CHALMERS

Evaluation of Optimization Solvers in Mathematica with focus on Optimal Control Problems

Master's Thesis in Engineering Mathematics and Computational Sciences

REBECKA NYLIN

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2013
Master's Thesis 2013

Evaluation of Optimization Solvers in Mathematica with focus on Optimal Control Problems

Rebecka Nylin

Department of Mathematical Science
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden

Abstract

This thesis involves optimization of optimal control problems using collocation methods in Wolfram SystemModeler and Mathematica. Three optimization solvers are evaluated and compared: FindMinimum in Mathematica, IPOPT and KNITRO. They are evaluated by letting them solve optimal control problems with use of the numerical method direct collocation. Two test problems are used; they represent a batch reactor and a free floating robot. An interface between IPOPT and Mathematica is created.

Direct collocation is based on discretizing the state and control variables of the optimal control problem with use of collocation points. In this thesis three numerical methods are used to approximate the derivatives of the state variables in the optimal control problem. The three methods are Euler method, backward differentiation method and the use of Lagrange basis polynomials. The results are obtained by investigating the optimal trajectories of the variables, the optimal cost function, the number of iterations and the total time spent during optimization.

The conclusion is that FindMinimum is not as good as the other two solvers. It performs worst in all studied aspects. The main reason is that it does not have a good algorithm to solve constrained optimization problems. IPOPT and KNITRO are equally good with respect to number of iterations and optimal results, but in the timing aspect is IPOPT slower because of slow function evaluations.

Acknowledgements

This thesis has been performed at Wolfram MathCore in Linköping. I would like to thank my supervisor Peter Aronsson and all colleagues at Wolfram MathCore.

I would also like to thank my supervisor and examiner Stig Larsson at the Department of Mathematical Sciences, Chalmers University.

Rebecka Nylin, Gothenburg June 7, 2013

Contents

1

Introduction

The background, purpose and problem formulation of the thesis is described in this section. The method is outlined and an explanation of the creation of an interface from Mathematica to the optimization solver IPOPT. The utilized software and different optimization solvers are also presented.

1.1 Background

Wolfram SystemModeler is a software for modeling and simulation which easily connects with Wolfram Mathematica for an integrated workflow. Wolfram SystemModeler is created by the company Wolfram MathCore. The company has a need to enlarge the software by integration of optimization in Wolfram SystemModeler. This is because optimization of dynamic models is often used in various applications for example in optimal control and design optimization.

1.2 Purpose

The thesis involves optimization with focus on collocation methods for optimal control problems in Wolfram SystemModeler and Mathematica. It involves comparison and evaluation of three optimization solvers for specified optimization problems. The optimization solvers used here are the function FindMinimum in Mathematica and the two solvers IPOPT and KNITRO which are software for solving large optimization problems. Using IPOPT required the development of an interface from Mathematica. Evaluation is performed by studying different test problems and comparing performance, accuracy and precision in the different solvers.

1.3 Problem analysis

The following tasks are to be fulfilled:

- Create an interface from Mathematica to the optimization solver IPOPT using Wolfram LibraryLink and MathLink, powerful ways to connect external code to Mathematica.
- Evaluate the optimization solvers FindMinimum, IPOPT and KNITRO by implementing and solving optimization problems.

1.4 Method

The work of the thesis can be divided into literature studies, learning the software, implementation and evaluation. Literature studies begin and proceed during the whole time since it is important to study the area and see the different algorithms of the solvers. These studies are mainly through printed literature and on web pages belonging to the developers of the different solvers. The software which are used in the thesis are Wolfram SystemModeler and Mathematica and knowledge in them are important since the major part of the work is performed with them. Implementation of the solvers is the major part of the work, it will involve some modeling in Wolfram SystemModeler to create the examples which should be used to evaluate the different solvers. But the emphasis is on the integration with Mathematica, involving implementation of the solvers, creation of interface to the solver IPOPT and evaluation of all solvers. The evaluation is performed by evaluating the different solvers on test problems created in Wolfram SystemModeler by comparing performance, accuracy and precision.

1.4.1 Implementation of interface from Mathematica to IPOPT

To use the optimization solver IPOPT in Mathematica an interface has to be created since the IPOPT code is written in C++. There are predefined interfaces for C, C++ and Fortran which can be used to easily define the problem formulation. In this thesis the problem is formulated in Mathematica and by using LibraryLink and MathLink the formulation is linked to the predefined interface for C. LibraryLink and MathLink provide the possibility to link external code with Mathematica in an efficient way with respect to speed and memory. IPOPT implements an interior point line search method and it requires the cost function, constraints, gradient of the cost function, Jacobian of the constraints and the Hessian of the Lagrangian function. These functions are defined and calculated in Mathematica and then

linked with MathLink to the problem formulation in C. A C compiler is needed; Microsoft Visual Studio 9.0 (2008) is used in this thesis.

1.5 Software

Different software are applied for different purposes including software for modeling and computation of the test problems and implementation of the collocation methods. Three optimization solvers are evaluated on the specified optimal control problems.

1.5.1 Modeling and computational software

Two software for modeling and computation are utilized and they are Wolfram SystemModeler and Mathematica. The test problems that are used for evaluation of the optimization solvers are implemented in Wolfram SystemModeler. The models are imported into Mathematica where the collocation method is implemented to solve the optimal control problem associated with the test problems.

Wolfram SystemModeler

Wolfram SystemModeler is a physical modeling and simulation tool developed by the company Wolfram MathCore. It is based on the free object-oriented modeling language Modelica and connects easily with Mathematica. Realistic models can be built with predefined standard Modelica components from a large library. Numerical experiments can then be performed on the model to evaluate and optimize it if desired [1].

Mathematica

Mathematica is a computational software program provided by the company Wolfram Research. It contains all elements of a project; calculations, visualizations, data storage and documentation together in one program. It is split in two parts; the kernel which interprets expressions and return result expressions and the front end which provides a graphical user interface allowing creation and editing of the program code together with the results including graphs, text and tables for example [2].

1.5.2 Optimization solvers

Three different optimization solvers are evaluated: IPOPT, KNITRO and the function FindMinimum in Mathematica.

IPOPT

IPOPT, short for Interior Point Optimizer, is a software package which is available online for usage [3]. It is distributed by COIN-OR, short for Computational Infrastructure for Operations Research, a project initiated to help the development of open source software [4]. The original was developed by Andreas Wächter under his supervisor Lorenz T. Biegler at the Chemical Engineering Department of Carnegie Mellon University during a dissertation research. IPOPT is written in the language C++, but it can also be used to generate a library which can be linked to the problem formulation [3].

IPOPT is designed to find local solutions of large nonlinear optimization problems of the form

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_L \leq g(x) \leq g_U \\ & x_L \leq x \leq x_U \end{aligned} \tag{1.1}$$

where $f(x)\colon \mathbb{R}^n \to \mathbb{R}$ is the cost function, x_L and x_U are possible lower and upper bounds on the state variable $x \in \mathbb{R}^n$. $g(x) : \mathbb{R}^n \to \mathbb{R}^m$ is the general nonlinear constraint function with lower and upper bounds g_L and g_U. Both functions $f(x)$ and $g(x)$ can be linear or nonlinear, convex or non-convex but should always be twice differentiable. The solver implements an interior point line search filter method that tries to find a local solution of the formulated optimization problem with associated constraints [3]. The mathematical details of the algorithm are presented in [5].

KNITRO

KNITRO, short for Nonlinear Interior point Trust Region Optimization (silent K), is a solver for nonlinear optimization problems produced by the company Ziena Optimization, Inc. It is designed for large problems and is highly regarded for its robustness and efficiency [6]. KNITRO runs as a solver in Mathematica with the application package KNITRO for Mathematica.

KNITRO solves optimization problems of the form (1.1). The functions $f(x)$ and $g(x)$ are assumed to be smooth. The solver implements three state-of-the-art interior-point and active-set methods for solving problems. The algorithms are the interior/direct algorithm, the interior/CG algorithm and the active set algorithm. This provides three different algorithms with different behaviors on the nonlinear optimization problems [7].

FindMinimum

FindMinimum is a function for solving optimization problems in Mathematica. It tries to find a local minimum numerically, so it does not guarantee a global minimum. It is defined for both unconstrained and constrained optimization. There are a number of methods available for unconstrained optimization but only one for constrained optimization. The method used for optimization with constraints is the interior point algorithm which requires first and second derivatives of the cost function and the constraints [8].

2

Theory

The theory of this thesis is described in this section. It consists of the method direct collocation, the description of optimal control and model predictive control together with different discretization and approximation schemes of ordinary differential equations.

2.1 Dynamic control systems

A dynamic control system is described by its input, state and output variables together with the system equations describing the systems behavior. The input variable is called the control variable and the output variable is often used for feedback. External signals and influences such as disturbances, noises and time delays can also affect the system behavior [9]. Dynamic systems can be described in different ways depending on the differential equation that best represents the dynamics of the system. Most often are ordinary differential equations used and they can be in either explicit form $\dot{x}(t) = f(x(t),u(t),t)$ or implicit form $f(x(t),\dot{x}(t),u(t),t) = 0$ where t represents time, $x(t)$ the state variable and $u(t)$ the control variable. The systems are mostly autonomous and t does not appear explicitly in the equations [10].

2.2 Model predictive control

Model predictive control descends from optimal control and the concept is described as using a dynamic model to forecast various system behaviors. The forecast is optimized to obtain the best choice at the present time based on some

demands and restrictions [11]. A future behavior of the state variable $x(t)$ is optimized by computing and manipulating a control variable $u(t)$. The optimization is performed within a limited time horizon by giving new information at the start of the horizon. The idea can be explained with a simple example in planning activities. Planning the working tasks in a specific plant is performed for the next 10 hours, but the plan is only implemented during the first hour. Thereafter the planning for the next 10 hours is repeated for every new hour based on how many tasks are fulfilled so far. A predetermined criterion is used to decide which plan is the best. How well and how fast the tasks are performed is based on how much effort the staff put in, how well they work together and if some machinery breaks down, these are the control variables. There are also limitations such as work hours, skills in engineering and available staff. This repeated planning is performed until all tasks are fulfilled [12].

2.3 Optimal control

There are different ways to meet the control objectives connected to control designs. Optimal control which is of interest here is one of the most used methods. Optimal control means to minimize or maximize a given cost function, defining the performance index of the model, within an optimization window connected to the studied problem or model [9]. It involves continuous functions such as $x(t)$ and $u(t)$ together with differential equations describing the paths of the variables. Optimal control can be seen as an infinite extension of nonlinear programming problems, which is characterized by a finite set of variables x and constraints c [13].

There are many applications of optimal control in both off-line and on-line settings. Off-line settings involve finding optimal trajectories for the transition between stationary operating conditions in a system. These can then be applied as a reference during manual control or as a target in automatic control together with feedback. On-line optimal control is most often performed in the form of model predictive control. The general problem in optimal control with the cost function $J(x(t),u(t),t)$ over the time interval $[t_0,t_f]$ has the following form

$$\begin{aligned} \underset{u(t)\ t\in[t_0,t_f]}{\text{minimize}} \quad & J(x(t),u(t),t) \\ \text{subject to} \quad & x(t_0) = x_0 \\ & x_L \leq x(t) \leq x_U \\ & g_L \leq g(x(t),u(t),t) \leq g_U \\ & f(x(t),\dot{x}(t),u(t),t) = 0 \end{aligned} \tag{2.1}$$

where the constraints are the initial constraints, the bounds of the variables, the path constraints and the nonlinear dynamic model description. The inequality signs can be replaced by equality signs and there are often more than one constraint describing the dynamic system.

The cost function in optimal control problems can often be divided into two parts

$$J(x(t),u(t),t) = \phi(x(t_0),u(t_0),t_0,x(t_f),u(t_f),t_f) + \int_{t_0}^{t_f} L(x(t),u(t),t)\,\mathrm{d}t$$

where ϕ is called the Mayer term and L is called the Lagrange integrand [14].

There are two major parts when solving an optimal control problem. They are the method for solving the optimization problem and the method for numerically solving the differential equations defining the system [13]. The optimization solvers mentioned in Section 1.5.2 are the methods used for solving the optimization problem. Various methods that solve the differential equations are discussed later in Section 2.4 and 2.6.

2.4 Indirect/Direct methods

There are two main types of numerical methods that solve optimal control problems, indirect methods and direct methods. Calculus of variations is used in indirect methods to determine the first-order optimality conditions of the optimal control problem of interest. This kind of method leads to a multiple-point boundary value problem which is solved to determine possible optimal trajectories called extremals. These extremals are then studied further to find out if they are a minimum or maximum. As the name suggests indirect methods solves the problem indirectly by transforming the problem into a boundary value problem. This means that the optimal solution is found by solving a system of differential equations. When applying direct methods are the state and control variable of the optimal control problem discretized in some manner. Both the state and control variable can be discretized or only one of them. The problem is then transcribed into a nonlinear optimization problem. There are many different optimization techniques to solve these kinds of nonlinear problems. The optimal solution is found by changing an infinite-dimensional optimal problem into a finite-dimensional one. The most common direct methods are single shooting, collocation and multiple shooting and the relation between the methods is presented in Figure 2.1 [15].

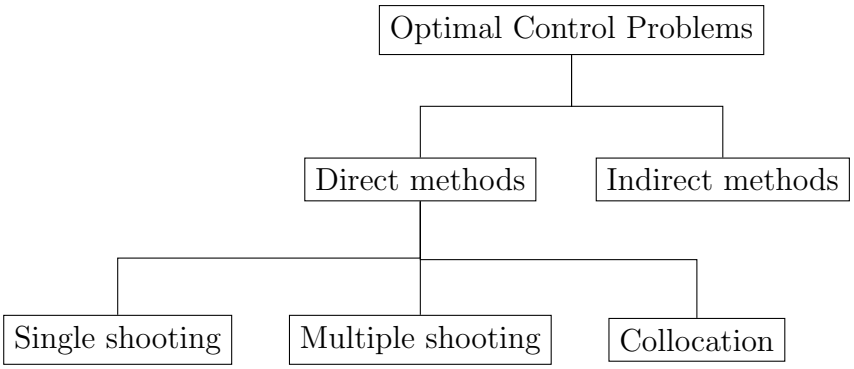


Figure 2.1: Flow chart describing the different methods that solve optimal control problems.

The method of single shooting relies on four steps. The first step is to guess the initial conditions of the optimal control problem and the second step is to propagate the differential equations from t_0 to t_f with the guessed initial conditions. This is called to "shoot" which gives the method its name. The third step is to evaluate the error at t_f and the fourth is to try to adjust the control variables to satisfy the constraints better, that is to repeat step one to three. The problem with this method is that small changes in the initial conditions can lead to large changes in the final conditions. Multiple shooting reduce the sensitivity in the initial conditions by dividing the problem into smaller steps. This increases the size of the problem but also the accuracy. Multiple shooting is also called parallel shooting due to the possibility of using a parallel processor for each step. However the most used method is collocation and it is described in more detail in the next section [13].

2.5 Collocation

Collocation is a method for solving differential equations, both ordinary and partial, and integral equations. The idea is to pick a space of finite dimension consisting of candidate solutions, often polynomials, and choosing a number of so called collocation points in the domain. The solution to select is the one that satisfies the given equation at the collocation points [14].

There are three different schemes for deciding the collocation points which are the most common ones. The three methods are the Gauss method, the Radau method and the Lobatto method. In the Gauss method neither of the two endpoints of the element is used as collocations point. In the Radau method at most one of the endpoints is used as a collocation point and in the Lobatto method both

of them are used as collocation points [15].

2.5.1 Direct collocation

Direct collocation is the most powerful method for solving general optimal control problems. The method approximates both the state and control variable with discretization using collocation points [15].

The time interval $[t_0,t_f]$ is divided and discretized into n elements by

$$t_i = t_0 + \sum_{k=0}^{i-1} h_k$$

where $i \in [1,\ldots,n-1]$, $t_n = t_f$ and h_k is the length of element k. A large number of elements leads to greater size of the optimization problem but also higher accuracy. n_c collocation points τ_j are introduced in each of the n elements which gives the following time discretization

$$t_{i,j} = t_0 + \left(\sum_{k=0}^{i-1} h_k + \tau_j h_i \right)$$

where $\tau_j \in [0,1]$, $i \in [1,\ldots,n]$, $j \in [1,\ldots,n_c]$ and element i is represented by the time points t_{i-1} at the beginning and t_i at the end. Inside the elements are both the state and control variable approximated with so called collocation polynomials. The approximated state and control variable in element i is denoted by x_i and u_i respectively. They are formed by selecting a number of collocation points, for simplicity they are all the same for every element. Figure 2.2 presents the idea of the discretization scheme as an example with three collocation points in each element i. The collocation polynomials are created by using the chosen numerical method to solve the ordinary differential equation. The different methods are explained in more detail in Section 2.6.

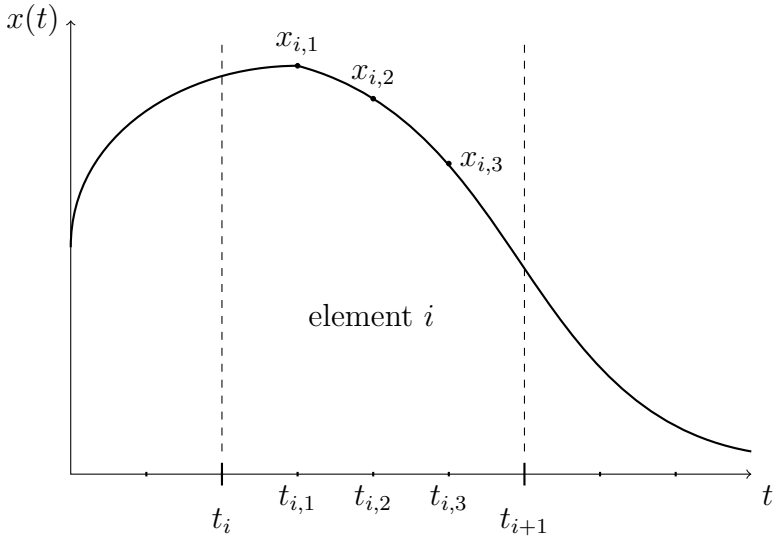


Figure 2.2: Presenting the collocation points in each element and the value of the state variable at the corresponding point. The figure is an example with three collocation points in each element.

The collocation points can be chosen differently and the three most common methods are mentioned earlier; Gauss, Radau and Lobatto. They have different numerical properties concerning stability and convergence.

Direct methods change infinite dimensional optimal control problems into finite dimensional. The optimization problem with infinite dimension in (2.1) is transformed into a finite dimensional problem with use of the constructed collocation polynomials. The cost function $J(x(t),u(t),t)$ are transformed by using a change of variable into

$$J(x(t),u(t),t) \approx \phi(x_{1,0},u_{1,0},t_0,x_{n,n_c},u_{n,n_c},t_f)+ \\ + \sum_{i=1}^{n} \left(h_i(t_f - t_0) \int_0^1 L_i(x_i(\tau),u_i(\tau),\tau)\,\mathrm{d}\tau \right) = \tilde{J}(Z)$$

where $t_{i,k}$ denote collocation point k in element i and $L_i(x_i(\tau),u_i(\tau),\tau)$ is the Lagrange integrand in element i. Z is the new optimization variable containing all values of both the state and control variable at the collocation points and t_0 and t_f if they are free. The integrand in $\tilde{J}(Z)$ is in this thesis approximated in different ways according to which method is used for solving the ordinary differential equation. The different approximations are described later.

The constraints corresponding to the nonlinear dynamic model description is only enforced at the collocation points and at t_0 after the transformation. The same holds for the path constraints. This implies that the transformed problem is sparse. It means that many elements of the constraints and mainly the Jacobian and the Hessian of the constraints are zero. Sparse optimization problems are possible to solve even though the problems are large. After the change into a finite dimensional problem with use of collocation points the formulation of the optimization problem is

$$\begin{aligned}
&\underset{Z}{\text{minimize}} && \tilde{J}(Z) && \text{(2.2a)}\\
&\text{subject to} && x_{1,0} = x_0 && \text{(2.2b)}\\
&&& x_{m,n_c} = x_{m+1,0} && \text{(2.2c)}\\
&&& x_L \leq x_{i,k} \leq x_U && \text{(2.2d)}\\
&&& g_L \leq g(x_{i,k},u_{i,k},t_{i,k}) \leq g_U && \text{(2.2e)}\\
&&& f(x_{i,k},\dot{x}_{i,k},u_{i,k},t_{i,k}) = 0 && \text{(2.2f)}\\
&&& \forall (i,k) \in \{(1,0)\} \cup ([1,\ldots,n] \times [1,\ldots,n_c]) && \text{(2.2g)}\\
&&& \forall m \in [1,\ldots,n-1]. && \text{(2.2h)}
\end{aligned}$$

The constraints (2.2b), (2.2d), (2.2e) and (2.2f) follows directly from only enforcing the initial constraints, the bounds of the variables, the path constraints and the system dynamics at the collocation points instead of over the whole time interval. Constraint (2.2c) is added to get continuity for the state variable [14] [16].

2.6 Numerical methods

Different numerical methods can be used to solve the ordinary differential equation defining the system. Euler method, backward differentiation methods and using Lagrange basis polynomials to approximate the variables are used in this thesis.

2.6.1 Euler method

The Euler method is an explicit one-step method of first order. It means that one collocation point is used which is easily applied together with the Radau method. The method is also named forward Euler since it evaluates in the starting point of each step. The derivative of the state variable is approximated by [13]

$$\dot{x}_i \approx \frac{x_{i+1} - x_i}{h_i}$$

where h_i is the length between x_i and x_{i+1}. The lengths of the elements are often equal, meaning $h_i = h = \frac{t_f - t_0}{n}$. In this context it represents the length of each element, since it uses only one collocation point in each element. This method is the most basic explicit method for solving ordinary differential equations numerically.

Using this method, or any other forward method, implies that the system dynamics cannot be imposed at the end of the time interval. Since Euler method is a one-step method it means that the ordinary differential equation describing the system is not imposed at the last collocation point.

The integral in the cost function has to be approximated since the problem is discretized and when using Euler method in this thesis is the integral approximated with the trapezoidal rule. It approximates the integral by

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\frac{f(a)+f(b)}{2}.$$

Given the form of the integral in the cost function the approximation is

$$\int_0^1 L_i(x_i(\tau),u_i(\tau),\tau)\mathrm{d}\tau \approx \frac{L(x_{i-1,1},u_{i-1,1},\tau_1) + L(x_{i,1},u_{i,1},\tau_1)}{2}$$

because in the Euler method there is only one collocation point τ_1 in each element and the integral is therefore approximated by the values of the cost function at each collocation point.

2.6.2 Lagrange basis polynomials

Lagrange interpolation polynomials can be used as candidate solutions with the collocation points as interpolation points. The state variables have to be continuous at the element boundaries and therefore an extra interpolation point is added at the start of each element denoted by $\tau_{i,0} := 0$. The Lagrange basis polynomials are given by

$$\tilde{\ell}_k(\tau) = \prod_{l\in[0,\ldots,n_c]\setminus\{k\}} \frac{\tau-\tau_l}{\tau_k-\tau_l} \text{ where } k \in [0,\ldots,n_c]$$

$$\ell_k(\tau) = \prod_{l\in[1,\ldots,n_c]\setminus\{k\}} \frac{\tau-\tau_l}{\tau_k-\tau_l} \text{ where } k \in [1,\ldots,n_c].$$

The polynomials are the same for all elements because the choice of the collocation points τ_j in each element are the same. The Lagrange basis polynomials have the

property

$$\ell_k(\tau_j) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k. \end{cases}$$

The collocation polynomials for the state and control variable are given by

$$x_i(\tau) = \sum_{k=0}^{n_c} x_{i,k} \tilde{\ell}_k(\tau)$$

$$u_i(\tau) = \sum_{k=1}^{n_c} u_{i,k} \ell_k(\tau)$$

where $x_{i,k} = x_i(\tau_k)$ and the same for $u_{i,k}$ for $i \in [1,\ldots,n]$. To approximate the derivative of the state variable $\dot{x}$ in each collocation point $t_{i,j}$ the collocation polynomial $\dot{x}_i$ is differentiated with respect to time. The chain rule gives the result

$$\dot{x}_{i,j} = \dot{x}_i(\tau_j) = \frac{1}{h_i(t_f - t_0)} \sum_{k=0}^{n_c} x_{i,k} \dot{\tilde{\ell}}_k(\tau_j)$$

where h_i is the length of element i. With this method is all h_i of equal length and normalized to $h_i = \frac{1}{n}$, which implies $\sum_{i=0}^{n-1} h_i = 1$.

An extra constraint

$$u_{1,0} - \sum_{k=1}^{n_c} u_{1,k} \ell_k(0) = 0 \tag{2.3}$$

is added to the optimization problem (2.2) as an extrapolation constraint. It is required since the initial value for the control variable is not given by the initial equation or the dynamic equations describing the system. It is therefore given by the collocation polynomial $u_{1,k}(\tau)$ instead, so $u_{1,0}$ is given by the extra constraint (2.3).

The integral in the cost function can be simplified by the following

$$\int_0^1 L_i(x_i(\tau),u_i(\tau),\tau)\,\mathrm{d}\tau \approx \sum_{k=1}^{n_c} \omega_k L_i(x_{i,k},u_{i,k},\tau_k)$$

where $\omega_k = \int_0^1 \ell_k(\tau)\mathrm{d}\tau$ are quadrature weights which gives the best approximation for this interpolation. The approximated cost function with Lagrange basis

polynomials is therefore

$$J(x(t),u(t),t) \approx \phi(x_{1,0},u_{1,0},t_0,x_{n,n_c},u_{n,n_c},t_f)+\sum_{i=1}^{n}\left(h_i(t_f-t_0)\sum_{k=1}^{n_c}\omega_k L_i(x_{i,k},u_{i,k},\tau_k)\right)$$

[14] [16].

2.6.3 Backward differentiation methods (BDF)

The backward differentiation methods are implicit methods for solving differential equations. They are multi-step methods where the method of first order is the backward Euler. The formulas for the methods of order 1-6 are

BDF 1	$\dot{x}_i \approx \frac{x_i - x_{i-1}}{h_i}$
BDF 2	$\dot{x}_i \approx \frac{x_i - \frac{4}{3}x_{i-1} + \frac{1}{3}x_{i-2}}{\frac{2}{3}h_i}$
BDF 3	$\dot{x}_i \approx \frac{x_i - \frac{18}{11}x_{i-1} + \frac{9}{11}x_{i-2} - \frac{2}{11}x_{i-3}}{\frac{6}{11}h_i}$
BDF 4	$\dot{x}_i \approx \frac{x_i - \frac{48}{25}x_{i-1} + \frac{36}{25}x_{i-2} - \frac{16}{25}x_{i-3} + \frac{3}{25}x_{i-4}}{\frac{12}{25}h_i}$
BDF 5	$\dot{x}_i \approx \frac{x_i - \frac{300}{137}x_{i-1} + \frac{300}{137}x_{i-2} - \frac{200}{137}x_{i-3} + \frac{75}{137}x_{i-4} - \frac{12}{137}x_{i-5}}{\frac{60}{137}h_i}$
BDF 6	$\dot{x}_i \approx \frac{x_i - \frac{360}{147}x_{i-1} + \frac{450}{147}x_{i-2} - \frac{400}{147}x_{i-3} + \frac{225}{147}x_{i-4} - \frac{72}{147}x_{i-5} + \frac{10}{147}x_{i-6}}{\frac{60}{147}h_i}$

In this context is h_i the distance between the collocation points, which means $h_i = \frac{t_f - t_0}{n \cdot n_c}$. For order higher than 6 the methods are not zero-stable and therefore not usable for numerical approximation of derivatives [17].

To impose the system dynamics over all collocation points is important. The ordinary differential equation is therefore approximated with BDF 1 in the second collocation point, BDF 2 in the third collocation point and so on up to the BDF of the chosen order. Figure 2.3 presents the idea of using BDF of lower order at the beginning of the time horizon to impose the system dynamics over the whole time horizon.

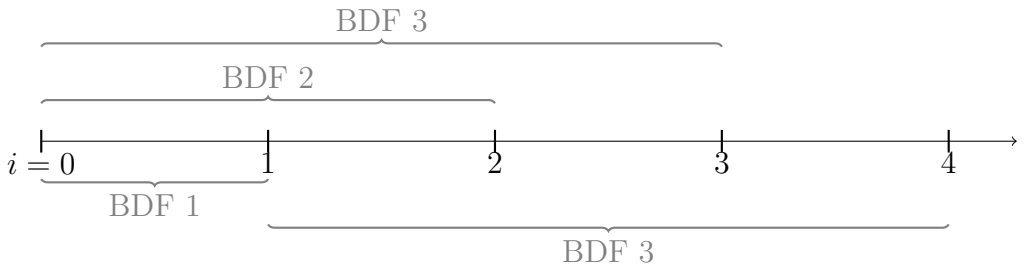


Figure 2.3: Describing the idea of BDF of lower order at the beginning of the time interval, in this figure the order of the method is chosen to be 3.

Approximating the integral when using the backward differentiation method is performed with the composite rule

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{n}\left(\frac{f(a)}{2} + \sum_{k=1}^{n-1}\left(f\left(a + k\frac{b-a}{n}\right)\right) + \frac{f(b)}{2}\right).$$

Applying this on the integrand in the cost function gives the following approximation

$$\int_0^1 L_i(x_i(\tau),u_i(\tau),\tau)\mathrm{d}\tau \approx$$

$$\approx \frac{1}{n_c}\left(\frac{L(x_{i-1,n_c},u_{i-1,n_c},\tau_0)}{2} + \sum_{k=1}^{n_c-1} L(x_{i,k},u_{i,k},\tau_{i,k}) + \frac{L(x_{i,n_c},u_{i,n_c},\tau_{n_c})}{2}\right).$$

3

Test problems

The optimization solvers FindMinimum, IPOPT and KNITRO are evaluated with two test problems: a batch reactor and a free floating robot. The test problems themselves are not studied in detail, they are just used to evaluate the optimization solvers.

3.1 Batch reactor

The first test problem is a chemical reactor. The model is to maximize the yield of $x_2(t)$ after one hour by manipulating the reaction temperature $u(t)$ within the time interval [0,1]. The problem formulation of the initial value problem is

$$\begin{aligned}
\underset{u(t)\ t\in[0,1]}{\text{minimize}} \quad & J(x(t),u(t),t) = -x_2(1) \\
\text{subject to} \quad & x_1(0) = 1 \\
& x_2(0) = 0 \\
& 0 \leq x_1(t) \leq 1 \\
& 0 \leq x_2(t) \leq 1 \\
& 0 \leq u(t) \leq 5 \\
& \dot{x}_1(t) = -\left(u(t) + \frac{u^2(t)}{2}\right) x_1(t) \\
& \dot{x}_2(t) = u(t)x_1(t)
\end{aligned} \tag{3.1}$$

The test problem is taken from [10] [18].

3.2 Free floating robot

The second test problem describes a free floating robot. The problem formulation of the boundary value problem is

$$
\begin{aligned}
\underset{u(t)\ t\in[0,5]}{\text{minimize}} \quad & J(x(t),u(t),t) = \frac{1}{2}\int_0^5 \sum_{i=1}^{4} u_i^2 \mathrm{d}t \\
\text{subject to} \quad & \dot{x}_1 = x_2 \\
& \dot{x}_2 = \frac{(u_1+u_3)c_5 - (u_2+u_4)s_5}{M} \\
& \dot{x}_3 = x_4 \\
& \dot{x}_4 = \frac{(u_1+u_3)s_5 + (u_2+u_4)c_5}{M} \\
& \dot{x}_5 = x_6 \\
& \dot{x}_6 = \frac{(u_1+u_3)D - (u_2+u_4)L_e}{I_n} \\
& x_i(0) = 0 \qquad 1 \leq i \leq 6 \\
& x_i(5) = 0 \qquad i \in [2,4,5,6] \\
& x_i(5) = 4 \qquad i \in [1,3]
\end{aligned}
\tag{3.2}
$$

where $s_5 = \sin(x_5)$, $c_5 = \cos(x_5)$, $M = 10$, $L_e = 5$, $D = 5$ and $I_n = 12$ [19].

4

Comparison of the solvers

The results are obtained by evaluating the optimization solvers with the test problems and the different numerical methods for solving the ordinary differential equations characterizing the system. The number of collocation points in each element and number of discretization elements are also varied to obtain values of them that give good results in most cases. The Radau collocation scheme with the right endpoint of the element as collocation point is used to obtain the results. Both FindMinimum and IPOPT require an initial starting point and in this thesis is 0 provided as a starting value for all variables. KNITRO computes an initial point based on the feasible set. To compare the different solvers and methods the following statistics are used

Total time The total time in seconds spent during the whole optimization process

Iterations The number of iterations needed by the optimization solver to solve the problem

The obtained results are also compared, such as the cost function and the optimal trajectories of the state and control variables.

4.1 Size of the problems

The number of variables and constraints define the size of the optimal control problem. Applying direct collocation leads to a finite dimensional problem which is larger with respect to number of variables and constraints than the original problem but sparse. The new optimization variable Z created after the transformation consists of $n_Z = N_s \cdot n \cdot n_c + (n \cdot n_c + 1)N_c$ variables, where N_s and N_c are the

number of state and control variables in the original problem. The number of constraints depends on the numerical method used to approximate the system equations. The number of constraints for the different methods are presented in Table 4.1.

Lagrange basis polynomials	$n_{cons}(n \cdot n_c + 1) + N_c + N_s(n-1) + N_b$
Euler method	$n_{cons} \cdot n \cdot n_c + N_b$
Backward differentiation methods	$n_{cons}(n \cdot n_c + 1 - n_c) + n_{cons}(n_c - 1) + N_b$

Table 4.1: Number of constraints in optimal control problems after applying direct collocation using the three numerical methods.

If the solver FindMinimum is used $N_s + N_c$ constraints have to be added since it takes the bounds of the variables as constraints while IPOPT and KNITRO are given the bounds as a separate option. n_{cons} is the number of constraints in the original problem and N_b is the number of boundary conditions. Both IPOPT and KNITRO make use of the transformed problems sparsity and keep track of the number of nonzero elements in the Jacobian and Hessian which increases their speed when solving large problems.

4.2 Accuracy and precision for the optimization solvers

There are different options for the optimization solvers. No changes in the standard options are made when comparing the optimization solvers to each other. There are different options for the three solvers which make it hard to obtain exactly the same accuracy and precision for all three and that is the reason for keeping the standard settings. It is still important to know the difference between them to fairly compare the solvers to each other.

FindMinimum offers the possibility to choose accuracy and precision by fulfilling the following two inequalities $||x_k - x^*|| \leq \max(10^{-a}, 10^{-p}||x_k||)$ and $\nabla f(x_k) \leq 10^{-a}$ where a represents accuracy and p precision. By default is the working precision for FindMinimum defined such that numbers contain just fewer than 16 digits, exactly 53 bits, and the accuracy and precision is defined to be half of the working precision.

KNITRO has several requirements to fulfill to terminate the optimization process, one is that the two following inequalities should be fulfilled

$$\varepsilon_{\text{feas}} \leq \max(\tau_1 \cdot tol_{\text{feas}}, tol_{\text{feas,abs}})$$
$$\varepsilon_{\text{opt}} \leq \max(\tau_2 \cdot tol_{\text{opt}}, tol_{\text{opt,abs}})$$

where $\varepsilon_{\text{feas}}$ is the feasibility error, ε_{opt} is the optimality error, tol_{feas} and tol_{opt} the final relative stopping tolerance for the feasibility/optimality error and $tol_{\text{feas,abs}}$ and $tol_{\text{opt,abs}}$ the final absolute stopping tolerance for the feasibility/optimality error. τ_1 and τ_2 are given by

$$\tau_1 = \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), (x_j^0 - b_j^U))$$

$$\tau_2 = \max(1, ||\nabla f(x^k)||_\infty)$$

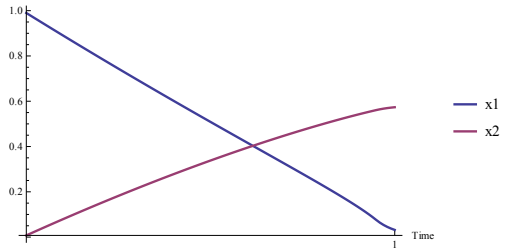
where x^0 is the initial point, c_i^L and c_i^U are the lower and upper bounds for the constraints and b_j^L and b_j^U are lower and upper bounds for the variables. If these two constraints are fulfilled the termination stops. The default values for tol_{feas} and tol_{opt} is 10^{-6} while $tol_{feas,abs}$ and $tol_{opt,abs}$ by default are 0. The process also terminates if the relative change in all components of the solution point estimate is less than x_{tol} which by default is 10^{-15}.

IPOPT has also several tolerance demands to obtain a successful termination of the optimization algorithm. The algorithm terminates if the (scaled) optimality error is less than the default value 10^{-8} and it also terminates if the maximum norm of the (unscaled) constraint violation is less than the default value 0.0001.

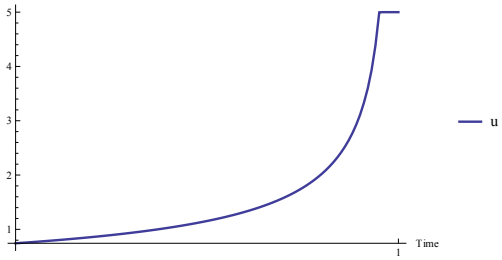
The KNITRO solver scales both the cost function and the constraints if necessary. IPOPT has by default a gradient-based scaling which means that it scales the problem so that maximum of the gradient at the starting point is 100 and the same value for the minimum of the gradient is 10^{-8}. This default option can cause problem for example when the value of some constraints are high and some are low.

4.3 Batch reactor

The results of the optimal control problem corresponding to a batch reactor are presented in Figure 4.1. The results are presented as the trajectories of the state and control variables.



(a) Trajectories of the state variables x_1 and x_2.



(b) Trajectory of the control variable u

Figure 4.1: Trajectories of the state and control variables obtained with KNITRO and Lagrange basis polynomials with $n = 20$ and $n_c = 5$.

4.3.1 Euler method

The results from the Euler method are obtained for $n = 10, 20, 40, 80, 160$ discretization elements. The number of iterations and the total time used during the optimization process is presented in Figure 4.2 and Figure 4.3. These results are obtained by solving optimal control problems with 31 variables and 20 constraints when $n = 10$ and 481 variables and 320 constraints when $n = 160$. There are no results for FindMinimum with $n = 160$ because the solver presents bad results already for $n = 80$ where the trajectories of the variables flatten out after $t = 0.5$ instead of decreasing respectively increasing for x_1 and x_2.

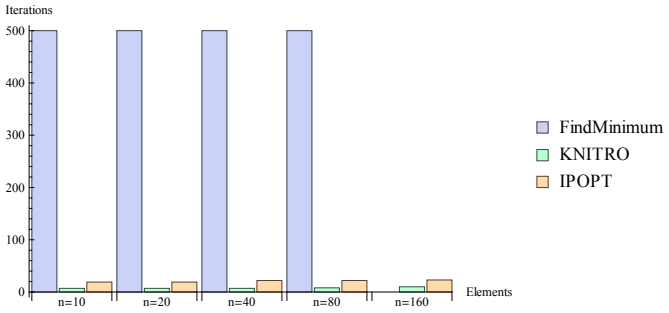


Figure 4.2: Number of iterations using Euler method with $n = 10, 20, 40, 80, 160$ and the three optimization solvers.

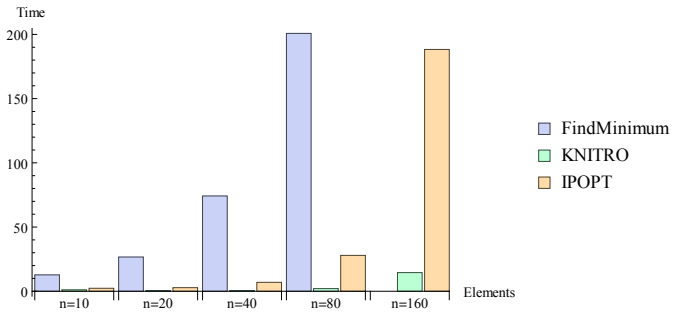


Figure 4.3: Total time spent during optimization with Euler method for $n = 10, 20, 40, 80, 160$ and the three optimization solvers.

FindMinimum presents a large number of iterations for every n and it stops at 500 iterations because it is the maximum number of iterations by default in FindMinimum. IPOPT uses about twice as many iterations as KNITRO but both use fewer than FindMinimum. With respect to time FindMinimum is presenting the longest times compared to the other two solvers. But there is also a significant difference between IPOPT and KNITRO. IPOPT takes more time than KNITRO. One thing to point out is the CPU time IPOPT spends in function evaluations which is presented in Table 4.2. It can be compared to the time in seconds KNITRO spends in function evaluations. There is a difference between CPU time and elapsed time but it still indicates the difference in time between IPOPT and KNITRO in the sense that function evaluations takes longer time for IPOPT.

	$n = 10$	$n = 20$	$n = 40$	$n = 80$	$n = 160$
IPOPT	0.304	1.009	4.780	25.63	184.9
KNITRO	0.043	0.014	0.013	0.03	0.05

Table 4.2: The total CPU time in seconds spent by IPOPT in function evaluations with the Euler method and the total time in seconds spent by KNITRO in evaluations with the Euler method.

The optimized values of the cost function is presented in Table 4.3. It shows that KNITRO and IPOPT finds the same optimal solution for each n. It also verifies the fact that FindMinimum obtains a bad result for $n = 80$.

	$n = 10$	$n = 20$	$n = 40$	$n = 80$	$n = 160$
IPOPT	-0.6055	-0.5902	-0.5808	-0.5771	-0.5753
KNITRO	-0.6055	-0.5902	-0.5808	-0.5771	-0.5753
FindMinimum	-0.5927	-0.5858	-0.5813	-0.4694	

Table 4.3: The optimal values of the cost function using the three optimization solvers with the Euler method for various number of discretization elements.

4.3.2 Backward differentiation methods

The results obtained by using the backward differentiation methods are derived by varying order of the method and number of discretization elements. Two set of results are presented. The first result is obtained with order 4 and $n = 10, 15, 20$ discretization elements. It corresponds to problems with between 121 and 241 variables together with between 80 and 160 constraints. The number of iterations and the time spent during the optimization is presented in Figure 4.4 and Figure 4.5.

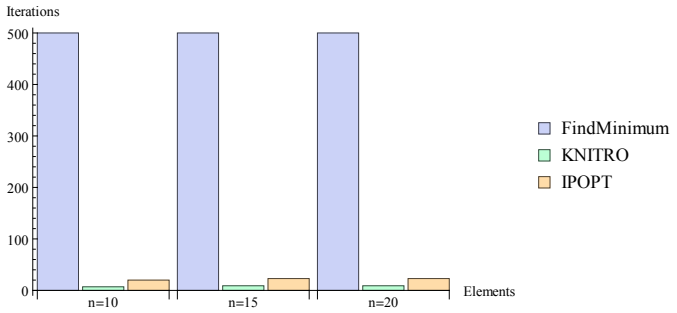


Figure 4.4: Number of iterations using BDF 4 with $n = 10, 15, 20$ and the three optimization solvers.

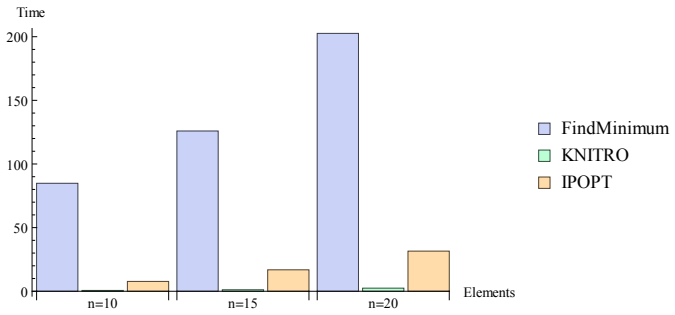


Figure 4.5: Total time spent during optimization with BDF 4 for $n = 10, 15, 20$ and the three optimization solvers.

FindMinimum presents a large number of iterations with this method too, and as with the Euler method it stops at 500 iterations since that is the maximum number. KNITRO and IPOPT present similar number of iterations but with KNITRO slightly fewer. FindMinimum spends the largest amount of time and KNITRO spends least time during the process. IPOPT and KNITRO provide the same optimal value of the cost function, while FindMinimum presents slightly different values compared to the other two.

The second set of results are obtained with 20 discretization elements and $n_c = 1, 2, 3, 4, 5, 6$ as the order of the backward differentiation method. These results correspond to problems consisting of between 61 and 361 variables together with between 40 and 240 constraints. Figure 4.6 and Figure 4.7 present the number of iterations used and the total time spent during the optimization.

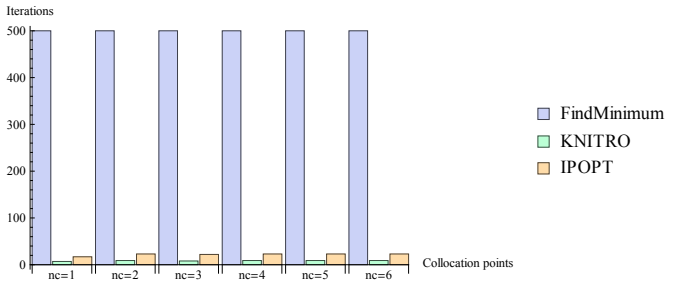


Figure 4.6: Number of iterations using BDF with $n = 20$, $n_c = 1, 2, 3, 4, 5, 6$ and the three optimization solvers.

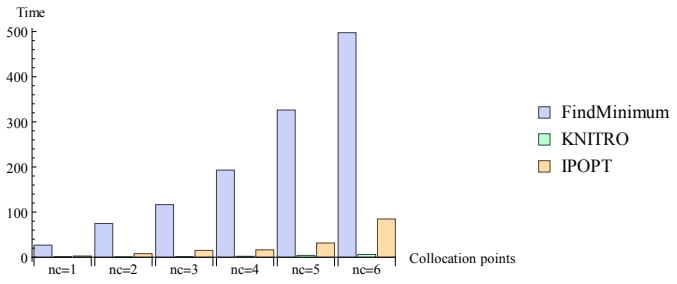


Figure 4.7: Total time spent during optimization with BDF for $n = 20$, $n_c = 1, 2, 3, 4, 5, 6$ and the three optimization solvers.

FindMinimum use its maximum number of iterations for all choices of n_c, while IPOPT use about twice as many as KNITRO. FindMinimum also spends more time than IPOPT and KNITRO do in all results. In the timing aspect IPOPT takes longer time than KNITRO, which spends least time in all cases. KNITRO and IPOPT find the same optimal values of the cost function and FindMinimum presents slightly different values. It also shows that for $n_c = 5$ and 6 the trajectories start to oscillate when using FindMinimum.

4.3.3 Lagrange basis polynomials

The presented results obtained by using Lagrange basis polynomials are produced in two different ways. The first set is obtained with $n = 20$ discretization elements and varying number of collocation points. In this set of results are the problems consisting of 121 variables and 121 constraints at minimum and 361 variables and 281 constraints at maximum. The second set is obtained with $n_c = 5$ and varying the number of discretization elements. The second set consists of between 76 and 301 variables and between 61 and 241 constraints. The results produced with $n = 20$ are presented in Figure 4.8 and Figure 4.9, which represents the number of iterations and the total time spent during the optimization process. No result was obtained for $n = 20$ and $n_c = 2$ with IPOPT because the optimization failed in that case.

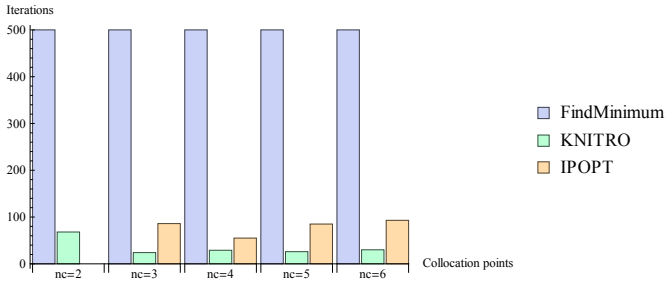


Figure 4.8: Number of iterations using Lagrange basis polynomials with $n = 20$, $n_c = 2, 3, 4, 5, 6$ and the three optimization solvers.

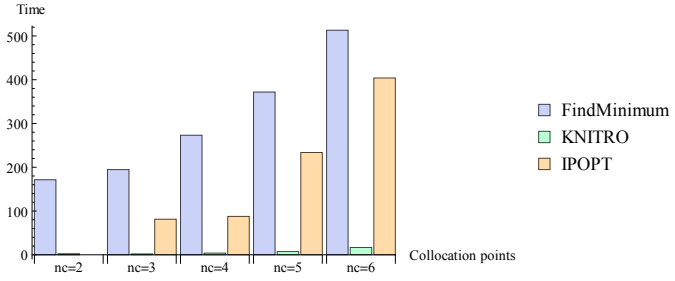


Figure 4.9: Total time spent during optimization with Lagrange basis polynomials for $n = 20$, $n_c = 2, 3, 4, 5, 6$ and the three optimization solvers.

As with the previously used methods FindMinimum needs its maximum number of iterations during the optimization. The other two solvers need less iterations, where KNITRO presents the lowest number. For $n_c = 2$ all optimization solvers presents wrong results, not the optimal value of the cost function. For low values of n_c FindMinimum presents the right optimized trajectories of x_1 and x_2 but the trajectory of the control variable u is different compared to the other results. But for higher values of n_c the correct optimized trajectories are obtained. FindMinimum takes the longest time during the optimization process with IPOPT taking a little less time. KNITRO is the fastest solver and takes just a small amount of time for all choices of n_c. In Table 4.4 is the optimal values of the cost function presented. It states the fact that for $n_c = 2$ none of the solvers provides the correct result.

	$n_c = 2$	$n_c = 3$	$n_c = 4$	$n_c = 5$	$n_c = 6$
IPOPT	-0.0284	-0.5726	-0.5732	-0.5735	-0.5735
KNITRO	-0.1317	-0.5735	-0.5735	-0.5735	-0.5735
FindMinimum	-23.74	-0.5226	-0.5731	-0.5697	-0.5688

Table 4.4: The optimal values of the cost function using the three optimization solvers with Lagrange basis polynomials for $n = 20$ and various number of collocation points.

The second set of results is presented in Figure 4.10 and Figure 4.11. The number of discretization elements is $n = 5, 10, 20$ and number of collocation points are fixed at $n_c = 5$.

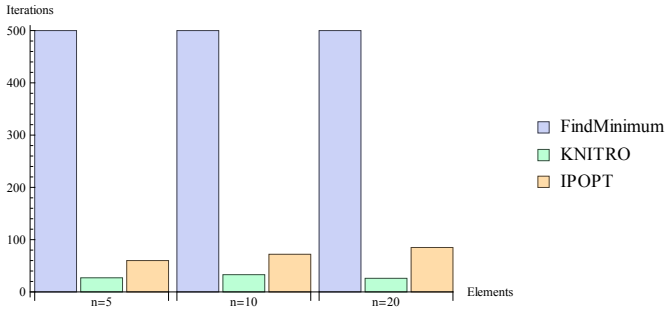


Figure 4.10: Number of iterations using Lagrange basis polynomials with $n_c = 5$, $n = 5, 10, 20$ and the three optimization solvers.

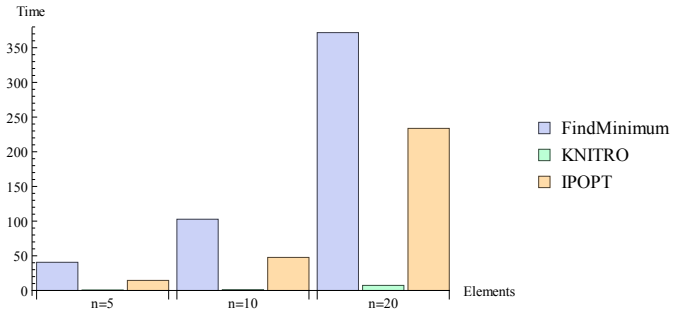
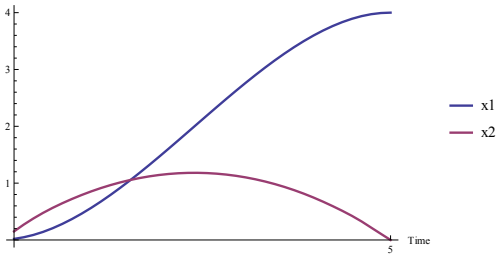


Figure 4.11: Total time spent during optimization with Lagrange basis polynomials for $n_c = 5$, $n = 5, 10, 20$ and the three optimization solvers.

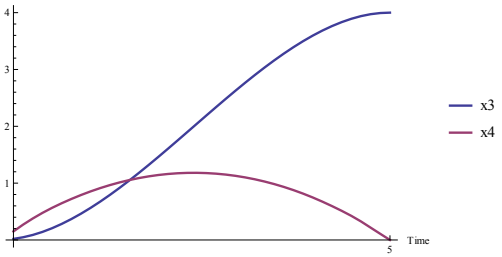
FindMinimum uses 500 iterations for all derived results and KNITRO and IPOPT use less, with KNITRO about half of the number IPOPT needs. The optimization is fast using KNITRO and slower using IPOPT and FindMinimum. All three solvers present good result in those cases, just some oscillations in the trajectory of u for FindMinimum. Only small differences between the optimal values of the cost function separates them.

4.4 Free floating robot

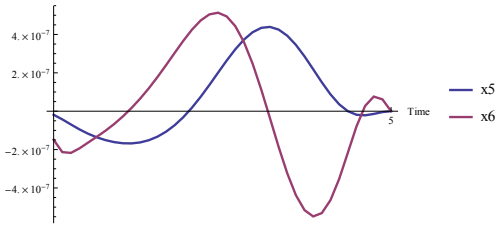
The trajectories corresponding to the optimal solution of the test problem with a free floating robot are presented in Figure 4.12. The result represents the six state variables and the four control variables. The beginning of the trajectories corresponding to the control variables in Figure 4.12 presents how the use of BDF with lower order in the beginning of the time interval affects the result. BDF 4 is applied and it leads to four directional changes.



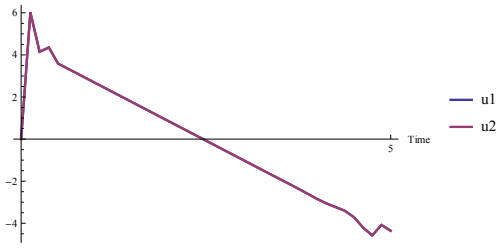
(a) Trajectories of the state variables x_1 and x_2.



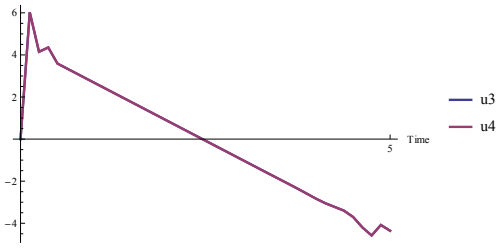
(b) Trajectories of the state variables x_3 and x_4.



(c) Trajectories of the state variables x_5 and x_6.



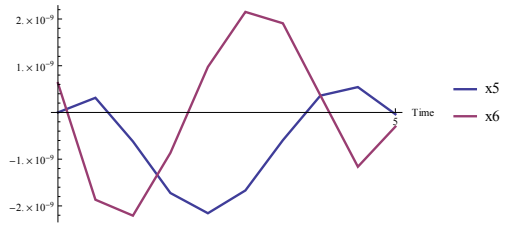
(d) Trajectories of the control variables u_1 and u_2.



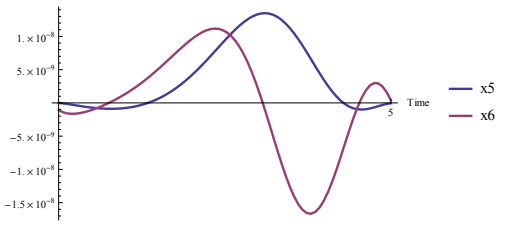
(e) Trajectories of the control variables u_3 and u_4.

Figure 4.12: Trajectories of the state and control variables obtained with KNITRO and backward differentiation methods using $n = 10$ and $n_c = 4$.

The difference in number of discretization elements and number of collocation points in each element is noticeable. Figure 4.13 shows how different the trajectories for the state variables x_5 and x_6 can behave. The trajectories are obtained with FindMinimum and Lagrange basis polynomials for $n = 5$, $n_c = 3$ and $n = 20$, $n_c = 5$ where the latter corresponds to the smooth curves. This difference shows that more discretization elements often results in more continuous trajectories. This is true at least up to a certain level and afterwards it can start to oscillate instead.



(a) Trajectories obtained with $n = 5$ discretization elements and $n_c = 3$ collocation points.



(b) Trajectories obtained with $n = 20$ discretization elements and $n_c = 5$ collocation points.

Figure 4.13: Trajectories of the state variables x_5 and x_6 for different number of discretization elements and collocation points presenting the possible differences.

4.4.1 Euler method

The results derived with the Euler method for the three solvers are produced with $n = 10, 20, 40, 80, 160$ discretization elements. This corresponds to problems with 104 variables and 66 constraints for $n = 10$ and 1604 variables and 966 constraints for $n = 160$. No results were obtained for KNITRO with $n = 160$ because the optimization could not terminate due to unavailable memory on the computer. Number of iterations during the optimization process is presented in Figure 4.14 and the total time spent during optimization is presented in Figure 4.15.

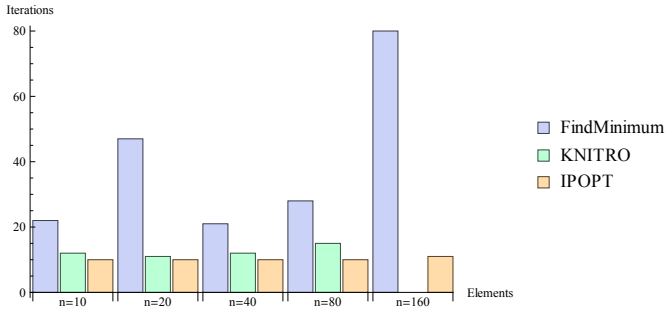


Figure 4.14: Number of iterations using Euler method with $n = 10, 20, 40, 80, 160$ and the three optimization solvers.

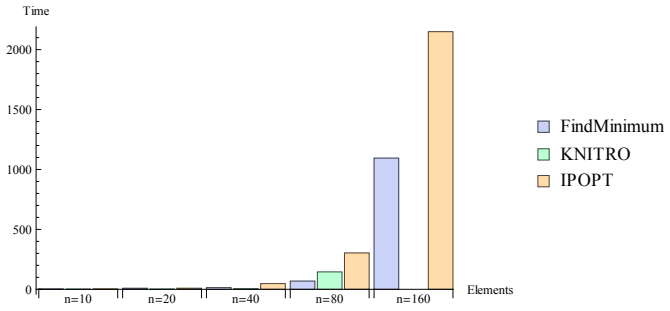


Figure 4.15: Total time spent during optimization with Euler method for $n = 10, 20, 40, 80, 160$ and the three optimization solvers.

FindMinimum presents the highest number of iterations for all n but the difference is not that big as in the first test problem. Both IPOPT and KNITRO present about the same number for all n while FindMinimum varies more. IPOPT spends most time during optimization compared to the other two but as in the previous test problem IPOPT spends much more time in function evaluations than the other two. FindMinimum is the solver that spends the lowest amount of time for $n = 80$ and 160. All three optimization solvers obtain the same optimal value of the cost function for each result. In Table 4.5 are the optimal values of the cost function presented.

	$n = 10$	$n = 20$	$n = 40$	$n = 80$	$n = 160$
Optimal value	317.1	340.9	359.2	370.7	377.1

Table 4.5: The optimal values of the cost function with Euler method and various number of discretization elements.

4.4.2 Backward differentiation methods

The results obtained using the backward differentiation methods are produced with different number of discretization elements and order of the method. Two different set of results are presented. They are obtained by either fixing the order to 4 and varying the discretization elements or by fixate the number of discretization elements to 10 and varying the order. In the first set of results with order 4 are the number of variables between 404 and 804 while the number of constraints is between 246 and 486. The number of iterations and time spent during the process for order 4 is presented in Figure 4.16 and Figure 4.17.

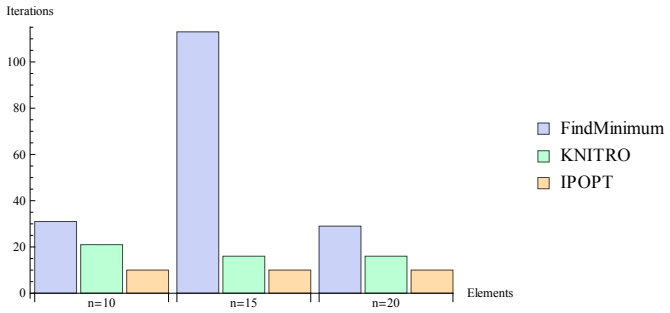


Figure 4.16: Number of iterations using BDF 4 with $n = 10, 15, 20$ and the three optimization solvers.

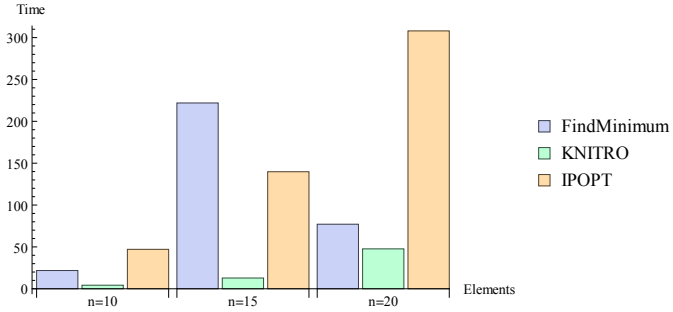


Figure 4.17: Total time spent during optimization with BDF 4 for $n = 10, 15, 20$ and the three optimization solvers.

The number of iterations vary for FindMinimum and for IPOPT and KNITRO the number is approximately the same for all n. But for all three solvers is the number of iterations at an acceptable level except for FindMinimum with $n = 15$. Also in the time aspect FindMinimum varies while IPOPT and KNITRO spend more time the bigger problem. IPOPT spends more time than the other two for higher n which derives from the higher amount of time it spends in function evaluations compared to FindMinimum and KNITRO. The same value of the cost function is obtained by the solvers for each set of results. Studying Table 4.6 a comparison between how much time IPOPT and KNITRO spends in function evaluations can be made. It is the CPU time IPOPT spends in function evaluations and the elapsed time KNITRO spends in evaluations which are represented. Even though the distinction between CPU time and elapsed time the difference is significant: IPOPT spends much more time in function evaluations than KNITRO.

	$n = 10$	$n = 15$	$n = 20$
IPOPT	45.59	137.7	305.4
KNITRO	0.106	0.138	0.359

Table 4.6: The total CPU time in seconds spent by IPOPT in function evaluations with BDF 4 and the total time in seconds spent by KNITRO in evaluations with BDF 4.

Figure 4.18 and Figure 4.19 presents the result when the number of discretization elements is fixed to 10 and the order of the method is varied. In this case are the number of variables between 104 and 604 with the number of constraints between 66 and 366.

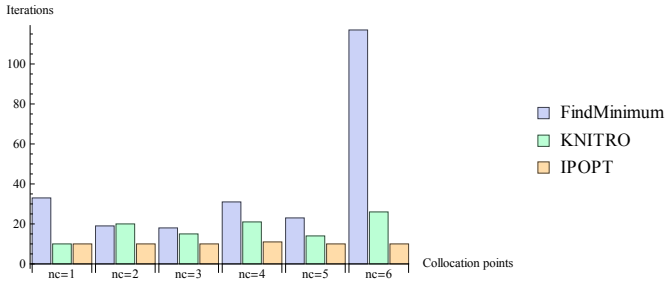


Figure 4.18: Number of iterations using BDF with $n = 10$, $n_c = 1, 2, 3, 4, 5, 6$ and the three optimization solvers.

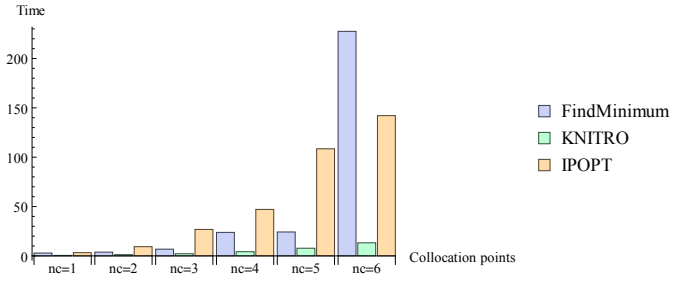


Figure 4.19: Total time spent during optimization with BDF for $n = 10$, $n_c = 1, 2, 3, 4, 5,6$ and the three optimization solvers.

The number of iterations needed by both KNITRO and FindMinimum vary more than it does for IPOPT. IPOPT keeps a constant level around 10 iterations. FindMinimum needs many iterations for $n_c = 6$ but the result is still the same. KNI-

TRO spends a low amount of time in the optimizations compared to the other two. IPOPT is slowest except for $n_c = 6$ when FindMinimum takes the longest time. Still IPOPT depends heavily on the time spent in function evaluations. The solvers obtain the same optimal value of the cost function for all n_c.

4.4.3 Lagrange basis polynomials

When Lagrange basis polynomials are used during optimization of the free floating robot problem it does not provide many good results. First of all it takes long time, so the studied results are obtained with the combinations of $n = 5, 10, 15$ and $n_c = 2, 3, 4$. FindMinimum provides results for all combinations but the wrong values. It does not find the correct optimal solution in any case. KNITRO fails to obtain results in all cases except for $n = 10$ and $n_c = 3$, in that case the result is correct but not with smooth trajectories. The optimization fails due to different reasons. The KNITRO solver leaves a message during failure. The reasons for failure are maximum number of iterations reached, converged to an infeasible point and the solver could not improve on current infeasible estimate. IPOPT is the solver that provides most correct optimizations even though it does not succeed in all cases. For $n_c = 2$ it fails for all choices of n because it converges to a point of local infeasibility. For $n_c = 3$ it provides good optimization results in all cases. For $n_c = 4$ IPOPT only solves the problem to an acceptable level. It means that it does not fulfill the desired tolerance, only the acceptable tolerance which is lower than the desired.

5

Discussion

The discussion consists of analysis regarding the time the optimization solvers spend during the optimization process and the iterations needed to obtain a result. Comparison of the numerical methods used to approximate the derivative of the state variable and a comparison of the three optimization solvers ends the discussion.

5.1 Timing aspect

FindMinimum has much better performance for unconstrained optimization than for constrained optimization. When constrained optimization problems are about to be solved with FindMinimum, it uses an implementation of an interior point method which is quite slow compared to the other two optimization solvers.

KNITRO spends the most time in construction of sparse partial derivative objects in Mathematica, numerical evaluation of functions and their derivatives in Mathematica and optimization calculations in KNITRO.

IPOPT is used by an interface from Mathematica to its interface in C and by callbacks to Mathematica to calculate the functions such as the cost function, constraints and their derivatives in each iteration. This implies that the time IPOPT takes to optimize the problem is heavily dependent on how fast these calculations are and how fast the result can be transferred from Mathematica to IPOPT.

All of these aspects should be taken into account when discussing the total time spent by the optimization solvers to solve the optimal control problems. With

few exceptions FindMinimum is the solver which takes most time during the optimization process. This is an expected result since FindMinimum does not handle constrained optimization that well. KNITRO is the fastest solver of the three in all presented results. The difference in time is large for larger problems and for smaller problems the difference between the three is small. IPOPT spends more time than KNITRO but mostly less time than FindMinimum. As mentioned IPOPT relies heavily on the implementation of the interface. The interface deals with sending information and handles the function evaluations which are the major time consumers in the optimization. IPOPT would perform much better if the link between Mathematica and IPOPT's C interface was faster and if the calculations were faster. It is definitely an improvement possibility and need of further work.

5.2 Number of iterations

The maximum number of iterations in FindMinimum is 500 and this number is reached in all results obtained with the batch reactor problem. In the free floating robot problem FindMinimum use most iterations in almost all cases and the number definitely vary more due to the size of the problem, while KNITRO and IPOPT use about the same number of iterations in all results. IPOPT use less iterations than KNITRO in the free floating robot problem and the other way around in the problem with the batch reactor. The number of iterations should be low if the optimization algorithm is good, otherwise it will lead to long optimization times.

5.3 Comparing the numerical methods

Lagrange basis polynomials presents good or decent results in the batch reactor problem using between three and six collocation points inside each element. Using two collocation points does not provide good results. The best results are provided with four or five collocation points. The number of discretization elements should not be too few which makes the trajectories non-smooth but also not too many which instead cause the trajectories to oscillate. In the test problem with the free floating robot optimization with Lagrange basis polynomials does not provide many good results. It mainly fails or presents wrong results. Overall is Lagrange basis polynomials therefore not a numerical method to recommend in this aspect.

Euler method is the most basic numerical method used. Despite this it presents good result, but not the best. The results are sufficiently good in both test problems which make it a useful numerical method even though it is a basic method. The more discretization elements used the better result in general.

Backward differentiation methods present good or descent results in all cases for both test problems. There is no big difference between the results depending on which order or number of discretization elements that is used. But BDF of order three and four present good results in both examples which make BDF 3 and BDF 4 methods to recommend. The results tend to be a little bit better with a higher number of discretization elements.

5.4 Comparing the optimization solvers

A comparison of the three optimization solvers is made concerning the total time spent during optimization, how many iterations are needed during the process and how correct the optimal trajectories and cost function are. FindMinimum is an optimization solver provided by Mathematica which does not have any good algorithms for solving constrained optimization. Also it does not take the sparsity of the problem into consideration which both IPOPT and KNITRO do. This fact made FindMinimum the solver with least expectations on beforehand. Despite this it presents the correct optimized values in most cases even though IPOPT and KNITRO were better. In the aspect of time and number of iterations it also presents the worst numbers. It varies in aspect of time and iterations, instead of increasing with increasing size of the problem. This makes it unreliable and therefore not the optimization solver to recommend in this case.

The comparison between IPOPT and KNITRO is more interesting. As mentioned throughout the thesis IPOPT is implemented with an interface from Mathematica to IPOPT's interface. The interface does not transfer information and calculate function evaluations as fast as desired. This fact makes IPOPT slower than KNITRO in aspect of function evaluations and results in more time spent in total than KNITRO. There is definitely more work to be done in the creation of the interface to make IPOPT competitive in aspect of time. Both solvers are similar in number of iterations used, sometimes IPOPT uses more and vice versa. Both solvers present good results over all and there is no big difference between the two according to optimal results. The only thing is that KNITRO fails in more cases than IPOPT and cannot terminate the optimization process due to unavailable memory, the iteration limit is reached or that it converges to an infeasible point. To separate the two solvers apart more test problems are needed and more test statistics should be provided. The main conclusion is that both IPOPT and KNITRO perform better than FindMinimum in all aspects; time, number of iterations, optimal trajectories and cost function. FindMinimum needs a better implementation of the algorithm solving constrained optimization to be competitive.

Bibliography

[1] Wolfram, "Wolfram SystemModeler." http://www.wolfram.com/system-modeler/, 2013.

[2] Wolfram, "Wolfram Mathematica 9." http://www.wolfram.com/mathematica/, 2013.

[3] 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*, vol. 106, no. 1, pp. 25–57, 2006.

[4] Computational Infrastructure for Operations Research. http://www.coin-or.org/, January 2012.

[5] A. Wächter, "An interior point algorithm for large-scale nonlinear optimization with applications in process engineering," Master's thesis, Carnegie Mellon University, 2002.

[6] Ziena Optimization LLC, "Knitro." http://www.ziena.com/knitro.htm, 2013.

[7] Ziena Optimization LLC, "Knitro 8.0 for Mathematica, User's manual." http://www.ziena.com/docs/Knitro80Mathematica_UserManual.pdf, November 2011.

[8] Wolfram, "Wolfram Mathematica 9, Documentation center: Constrained Optimization." http://reference.wolfram.com/mathematica/tutorial/ConstrainedOptimizationOverview.html, 2013.

[9] W. H. Kwon and S. H. Han, *Receding Horizon Control: Model Predictive Control for State Models*. Advanced Textbooks in Control and Signal Processing, Springer, 2005.

[10] J. Tamimi, *Development of Efficient Algorithms for Model Predictive Control of Fast Systems*. VDI Verlag, 2011.

[11] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[12] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB®*. Advances in Industrial Control, Springer, 2009.

[13] J. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*. Advances in Design and Control Series, Society For Industrial Mathematics, 2001.

[14] F. Magnusson and J. Åkesson, "Collocation Methods for Optimization in a Modelica Environment," in *Proceedings of the 9th International Modelica Conference*, 2012.

[15] Anil V. Rao, "Survey of numerical methods for optimal control," in *2009 AAS/AIAA Astrodynamical Specialist Conference*, 2009.

[16] J. Åkesson, "Lecture notes in numerical methods for dynamic optimization: Direct methods for Dynamic Optimization - Collocation." `http://www.control.lth.se/user/jakesson/DynamicOptimization2009/dynopt_l4b_slides.pdf`, 2009.

[17] Wikipedia, "Backward differentiation formula." `http://en.wikipedia.org/wiki/Backward_differentiation_formula`, April 2013.

[18] B. Bachmann, L. Ochel, V. Ruge, M. Gebremedhin, P. Fritzson, V. Nezhadali, L. Eriksson, and M. Sivertsson, "Parallel multiple-shooting and collocation optimization with OpenModelica," in *Proceedings of the 9th International Modelica Conference*, 2012.

[19] "A Java Application for the solution of Optimal Control Problems, Example 6b." `http://abs-5.me.washington.edu/dynOpt/ug.html`, 2013.