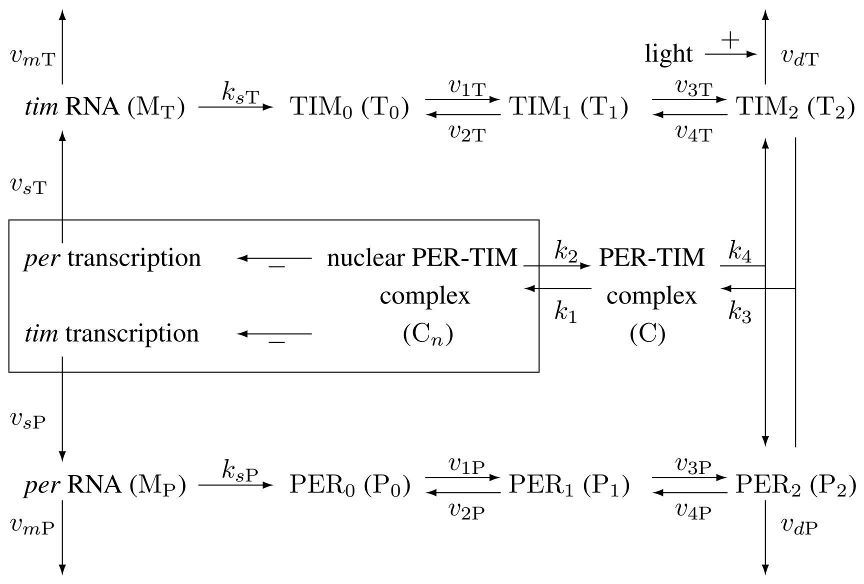

Figure 6.6: Model for the circadian oscillator

per and *tim* mRNAs are produced in the cell nucleus and transported into the cytoplasm. After being translated into PER and TIM proteins and multiple phosphorylation both proteins are able to form a complex that eventually moves back into the nucleus. This can lead to a damped expression of *per* and *tim* genes. This negative feedback is the basis of oscillations in the concentration of the involved molecules. Since light is supposed to increase the degradation of TIM_2 the molecular clock is coupled to the light-dark-cycle of the environment.

The Leloup-Goldbeter model consists of 10 ordinary differential equations involving 38 parameters. We list the differential equations and the parameters at the end of this subsection. We chose parameters according to [21] which yields self-sustained oscillations with a period close to 24 h for all involved molecules.

The considered problem is the following. We suppose a 12 h phase shift and we want to identify the optimal strength and timing of light stimuli to return to the origin phase (zero phase). According to [16] the zero phase is defined as the curve where the minimum in *per* mRNA occurs after 12 h.

The corresponding initial values are

$$\begin{array}{llll}
y_1(0) = 1.5587 & y_6(0) = 0.4477 & y_{11}(0) = 0.032542 & y_{16}(0) = 0.013224 \\
y_2(0) = 0.4474 & y_7(0) = 0.3957 & y_{12}(0) = 0.013224 & y_{17}(0) = 0.021253 \\
y_3(0) = 0.3936 & y_8(0) = 0.2728 & y_{13}(0) = 0.021253 & y_{18}(0) = 0.035094 \\
y_4(0) = 0.2545 & y_9(0) = 0.1429 & y_{14}(0) = 0.035094 & y_{19}(0) = 0.35095 \\
y_5(0) = 1.5587 & y_{10}(0) = 0.5595 & y_{15}(0) = 0.032542 & y_{20}(0) = 1.7829
\end{array}$$

where $y_1, \dots, y_{10}$ are the values for the zero phase, i.e. the uncontrolled system, and $y_{11}, \dots, y_{20}$ are the state variables which are influenced by the control function (see table 6.3). We define $y_0 := (y_1(0), \dots, y_{20}(0))^\top$.
We refer to the angle criteria as introduced in [26] to express the phase shift between the zero phase and the shifted one as a function of the state variables. For

$$x_j := \begin{pmatrix} y_j \\ \dot{y}_j \end{pmatrix}$$

the angles can be computed as

$$\alpha^{(j)}(t) := \arccos \frac{\langle x_j, x_{j+10} \rangle_2}{\|x_j\|_2 \|x_{j+10}\|_2}$$

for $j = 1, \dots, 10$. This leads to the optimization criterion

$$\min \int_0^T \|\alpha(t)\|_2 \mathrm{d}t$$

with $\alpha = (\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(10)})$ being the vector that holds all angles $\alpha^{(j)}$. We set $T = 72$ and restrict the control function by $u \in [1, 3]$. The entire formulation is

$$\begin{aligned}
&\min \int_0^{72} \|\alpha(t)\|_2 \mathrm{d}t \\
\text{s.t.} \quad &\dot{y}(t) = \begin{pmatrix} f_z(y(t)) \\ f_s(y(t), u(t)) \end{pmatrix} \qquad t \in [0, 72] \\
&y(0) = y_0 \\
&u(t) \in [1, 3]
\end{aligned} \tag{6.7}$$

where $f_z(y(t))$ stands for the uncontrolled system and $f_s(y(t), u(t))$ for the system influenced by the control function (see table 6.3). Results are depicted in figure 6.7 and table 6.4.

$$\begin{aligned}
\dot{y}_1 &= v_{sP}\frac{K_{IP}^n}{K_{IP}^n + y_{10}^n} - v_{mP}\frac{y_1}{K_{mP} + y_1} - k_d y_1\\
\dot{y}_2 &= k_{sP} y_1 - v_{1P}\frac{y_2}{K_{1P} + y_2} + v_{2P}\frac{y_3}{K_{2P} + y_3} - k_d y_2\\
\dot{y}_3 &= v_{1P}\frac{y_2}{K_{1P} + y_2} - v_{2P}\frac{y_3}{K_{2P} + y_3} - v_{3P}\frac{y_3}{K_{3P} + y_3} + v_{4P}\frac{y_4}{K_{4P} + y_4} - k_d y_3\\
\dot{y}_4 &= v_{3P}\frac{y_3}{K_{3P} + y_3} - v_{4P}\frac{y_4}{K_{4P} + y_4} - k_3 y_4 y_8 + k_4 y_9 - v_{dP}\frac{y_4}{K_{dP} + y_4} - k_d y_4\\
\dot{y}_5 &= V_{sT}\frac{K_{IT}^n}{K_{IT}^n + y_{10}^n} - v_{mT}\frac{y_2}{K_{mT} + y_5} - k_d y_5\\
\dot{y}_6 &= k_{sT} y_5 - v_{1T}\frac{y_6}{K_{1T} + y_6} + v_{2T}\frac{y_7}{K_{2T} + y_7} - k_d y_6\\
\dot{y}_7 &= v_{1T}\frac{y_6}{K_{1T} + y_6} - v_{2T}\frac{y_7}{K_{2T} + y_7} - v_{3T}\frac{y_7}{K_{3T} + y_7} + v_{4T}\frac{y_8}{K_{4T} + y_8} - k_d y_7\\
\dot{y}_8 &= v_{3T}\frac{y_7}{K_{3T} + y_7} - v_{4T}\frac{y_8}{K_{4T} + y_8} - k_3 y_4 y_8 + k_4 y_9 - u v_{dT}\frac{y_8}{K_{dT} + y_8} - k_d y_8\\
\dot{y}_9 &= k_3 y_4 y_8 - k_4 y_9 - k_1 y_9 + k_2 y_{10} - k_{dC} y_9\\
\dot{y}_{10} &= k_1 y_9 - k_2 y_{10} - k_{dN} y_{10}
\end{aligned}$$

Table 6.3: Differential equations for the Leloup-Goldbeter model. For $u \equiv 1$ this describes the uncontrolled case. The corresponding parameters can be found in table 6.5.

Number of multiple shooting nodes	25
Control parametrization (piecewise, see section 5.2.2)	constant
Tolerance integrator (TOL in section 3.3.2)	10^{-7}
Number of iterations needed by IPOPT (section 4.2)	104
Computing time	166.774 s

Table 6.4: Numerical settings and results for example 6.2.3

parameter	process	nominal value $\hat{p_d}$
n	transcriptional repression	4
K_{IP}	PER auto-inhibition	1 nM
K_{IT}	TIM auto-inhibition	1 nM
v_{1P}	PER phosphorylation I	8 nMh^{-1}
v_{1T}	TIM phosphorylation I	8 nMh^{-1}
v_{2P}	PER dephosphorylation I	1 nMh^{-1}
v_{2T}	TIM dephosphorylation I	1 nMh^{-1}
v_{3P}	PER phosphorylation II	8 nMh^{-1}
v_{3T}	TIM phosphorylation II	8 nMh^{-1}
v_{4P}	PER dephosphorylation II	1 nMh^{-1}
v_{4T}	TIM dephosphorylation II	1 nMh^{-1}
K_{1P}	PER phosphorylation I	2 nM
K_{1T}	TIM phosphorylation I	2 nM
K_{2P}	PER dephosphorylation I	2 nM
K_{2T}	TIM dephosphorylation I	2 nM
K_{3P}	PER phosphorylation II	2 nM
K_{3T}	TIM phosphorylation II	2 nM
K_{4P}	PER dephosphorylation II	2 nM
K_{4T}	TIM dephosphorylation II	2 nM
v_{sP}	PER transcription	1 nMh^{-1}
v_{sT}	TIM transcription	1 nMh^{-1}
v_{mP}	per mRNA degradation	0.7 nMh^{-1}
v_{mT}	per mRNA degradation	0.7 nMh^{-1}
v_{dP}	PER II degradation	2 nMh^{-1}
v_{dT}	TIM II degradation	2 nMh^{-1}
K_{mP}	PER mRNA degradation	0.2 nM
K_{mT}	TIM mRNA degradation	0.2 nM
K_{dP}	PER II degradation	0.2 nM
K_{dT}	TIM II degradation	0.2 nM
k_{sP}	PER translation	0.9 h^{-1}
k_{sT}	TIM translation	0.9 h^{-1}
k_1	transport cytoplasm $\rightarrow$ nucleus	0.6 h^{-1}
k_2	transport nucleus $\rightarrow$ cytoplasm	0.2 h^{-1}
k_3	PER-TIM association	1.2 nM^{-1}h^{-1}
k_4	complex dissociation	0.6 h^{-1}
k_d	unspecific degradation	0.01 h^{-1}
k_{dC}	nucleus unspecific degradation	0.01 h^{-1}
k_{dN}	cytoplasm unspecific degradation	0.01 h^{-1}

Table 6.5: Parameters for the Leloup-Goldbeter model

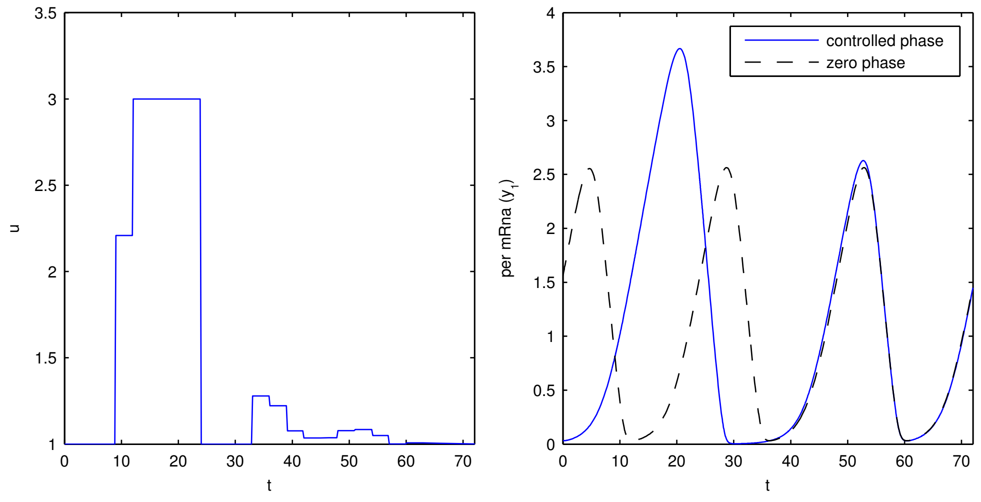

Figure 6.7: Computed control function and a state trajectory for the problem (6.7)

6.2.4 Calcium oscillator

The calcium oscillator model is taken from reference [20]. Here, a biochemical process of intracellular calcium spiking in hepatocytes induced by an extracellular increase in adenosine triphosphate (ATP) concentration is described. As external control an inhibitor $u(t)$ of PLC activation by the G protein is considered. For detailed information about the model from a biochemical point of view see [18].
The mathematical description leads to a nonlinear system of ordinary differential equations with 4 state variables

$$\begin{aligned}
\dot{y}_1 &= k_1 + k_2 y_1 - \frac{k_3 y_1 y_2}{y_1 + k_4} - \frac{k_5 y_1 y_3}{y_1 + k_6} \\
\dot{y}_2 &= (1-u) k_7 y_1 - \frac{k_8 y_2}{y_2 + k_9} \\
\dot{y}_3 &= \frac{k_{10} y_2 y_3 y_4}{y_4 + k_{11}} + k_{12} y_2 + k_{13} y_1 - \frac{k_{14} y_3}{y_3 + k_{15}} - \frac{k_{16} y_3}{y_3 + k_{17}} + \frac{y_4}{10} \\
\dot{y}_4 &= -\frac{k_{10} y_2 y_3 y_4}{y_4 + k_{11}} + \frac{k_{16} y_3}{y_3 + k_{17}} - \frac{y_4}{10}
\end{aligned} \tag{6.8}$$

with the control function $u \in [0,1]$ and the following parameter values
$k_1 = 0.09,\ k_2 = 2.30066,\ k_3 = 0.64,\ k_4 = 0.19,\ k_5 = 4.88,\ k_6 = 1.18,$
$k_7 = 2.08,\ k_8 = 32.24,\ k_9 = 29.09,\ k_{10} = 5.0,\ k_{11} = 2.67,\ k_{12} = 0.7,$
$k_{13} = 13.58,\ k_{14} = 153.0,\ k_{15} = 0.16,\ k_{16} = 4.85$ and $k_{17} = 0.05$.

To demonstrate the bursting-type oscillations in this model, we show in figure 6.8 the trajectory of variable y_3 in the uncontrolled case, i.e. with $u \equiv 0$.

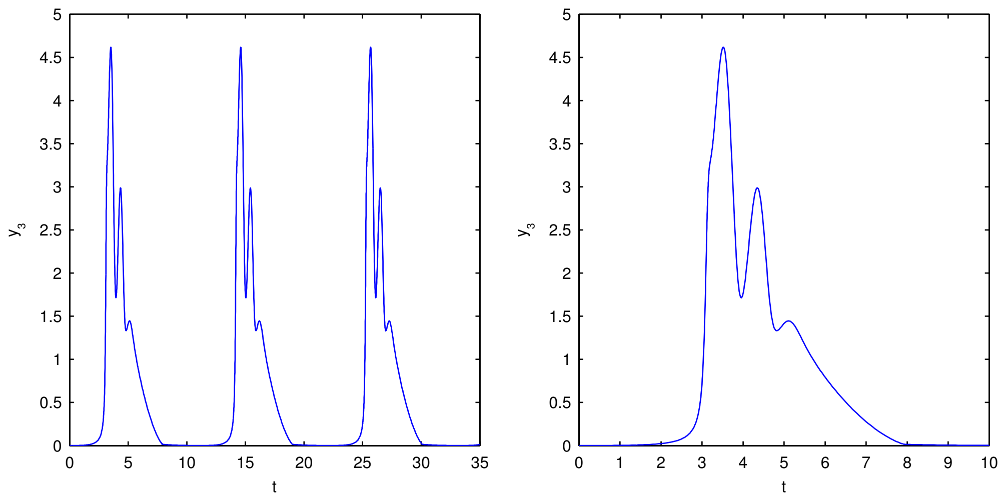

Figure 6.8: Bursting-type oscillations of species y_3 with $u \equiv 0$

As in [20] we consider initial values $y_1(0) = 0.039657$, $y_2(0) = 1.097992$, $y_3(0) = 0.001416$ and $y_4(0) = 1.65431$ and define $y_0 := (y_1(0), \ldots, y_4(0))^\top$. The aim of our approach is to compute the optimal control which interrupts the oscillation and leads to a steady state y^s. As steady state values we obtained

$$\begin{aligned} y_1^s &= 6.786776476776 \\ y_2^s &= 22.65835682778 \\ y_3^s &= 0.384305710319 \\ y_4^s &= 0.289777369811 \end{aligned}$$

by setting $\dot{y}_i = 0$ in (6.8) and solving the resulting nonlinear equation system with MATLAB®. We define $y^s := (y_1^s, \ldots, y_4^s)^\top$. Model (6.8) is used in two different optimization problems, a least-square and a time optimal formulation. For the first one we formulate the optimization criterion

$$\min \int_0^T \sum_{i=1}^{4} (y_i(\tau) - y_i^s)^2 d\tau$$

and set $T = 15$.

Hence the least-square problem formulation is given as

$$\begin{aligned} &\min \int_0^{15} \sum_{i=1}^{4} (y_i(\tau) - y_i^s)^2 d\tau \\ \text{s.t.} \quad &\dot{y}(t) = f(y(t), u(t)) \qquad t \in [0, 15] \\ &y(0) = y_0 \\ &u(t) \in [0, 1] \end{aligned} \tag{6.9}$$

where $f(y(t), u(t))$ is defined as the nonlinear system of ordinary differential equations in (6.8). In figure 6.9 and table 6.6 the results are shown and we can observe that the steady state is unstable, since the oscillation starts again after the control is turned off ($t > 15$).

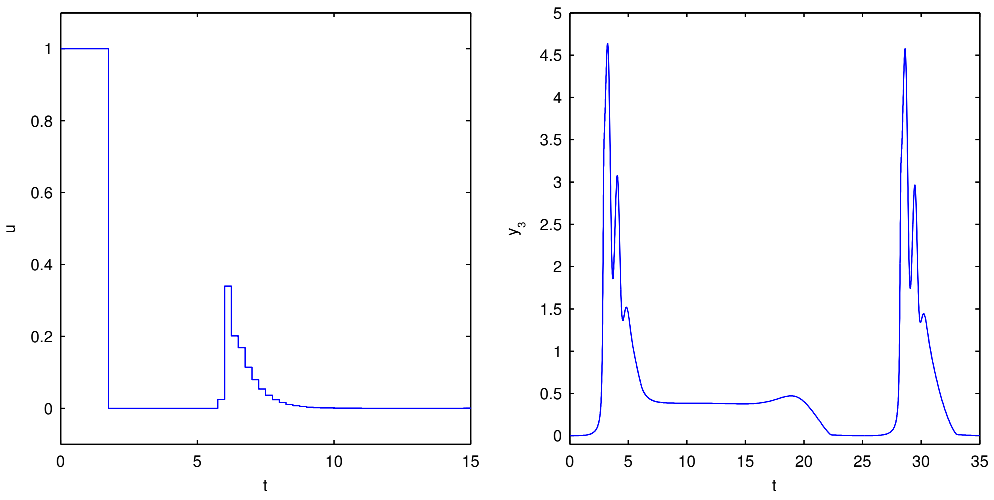

Figure 6.9: Control function and trajectory of y_3 for problem (6.9)

Number of multiple shooting nodes	61
Control parametrization (piecewise, see section 5.2.2)	constant
Tolerance integrator (TOL in section 3.3.2)	10^{-7}
Number of iterations needed by IPOPT (section 4.2)	110
Computing time	77.713 s

Table 6.6: Numerical settings and results for problem (6.9)

The time optimal formulation for this model is

$$
\begin{aligned}
& \min T \\
\text{s.t.} \quad & \dot{y}(t) = f(y(t), u(t)) \qquad t \in [0, T] \\
& y(0) = y_0 \\
& \psi(y(T)) = y^s - y(T) \\
& u(t) \in [0, 1]
\end{aligned}
\tag{6.10}
$$

for a free end time T and again $f(y(t), u(t))$ is defined as the nonlinear system of ordinary differential equations in (6.8). We forced an equidistant time grid (recall section 5.2.1). See table 6.7 and figure 6.10 for the results.

Number of multiple shooting nodes	201
Control parametrization (piecewise, see section 5.2.2)	constant
Tolerance integrator (TOL in section 3.3.2)	10^{-7}
Number of iterations needed by IPOPT (section 4.2)	245
Computing time	823.475 s

Table 6.7: Numerical settings and results for problem (6.10)

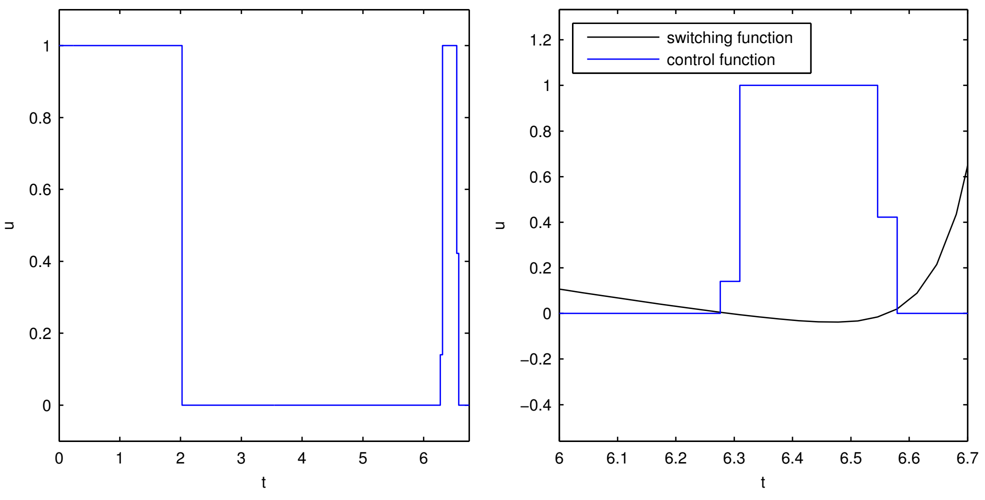

Figure 6.10: Computed control function for (6.10). On the right hand side it is scaled to the time range [6.0, 6.7] and we illustrate an approximation of the corresponding switching function.

The computed control function is almost of bang-bang type, but not exactly. The illustrated approximation of the switching function is obtained by the values of the adjoint variables on every node of the discretization. The isolated zeros of this function determine the switching points. With the discretization used here (i.e. 200 equidistant intervals), we can not hit those zeros exactly, since apparently at least one switching point lies inside an interval. We use the results of this formulation for our second attempt. We assume a discretization with 4 intervals of free length and fix the control function to be bang-bang. We adapt formulation (6.10) and solve

$$\begin{aligned}
&\min\ T_4 \\
\text{s.t.}\quad &\dot{y}(t) = f(y(t), u(t)) \qquad t \in [0, T_4] \\
&y(0) = y_0 \\
\psi(y(T_4)) = &\begin{pmatrix} y_1^s - y_1(T_4) \\ \left(y_2^s - y_2(T_4)\right)^2 + \left(y_3^s - y_3(T_4)\right)^2 \\ y_4^s - y_4(T_4) \end{pmatrix} \\
u(t) = &\begin{cases} 1 & 0 \le t < T_1 \\ 0 & T_1 \le t < T_2 \\ 1 & T_2 \le t < T_3 \\ 0 & T_3 \le t \le T_4 \end{cases}
\end{aligned} \tag{6.11}$$

which is a switching point optimization with the understanding of the end time T_4 as $T_4 = \Delta t_1 + \Delta t_2 + \Delta t_3 + \Delta t_4$, where $\Delta t_j := T_j - T_{j-1}$ for $j = 1, \ldots, 4$ are optimization variables (see section 5.2.1). We use initial values

$$\begin{aligned}
\Delta t_1 &= 2.02179177509 \\
\Delta t_2 &= 4.28284039624 \\
\Delta t_3 &= 0.25518524552 \\
\Delta t_4 &= 0.69521017694
\end{aligned}$$

and show results of the computation in table 6.8 and figure 6.11.
Formulation (6.11) is equivalent to the formulation in (6.5). Solving (6.11) with DOT includes solving the differential equation system on subintervals and consequently multiple shooting constraints (5.8) are added. Thus the Jacobian matrix A_{dot} and Hessian matrix H_{dot} we use (and obtain) in DOT are in fact of a higher dimension then $\tilde{\Psi}_{\tilde{x}}$ and $\tilde{\mathcal{L}}_{\tilde{x}\tilde{x}}$ as corresponding to (6.5). Since it is inconvenient to calculate $\tilde{\Psi}_{\tilde{x}}$ and $\tilde{\mathcal{L}}_{\tilde{x}\tilde{x}}$ out of A_{dot} and H_{dot}, we use directly A_{dot} and H_{dot} to verify optimality with the help of second order sufficient conditions.

Let N_{dot} denote the matrix which columns span the kernel of A_{dot}. This matrix can be obtained with MATLAB®. We get

$$N_{\text{dot}}^{*} H_{\text{dot}} N_{\text{dot}} = 0.02411965899 > 0$$

which guarantees the optimality of the calculated solution.

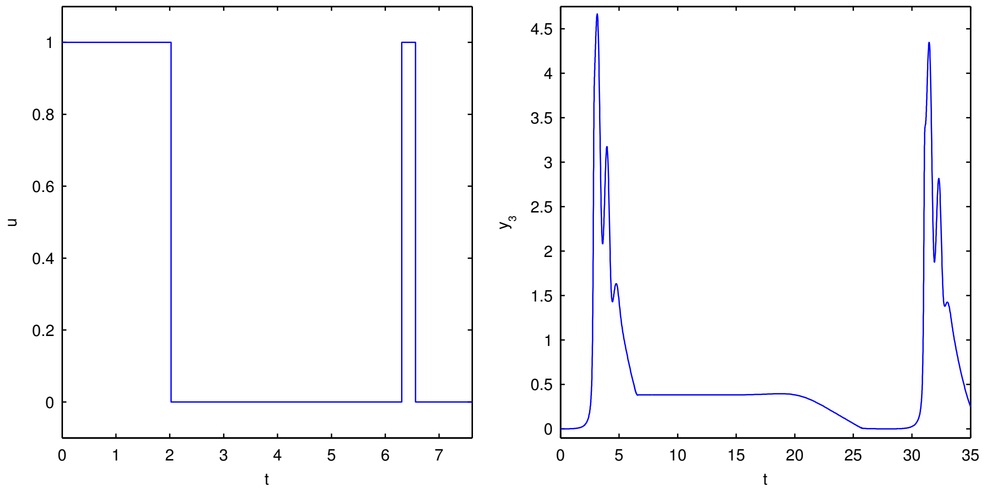

Figure 6.11: Results for the time optimal problem formulation (6.11)

Number of multiple shooting nodes	5
Control parametrization (piecewise, see section 5.2.2)	constant
Tolerance integrator (TOL in section 3.3.2)	10^{-7}
Number of iterations needed by IPOPT (section 4.2)	22
Computing time	9.505 s

Table 6.8: Numerical settings and results for problem (6.11)

Attempts with different initial values for Δt_i $(i = 1, \ldots, 4)$ showed that the system is highly sensitive, e.g. a difference of 10^{-5} in those values can cause a difference of 10^{-1} in the resulting value of the objective function for the solution. In fact, the algorithm could not yield a solution for all initial guesses or all different orders of integration tolerances. We suffer from numerical inaccuracies (e.g. error propagation) due to the coarse discretization grid on only 5 multiple shooting nodes. We tried to avoid these inaccuracies by inserting more nodes and reducing the maximum integration step size, but we encountered the limits of numerical capacities and ran out of memory. This model shows which significant difficulties can occur while tackling a nonlinear problem numerically.

CHAPTER 7

Summary and outlook

7.1 Summary

Growing interest in numerical methods to solve optimal control problems has been stimulated by the rising complexity of those problems. There is, for example, no reliability that we can find an analytical solution for general nonlinear problems. Since such problems occur in a wide field of applications, it is desirable to have optimization algorithms which can easily be adapted to the different types of problems. This adaption requires automatisms for different tasks presented in this thesis. We present numerical methods concerning differentiation (chapter 2), integration of ordinary differential equations (chapter 3) and optimization (chapter 4), always with a detailed demonstration of the specific methods used in DOT, the numerical tool for optimal control of ordinary differential equations developed in this thesis.

The automatic differentiation technique yields highly accurate values for the derivatives and we do not have to analytically differentiate any function appearing in the problem. The BDF algorithm, which is used to solve ordinary differential equations, is implicit and ideally suited for stiff problems, e.g. the calcium oscillation process we treat in chapter 6. The interior point optimization algorithm utilized in DOT guarantees (under certain assumptions) global convergence and is a very good alternative to the active set strategies. Furthermore, the IPOPT software package used here provides many options to adapt the optimization algorithm, e.g. it is possible to adjust the values of all constants we mention in chapter 4.

DOT (chapter 5) is a direct optimal control tool and its implementation is the result of this thesis. We present the main theoretical foundations and its practical implementation. The interface which is explained in the appendix provides an environment to formulate the problem. The applications (chapter 6) show that DOT can be successfully used to handle intricate models.

7.2 Outlook

Potential improvements and expansions for the DOT - code:

- The implemented BDF method is implicit thus it could be useful to additionally implement an explicit one (e.g. a Runge-Kutta method) to have an alternative integrator for handling non-stiff problems.
- The interior point algorithm yields solutions with satisfy the KKT conditions, but these are only necessary ones. To ensure the optimality of a solution it would be helpful to have an automatic verification of second order sufficiency conditions (see e.g. [22]).
- Up to now only problems in an autonomous formulation can be handled. Although it is possible to transform a non-autonomous problem into an autonomous one, this part definitely has the potential to be expanded.
- The interface of DOT could be more user-friendly, e.g. concerning problem input or solution output. Programming experience (in C++) is needed to formulate the problem, but it is conceivable to simplify this process with a basic input/output file transfer, where only the corresponding values have to be entered (instead of first declaring variables and so on).
- Last but not least there is always the ambition of speeding up an algorithm and since this is the first implementation of DOT, there are, most likely, issues which can be improved.

APPENDIX A

DOT user guide

A.1 Software packages

In DOT we use the following free available software:

- **IPOPT**: An interior point method written in C++, which fundamentals are outlined in section 4.2. To get IPOPT visit

  `https://projects.coin-or.org/Ipopt`

- **cppAD**: A C++ software package using automatic differentiation techniques which are presented in sections 2.2 and 2.3. This package can be downloaded from

  `http://www.coin-or.org/CppAD`

- **BDF integrator**: A variable step variable order backward differentiation formula as presented in section 3.3. It has been implemented by Dominik Skanda and is part of the DOT package.

A.2 The interface

The program is interfaced through the class `ProblemInterface` which provides methods to prepare and solve problems of the form (5.12). After creating an object of this class via

```
ProblemInterface MyProblem
```

the settings should be adapted by

```
MyProblem.option()
```

where `option()` is the method to call. In the following sections a list with all member methods of an interface is given. It is not necessary to call all of them at any time.

A.2.1 Basic info

```
setGeneralInfo(int numOfRHS, int numOfParameter, double relTol,
               double absTol, int maxsteps)
```

- `numOfRHS` number of right hand sides (or model stages) you want to handle in your problem
- `numOfParameter` number of all system parameters
- `relTol` relative tolerance for the integrator
- `abstol` absolute tolerance for the integrator
- `maxsteps` maximum number of steps for the integrator

The `setGeneralInfo` - option is the most important one and has to be called first, since otherwise declaration problems occur!

```
setRHS(int index, rhs_ptr rhs, obj_ptr obj, int numOfSpec,
       int numOfInt,int numOfCon)
```

- `index` index number of the right hand side you want to address
- `rhs` template function for the right hand side
- `obj` template function for the objective function

- `numOfSpec` number of variables
- `numOfInt` number of discretization intervals
- `numOfCon` number of control functions

This should be called in second place. The info given here is needed in further options. The data types `rhs_ptr` and `obj_ptr` are pointers for template functions, section A.3 shows how to define such a function.

```
setControlInfo(int index, int** orderOfControl)
```

- `index` index number of the right hand side you want to address
- `orderOfControl[i][j]` order of the `j`-th control function starting at node `i`

This option does not have to be called if there is no control function.

```
setNodesOfEval(int index, bool* evalObj)
```

- `index` index number of the right hand side you want to address
- `evalObj[i]` true/false whether the objective function should be evaluated on time node `i` or not

If this option is not called, the objective function will be evaluated only at the last node.

A.2.2 Constraints

```
setAdditionalConstraints(int index, constr_ptr conBeg,
                         int numOfConBeg, constr_ptr conInt,
                         int numOfConInt, constr_ptr conEnd,
                         int numOfConEnd)
```

- `index` index number of the right hand side you want to address
- `conBeg` template function for constraints at the beginning
- `numOfConBeg` number of constraints at the beginning
- `conInt` template function for constraints at the interior nodes
- `numOfConInt` number of constraints at the interior nodes

- `conEnd` template function for constraints at the end
- `numOfConEnd` number of constraints at the end

If there are no additional constraints at some points call this option with NULL and 0 at the corresponding place. The data type `constr_ptr` is again a function pointer, specific information can be found in section A.3.

`setTimeConstraint(time_ptr conTime, int numOfTimeConstraints)`

- `conTime` template function for the time constraints, e.g. forcing equidistant time grid
- `numOfTimeConstraints` how many constraints appear in `conTime`

This includes the time constraints for the whole system (even if there are several model stages). See section A.3 for more information about the general formulation of these template functions.

A.2.3 Initial values

`setInitialSpecies(int index, double** initialValue)`

- `index` index number of the right hand side you want to address
- `initialValue[i][j]` the initial values on node `i` of species `j`

These initial values can also be set by a first integration of the system. See section A.2.5 for further information.

`setInitialTimeGrid(int index, double* initialValue)`

- `index` index number of the right hand side you want to address
- `initialValue[i]` the initial length of time intervall `i`

`setInitialParameter(double* initialValue)`

- `initialValue[i]` the initial value of parameter `i`

`setInitialCoefficients(int index, double*** initialValue)`

- `index` index number of the right hand side you want to address
- `initialValue[i][j][k]` the value on node `i` of the `k`-th coefficient of control function `j`

A.2.4 Bounds

```
setBoundTimeGrid(int index,
        double* minBound, double* maxBound, bool* isFree)
```

- `index` index number of the right hand side you want to address
- `minBound[i]` lower bound of time interval `i`
- `maxBound[i]` upper bound of time interval `i`
- `isFree[i]` true/false whether the length of interval `i` is free or fix

```
setBoundSpecies(int index,
        double** minBound, double** maxBound)
```

- `index` index number of the right hand side you want to address
- `minBound[i][j]` lower bound on node `i` of species `j`
- `maxBound[i][j]` upper bound on node `i` of species `j`

```
setBoundParameter(double* minBound, double* maxBound,
        bool* isFree)
```

- `minBound[i]` lower bound of parameter `i`
- `maxBound[i]` upper bound of parameter `i`
- `isFree[i]` true/false whether parameter `i` is free or not

```
setBoundCoefficients(int index,
        double*** minBound, double*** maxBound)
```

- `index` index number of the right hand side you want to address
- `minBound[i][j][k]` lower bound on node `i` of the `k`-th coefficient of control function `j`
- `maxBound[i][j][k]` upper bound on node `i` of the `k`-th coefficient of control function `j`

```
setBoundAdditionalConstraints(int index,
        double* minBoundBeg, double* maxBoundBeg,
        double* minBoundInt, double* maxBoundInt,
        double* minBoundEnd, double* maxBoundEnd)
```

- `index` index number of the right hand side you want to address
- `minBoundBeg[i]` lower bound of the `i`-th constraint at the beginning
- `maxBoundBeg[i]` upper bound of the `i`-th constraint at the beginning
- `minBoundInt[i]` lower bound of the `i`-th constraint at interior nodes
- `maxBoundInt[i]` upper bound of the `i`-th constraint at interior nodes
- `minBoundEnd[i]` lower bound of the `i`-th constraint at the end
- `maxBoundEnd[i]` upper bound of the `i`-th constraint at the end

If this option is not called, by default the lower and upper bounds for the constraints will be zero.

A.2.5 Tackling the problem

When all settings of the problem are done, these commands can be used to start the optimization tool.

`initializeIntegrator()`

This option has to be called to initialize the integrator

`getInitialSolution(int index)`

Use this function to integrate right hand side `index` first, e.g. to get initial values on each node, which are automatically stored

`optimize()`

Solve the problem!

A.2.6 Output

If the optimizer has found a solution, use these options to get the corresponding solution values.

`getTimeGridSolution(int index, double* solutionVector)`

- `index` index number of the right hand side you want to address
- `solutionVector[i]` stores the length of intervall `i`

`getSpeciesSolution(int index, double** solutionVector)`

- `index` index number of the right hand side you want to address
- `solutionVector[i][j]` stores the solution on node `i` of species `j`

`getParameterSolution(double* solutionVector)`

- `solutionVector[i]` stores the solution parameter `i`

`getControlSolution(int index, double** solutionVector)`

- `index` index number of the right hand side you want to address
- `solutionVector[i][j]` stores the value of control function `j` starting at node `i`

`getControlCoefficients(int index, double*** solutionVector)`

- `index` index number of the right hand side you want to address
- `solutionVector[i][j][k]` stores the value of the `k`-th coefficient of control function `j` on the `i`-th node

A.2.7 Additional options

```
setErrorOfSensitivities(bool controlError,
        double absTolSens, double relTolSens)
```

- `controlError` true/false whether the step size control of the integrator should be used or not
- `absTolSens` value for the absolute tolerance
- `relTolSens` value for the relative tolerance

`setToleranceIPOPT(double tol)`

Set the IPOPT convergence tolerance to `tol`

`derivativeTest(int index)`

Use the IPOPT derivative check. Set `index`
0 for first-order
1 for only-second-order
2 for first and second order check

Additionally we can use the `ipopt.opt` file to adjust all IPOPT options we want to.

A.3 The templates

In DOT we use template functions to describe all appearing functions, e.g. system dynamics or constraint functions. Since there are specifications on each function, i.e. each template, we give here an overview together with few examples.

right hand side (rhs)

To state the differential equation system, we have to use the body

```
template<class T1,class T2,class T3,class T4,class T5>
static int rhs(T1 ydot,T2 y,T3 p,T4 u,T5 t)
{
 // implement the system dynamics here
}
```

with system variables `y`, `p` for the parameters and `u` for the control functions. Note that the option for non-autonomous systems is given, since there is a time vector `t`, but the implementation is not yet complete. By now we only solve autonomous systems.
Recall the energy-optimal car (EOCAR) example of section 6.2.2, the corresponding system dynamics are formulated in the following way:

```
template<class T1,class T2,class T3,class T4,class T5>
static int rhs(T1 ydot,T2 y,T3 p,T4 u,T5 t)
{
 // the dynamic of the system
 ydot[0]=y[1];
 ydot[1]=u[0];
 // the Lagrange term
 ydot[2]=u[0]*u[0];
 return(0);
}
```

Note, that the 3rd state (`ydot[2]` respectively `y[2]`) appearing here, is the result of the reformulation as a Mayer problem.

objective function

The body of the objective function is

```
template<class T1,class T2,class T3>
static int obj(T1 &value,T2 &y,T3 &p)
{
 // fill me
}
```

In `value` the value of the objective function is stored. The system variables are denoted by `y` and `p` still denotes the parameters.
For the EOCAR example the objective function is

```
template<class T1,class T2,class T3>
static int obj(T1 &value,T2 &y,T3 &p)
{
 value=y[2];
 return(0);
}
```

constraints

Generally constraints should be formulated in the following way (we show the special case with time constraints afterwards)

```
template<class T1,class T2,class T3,class T4,class T5,class T6>
static int constraint(T1 &g,T2 &y,T3 &p,T4 &uL,T5 &uR,T6 &t)
{
\\ unlimited freedom
}
```

where `g` stands for the constraint (whose bounds should be set accordingly). The control functions can be considered to end in this node (this is `uL`) or to start at this node (denoted by `uR`). This gives the option to enforce continuous control functions by formulating interior constraints

```
g[0]=uR[0]-uL[0]
```

and setting the corresponding lower and upper bounds zero.
According to the EOCAR example the constraints at the first node can be written as

```
template<class T1,class T2,class T3,class T4,class T5,class T6>
static int conBeg(T1 &g,T2 &y,T3 &p,T4 &uL,T5 &uR,T6 &t)
{
 g[0]=y[0];
 g[1]=y[1];
 g[2]=y[2];
 g[3]=uR[0];
 return(0);
}
```

where we have constraints for the species and for the control function. In this case the values for the species should be fixed (lower Bound = upper Bound) to fix the starting point. By setting the bounds of `g[3]` correctly we can ensure to keep the control function within a desired interval.
At the interior points, we only postulate continuous connecting conditions and of course the control function should still be within an interval.

```
template<class T1,class T2,class T3,class T4,class T5,class T6>
static int conInt(T1 &g,T2 &y,T3 &p,T4 &uL,T5 &uR,T6 &t)
{
 g[0]=uR[0]-uL[0];
 g[1]=uR[0];
 return(0);
}
```

with zero as upper and lower bound for `g[0]`.

time constraints

The body for time constraints is

```
template<class T1,class T2,class T3,class T4,class T5>
static int conTime(T1 &g,T2 &p,T3 **timeGrid,
                   T4 numOfInt,T5 numOfRHS)
{
\\ i feel kind of inner emptiness...somehow depressing..
}
```

with `timeGrid[i][j]` denoting the length of the `j`-th interval of the `i`-th right hand side.
Here, we present the possibility of forcing an equidistant time grid for one right hand side

```
template<class T1,class T2,class T3,class T4,class T5>
static int conTime(T1 &g,T2 &p,T3 **timeGrid,
                   T4 numOfInt,T5 numOfRHS)
{
 int i, count;
 count=0;
  for(i=0;i<numOfInt[0];i++)
   {
    g[count]=1./numOfInt[0]*p[0]-timeGrid[0][i];
    count++;
   }
 return(0);
}
```

where `p[0]` should be the parameter for the overall time.

Bibliography

[1] J. Albersmeyer and H. G. Bock, *Sensitivity generation in an adaptive BDF-method*, in Modeling, Simulation and Optimization of Complex Processes: Proceedings of the Third International Conference on High Performance Scientific Computing, Springer, 2008.

[2] U. Ascher and L. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*, SIAM, Philadelphia, 1998.

[3] H. G. Bock and K. J. Plitt, *A multiple shooting algorithm for direct solution of optimal control problems*, in Proceedings of the Ninth IFAC World Congress, Budapest, Pergamon, Oxford, 1984.

[4] R. H. Byrd, G. Liu, and J. Nocedal, *On the local behavior of an interior point method for nonlinear programming*, Addison Wesley Longman, 1997.

[5] G. D. Byrne and A. C. Hindmarsh, *A polyalgorithm for the numerical solution of ordinary differential equations*, ACM Transactions on Mathematical Software, 1 (1975), pp. 71–96.

[6] M. Calvo, J. I. Montijano, and L. Rández, *On the change of step size in multistep codes*, Numerical Algorithms, 4 (1993), pp. 283–304.

[7] B. Christianson, *Reverse accumulation and accurate rounding error erstimates for taylor series coefficient*, Optimization Methods and Software, 1 (1992), pp. 81–94.

[8] E. A. Coddington and N. Levinson, *Theory of ordinary differential equations*, McGraw-Hill, 1955.

[9] C. W. Cryer, *On the instability of high order backward-difference multistep methods*, BIT Numerical Mathematics, 12 (1972), pp. 17–25.

[10] G. Dahlquist, *Convergence and stability in numerical integration of ordinary differential equations*, Mathematica Scandinavica, 4 (1956), pp. 33–53.

[11] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Mathematical Programming, 91 (2002), pp. 239–269.

[12] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, no. 19 in Frontiers in Appl. Math., SIAM, Philadelphia, PA, 2000.

[13] R. Gunawan, M. Y. L. Jung, E. G. Seebauer, and R. D. Braatz, *Optimal control of rapid thermal annealing in a semiconductor process*, Journal of Process Control, 14 (2004), pp. 423 – 430.

[14] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, no. 8 in Springer Series in Computational Mathematics, Springer, Berlin, second ed., 2000.

[15] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2Ed. Springer-Verlag, 2002, 2002. Index.

[16] J. C. Hall and M. Rosbash, *Genes and biological rhythms*, Trends in Genetics, 3 (1987), pp. 185–191.

[17] H. Knobloch and F. Kappel, *Gewöhnliche Differentialgleichungen*, B.G. Teubner, 1974.

[18] U. Kummer, L. F. Olsen, C. J. Dixon, A. K. Green, E. Bornberg-bauer, and G. Baier, *Switching from simple to complex oscillations in calcium signaling*, Biophysical Journal, 79 (2000), pp. 1188–1195.

[19] D. Lebiedz, *Optimal control, model- and complexity-reduction of self-organized chemical and biochemical systems: A scientific computing approach.* Habilitation thesis, University of Heidelberg, 2006.

[20] D. Lebiedz, S. Sager, H. G. Bock, and P. Lebiedz, *Annihilation of limit cycle oscillations by identification of critical phase resetting stimuli via mixed-integer optimal control methods*, Physical Review Letters, 95 (2005), p. 108303.

[21] J. C. Leloup and A. Goldbeter, *A model for circadian rhythms in drosophila incorporating the formation of a complex between the per and tim proteins.*, Journal of Biological Rhythms, 13 (1998), pp. 70–87.

[22] H. Maurer, C. Büskens, J.-H. R. Kim, and C. Y. Kaya, *Optimization methods for the verification of second order sufficient conditions for bang-bang controls*, Optimal Control Applications and Methods, 26 (2005), pp. 129–156.

[23] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, second ed., 2006.

[24] A. Nordsieck, *On numerical integration of ordinary differential equations*, Mathematics of Computation, 16 (1962), pp. 22–49.

[25] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *L. S. Pontryagin Selected Works, Volume 4: The Mathematical Theory of Optimal Processes*, Gordon and Breach, Montreux, Switzerland, 1986.

[26] M. Rehberg and D. Lebiedz, *Phase tracking of circadian rhythm by model-based optimal control*, in Proceedings of the Third International Conference on Foundations of Systems Biology in Engineering (FOSBE 2009), 2009.

[27] N. Sadati and A. Babazadeh, *Optimal control of robot manipulators with a new two-level gradient-based approach*, Electrical Engineering (Archiv fur Elektrotechnik), 88 (2006), pp. 383–393.

[28] J. Stoer and R. Bulirsch, *Numerische Mathematik*, vol. 2, Springer, Berlin, fifth ed., 2005.

[29] A. Swierniak, U. Ledzewicz, and H. Schättler, *Optimal control for a class of compartmental models in cancer chemotherapy*, International Journal of Applied Mathematics and Computer Science, 13 (2003), pp. 357–368.

[30] A. Wächter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*, PhD thesis, Carnegie Mellon University, 2002.

[31] A. Wächter and L. T. Biegler, *Line search filter methods for nonlinear programming: Local convergence*, SIAM Journal on Optimization, 16 (2005), pp. 32–48.

[32] A. Wächter and L. T. Biegler, *Line search filter methods for nonlinear programming: Motivation and global convergence*, SIAM Journal on Optimization, 16 (2005), pp. 1–31.

[33] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106 (2006), pp. 25–57.

[34] O. B. Widlund, *A note on unconditionally stable linear multistep methods*, BIT Numerical Mathematics, 7 (1967), pp. 65–70.