NTNU – Trondheim
Norwegian University of
Science and Technology

Modeling and Dynamic Optimization in Oil Production

Konstantin Nalum

Master of Science in Engineering Cybernetics
Submission date: June 2013
Supervisor: Bjarne Anton Foss, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of Science and Technology

Faculty of Information Technology, Mathematics and Electrical Engineering

Department of Engineering Cybernetics

Master project

Name of candidate: Konstantin Nalum

Subject: Engineering Cybernetics

Title: Modeling and dynamic optimization in oil production
Title (in Norwegian):

In oil and gas gathering systems there exist optimization opportunities which are not being exploited, partly due to the lack of efficient dynamic optimization tools. In this project dynamic optimization of an oil gathering network will be proposed. The implementation of this requires special attention to efficient algorithms to make the application possible in real-time. This study will judge the capability of research algorithms to tackle this problem. To this end, the model developed in [1] should be implemented in JModelica [2] and interfaced to CasADi [3] to take advantage of standard modeling languages and automatic differentiation tools. Instances of the dynamic optimization problem should be solved with IPOPT [4] relying on information provided by JModelica and CasADi.

Direct methods parameterize control and states of dynamical systems to formulate a dynamic optimization problem as a nonlinear programming (NLP) problem. The performance of direct methods such as Single shooting, Multiple shooting and Direct collocation should be compared using the tools mentioned above.

The master project is a continuation of previous work [1].

Task description:

1. Literature review on methods for optimization and control of dynamic systems. This is an extensive area so the review should be limited to methods of particular interest to the current study.
2. Implement the well, manifold and pipeline-riser models in JModelica.
3. Based on [1], determine a set of scenarios to test dynamic optimization algorithms.
4. Dynamic optimization of nonlinear models for real-time applications is computationally challenging. Study alternative algorithms such as Single shooting, Multiple shooting and Direct collocation, and assess them with respect to computation time and robustness.
5. Recommend a solution which provides a reasonable compromise between complexity, efficiency, accuracy and robustness.
6. Make a performance comparison of the new platform and the platform used in [1,5].

The thesis report may include a draft paper to a selected conference with the main results of this work.

[1] K. Nalum. Model predictive control of well-pipeline systems . Project report 2012.
[2] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org—Languages and tools for solving large-scale dynamic optimization problems. Computers & Chemical Engineering. 2010.
[3] J. Andersson, J. Åkesson, F. Casella and M. Diehl. Integration of CasADi and Jmodelica.org. Linkoping University Electronic Press. 2011.
[4] 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. 2006.
[5] A. Codas, M. A. Aguiar, K. Nalum and B. Foss. Differentiation Tool Efficiency Comparison for Nonlinear Model Predictive Control Applied to Oil Gathering Systems. Submitted to the 9th IFAC symposium on nonlinear control systems. 2013.

Start date: 15.01.2013

End date: 11.06.2013

Co-supervisor: Andres Codas, NTNU

Trondheim, 15.01.2013

Bjarne Foss
Professor/supervisor

Preface

This master's thesis was written during the spring semester of 2013 as a part of the M.Sc. degree in Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The thesis is a continuation of my project work in the fall of 2012.

I would like to thank my supervisor Prof. Bjarne Foss and co-supervisor Andres Codas at the university for valuable advice and guidance throughout this last year.

Trondheim
June 7, 2013

Konstantin Nalum

Abstract

The concept of Real-Time Production Optimization is a key for improving operational performance and productivity in the process industry. This includes utilization of computational technology, combining real-time measurements with mathematical models and optimization techniques, to offer a tool for decision support.

In this thesis, dynamic optimization is applied to a subsea oil gathering network consisting of wells and flowlines, described by continuous-time nonlinear differential algebraic equations. The goal is to combine modeling and optimization tools to suggest operational conditions that optimize production over a short-term horizon. Three different approaches for optimal control are presented and assessed: direct collocation, single shooting and multiple shooting. These are implemented in the JModelica.org framework coupling state of the art numerical packages for modeling and optimization of dynamic systems. This includes IPOPT for solving nonlinear programs, CVODES providing integration of continuous-time ordinary differential equations with sensitivity information, CasADi offering efficient automatic differentiation and Modelica as a high-level modeling language.

Experiments show that direct collocation and single shooting methods are significantly faster than the multiple shooting implementation. A set of test scenarios for production optimization has been assessed, indicating an increase of $0.5 - 3.2\%$ of production rates by tracking an unreachable reference, leading to a steady-state profile for production rates. When maximizing the objective function over 12 hours, the suggested solution increases the objective function value by $3.4 - 15\%$.

Sammendrag

Sanntids produksjonsoptimalisering er viktig for å bedre operasjonell drift og produktivitet i prosessindustrien. Dette inkluderer utnyttelse av teknologi for å kombinere sanntidsmålinger med matematiske modeller og optimaliseringsteknikker for å tilby et verktøy for beslutningsstøtte.

I denne avhandlingen er dynamisk optimalisering anvendt på et undervanns oljeproduksjonssystem bestående av brønner og produksjonslinjer, beskrevet av et kontinuerlig-tid ulineært differensial-algebraisk system. Målet er å kombinere modellering og optimaliseringsverktøy for å foreslå operasjonelle strategier som optimaliserer produksjonen over en korttidshorisont. Tre ulike tilnærminger for optimalregulering er presentert og vurdert: direkte kollokasjon, single shooting og multiple shooting. Disse er implementert i rammeverket JModelica.org som kobler numeriske pakker for modellering og optimalisering av dynamiske systemer. Dette inkluderer IPOPT for å løse ulineære programmer, CVODES for integrasjon av kontinuerlig-tid ordinære differensialligninger med sensitivitetsanalyse, CasADi for effektiv automatisk derivasjon, og Modelica som er et høy-nivå modelleringsspråk.

Eksperimenter indikerer at metodene direkte kollokasjon og single shooting er betydelig raskere enn multiple shooting implementasjonen. Et sett av tester for produksjonsoptimalisering har blitt vurdert og tyder på en økning på $0.5 - 3.2\%$ i produksjonsrater ved å følge en uoppnåelig referanseverdi, noe som leder til stasjonær oppførsel. Ved maksimering av objektivfunksjonen over 12 timer, vil den foreslåtte løsningen øke objektivfunksjonsverdien med $3.4 - 15\%$.

Contents

Abbreviations

AD	Automatic Differentiation
AM	Adams-Moulton
BDF	Backward Differentiation Formula
CasADi	Computer Algebra System for Automatic Differentiation
DAE	Differential Algebraic Equation
DC	Direct Collocation
DOP	Dynamic Optimization Problem
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
IP	Interior Point
IPOPT	Interior Point Optimizer
IRK	Implicit Runge-Kutta
IVP	Initial Value Problem
JMI	JModelica Model Interface
MPC	Model Predictive Control
MS	Multiple Shooting
NLP	Non Linear Program
OC	Optimal Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
RTO	Real Time Optimization
SQP	Sequential Quadratic Programming
SS	Single Shooting

Chapter 1

Introduction

This chapter gives a brief introduction to production optimization in oil gathering systems, describing the difference between a static and dynamic approach. Improvement of operational performance of a plant can be achieved through utilization of capacity and production planning. This motivates application of real-time optimization as a decision support tool described in Section 1.1. Static real-time optimization has been tested with excellent results [19, 20] and several commercial products for this purpose are available [18]. In this thesis, a dynamic approach for production optimization is explored, combining dynamic models with optimal control to improve operational performance.

In Section 1.2, a historical overview of dynamic optimization based on [2] is given, presenting development in this area from the 1950's until today. Several methods for solving optimal control problems exist, where direct methods are treated in this thesis.

Finally, Section 1.3 provides the scope and emphasis of this thesis, giving an outline for the work that has been done.

1.1 Production optimization in oil gathering systems

An oil gathering system consists of a flowline network and processing facilities used to transport flow of oil and gas from wells to a temporary storage facility. For example, in a subsea petroleum production plant, the fluid flow is directed from wells through manifolds and flowlines, to the topside separator, as illustrated by Figure 1.1. Wells can be naturally flowing or artificially lifted, where the latter is necessary when the down hole pressure is too low [15]. Among the artificial lifting techniques, gas lift is widely used [10] and provides means to maintain or increase production by injecting lift-gas in the bottom of the well. However, large gas injection rates can incur in expensive operations and production loss due to excessive friction.

Lift-gas injection rates should be decided to control well flows in terms of production rates. In addition, the routing from wells to separator leads to a non-trivial decision problem, where the routing configuration may have a great impact on production performance. Controlling valve openings and routing configurations are decisions to be taken by the operator in order to maintain or improve production rates. In large oil fields, the amount of decisions can be extensive, motivating the implementation of an optimization tool to assist the operator to perform this task.

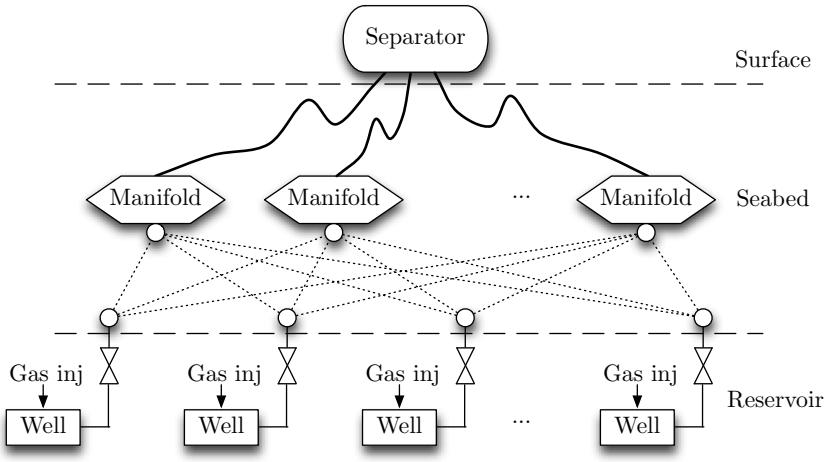


Figure 1.1: Cluster topology of a well-flowline network.

The development and operation of a petroleum field asset requires planning and decision taking on several horizons. It is natural to divide these decisions into their time scales according to Figure 1.2. This thesis considers Production Optimization, that is decisions on a short-term time horizon. Here, decisions regarding production and possibly injection rates as well as routing configurations have to be taken, where the goal is to maximize daily production rates or to keep production at specified rates [18]. For small time scales, effects such as changes in reservoir conditions are not considered, as these are treated by higher layers in the control hierarchy, emphasizing long term recovery strategies.

The incorporation of automatic control combined with real-time measurements and the exploitation of data information generated from a production plant are key procedures in what is called Integrated Operations. This seeks to improve operational performance of a production plant by assisting the operator in complex decisions by providing a decision support tool. A term covering these features is known as *Real Time Optimization* (RTO), which can be described as *a method for complete or partial automation of the process of finding good (optimal) control settings* [10].

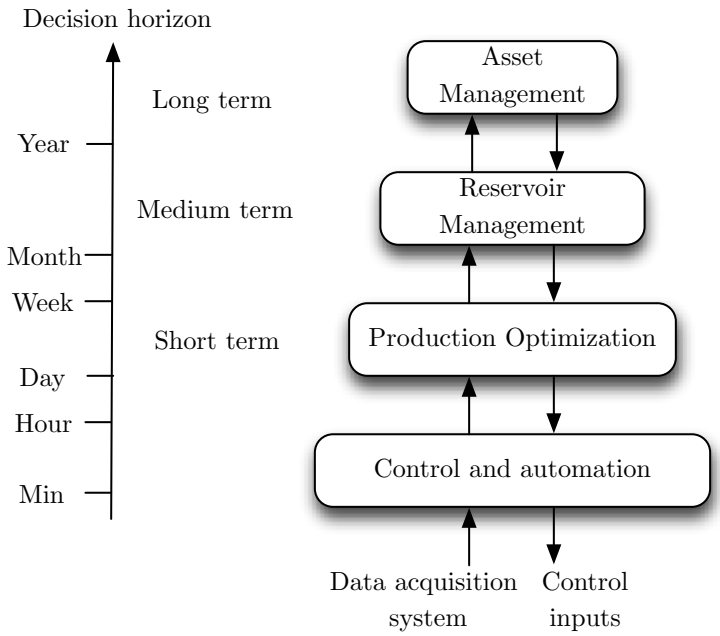
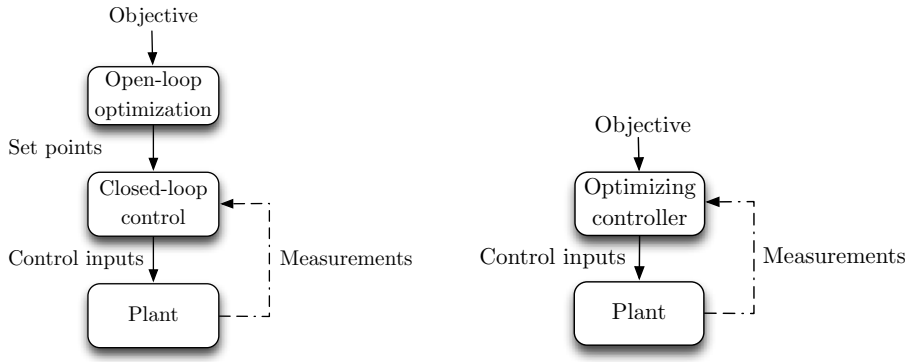


Figure 1.2: Multi-level control hierarchy [18].

Optimal production strategies propose ways to utilize the oil field capacity to avoid bottlenecks and improve overall operation. This is usually achieved through mathematical modeling of the plant, giving a relation between control inputs and measured outputs, enabling optimization-based control. In a RTO context, the models used for this purpose are in most cases static, meaning that they are valid for steady-state analysis only. However, a steady-state optimization approach has showed industrial success through commercial products like GAP by Petroleum Experts and MaxPro by FMC Technologies [18]. Problems that can be analyzed using this methodology are for instance decision of production valve settings, lift-gas injection rates and well-flowline routing configuration [14, 21]. RTO appears as static optimization problems based on linear or nonlinear models with possibly a mixed-integer formulation, where the latter appears in the case of well-flowline routing decisions.

From a short-term production planning view, RTO is a decision support tool aimed to improve operational performance, e.g. by offering optimized control settings to the operator for bringing the overall plant closer to an optimum. RTO solutions lacking dynamic models can propose infeasible transient operation. A dynamic description must be used to predict this behavior and control the plant to obey constraints and limitations. A suitable framework for this purpose can be taken from the theory of dynamic optimization enabling optimization of dynamic models to produce time-varying control settings, which then can be applied to the plant. These can be applied in an open-loop manner or in closed-loop by including measurement feedback, as illustrated by Figure 1.3. The latter is used in solutions such as *Model Predictive Control* (MPC).



(a) Open-loop optimization separated from closed-loop control.

(b) Integrated optimization and control (closed-loop optimization).

Figure 1.3

1.2 Dynamic optimization

In dynamic optimization, the goal is to find solutions of optimization problems constrained by dynamic models represented by *ordinary differential equations* (ODE) or *differential algebraic equations* (DAE). The *optimal control problem* (OCP) is concerned with finding inputs in order to control a system where a specified objective function is optimized. These problems apply to a wide area of processes ranging from aerospace to chemical process industry. An important application is within MPC, where dynamic optimization is combined with a receding horizon principle to form a closed-loop feedback controller where optimization is performed on-line. An overview of optimal control methods is illustrated in Figure 1.4.

In 1957, the principle of optimality was formulated by R. Bellman, which also showed that dynamic programming was applicable for deriving an optimal control law from the solution of the Hamilton-Jacobi-Bellman equation. However, using this approach for large OCPs, the presence of nonlinear dynamics and state or control variable constraints may lead to computationally intractable problems.

Another approach extending results from the area of calculus of variations is the maximum principle formulated in 1962 by L. Pontryagin and co-workers, known as indirect methods. Although providing an analytical approach for solving the OCP, disadvantages include difficulties for handling state bounds and large nonlinear systems.

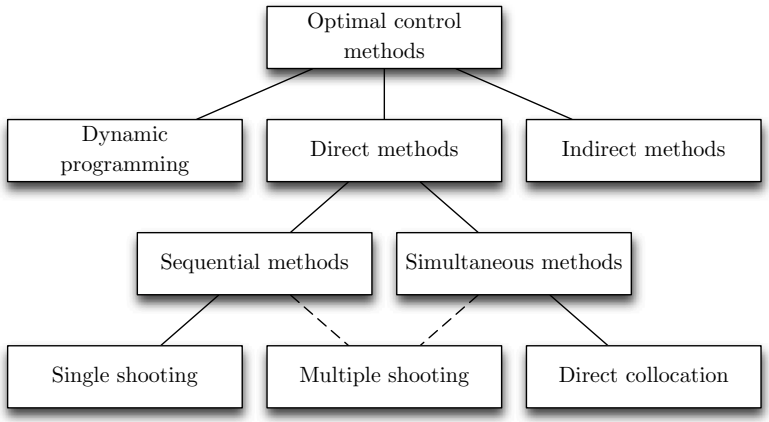


Figure 1.4: Dynamic optimization overview.

Direct methods form a family that has been developed during the last two decades. These are based on approximating the infinite dimensional dynamic optimization problem into a finite dimensional static problem, as illustrated by Figure 1.5. Approaches within the category of direct methods differ in the way the finite dimensional optimization problem is formulated.

Sequential (single shooting) methods address this problem in a reduced space treating only control inputs as decision variables. This is known as partial discretization [13], making states implicitly defined by the control inputs. In this way, simulation and optimization is done in a sequential manner.

Simultaneous methods formulate the optimization problem using both states and control inputs as decision variables, leading to a full space problem. In this case, a full discretization of all variables occurs. Although leading to a significantly larger *nonlinear program* (NLP) in terms of decision variables and constraints, the problem is sparse, and structure can be exploited to improve efficiency. A simultaneous method known as direct collocation is based on approximating variable profiles by piecewise polynomials and solving the model equations at specific points in time.

Multiple shooting can be viewed as a combination of sequential and simultaneous methods, discretizing both states and controls over several time segments, where each of these are integrated independently.

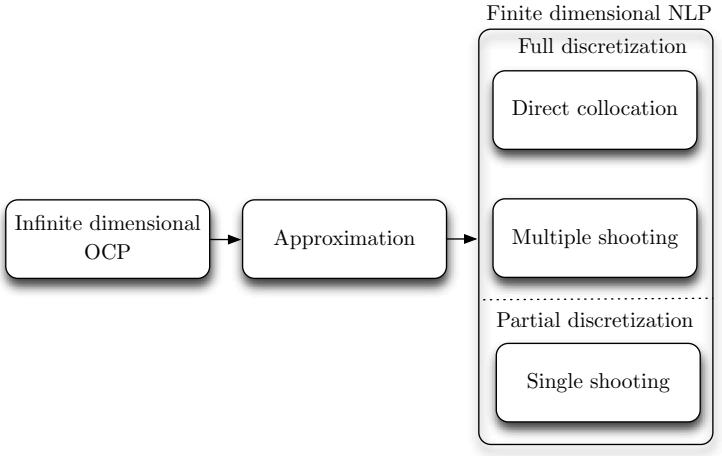


Figure 1.5: The infinite dimensional OCP is approximated by a finite dimensional NLP after the application of an approximation procedure.

1.3 Scope and emphasis

This thesis will motivate the use of dynamic optimization applied to an oil gathering system consisting of a well-flowline network. Chapter 2 presents an optimal control problem formulation constrained by a system of continuous-time nonlinear differential algebraic equations. This problem is approached by means of three methods: direct collocation, single shooting and multiple shooting, all of which are briefly described here.

In Chapter 3, an overview of the JModelica.org framework is given. Implementations of shooting and direct collocation methods are illustrated by an example. The well-flowline model is presented in Chapter 4, summarizing the amount of variables and equations. An optimal control problem for this model is formulated in Chapter 5.

A performance assessment description is provided in Chapter 6 with all results given in Appendix D. Finally, Chapter 7 presents a discussion of the results followed by a conclusion in Chapter 8.

Chapter 2

Optimization and control of dynamic systems

Dynamic optimization methods presented in this thesis are based on a direct approach. In Section 2.1, a general optimal control problem constrained by DAE system equations is defined. Different formulations and the relation between them are explained.

Section 2.2-2.4 presents the direct methods: single shooting, multiple shooting and direct collocation, with theory based on [9]. These approximate the infinite dimensional optimal control problem by a finite dimensional NLP that is suited for numerical algorithms. Shooting methods rely on embedded integrators with sensitivity capabilities for obtaining variable profiles and derivative information. The direct collocation method is based on orthogonal collocation over finite elements with Gauss-Radau interpolation points and Lagrange polynomials.

To improve performance of large-scale optimization algorithms, efficient derivative computations are necessary, where automatic differentiation is presented in Section 2.5. This technique provides partial derivatives of system equations and constraints with accuracy up to machine precision by applying the chain rule and using pre-defined look-up tables for derivatives of elementary functions.

2.1 Optimal control problem formulation

Dynamic optimization problems (DOP) are concerned with the solution of decision-making problems constrained by differential or differential-algebraic equations. A DOP can be further categorized into parameter estimation and optimal control problems [24], where the latter is considered here.

The process model used in this thesis can be represented by a set of fully implicit DAEs written as an *initial value problem* (IVP):

$$F(x(t), \dot{x}(t), u(t), t) = 0, \tag{2.1a}$$
$$h(x(t_0)) = 0, \tag{2.1b}$$

where $x : [t_0, t_f] \to \mathbb{R}^{n_x}$ are states, $u : [t_0, t_f] \to \mathbb{R}^{n_u}$ are control inputs, $F : \mathbb{R}^{n_x \times n_x \times n_u \times 1} \to \mathbb{R}^{n_x}$ are system functions and $x(t_0)$ are initial conditions. Start and end time are given by t_0 and t_f respectively.

A more structured formulation of the DAE is given by the semi-explicit form, where $x(t)$ is partitioned into differential states $z(t)$ and algebraic variables $y(t)$. Further, a time-invariant system is considered, such that the time variable t does not appear explicitly in the system equations. A semi-explicit DAE is written as:

$$\dot{z}(t) = f(z(t), y(t), u(t)), \tag{2.2a}$$
$$z(t_0) = z_0, \tag{2.2b}$$
$$0 = g(z(t), y(t), u(t)), \tag{2.2c}$$

where $z : [t_0, t_f] \to \mathbb{R}^{n_z}$ and $y : [t_0, t_f] \to \mathbb{R}^{n_y}$. Differential and algebraic equations are given by $f : \mathbb{R}^{n_z \times n_y \times n_u} \to \mathbb{R}^{n_z}$ and $g : \mathbb{R}^{n_z \times n_y \times n_u} \to \mathbb{R}^{n_y}$, while $z(t_0)$ are initial conditions.

Further, assuming invertibility of $g(z(t), y(t), u(t))$ or equivalently requiring that $\frac{\partial g}{\partial y}$ is nonsingular for all $z(t), y(t), u(t)$, this allows implicit elimination of the algebraic variables $y(t)$. In this way, the DAE may be treated as an ODE such that (2.2) can be written as an equivalent ODE IVP:

$$\dot{z}(t) = f(z(t), y[z(t), u(t)], u(t)) = \overline{f}(z(t), u(t)), \tag{2.3a}$$
$$z(t_0) = z_0. \tag{2.3b}$$

This corresponds to an index-1 property of the DAE system (2.1). Existence and uniqueness of the solution of (2.3) can be guaranteed by Theorem 8.1 in [9].

A general OCP formulation based on the semi-explicit DAE system (2.2) can be written as:

$$\min_{z,y,u} \ \Phi(z,y,u) = \int_{t_0}^{t_f} L(t,z,y,u)dt + E(t_f, z(t_f), y(t_f), u(t_f)) \tag{2.4a}$$

$$\text{s.t. } \dot{z} = f(z,y,u), \tag{2.4b}$$

$$z(t_0) = z_0, \tag{2.4c}$$

$$g(z,y,u) = 0, \tag{2.4d}$$

$$g_I(z,y,u) \leq 0, \tag{2.4e}$$

$$z_L \leq z \leq z_U, \tag{2.4f}$$

$$y_L \leq y \leq y_U, \tag{2.4g}$$

$$u_L \leq u \leq u_U, \tag{2.4h}$$

$$t \in [t_0, t_f].$$

DAE system equations are given as equality constraints in (2.4b)-(2.4d). The constraints (2.4e) are known as path inequalities, with $g_I : \mathbb{R}^{n_z \times n_y \times n_u} \to \mathbb{R}^{n_I}$. Variable bounds are given in (2.4f)-(2.4h). Additional point or end time constraints may also be added. The variables z, y, u are all functions of time t even though the explicit time argument notation is for simplicity not used further.

The objective function (2.4a) consists of two parts, namely an integral term accounting for the cost throughout the horizon $[t_0, t_f]$, and a terminal cost at the final time. An objective function containing only the first part

$$\int_{t_0}^{t_f} L(t,z,y,u)dt \tag{2.5}$$

leads to a Lagrange problem where $L(t,z,y,u)$ is called a Lagrange integrand, while the Mayer problem contains only the terminal cost

$$E(t_f, z(t_f), y(t_f), u(t_f)). \tag{2.6}$$

A combination of these two yields a Bolza functional, and the OCP arising from such a formulation is known as the Bolza problem. It should be noted that these formulations are equivalent, and they may be interchanged. For instance, the Bolza functional in (2.4a) may be written in Mayer form by defining an additional state

$$\dot{z}_c(t) = L(t,z,y,u) \tag{2.7}$$

with $z_c(t_0) = 0$. In this way, integration of (2.7) leads to

$$z_c(t) = \int_{t_0}^{t} L(t,z,y,u)dt \tag{2.8}$$

and by setting $t = t_f$, the objective function (2.4a) can be written as

$$\Phi(t_f) = z_c(t_f) + E(t_f, z(t_f), y(t_f), u(t_f)), \tag{2.9}$$

where $z_c(t_f)$ may be incorporated into the function E, yielding a Mayer type cost function. In this way, the Lagrange integrand becomes an additional state. The resulting OCP can then be written as:

$$\begin{aligned}
\min_{z,y,u} \ & \Phi(t_f) = E(t_f, z(t_f), y(t_f), u(t_f)) && \text{(2.10a)}\\
\text{s.t. } & \dot{z} = f(z, y, u), && \text{(2.10b)}\\
& z(t_0) = z_0, && \text{(2.10c)}\\
& g(z, y, u) = 0, && \text{(2.10d)}\\
& \dot{z}_c = L(t, z, y, u), && \text{(2.10e)}\\
& z_c(t_0) = 0, && \text{(2.10f)}\\
& g_I(z, y, u) \leq 0, && \text{(2.10g)}\\
& z_L \leq z \leq z_U, && \text{(2.10h)}\\
& y_L \leq y \leq y_U, && \text{(2.10i)}\\
& u_L \leq u \leq u_U, && \text{(2.10j)}\\
& t \in [t_0, t_f].
\end{aligned}$$

Here, DAE system equations consist of model equations (2.10b)-(2.10d) and the additional cost state (2.10e)-(2.10f).

As the OCP includes a dynamic constraint in the form of a continuous-time differential equation, it contains an infinite number of decision variables: $z(t), y(t), u(t)$ for $t \in [t_0, t_f]$. To solve it using a numerical optimization algorithm, it has to be approximated by a finite dimensional NLP. An approximation procedure discretizes the continuous-time formulation, into a discrete formulation making it suited for numerical algorithms. The methods presented in Section 2.2-2.4 involve different techniques of this procedure.

2.2 Single shooting

In the OCP (2.10), the decision variables consist of states, algebraic variables and control inputs. However, from the DAE system equations (2.10b)-(2.10f), states and algebraic variable profiles may be obtained by DAE integration. In this way, states and algebraic variables can be treated as implicit functions of control inputs. That is, given a sequence of $u(t)$, it is possible to obtain the profiles $z(t)$ and $y(t)$. These are then eliminated from the optimization problem, leaving only the control inputs left as decision variables, leading to a reduced space problem. This method is commonly known as single shooting.

The sequential manner of this method is in the sense that integration and optimization are performed sequentially. An integrator provides variable profiles to the NLP solver, which in turn optimizes the objective function by producing a new control input sequence, as illustrated by Figure 2.1. NLP optimization algorithms depend on derivatives of objective and constraint functions with respect to control inputs. These are referred to as sensitivity calculations and can be performed in 3 ways: perturbation, direct (forward) sensitivity and adjoint sensitivity. As the perturbation method is based on a finite difference approximation, this method is the simplest to implement. However, it leads to problems such as truncation and round-off errors which limits the accuracy of derivatives, therefore it will not be treated further.

The single shooting method is convenient due to its simplicity and reduced size NLP. In addition, as the dynamic model constraints are fulfilled with high accuracy in each iteration, the optimization procedure can be terminated prematurely, resulting in a suboptimal control profile, which may still be an improvement over the initial guess [2]. However, disadvantages include difficulty of handling unstable systems and state path constraints.

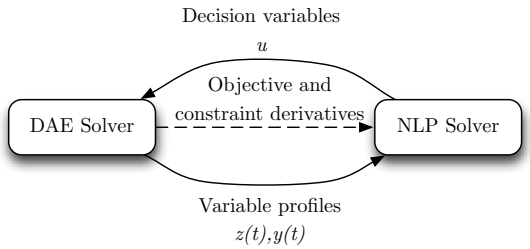


Figure 2.1: Sequential method strategy. DAEs are solved in an outer loop providing variable profiles and derivative information to the NLP solver, which in turn uses these to compute a new sequence of control inputs u.

Formulation of a single shooting NLP

Given the OCP (2.10), control inputs need to be discretized into a finite amount of parameters making them decision variables in the resulting NLP. A common parameterization for these is a piecewise constant control input given by

$$u(t) = u^l, t \in \left(\frac{t_f(l-1)}{N}, \frac{t_f l}{N} \right], l = 1, ..., N \tag{2.11}$$

where N is the amount of intervals where control inputs are kept constant and t_f is final time, as illustrated by Figure 2.2. Other parameterizations may also be used, e.g. a piecewise linear or polynomial representation. In this way, the control inputs become discrete. The OCP (2.10) can be approximated by the following NLP with $U = \{u^l \in \mathbb{R}^{n_u}, l = 1, ..., N\}$ addressing a single shooting problem:

$$\min_{U \in \mathbb{R}^{N n_u}} \varphi(U) \tag{2.12a}$$

$$\text{s.t.} \quad c_E(U) = 0, \tag{2.12b}$$

$$c_I(U) \leq 0, \tag{2.12c}$$

where the DAE system (2.10b)-(2.10f) is solved by an integrator making it possible to view the objective and constraint functions in (2.12) as implicit functions of U, that is:

$$\varphi(U) = \varphi(z(U), y(U), U), \tag{2.13}$$

$$c_E(U) = c_E(z(U), y(U), U), \tag{2.14}$$

$$c_I(U) = c_I(z(U), y(U), U). \tag{2.15}$$

The NLP arising from a single shooting formulation has Nn_u decision variables, and can be solved by an appropriate solver, e.g. algorithms based on *Sequential Quadratic Programming* (SQP) or *Interior Point* (IP) methods. In this thesis, IPOPT [29] is used, which is a large-scale interior point optimizer for mathematical optimization problems of the form:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.16a}$$

$$\text{s.t.} \quad g_L \leq g(x) \leq g_U, \tag{2.16b}$$

$$x_L \leq x \leq x_U, \tag{2.16c}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are objective and constraint functions respectively, with x being decision variables. Since the single shooting NLP (2.12) can be formulated as (2.16), a local solution of the optimization problem can be found. SQP and IP methods require derivative information of both objective and constraint functions. These computations are known to be the most time-consuming part of a single shooting algorithm [22].

Gradient computation and sensitivity calculations

To improve efficiency of the NLP solver for finding a solution of the single shooting problem (2.12), the gradients $\nabla_U \varphi$, $\nabla_U c_E$ and $\nabla_U c_I$ are required. Let Ψ contain objective and constraint functions:

$$\Psi = \begin{bmatrix} \varphi(U) & c_E(U)^T & c_I(U)^T \end{bmatrix}. \tag{2.17}$$

Consider a single time period with only one parameterized control input $u(t) = u,\ t \in [0, t_f]$. By applying the chain rule together with (2.13)-(2.15) and (2.17), the gradient can be written as:

$$\nabla_u \Psi = \begin{bmatrix} \frac{\partial \varphi}{\partial u} & \frac{\partial c_E^T}{\partial u} & \frac{\partial c_I^T}{\partial u} \end{bmatrix} \tag{2.18}$$

$$= \begin{bmatrix} \frac{dz}{du}\frac{\partial \varphi}{\partial z} + \frac{dy}{du}\frac{\partial \varphi}{\partial y} + \frac{\partial \varphi}{\partial u} & \frac{dz}{du}\frac{\partial c_E^T}{\partial z} + \frac{dy}{du}\frac{\partial c_E^T}{\partial y} + \frac{\partial c_E^T}{\partial u} & \frac{dz}{du}\frac{\partial c_I^T}{\partial z} + \frac{dy}{du}\frac{\partial c_I^T}{\partial y} + \frac{\partial c_I^T}{\partial u} \end{bmatrix} \tag{2.19}$$

$$= S(t_f)^T \frac{\partial \Psi}{\partial z} + R(t_f)^T \frac{\partial \Psi}{\partial y} + \frac{\partial \Psi}{\partial u}, \tag{2.20}$$

where $S(t) = \frac{dz}{du}^T$ and $R(t) = \frac{dy}{du}^T$ are known as the sensitivity matrices.

Direct sensitivity methods are based on taking the derivative of DAE system equations with respect to u, that is for $t_0 = 0$:

$$\frac{d}{du} \begin{cases} \dfrac{dz}{dt} = f(z, y, u), \\ z(0) = z_0, \\ g(z, y, u) = 0. \end{cases} \tag{2.21}$$

This leads to the sensitivity equations:

$$\frac{dS}{dt} = \frac{\partial f}{\partial z}^T S(t) + \frac{\partial f}{\partial y}^T R(t) + \frac{\partial f}{\partial u}, \tag{2.22a}$$

$$S(0) = \frac{\partial z_0}{\partial u}^T, \tag{2.22b}$$

$$0 = \frac{\partial g}{\partial z}^T S(t) + \frac{\partial g}{\partial y}^T R(t) + \frac{\partial g}{\partial u}^T, \tag{2.22c}$$

resulting in $(n_z + n_y)n_u$ differential equations, which are linear time-variant.

Another method for obtaining the derivatives of objective and constraint functions are adjoint sensitivity calculations, which is described in [9].

The direct sensitivity method is efficient with few decisions u and many constraints, while the adjoint method is more efficient for many decision variables u and few constraints, thus these methods complement each other. Both direct and adjoint sensitivity methods require computation of partial derivatives related to objective and constraint functions. For this task, it is possible to use automatic differentiation presented in Section 2.5.

Efficient DAE/ODE integrators with sensitivity capabilities improve the performance of the single shooting approach. In this thesis CVODES [28] is used, which implements an ODE integrator providing forward and adjoint sensitivity analysis. As the DAE system and sensitivity equations share the same Jacobian matrices, this fact is exploited such that sensitivity equations are solved simultaneously with the state equations. Other improvements include taking advantage of sparsity and structure of the Jacobian, which are common features in large-scale DAE/ODE solvers.

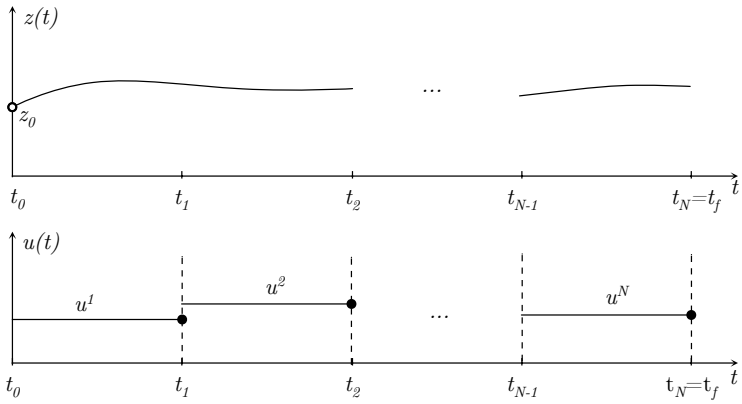


Figure 2.2: In single shooting, variable profiles are obtained by solving a DAE system with parameterized control inputs.

2.3 Multiple shooting

The single shooting approach leads to a reduced space NLP by efficiently eliminating state and algebraic variables through an embedded integrator. However, this strategy is not well suited for unstable dynamic systems as it can lead to unbounded state profiles, or convergence difficulties in the NLP solver. These problems can be avoided by including state variable information into the NLP formulation.

Multiple shooting can be viewed as an extension of the single shooting approach. Here, the time horizon is divided into a number of segments, as illustrated in Figure 2.3. In each of these segments, the DAE system is solved independently. To enforce continuity of the state profiles, additional state equality constraints are introduced to ensure that the defects: $d^l = z^{l-1}(t_{l-1}) - z_0^l \ , \ l = 2, ..., N$ are equal to zero. As for single shooting, control inputs are parameterized into a finite amount of variables. In addition, states at the beginning of each segment appear as decision variables. As integration is performed over shorter time intervals, numerical stability properties of the algorithm are improved. Additional inequality constraints for the state variables at the grid points $t_l \ , \ l = 1, ..., N$ are also easier to include, as these are now decision variables.

Although multiple shooting offers numerical advantages over a single shooting approach, it should be emphasized that the resulting NLP becomes larger and more sensitivity information is required. To improve performance, exploitation of structure and parallelization should be applied.

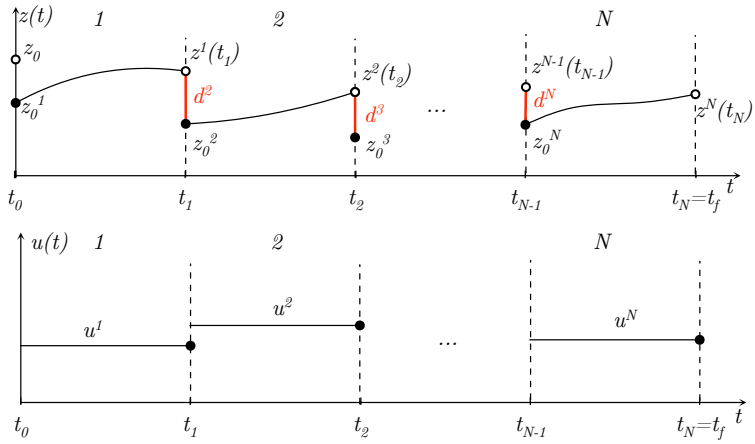


Figure 2.3: A multiple shooting time horizon is divided into segments where each one is integrated separately.

Let the time horizon be divided into $l = 1, ..., N$ segments, where these are connected at the grid points t_l. A multiple shooting approach leads to a NLP with DAE integration solved in an outer loop, and can be formulated as:

$$\begin{aligned}
\min_{V \in \mathbb{R}^{n_v}} \quad & \varphi(z^l(t_l), y^l(t_l), u^l) && \text{(2.23a)}\\
\text{s.t.} \quad & z^{l-1}(t_{l-1}) - z_0^l = 0 \ , \ l = 2, ..., N && \text{(2.23b)}\\
& z_0 - z_0^1 = 0, && \text{(2.23c)}\\
& g_I(z^l(t_l), y^l(t_l), u^l) \leq 0, && \text{(2.23d)}\\
& u_L \leq u^l \leq u_U, && \text{(2.23e)}\\
& y_L \leq y^l(t_l) \leq y_U, && \text{(2.23f)}\\
& z_L \leq z^l(t_l) \leq z_U, && \text{(2.23g)}\\
& l = 1, ..., N, \\
& \dot{z}^l = f(z^l(t), y^l(t), u^l), && \text{(2.23h)}\\
& z^l(t_{l-1}) = z_0^l, && \text{(2.23i)}\\
& g(z^l(t), y^l(t), u^l) = 0, && \text{(2.23j)}\\
& t \ \in \ (t_{l-1}, t_l] \ , \ l = 1, ..., N.
\end{aligned}$$

Eq (2.23b) enforces continuity of state profiles between segments. Initial conditions are defined in (2.23c), where z_0 are given values. Inequality constraints (2.23d) and variable bounds (2.23e)-(2.23g) are posed directly at grid points. In addition, the DAE system (2.23h)-(2.23j) is solved independently for each segment in an outer loop, thus obtaining state and variable profiles.

Here, decision variables are states at the beginning of each segment with corresponding parameterized control inputs, that is:

$$V = \{(z_0^l, u^l) | z_0^l \in \mathbb{R}^{n_z}, u^l \in \mathbb{R}^{n_u}, l = 1, ..., N\}. \tag{2.24}$$

For an equal segment and control input discretization, this lead to a total number of $n_v = N(n_z + n_u)$ decision variables and Nn_z additional equality constraints from the continuity enforcement and initial condition.

2.4 Direct collocation

Within simultaneous methods for dynamic optimization, the direct collocation method is based on a NLP formulation without embedding a DAE integrator. Instead, variable profiles are approximated by piecewise polynomials throughout the time horizon. The infinite dimensional OCP is then approximated into a finite dimensional NLP, consisting of algebraic equations only, which can be solved by numerical algorithms. In this representation, states, algebraic variables and control inputs are decision variables, making it straightforward to define constraints on these. The resulting NLP for collocation is large because of its amount of decision variables and constraints arising from the collocation equations. However, due to its sparsity and structure, it is essential to exploit this in order to improve efficiency.

It should be noted that collocation itself is a numerical method for the solution of a DAE system, and can be used for simulation purposes [24]. Such an approximation scheme can be shown to be equivalent to a special class of *Implicit Runge-Kutta* (IRK) methods as presented in [9], inheriting the numerical stability properties of these.

Unlike shooting approaches that obtain variable profiles by integration of DAE system equations, collocation is based on approximating these profiles with polynomials. Collocation parameters should be chosen to yield a good approximation. A trade-off between a fine grained vs. coarse approximation arise from the choice of parameter values, which can have a great impact on solution time.

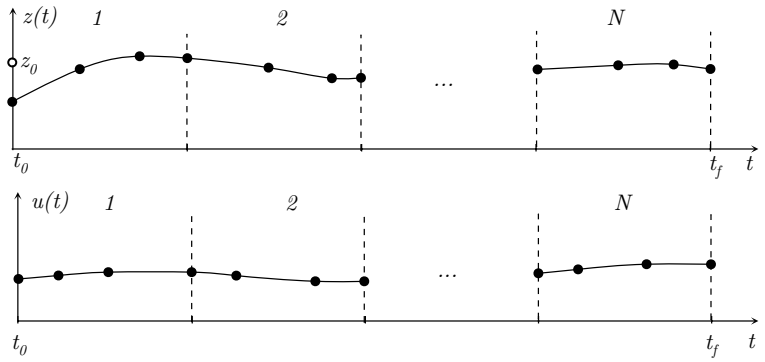


Figure 2.4: Decision variables are discrete points (interpolation points) marked by the dots, with interpolated variable profiles based on polynomial approximations.

2.4.1 Collocation - A numerical method for solving an ODE

Consider the following ODE where $z(t)$ are differential states:

$$\dot{z}(t) = f(z(t), t), \tag{2.25a}$$
$$z(0) = z_0. \tag{2.25b}$$

Let the time horizon be divided into N elements with index $i = 1, ..., N$, each with length h_i, as illustrated in Figure 2.4. Within each element, there are $K + 1$ interpolation points[1] with index $k = 0, ..., K$. In this way, an interpolation point can be denoted by τ_k, see Figure 2.5. The solution of (2.25) can be represented as a polynomial approximation of degree K within each element, denoted by $z^K(t)$. In collocation methods, representations based on Lagrange interpolation polynomials are used. In the following, consider a one-element approximation ($N = 1$) for element i:

$$\left.\begin{aligned} z^K(t) &= \sum_{j=0}^{K} \ell_j(\tau) z_{i,j}, \\ t &= t_{i-1} + h_i \tau, \end{aligned}\right\} \qquad t \in [t_{i-1}, t_i], \tau \in [0, 1], \tag{2.26}$$

where $\ell_j(\tau)$ are called Lagrange polynomials. These are defined as

$$\ell_j(\tau) = \prod_{k=0, k\neq j}^{K} \frac{\tau - \tau_k}{\tau_j - \tau_k}, \tag{2.27}$$

with $\tau_0 = 0, \tau_j < \tau_{j+1}, j = 0, ..., K - 1$.

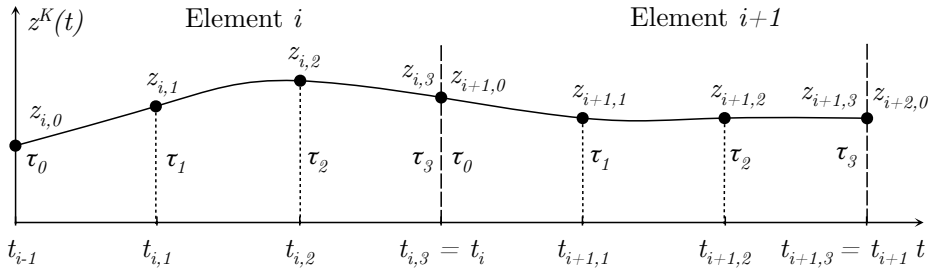


Figure 2.5: Two elements with $K = 3$. At the element boundaries, interpolation points overlap, enforcing continuity of variable profiles. The value of τ_k depends on K and collocation type.

[1] Also known as collocation points.

Enforcing the model equation (2.25a) at the interpolation points τ_k or equivalently $t_{i,k}$ leads to

$$\dot{z}^K(t_{i,k}) = f(z^K(t_{i,k}), t_{i,k}) \ , k = 1, ..., K. \tag{2.28}$$

Since the Lagrange polynomial representation has the property that $z^K(t_{i,j}) = z_{i,j}$ where $t_{i,j} = t_{i-1} + \tau_j h_i$, (2.28) becomes

$$\dot{z}^K(t_{i,k}) = f(z_{i,k}, t_{i,k}) \ , k = 1, ..., K. \tag{2.29}$$

Further, the following property is applied in (2.31):

$$\frac{dz^K}{d\tau} = h_i \frac{dz^K}{dt}. \tag{2.30}$$

Taking the derivative of $z(t)$ in (2.26) w.r.t τ and inserting (2.29) and (2.30) yields:

$$\frac{dz^K}{d\tau}(t_{i,k}) = \sum_{j=0}^{K} \frac{d\ell_j}{d\tau}(\tau_k) z_{i,j}, \ k = 1, ..., K \tag{2.31}$$

$$h_i \frac{dz^K}{dt}(t_{i,k}) = \sum_{j=0}^{K} \frac{d\ell_j}{d\tau}(\tau_k) z_{i,j}, \ k = 1, ..., K \tag{2.32}$$

$$h_i f(z_{i,k}, t_{i,k}) = \sum_{j=0}^{K} \frac{d\ell_j}{d\tau}(\tau_k) z_{i,j}, \ k = 1, ..., K \tag{2.33}$$

where (2.33) are known as the collocation equations. After determining the interpolation points τ_k, the collocation equations and initial condition (2.25b) result in a nonlinear system of equations which can be incorporated directly into an NLP formulation.

For multiple elements ($N > 1$), it is also necessary to enforce continuity of the state profiles across element boundaries. With Lagrange interpolation profiles, this is given as

$$z_{i+1,0} = \sum_{j=0}^{K} \ell_j(1) z_{i,j}, \ i = 1, ..., N-1 \tag{2.34}$$

where the first interpolation point in element $i + 1$ should be equal to the last interpolation point in the previous element to enforce continuity. Initial conditions are written as

$$z_{1,0} = z_0. \tag{2.35}$$

Determination of the values for τ_k is aimed to give the most accurate approximation of the states and depends on the type of collocation scheme used [9]. Further in this thesis, Gauss-Radau collocation will be used, due to the compatibility with the NLP formulation and stability properties.

2.4.2 NLP formulation from direct collocation

As the derivation in Section 2.4.1 is based on an ODE, a further generalization treating a semi-explicit DAE can be done by representing algebraic variables and control inputs by Lagrange interpolation profiles:

$$y(t) = \sum_{j=1}^{K} \overline{\ell}_j(\tau) y_{i,j}, \tag{2.36}$$

$$u(t) = \sum_{j=1}^{K} \overline{\ell}_j(\tau) u_{i,j}, \tag{2.37}$$

where

$$\overline{\ell}_j(\tau) = \prod_{k=1, k\neq j}^{K} \frac{\tau - \tau_k}{\tau_j - \tau_k}. \tag{2.38}$$

That is, given points $z_{i,j}, y_{i,j}$ and $u_{i,j}$, the interpolation equations (2.26), (2.36) and (2.37) are used to construct interpolated variable profiles.

For a DAE system, collocation equations and algebraic system equations can be written as:

$$\sum_{j=0}^{K} \frac{d\ell_j}{d\tau}(\tau_k) z_{i,j} - h_i f(z_{i,k}, y_{i,k}, u_{i,k}) = 0, \tag{2.39}$$

$$g(z_{i,k}, y_{i,k}, u_{i,k}) = 0, \tag{2.40}$$

$$i \in 1, ..., N \quad , \quad k \in 1, ..., K.$$

In this way, an infinite dimensional OCP can be approximated into a finite dimensional NLP with a total number of $n_w = (n_z + n_y + n_u)NK + Nn_z$ decision variables being:

$$z_{i,k} \in W \quad i = 1, ..., N \ , \ k = 0, ..., K \tag{2.41}$$

$$y_{i,k} \ , \ u_{i,k} \in W \quad i = 1, ..., N \ , \ k = 1, ..., K \tag{2.42}$$

where $W \in \mathbb{R}^{n_w}$ is the set of decision variables.

In addition, the collocation equations (2.39), algebraic system equations (2.40) and continuity equations (2.34) adds a number of $NK(n_z + n_y) + (N-1)n_z$ equality constraints. The resulting NLP from direct collocation can be formulated as:

$$\min_{W \in \mathbb{R}^{n_w}} \ \varphi(W) \tag{2.43a}$$

$$\text{s.t.} \ \sum_{j=0}^{K} \frac{d\ell_j}{d\tau}(\tau_k) z_{i,j} - h_i f(z_{i,k}, y_{i,k}, u_{i,k}) = 0, \tag{2.43b}$$

$$g(z_{i,k}, y_{i,k}, u_{i,k}) = 0, \tag{2.43c}$$

$$g_I(z_{i,k}, y_{i,k}, u_{i,k}) \leq 0, \tag{2.43d}$$

$$z_L \leq z_{i,k} \leq z_U, \tag{2.43e}$$

$$y_L \leq y_{i,k} \leq y_U, \tag{2.43f}$$

$$u_L \leq u_{i,k} \leq u_U, \tag{2.43g}$$

$$k \in 1, ..., K \quad , \quad i \in 1, ..., N$$

$$z_{i+1,0} = \sum_{j=0}^{K} \ell_j(1) z_{i,j}, \quad i = 1, ..., N-1 \tag{2.43h}$$

$$z_{1,0} = z_0. \tag{2.43i}$$

Eq (2.43b) are the collocation equations for the DAE, together with the algebraic system equations (2.43c). Path inequalities are posed at the interpolation points in (2.43d). Variable bounds are given as (2.43e)-(2.43g). Continuity of state profiles is enforced by (2.43h), while initial conditions are finally defined in (2.43i), where z_0 are given values.

Alternative NLP formulations based on direct collocation may yield a different amount of decision variables and equality constraints. The collocation algorithm in JModelica.org as presented in [24] has: $(2n_z + n_y + n_u)(1 + NK) + (N-1)n_z$ decision variables and $(1 + NK)(2n_z + n_y) + (N-1)n_z + n_u$ equality constraints from DAE system and collocation equations.

2.5 Automatic differentiation

Automatic differentiation (AD) is a technique for evaluating derivatives of computer represented functions up to machine precision [7]. AD exploits the fact that every computer represented function can be given as a sequence of elementary arithmetic operations such as addition, subtraction and multiplication, as illustrated in Figure 2.6. By breaking the problem down to these operations, it is trivial to differentiate each of these elementary functions using pre-defined look-up tables and then applying the chain rule. Two main approaches are available for AD, that is the forward and reverse mode [26].

A software implementation of AD can be viewed as a semantic transformation problem, namely to convert a code for computing a function into a code that computes the derivatives of that function [12]. Two approaches exist:

- **Operator overloading:** Redefinition of elementary operators. For example, the multiplication operator $*$ can be redefined to not only compute the expression $z = x * y$, but also its derivative according to the product rule. That is, each occurrence of a multiplication operator would produce $\nabla z = x\nabla y + y\nabla x$.
- **Source code transformation:** Rewriting the code explicitly. A transformation procedure may traverse the code and augment it with derivatives of its expressions. An assignment $z = x * y$ would result in adding an expression $\nabla z = x\nabla y + y\nabla x$.

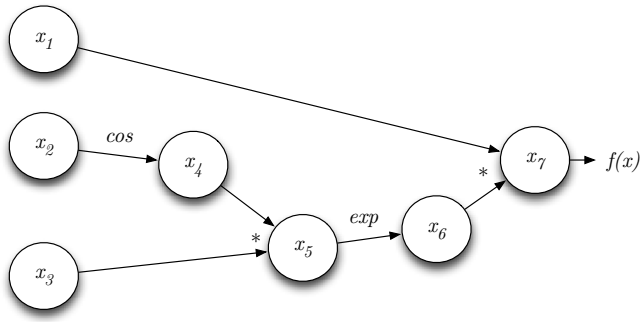


Figure 2.6: A function $f(x) = x_1 e^{\cos(x_2)x_3}$ may be decomposed into auxiliary variables x_4, x_5, x_6 and x_7 and elementary operations between these illustrated by a graph. For computing the gradient of a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the forward mode is best suited when $m > n$, while the reverse mode is better for the opposite [26].

Chapter 3

The JModelica.org framework

This chapter introduces the JModelica.org framework. Section 3.1-3.3 presents JModelica.org, the Modelica modeling language and its extension Optimica used to formulate dynamic optimization problems [2, 3]. Tools that are integrated in the framework include CVODES from the Sundials suite for solving differential equations, CasADi for providing efficient automatic differentiation and the interior point optimizer IPOPT for solving large-scale NLPs. Further, Python is used as a glue language between the different tools.

In Section 3.5, an example illustrates how to solve an OCP using the tools from JModelica.org. This includes modeling and formulating an OCP with Modelica and Optimica respectively. The OCP is solved by means of direct collocation, single shooting and multiple shooting methods. Implementations are given in Section 3.5.1-3.5.3. The presentation of the shooting methods is based on [5, 7, 27].

In addition, important remarks regarding the use of direct collocation methods are provided in Section 3.6-3.8. These include improvement of numerical efficiency and robustness through providing initial guess profiles, and specification of interpolation points for the collocation procedure. These have proven to be crucial for improving convergence and performance of the direct collocation method [1]. The performance of the three methods is also affected by scaling, which is discussed in Section 3.9.

3.1 The Modelica modeling language

Modelica is a high-level, object-oriented, equation-based language for modeling complex physical systems. Originated in the mid nineties, the first version of the Modelica language specification was published in 1997 by the Modelica Association. Being an object-oriented language, it supports concepts such as packages, classes, inheritance and components, which enables structuring and reuse of models. It is based on acausal equations, rather than assignment statements, which means that the order of computations is not decided at modeling time.

Models are defined by differential and algebraic equations, resulting in an ODE or DAE. In addition to equations and derivatives, discrete events may be formulated using functions and logic statements, offering a wide area of application, including hybrid systems. Built-in libraries include electrical, thermal, fluid and mechanic subcomponents where such components are coupled by connectors representing an input-output syntax between them. A graphical user interface for block diagram modeling is supported, enabling the user to construct models in a drag-and-drop manner.

Code written in the Modelica language is transformed to a suitable model code, which can be interfaced with a simulation algorithm in a procedure illustrated by Figure 3.1. Flattening of the model refers to transforming the hierarchical Modelica code into a representation consisting of variables and equations. Afterwards, symbolic manipulation is performed to represent the model as a structured hybrid DAE of index-1. The final step includes generation of efficient code that is suitable for integration with numerical algorithms for simulation; see [3, 4] for an overview of the compilation process.

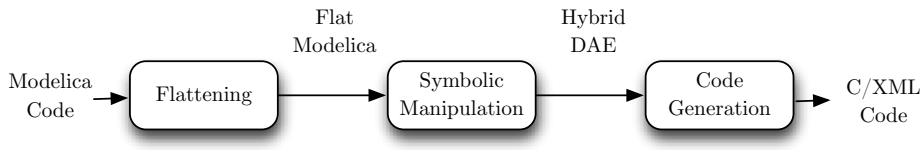


Figure 3.1: Modelica compilation process [3].

3.2 The JModelica.org environment

As Modelica offers a broad platform for modeling and simulation of systems, it does not support optimization of these. A framework incorporating this feature is JModelica.org, which is an *"extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems"*[1]. JModelica.org integrates state of the art algorithms for simulation, optimization and automatic differentiation. Further, it provides an interface for dynamic optimization of models written in the Modelica language using Python as a glue language for interacting between the different components. Some of the different components included in JModelica.org are shown in Figure 3.2.

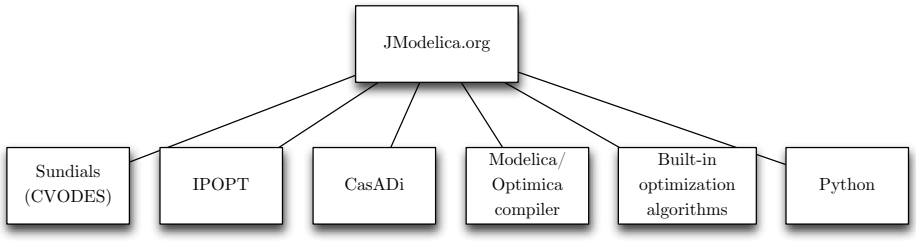


Figure 3.2: JModelica.org components.

After the user expresses a model using Modelica, a dynamic optimization problem may be formulated using the Modelica-extension Optimica. A built-in compiler in JModelica.org offers compilation of Modelica and Optimica code, which can generate C and XML files to be used for numerical simulation and optimization. An XML output consists of model meta data such as names, attributes and type for variables. The XML output follows a structure according to the *Functional Mock-up Interface*[2] (FMI), which is a standardized interface used in computer simulation. In practice, this enables the compiler to create *Functional Mock-up Units* (FMU), which are suited for numerical packages.

JModelica.org links sophisticated packages, such as CVODES [28] for simulation, and CasADi [6] for efficient automatic differentiation. A complete overview of the platform architecture, and the component relations is illustrated in Figure 3.3.

[1] JModelica.org http://www.jmodelica.org
[2] FMI https://www.fmi-standard.org

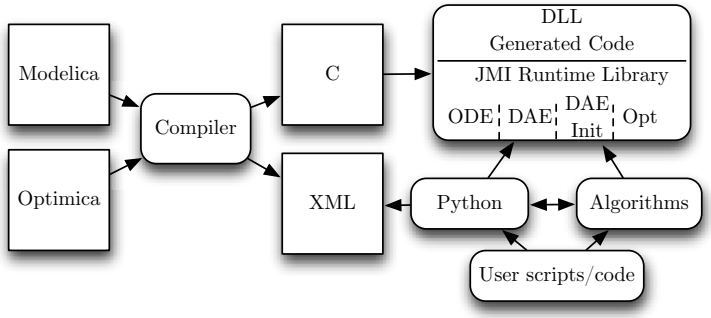


Figure 3.3: JModelica.org platform architecture [3].

The current edition of JModelica.org (version 1.9) includes 4 different built-in optimization algorithms, namely dynamic optimization of DAEs using direct collocation with CasADi, direct collocation with CppAD, pseudospectral optimization and derivative-free optimization.

The direct collocation method results in a NLP that is solved using IPOPT. It applies to continuous-time DAEs that do not contain functions representing discrete/hybrid behavior. IPOPT is prepared for solving large-scale NLPs and provides heuristics for exploitation of the sparsity structure [5].

JModelica.org uses CasADi or CppAD to compute derivatives based on AD. Since CasADi is included in the JModelica.org framework, this also enables efficient implementation of other algorithms for optimization based on direct (or indirect) approaches, e.g. shooting methods. In addition, CasADi provides interfacing between the Sundials and IPOPT packages, making it well suited for this purpose. An overview of the algorithms for dynamic optimization is given in Figure 3.4.

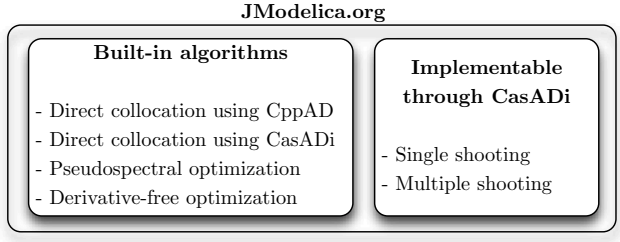


Figure 3.4: JModelica.org and CasADi algorithms for dynamic optimization.

3.3 Optimica

In order to formulate a dynamic optimization problem using the JModelica.org framework, the user needs to supply a model and an optimization formulation. Having the model code written in Modelica, an optimization formulation at a high abstraction level may be defined using the Optimica extension [2]. Optimica includes new language constructs enabling the user to express the dynamic optimization problem. This includes specifying optimization variables, objective function and constraints.

Although the formulation of the problem is expressed in Optimica, there is no information provided regarding how to solve it, i.e. approximation of the infinite dimensional optimization problem into a finite dimensional NLP. This is specified in the Python script, where the user can choose to use the built-in collocation algorithm or implement a shooting method. The approximation procedure that transforms the dynamic optimization problem into a NLP based on a simultaneous approach is done automatically by specifying this method in Python. It is also possible to specify additional options for the approximation procedure, e.g. number of elements and interpolation points for direct collocation.

Approximation of the OCP results in a NLP, which is solved by IPOPT. Here, algorithmic parameters for the NLP solver may be specified, e.g. tolerances and termination criteria. Figure 3.5 illustrates the workflow of importing Modelica/Optimica code through Python.

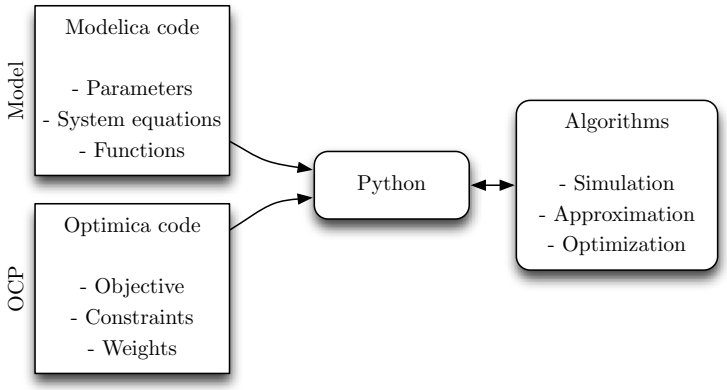


Figure 3.5: Modelica and Optimica relation in the JModelica.org platform.

3.4 CasADi

A package for automatic differentiation contained in JModelica.org is CasADi, best described as a minimalistic *Computer Algebra System for Automatic Differentiation*. CasADi can be viewed as a framework for efficient implementation of derivative based algorithms for dynamic optimization. It enables AD in forward and reverse mode by implementing a hybrid operator overloading/source code transformation approach. Expressions can be represented in two ways, either as scalar expressions (SX) or matrix expressions (MX), which have different properties in terms of efficiency and generality. The reader is referred to [5, 6, 7, 8] for a comprehensive description of CasADi.

In JModelica.org's built-in collocation algorithm `LocalDAECollocationAlg`, CasADi is interfaced to IPOPT, enabling AD for computing derivatives of objective and constraint functions. CasADi also provides a quick way for implementing shooting algorithms. Some main features are model import from Modelica/Optimica and symbolic manipulation, e.g. transformation from a DAE to ODE. In addition is has an interface to integrators such as CVODES and IDAS, both providing sensitivity information and NLP solvers like IPOPT and KNITRO as shown in Figure 3.6.

For shooting problems implemented with a Sundials integrator, CasADi automatically formulates forward and adjoint sensitivity equations and provides Jacobian information needed by the linear solvers [5]. In this way, the user needs only to instantiate the approximation procedure, e.g. control input discretization and construction of objective and constraint functions.

Figure 3.6: CasADi provides an interface to numerical integrators with sensitivity capabilities from Sundials: CVODES and IDAS, for ODE and DAE respectively, and NLP solvers: IPOPT and KNITRO.

3.5 Example: Coupled tanks

To illustrate the use of the JModelica.org framework, an example of performing dynamic optimization applied to a simple nonlinear system is given. Consider a model of two coupled liquid tanks, where the inflow is given as a control input, as shown in Figure 3.7. Let the model be described by the ODE:

$$\dot{z}_1 = -\overbrace{\frac{a_1}{A_1}\sqrt{2gz_1}}^{w_1} + \overbrace{\frac{a_2}{A_2}\sqrt{2gz_2}}^{w_2}, \tag{3.1a}$$

$$\dot{z}_2 = -\underbrace{\frac{a_2}{A_2}\sqrt{2gz_2}}_{w_2} + \underbrace{\frac{k}{A_u}u}_{w_{in}}, \tag{3.1b}$$

where z_1 and z_2 are liquid levels and u is liquid inflow. The parameters a_i, A_i, k, g and A_u are given. This model is written in Modelica according to Listing 3.1. An OCP describing the objective of controlling the liquid level z_1 to a constant reference value z_1^r can be formulated as the Mayer problem (3.2) with (3.2b) being system equations (3.1). An additional cost state z_c for the objective function (3.2d) is included. The OCP (3.2) is written in Optimica according to Listing 3.2.

$$\min_{z,u} \; z_c(t_f) \tag{3.2a}$$

$$\text{s.t. } \dot{z} = f(z,u), \tag{3.2b}$$

$$z(0) = z_0, \tag{3.2c}$$

$$\dot{z}_c = (z_1 - z_1^r)^2 + u^2, \tag{3.2d}$$

$$z_c(0) = 0, \tag{3.2e}$$

$$z_{min} \leq z \leq z_{max}, \tag{3.2f}$$

$$u_{min} \leq u \leq u_{max}, \tag{3.2g}$$

$$t \in [0, t_f].$$

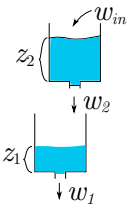


Figure 3.7: Coupled tanks.

Listing 3.1: Modelica code for coupled tanks example.

```
model tank
  import SI = Modelica.SIunits;
  parameter SI.Area a1=0.03;
  parameter SI.Area a2=0.02;

  parameter SI.Area A1=8.4;
  parameter SI.Area A2=7.1;
  parameter SI.Area A_u=1;

  parameter SI.Acceleration g = 9.81;
  parameter Real k = 0.4;     // [.]

  Real z1(start=0.4,fixed=true); // State z1 with initial value  [m]
  Real z2(start=3,fixed=true);  // State z2 with initial value [m]

  parameter Real u_initGuess = 0.1;
  input Real u(start=u_initGuess,fixed=true); // Control input with initial guess
      [m^3/s]

equation
  der(z1) = -(a1/A1)*sqrt(2*g*z1)+(a2/A2)*sqrt(2*g*z2);
  der(z2) = -(a2/A2)*sqrt(2*g*z2)+(k/A_u)*u;
end tank;
```

Listing 3.2: Optimica code for coupled tanks example.

```
optimization ocp(objective=zc(finalTime),startTime=0,finalTime=600)
  extends tank;             // Include tank model equations
  Real zc(start=0,fixed=true);
  constant Real z1r = 2;      // Reference value
equation
  // Cost function state
  der(zc) = (z1-z1r)^2+u^2;
constraint
  // State bounds z_min <= z <= z_max
  z1<=5;
  z1>=0.1;

  z2<=5;
  z2>=0.1;

  // Control input bounds u_min <= u <= u_max
  u<=0.3;
  u>=0;
end ocp;
```

3.5.1 Direct collocation

When using JModelica.org's built-in algorithm for solving the OCP (3.2) with direct collocation, the model and OCP needs to be imported through Python. In this example, both model and OCP are given in the same file although these can be separated.

In Listing 3.3, the model code is compiled to a FMU. Further, this is scaled and imported through `CasadiModel()`, which creates an object suitable for optimization with CasADi. The function `optimize()` uses this scaled model to solve the OCP using the algorithm `LocalDAECollocationAlg`, which is direct collocation utilizing CasADi for automatic differentiation. Approximation of the infinite dimensional OCP and solving the resulting NLP with IPOPT is done by the `optimize()` function.

Listing 3.3: Direct collocation implementation.

```
from pymodelica import compile_fmux,compile_fmu
from pyjmi import CasadiModel

fmux = compile_fmux("ocp", "tank.mop")
model = CasadiModel(fmux, scale_variables=True)

res = model.optimize(algorithm="LocalDAECollocationAlg")
```

3.5.2 Single shooting

The single shooting implementation assessed in this thesis is based on [5]. Some main remarks from the code given in Listing 3.4 are:

- The file containing Modelica and Optimica code is imported through Python by the `SymbolicOCP()` class in lines 7-12. The `ocp` object instantiated from this class contains all information related to the OCP, e.g. states, variables and equations. On line 32, an `ode` object is instantiated as a `SXFunction()`, which is a CasADi-type function suited for automatic differentiation. This results in a symbolic expression for the ODE.

- A `CVodesIntegrator` is instantiated in lines 34-37 by supplying it with the `ode` object. Notice that the integration interval is set to $\frac{t_f}{N}$, such that integration occurs for each parameterized control input u^i , $i = 1, ..., N$.

- In lines 52-55, a relation between control inputs and the states is constructed using the `call()` function of the integrator object. For each u^i, integration occurs over $[t_{i-1}, t_i]$ according to Figure 3.8. This results in a completely symbolic representation of the states as a function of the control inputs.

- Finally, `IpoptSolver` is provided with the objective function, state constraint functions, variable bounds and initial guess in lines 65-75, leading to a NLP formulation similar to (3.3).

The NLP solved by IPOPT in the single shooting implementation can be written as:

$$\min_{U \in \mathbb{R}^{N n_u}} f(U) \tag{3.3a}$$

$$\text{s.t.} \quad g_L \leq g(U) \leq g_U, \tag{3.3b}$$

$$u_L \leq U \leq u_U, \tag{3.3c}$$

where the expressions in (3.3) are given in Appendix B.3. Constraints on control inputs can be directly cast into the variable bounds (3.3c). State constraints however must be written in terms of U in (3.3b).

It should be noted that formulation of sensitivity equations and AD is done automatically by CasADi. These computations are implicit in the example code. In particular, CasADi provides an interface between the `CVodesIntegrator` and IPOPT. In this way it is possible to work with symbolic expressions to construct objective and constraint functions.

Constructing objective and constraint functions

The most important part of the single shooting implementation given in Listing 3.4 is the recursive integrator call construction in lines 52-55. This procedure can be illustrated by a graph consisting of nodes and edges as shown in Figure 3.9. Nodes correspond to a specific variable, e.g. states or control inputs at time points. On the left side, inputs are given by initial conditions z_0 and the set of all parameterized control inputs U, the latter being decision variables. Thus, it is possible to propagate the variables through the graph to obtain objective and constraint functions f and g respectively.

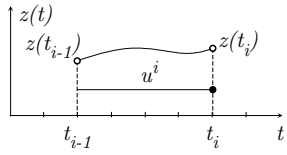


Figure 3.8: Integrator call from $z(t_{i-1})$ to $z(t_i)$.

In CasADI, integrators are considered to be functions that can be embedded in an optimization formulation. Let $F(.)$ be an integrator such that:

$$z(t_i) = F(t_{i-1}, t_i, z(t_{i-1}), u^i) \ , \ i \in 1, ..., N \tag{3.4}$$

where t_{i-1} and t_i are time points and u^i are parameterized control inputs as illustrated by Figure 3.8. By providing these arguments to the integrator, it performs integration in the time interval $[t_{i-1}, t_i]$ and outputs the state at end time t_i. These integrator calls can be embedded in the integrator call construction, which can be visualized by a directed acyclic graph shown in Figure 3.9.

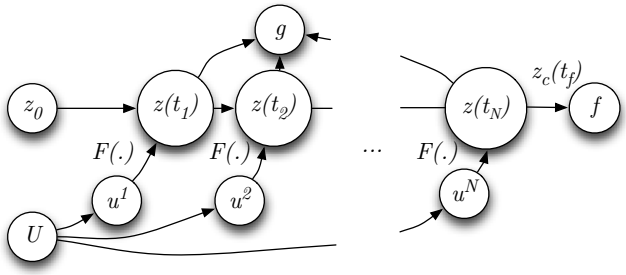


Figure 3.9: Computational graph for constructing f and g in the single shooting example. The only decision variables are $u^i, i = 1, ..., N$.

Listing 3.4: Single shooting implementation.

```
import numpy as NP
from pymodelica import compile_jmu
from casadi import *
import zipfile

# Import Modelica and Optimica code
jmu_name=
    compile_jmu("ocp","tank.mop",'optimica','ipopt',{'generate_xml_equations':True})
sfile  = zipfile.ZipFile('ocp.jmu','r')
mfile  = sfile.extract('modelDescription.xml','.')

ocp = SymbolicOCP()
ocp.parseFMI('modelDescription.xml',{'scale_variables':True,'sort_equations':False})

tf=ocp.tf                  # Final time
N=60                       # Control input discretization

# Assign states, control inputs and model equations from ocp object
# to symbolic variable names: z,u,f_rhs
def toArray(v, der=False):
  ret = []
  for i in v:
    if der:
      ret.append(i.der())
    else:
      ret.append(i.var())
  return NP.array(ret,dtype=SX)
ocp.makeExplicit()
z = toArray(ocp.xd)                # States
u = toArray(ocp.u)                 # Control inputs
f_rhs = ocp.ode                    # Right hand side of ODE: f(z,u)

ode=SXFunction(daeIn(z,[],u,[],[]),daeOut(f_rhs)) # Define ODE: f=(z,u)

odeInt = CVodesIntegrator(ode)                  # Instantiate a CVodesIntegrator
odeInt.setOption("tf",tf/N)                     # Integration length for each u^i
odeInt.setOption("steps_per_checkpoint",1000)  # Additional options
odeInt.init()

# Initial conditions z0 (numerical values)
C0   = 0;                                  # Init. value for cost function state
z10  = ocp.variable('z1').getStart()       # Init. value for state z1
z20  = ocp.variable('z2').getStart()       # Init. value for state z2

U = msym("U",N)                    # Symbolic control input vector
Z = msym([C0,z10,z20])             # Symbolic state vector

# Local expressions for state variables
Z1 = MX("Z1",N)                 # Symbolic state z1 vector
Z2 = MX("Z2",N)                 # Symbolic state z2 vector

# Build up a graph of integrator calls
for k in range(N):
  [Z,_,_,_] = odeInt.call([Z,U[k]])      # Integrator call
  Z1[k] = Z[1]                           # State z1
  Z2[k] = Z[2]                           # State z2

# Z[0] contains cost state zc at final time t_N
f = MXFunction([U],[Z[0]])                 # NLP Objective function f
g = MXFunction([U],[vertcat((Z1,Z2))])     # NLP Constraint function g

```

```
# IpoptSolver with formulation:
# min f(U)
# s.t   g_L <= g <= g_U
#       u_L <= U <= u_U
solver = IpoptSolver(f,g)
solver.setOption('tol',0.1)
solver.init()

solver.setInput(0*NP.ones(N), NLP_LBX)      # u_L
solver.setInput(0.3*NP.ones(N), NLP_UBX)    # u_U

solver.setInput(NP.ones(N)*0.1,NLP_X_INIT)   # Initial guess

solver.setInput(vertcat((NP.ones(N)*0.1,NP.ones(N)*0.1)),NLP_LBG) # g_L
solver.setInput(vertcat((NP.ones(N)*5,NP.ones(N)*5)),NLP_UBG)     # g_U

solver.solve()
```

3.5.3 Multiple shooting

An implementation of a multiple shooting method is given in Listing 3.5. On line 49, V is defined to be the vector containing all decision variables, i.e. $z_0^i, u^i, i = 1, ..., N$. Decision variables are partitioned into local expressions from V in lines 52-55. Variable bounds v_L and v_U are defined in lines 58-68. The NLP solved by IPOPT in the multiple shooting implementation becomes:

$$\min_{V \in \mathbb{R}^{n_v}} f(V) \tag{3.5a}$$

$$\text{s.t. } g_L \leq g(V) \leq g_U, \tag{3.5b}$$

$$v_L \leq V \leq v_U, \tag{3.5c}$$

where $n_v = N(n_z + n_u)$ and the expressions in (3.5) are given in Appendix B.4. Here, state and control input constraints can be directly cast into the variable bounds (3.5c). Continuity constraints and initial condition are contained in (3.5b).

Constructing objective and constraint functions

For each time segment $[t_{i-1}, t_i]$, $i = 1, ..., N$, an integrator call is made. A single call can be represented by Figure 3.10. For the last time segment, $z^N(t_N) = z(t_f)$ is obtained. This consists of states at the final time point, including the cost state $z_c(t_f)$. The objective function f is defined on line 105 by providing the decision variable vector V as input, and the cost state $z_c(t_f)$ at the final time as output.

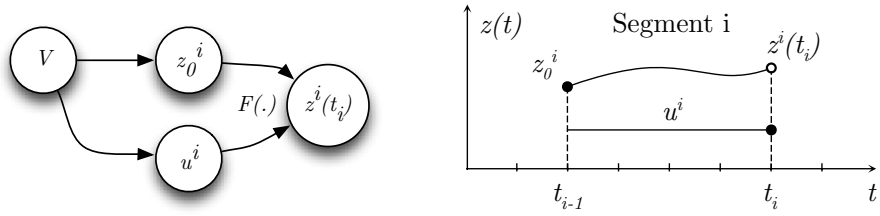


Figure 3.10: Computational graph for an integrator call. A state variable z_0^i with a control input u^i in segment i are used to compute the state at the end of the segment denoted by $z^i(t_i)$. This is done by the integrator call $F(.)$ as presented in (3.4), with z_0^i as input argument for initial state instead of $z(t_{i-1})$. The procedure is repeated for all segments $i = 1, ..., N$. The only decision variables here are z_0^i, u^i , $i = 1, ..., N$ which are contained in V, while $z^i(t_i)$ is the integrator output.

The initial condition g_1 is posed on line 89, while continuity of state profiles across segments $g_i, i = 2, ..., N$ is defined on line 101. A vector-function g containing these constraints is then composed on line 106. Figure 3.11 illustrates the procedure of constructing f and g.

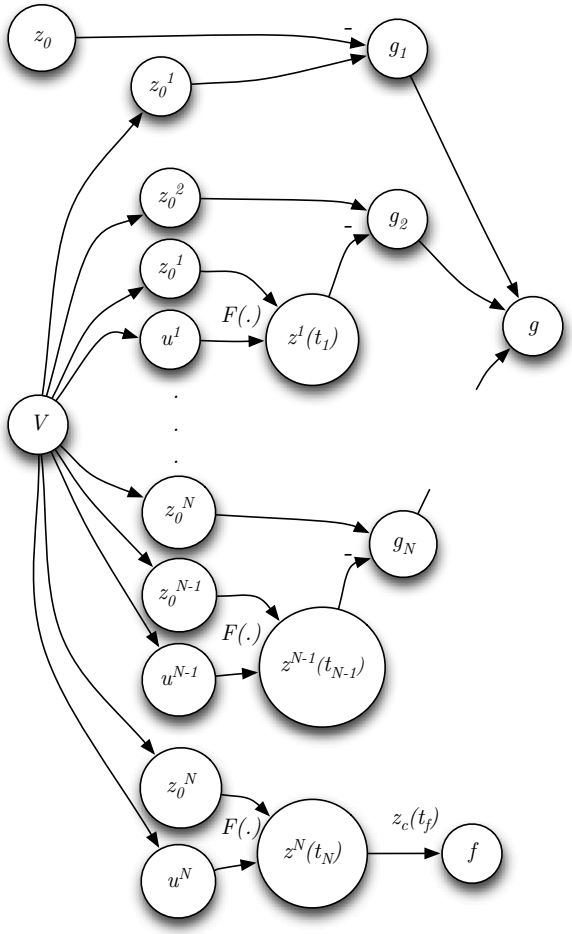


Figure 3.11: Computational graph for constructing f and g. At the top, initial conditions g_1 are defined from a numerical value z_0 and the decision variable z_0^1. Continuity constraints are given as: $g_i = z_0^i - z^{i-1}(t_{i-1})$, $i = 2, ..., N$. A complete constraint is then assembled as g. At the bottom, states at the final time give $z_c(t_f)$, which is defined as the objective function.

Listing 3.5: Multiple shooting implementation.

```
import numpy as NP
from pymodelica import compile_jmu
from casadi import *
import zipfile

# Import Modelica and Optimica code
jmu_name =
     compile_jmu("ocp","tank.mop",'optimica','ipopt',{'generate_xml_equations':True,
     'generate_fmi_me_xml':False})
sfile = zipfile.ZipFile('ocp.jmu','r')
mfile = sfile.extract('modelDescription.xml','.')

ocp = SymbolicOCP()
ocp.parseFMI('modelDescription.xml',{'scale_variables':True,'sort_equations':False})
ocp.sortType()

tf=600   # Final time
N=60     # Control input discretization and number of segments

nz=2+1   # State dimension (differential states + cost state)
nu = 1   # Control input dimension

# Assign states, control inputs and model equations from ocp object
# to symbolic variable names (z,u,f)
def toArray(v, der=False):
  ret = []
  for i in v:
    if der:
      ret.append(i.der())
    else:
      ret.append(i.var())
  return NP.array(ret,dtype=SX)
ocp.makeExplicit()
z = toArray(ocp.xd)      # States
u = toArray(ocp.u)       # Control inputs
f_rhs=ocp.ode            # Right hand side of ODE: f(z,u)

ode=SXFunction(daeIn(z,[],u,[],[]),daeOut(f_rhs)) # Generate ODE: f_rhs=(z,u)

odeInt = CVodesIntegrator(ode)      # Instantiate a CVodesIntegrator
odeInt.setOption("tf",tf/N)         # Integration length for each u^i
odeInt.setOption("steps_per_checkpoint",1000) # Additional options
odeInt.setOption("abstol",1e-5)
odeInt.setOption("reltol",1e-5)
odeInt.init()

nv = N*(nz+nu)           # Total amount of decision variables

# Decision variable vector [u,z] has the form:
# V = [u_1 ... u_N zc_1 z1_1 z2_1 ... zc_N z1_N z2_N]
V = msym("V", nv)

# Local expressions for state variables and control inputs
U = V[0:N*nu]                # Symbolic control input vector
ZC = V[N*nu+0:nv:nz]         # Symbolic zc vector
Z1 = V[N*nu+1:nv:nz]         # Symbolic z1 vector
Z2 = V[N*nu+2:nv:nz]         # Symbolic z2 vector

# Variable bounds v_L and v_U
v_L = (-inf)*NP.ones(nv)
v_U = (inf)*NP.ones(nv)
```

```
60
61  v_L[0:N*nu]    = 0               # u_min = 0
62  v_U[0:N*nu]    = 0.3             # u_max = 0.3
63
64  v_L[N*nu+1:nv:nz] = 0.1          # z1_min = 0.1
65  v_L[N*nu+2:nv:nz] = 0.1          # z2_min = 0.1
66
67  v_U[N*nu+1:nv:nz] = 5            # z1_max = 5
68  v_U[N*nu+2:nv:nz] = 5            # z2_max = 5
69
70  # Initial conditions z0 (numerical values)
71  C0 = 0                                  # Init. value for cost function state
72  z10  = ocp.variable('z1').getStart()    # Init. value for state z1
73  z20  = ocp.variable('z2').getStart()    # Init. value for state z2
74
75  # Initial guess vector has the form:
76  # VINIT = [u_g ... u_g C0 z10 z20 ... C0 z10 z20]
77  u_g = 0.3
78  initU = NP.ones(N*nu)*u_g
79  initV2 = NP.transpose(NP.tile([C0,z10,z20],(N)))
80  initVec = NP.concatenate((initU,initV2))
81  VINIT = initVec
82
83  # Constraint functions g_L <= g <= g_U
84  g_i = []                        # g_i
85  g_L=NP.zeros([N,nz])            # g_L
86  g_U=NP.zeros([N,nz])            # g_U
87
88  # First constraint (initial conditions): g1 = z1 - z0
89  g_i.append(vertcat((ZC[0],Z1[0],Z2[0]))-vertcat((C0,z10,z20)))
90
91  # Build up a graph of integrator calls
92  for k in range(N):
93    Zk = vertcat((ZC[k],Z1[k],Z2[k]))
94
95    if k!=N-1:
96     Zk_next = vertcat((ZC[k+1],Z1[k+1],Z2[k+1]))
97
98    Zk_end = odeInt.call([Zk,U[k]])[0]     # Integrator call
99
100   if k!= N-1:
101    g_i.append(Zk_next - Zk_end)           # Continuity constraints
102
103
104 # Zk_end[0] contains cost state zc at final time t_N
105 f = MXFunction([V],[Zk_end[0]])         # NLP Objective function f
106 g = MXFunction([V],[vertcat(g_i)])      # NLP Constraint function g
107
108 # IpoptSolver with formulation
109 # min f(V)
110 # s.t    g_L <= g(V) <= g_U
111 #        v_L <= V <= v_U
112 solver = IpoptSolver(f,g)
113 solver.setOption('tol',0.1)
114 solver.init()
115
116 solver.setInput(v_L, NLP_LBX)
117 solver.setInput(v_U, NLP_UBX)
118 solver.setInput(VINIT, NLP_X_INIT)
119
120 solver.setInput(vertcat((NP.concatenate(g_L))),NLP_LBG)
121 solver.setInput(vertcat((NP.concatenate(g_U))),NLP_UBG)
122 solver.solve()
```

3.6 Providing an initial guess to direct collocation methods

For the NLP resulting from direct collocation, it is necessary to provide an initial guess for states, algebraic variables and control inputs. For collocation methods, a good initial guess can improve convergence and robustness for the algorithm [1]. A straightforward way of doing this is by simulating the system with an initial guess of the control inputs, thus getting state and algebraic variable profiles, which then can be used as the initial guess profile as shown in Figure 3.12.

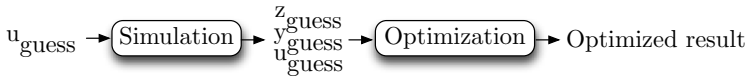


Figure 3.12: Initial guess simulation in JModelica.org.

In JModelica.org's built-in collocation algorithm, this may be done according to Listing 3.6. As the OCP (3.2) is formulated as a Mayer problem, an initial simulation provides state profiles for z_1, z_2 and the cost state z_c, given an initial guess for the control input u. A dedicated model for this initial simulation purpose is given in Listing 3.7.

Listing 3.6: Providing initial guess. Python code segment.

```python
from pymodelica import compile_fmux,compile_fmu
from pyjmi import CasadiModel
from pyfmi import FMUModel

fmux = compile_fmux("ocp", "tank.mop")
model = CasadiModel(fmux, scale_variables=True)

# Simulate tank_init_optimization
init_sim_fmu = compile_fmu("tank_init_optimization","tank.mop")
init_sim_model = FMUModel(init_sim_fmu)
init_res = init_sim_model.simulate(start_time=0, final_time=600)

opt_opts = model.optimize_options()
opt_opts['init_traj'] = init_res.result_data  # Provide initial guess trajectory

res = model.optimize(algorithm="LocalDAECollocationAlg",options=opt_opts)
```

Listing 3.7: Providing initial guess. Modelica code segment.

```
model tank_init_optimization
  extends tank;
  Real cost(start=0,fixed=true);
equation
  der(cost) = (z1-2)^2+u^2;
end tank_init_optimization;
```

3.7 Output from direct collocation

Direct collocation leads to a NLP given in Section 2.4.2, where the decision variables are discrete points in time. For the tank example, which does not contain algebraic variables, these are given by:

$$z_{i,k} \ , \ i = 1, ..., N \ , \ k = 0, ..., K \tag{3.6}$$

$$u_{i,k} \ , \ i = 1, ..., N \ , \ k = 1, ..., K. \tag{3.7}$$

Solving the NLP results in a sequence of these points shown in Figure 3.13.

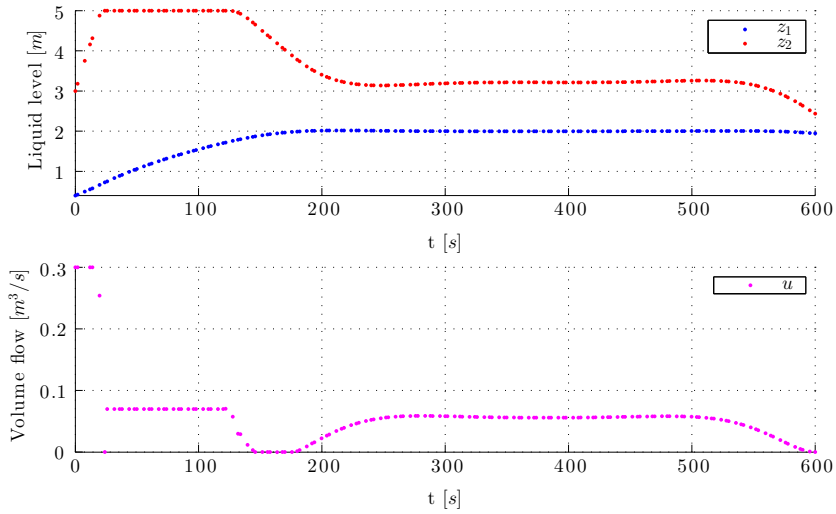


Figure 3.13: Optimized sequence of states and control inputs from the direct collocation algorithm are discrete points in time, here using N=50 and K=3.

By applying the Lagrange interpolation equations (2.26) and (2.37) to these points, it is possible to obtain interpolated variable profiles as shown in Figure 3.14. Here, interpolated variable profiles are shown together with their interpolation points at the beginning of each element. Finally, the result of the collocation procedure can be given as interpolated variable profiles for the entire horizon as shown in Figure 3.15.

It should be noted that the collocation method offered by JModelica.org enables to choose the output form, either as a sequence of interpolation points, or as interpolated variable profiles.

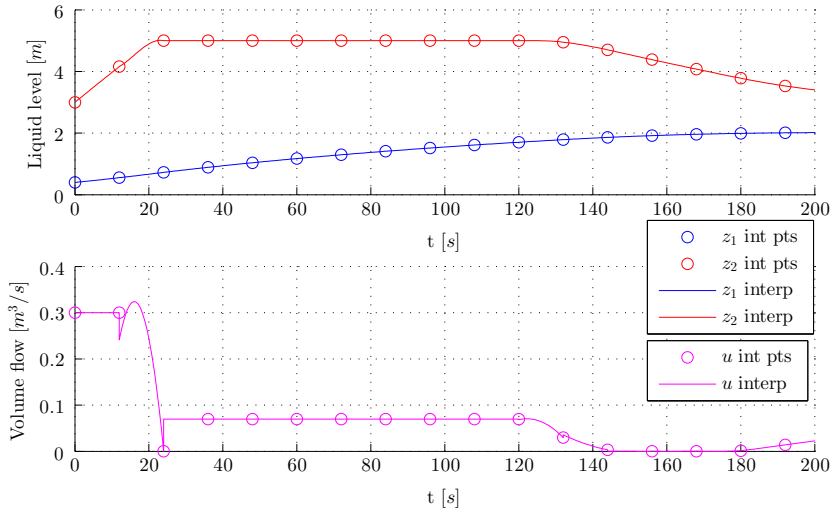


Figure 3.14: Interpolation points at the beginning of each element (int pts) with their interpolated profiles (interp). A closer look.

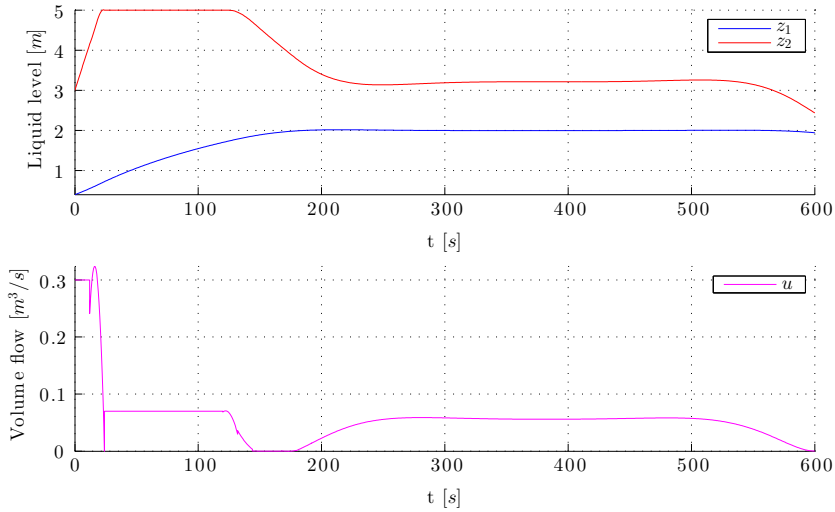


Figure 3.15: Interpolated variable profiles. The final output from direct collocation.

3.8 Choice of elements and interpolation points

When using the direct collocation algorithm, decreasing N and K results in less decision variables and constraints, which can lead to a reduction of running time. These two parameters decide the approximation accuracy of state variable profiles, such that increasing them will often lead to a better approximation, while a decrease can lead to a worse. From this it can be argued that there is a trade-off between running time and accuracy.

This trade-off motivates the comparison of collocation outputs in terms of approximated state and algebraic variable profiles with the output of numerical integration when applying an optimized control profile. Let the optimized profiles from a collocation algorithm be denoted by z^{opt} and u^{opt}, as illustrated by Figure 3.16. This corresponds to optimized state and control profiles respectively. When applying u^{opt} to a numerical integrator, the result is an output denoted by z^{sim} and u^{sim} being simulated state and control input profiles respectively. Clearly $u^{opt} = u^{sim}$, but this is not the case for z^{opt} and z^{sim}.

To illustrate the effect of varying N, Figure D.1 shows optimized control inputs u^{opt} and their corresponding state profiles z^{opt} for different values of N, together with simulated state profiles from an integrator. For N=2, the approximation is poor. For a low number of elements, it is also clear that the resulting variable profiles can violate constraints. These issues also apply to DAE systems with algebraic variables.

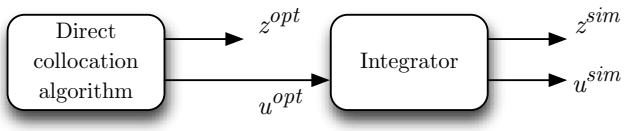


Figure 3.16: The output of a direct collocation algorithm consisting of interpolated variable profiles and control inputs should be compared to an integrator provided with the optimized control inputs.

3.9 Scaling of models

The performance of an optimization algorithm depends crucially on how the problem is formulated, e.g. scaling of functions and variables. A problem is said to be poorly scaled if changes in a variable x to a certain direction produce much larger variations in the value of a function f than any other direction [26].

Poorly scaled functions arise from physical and chemical systems having a number of processes taking place at very different rates. Scaling issues can also arise when some decision variables differ by many orders of magnitude from each other, far from an order of one. Consider for instance the function

$$y = \sqrt{x - c}, \tag{3.8}$$

where c is a constant in the order of 10^6, while decision variables y and x are in the order of 10^0 and 10^6 respectively. This may lead to numerical problems in an optimization algorithm.

In JModelica.org it is possible to enable scaling of models by using rough estimates for the numerical value of their variables. That is, given an a priori knowledge of which order of magnitude a specified variable operates in, it is possible to provide this information in Modelica through the `nominal` attribute. Eq. (3.8) can be scaled in Modelica by providing a nominal attribute for the variable x as shown in Listing 3.8.

Listing 3.8: Nominal attribute for the variable x in Modelica.

```
1 Real x(nominal=10^6);
```

This implies that each variable is divided by its nominal value, such that from an algorithm point of view they will take on values close to one, which is desirable. In order to ensure that the model equations are fulfilled, the nominal value needs to be multiplied with the scaled variable in the model equations. For (3.8) this corresponds to a scaled variable and function given as:

$$x_{scaled} = \frac{x}{10^6}, \tag{3.9}$$

$$y = \sqrt{10^6 x_{scaled} - c}, \tag{3.10}$$

such that the new decision variable x_{scaled} is in the order of one, while maintaining the same model equation as before.

Chapter 4

Modeling

In this chapter, a model of a subsea petroleum production plant consisting of wells, manifolds and flowlines is presented, based on previous work done in [17], [11] and [25]. A summary of all equations used to construct the well-flowline model is given in Appendix B, with a nomenclature of some of the main variables in Appendix A.

Section 4.1-4.4 present a component-based approach of constructing an overall model, by defining an input-output relationship for each component. A description on how these components are implemented and connected in Modelica is provided in Section 4.6. Finally, a test simulation of the complete model, comparing it to a similar one in Matlab [25] is given in Section 4.7.

4.1 Well

A simple model of a 2-phase artificially gas lifted well was presented in [17], which is known as the Eikrem model. Further development in [11] has led to a 3-phase model with gas, oil and water as separate phases. The primary aim of this model is to capture the dynamic behavior of a phenomenon known as casing-heading, which leads to oscillating production for low gas injection rates.

Referring to Figure 4.1, the well geometry is modeled as a vertical cylinder, consisting of two concentric inner cylinders being the tubing and annulus. The injected gas flow is controlled by a gas lift choke at the wellhead. This gas flows along the annulus to the bottom of the well, through the injection valve and into the tubing, helping to enhance the production [25].

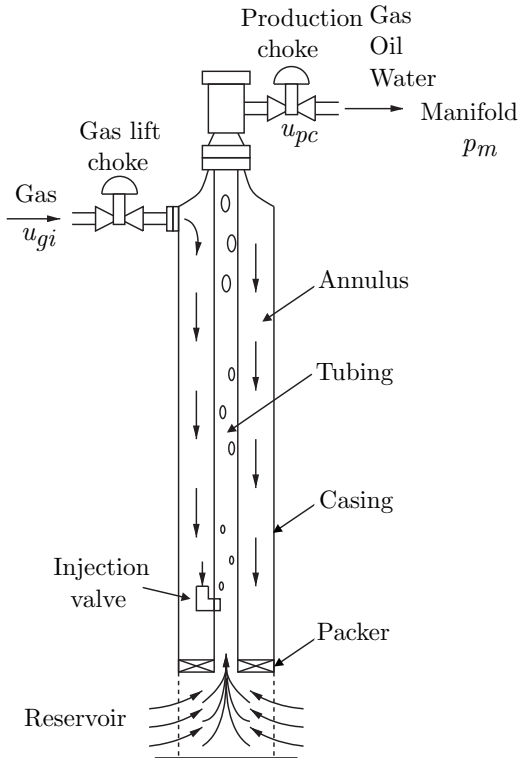


Figure 4.1: Production well with gas-lift [17].

Model structure

The well model is based on a first-principle approach. The governing state equations are written as mass balances. Liquid fluids oil and water are assumed to be incompressible, whereas gas follows the ideal gas law. Differential states are mass of gas in annulus m_{ga}, mass of gas in tubing m_{gt} and mass of liquid in tubing m_{lt}.

$$\text{States:} \quad z_w = \begin{bmatrix} m_{ga} \\ m_{gt} \\ m_{lt} \end{bmatrix}. \tag{4.1}$$

Further, there are two variables that are used to control the well. These are the production choke valve setting u_{pc} and the mass flow of lift-gas into the annulus u_{gi}. By adjusting these, it is possible to control outflow of the well. Here, the production choke valve setting will not be used as a control input, and will in practice be set to 1 (fully open). It is therefore not shown in the subsystem description.

$$\text{Control input:} \quad u_{gi}. \tag{4.2}$$

The well model can be viewed as a subsystem illustrated by Figure 4.2 with inputs u_{gi} and p_m, which is lift-gas injection rate and manifold pressure respectively. Outputs are mass flow of gas, oil and water, denoted by w_{gp}, w_{op} and w_{wp}. From the well model equations given in Appendix B.1, it can be written as:

$$\dot{z}_w = f_w(z_w, y_w, u_{gi}, p_m), \tag{4.3a}$$
$$0 = g_w(z_w, y_w, u_{gi}, p_m), \tag{4.3b}$$

where $z_w \in \mathbb{R}^3$ are states, $y_w \in \mathbb{R}^{18}$ are algebraic variables (not including p_m) and $u_{gi} \in \mathbb{R}$ is the control input. Manifold pressure $p_m \in \mathbb{R}$ is here represented as an external input, although not a controlled one. Differential and algebraic equations are given by the mappings f_w and g_w respectively.

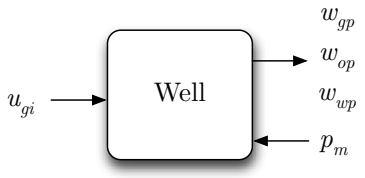
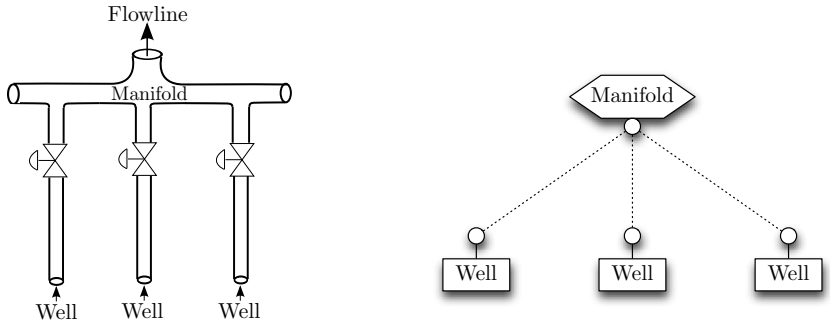


Figure 4.2: Well subsystem.

4.2 Manifold

The subsea manifold is a hydraulic component that regulates fluid flow between several inputs and one output. As illustrated by Figure 4.3, inputs are flows from wells, while the output is connected to a flowline.



(a) Hydraulic manifold.

(b) Simplified manifold topology.

Figure 4.3

In this thesis, a constant routing between wells and flowlines is assumed. The manifold can be viewed as a subsystem connecting wells with flowlines. Inputs are flow of gas, oil and water from each well, and pressure p_m at the flowline inlet. Outputs are the sum of all well flows and routed pressures back to the wells as shown in Figure 4.4. Let $j \in \mathcal{J}$ be the set of wells that are connected to a single flowline such that the manifold equations becomes:

$$w_{g,in} = \sum_{j \in \mathcal{J}} w_{gp,j} \ , \ w_{o,in} = \sum_{j \in \mathcal{J}} w_{op,j} \ , \ w_{w,in} = \sum_{j \in \mathcal{J}} w_{wp,j}, \tag{4.4}$$

$$w_{in} = \begin{bmatrix} w_{g,in} & w_{o,in} & w_{w,in} \end{bmatrix}^T . \tag{4.5}$$

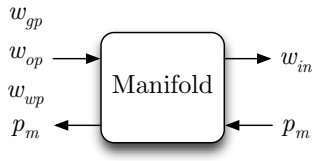


Figure 4.4: Manifold subsystem.

4.3 Flowline

A flowline consisting of a long horizontal pipeline connected to a vertical riser is modeled as two cylindrical volumes connected at a low point referring to Figure 4.5. Inlet mass flows: $w_{g,in}$, $w_{o,in}$ and $w_{w,in}$ are assumed to be stratified, which means that gas is covering the top layer with the liquids oil and water below. At the low point, these phases become mixed, yielding a mixture of gas, oil and water at the flowline outlet [25]. This flowline model is based on a two-phase gas-liquid model from [23] with modifications done in [25].

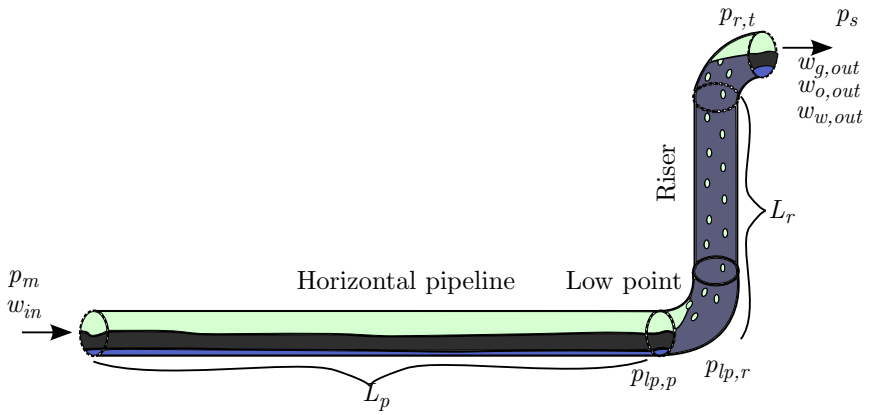


Figure 4.5: Flowline geometry consisting of a horizontal pipeline and a vertical riser with lengths L_p and L_r respectively.

Referring to Figure 4.5, flows are driven by pressures, which are:

- p_m : Pressure at the flowline inlet. This is the same pressure as in the manifold that is connected to the flowline.
- $p_{lp,p}$: Pressure at low point in the horizontal pipeline.
- $p_{lp,r}$: Pressure at low point in the riser.
- $p_{r,t}$: Pressure at top of the riser.
- p_s : Separator pressure, given as a constant boundary condition.

Model structure

The governing state equations for the flowline model are mass balances for each phase in pipeline and riser. Differential states are mass of each phase in the pipeline and riser part, where nomenclature is given in Appendix A.

$$\text{States:} \quad z_f = \begin{bmatrix} m_{gp} \\ m_{op} \\ m_{wp} \\ m_{gr} \\ m_{or} \\ m_{wr} \end{bmatrix}. \tag{4.6}$$

All differential and algebraic equations for the flowline model are given in Appendix B.2. These can be written as:

$$\dot{z}_f = f_f(z_f, y_f, w_{in}), \tag{4.7a}$$
$$0 = g_f(z_f, y_f, w_{in}), \tag{4.7b}$$

where $z_f \in \mathbb{R}^6$ are flowline states and $y_f \in \mathbb{R}^{40}$ are algebraic variables (not including w_{in}). The inlet flows of gas, oil and water contained in $w_{in} \in \mathbb{R}^3$ are here treated as external inputs. Differential and algebraic equations are given by the mappings f_f and g_f respectively.

The flowline model can be viewed as a subsystem with inputs being inlet flows of each phase from the manifold connected to the specific flowline. Outputs are pressure at the flowline inlet and outlet flows at the topside according to Figure 4.6.

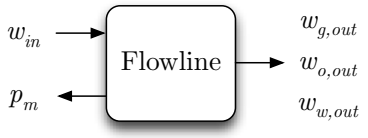


Figure 4.6: Flowline subsystem.

4.4 Overall model

By connecting the well and flowline subsystems through manifolds, it is possible to construct a general model of a subsea production plant consisting of a well-flowline network. Well and flowline model equations are both represented as semi-explicit DAEs, and by coupling these together, an overall model can be formed. In this way, the external inputs for well and flowline can be coupled according to Figure 4.7.

For a general number of wells and flowlines, let:

$$z = \begin{bmatrix} z_{w,1} \\ \vdots \\ z_{w,J} \\ z_{f,1} \\ \vdots \\ z_{f,L} \end{bmatrix}, \quad y = \begin{bmatrix} y_{w,1} \\ \vdots \\ y_{w,J} \\ y_{f,1} \\ \vdots \\ y_{f,L} \end{bmatrix}, \quad u = \begin{bmatrix} u_{gi,1} \\ \vdots \\ u_{gi,J} \end{bmatrix}, \tag{4.8}$$

be states, algebraic variables and control inputs for the full model, indexed according to Table 4.1. It is then possible to write the overall well-flowline network as a semi-explicit DAE of index-1. In the overall model, algebraic variables p_m and w_{in} are not treated as external inputs. The overall model is written as:

$$\dot{z} = f(z, y, u), \tag{4.9a}$$
$$0 = g(z, y, u), \tag{4.9b}$$

where $z \in \mathbb{R}^{3J+6L}$, $y \in \mathbb{R}^{19J+43L}$ and $u \in \mathbb{R}^{J}$ are states, algebraic variables and control inputs respectively. External inputs p_m and w_{in} have been added as algebraic variables, increasing the dimension of y to $(18+1)J$ and $(40+3)L$. An equal number of algebraic equations is also the result of adding manifold equations. A complete overview of dimensions related to system equations and variables can be seen in Table 4.2. A general well-flowline model is illustrated in Figure 4.8.

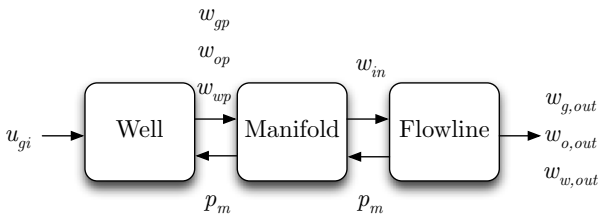


Figure 4.7: Connecting a single well to a flowline through the manifold.

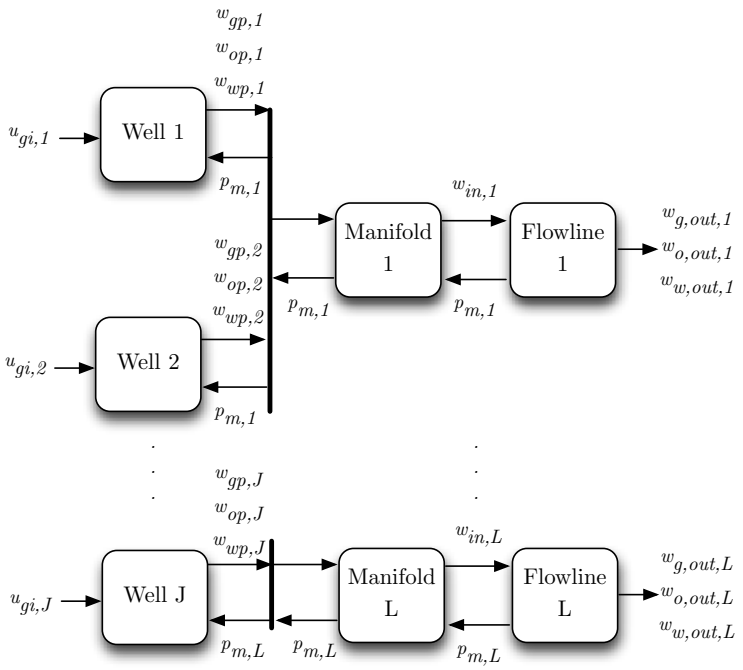


Figure 4.8: An overall model consists of connected subsystems. Sets of wells are connected to a flowline through its manifold. All wells connected to the same manifold meet the same manifold pressure p_m.

Set	Index	Description
$\mathcal{J} = \{1, ..., J\}$	j	Wells
$\mathcal{L} = \{1, ..., L\}$	l	Flowlines

Table 4.1: Sets and indices for a general well-flowline model.

	Well	Flowline	Overall (J wells, L flowlines)
Differential states	3	6	$3J + 6L$
Algebraic variables	18	40	$19J + 43L$
Control inputs	1	0	J
External inputs	1	3	0
Differential equations	3	6	$3J + 6L$
Algebraic equations	18	40	$19J + 43L$

Table 4.2: Dimensions of system equations and variables. External inputs for each subsystem become algebraic variables in the overall model.

4.5 Approximating discontinuities

When using JModelica.org's direct collocation algorithm and the CasADi package for automatic differentiation, there is currently no support for hybrid/discrete behavior. In the well-flowline model, valve equations are modeled using max(.) functions, which are not supported by CasADi. To handle this, a smooth approximation is used, that is

$$\max(x, y) \approx f(x, y) = \frac{\sqrt{(x-y)^2 + \varepsilon^2}}{2} + \frac{x+y}{2}, \tag{4.10}$$

where the parameter ε determines the approximation error. In the model equations (B.6)-(B.7) and (B.62)-(B.67), a function $\max(0, y)$ is used such that (4.10) becomes

$$f(y) = \frac{\sqrt{y^2 + \varepsilon^2}}{2} + \frac{y}{2}, \tag{4.11}$$

where $f \in C^2, \forall \varepsilon \neq 0$. This can be visualized for different values of ε shown in Figure 4.9. The maximum absolute value of the error occurs at $y = 0$ and is given as

$$|\max(0, y = 0) - f(y = 0)| = |0 - \frac{\varepsilon}{2}| = \frac{\varepsilon}{2}. \tag{4.12}$$

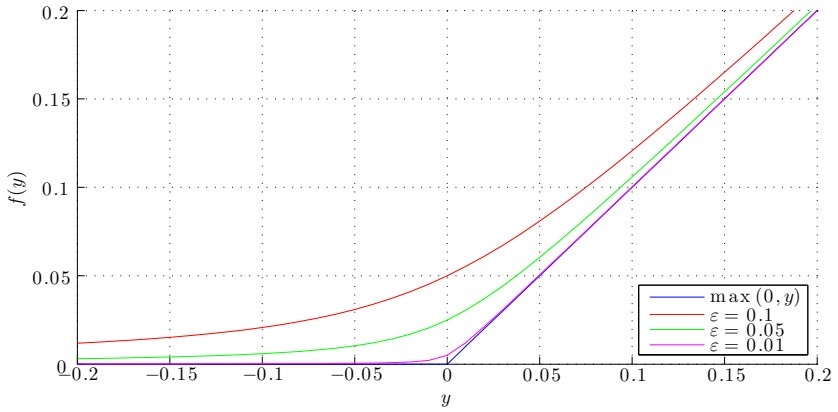


Figure 4.9: Comparison of $\max(0, y)$ and the smooth approximation.

4.6 Modelica implementation

A well-flowline network as presented in Section 4.4 has a structure that is suited for an object-oriented implementation approach. Both well and flowline can be treated as independent subsystems, which may be connected through a manifold. An implementation in the Modelica language is done by defining each subsystem with its parameters and equations and connecting them through the `connector` primitive which is a feature used to describe interactions between subsystems in Modelica. The connector will in this case represent the manifold linking flow and pressure between a well and flowline as illustrated by Figure 4.10.

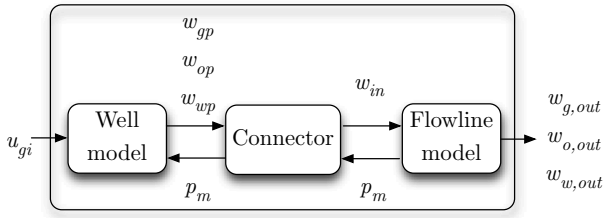


Figure 4.10: A well and flowline model interacts through a connector.

A general model can be made by using multiple connectors to connect each well with a flowline as shown in Figure 4.11. The connector will provide necessary model equations for routing flows and pressures between a connected well and flowline.

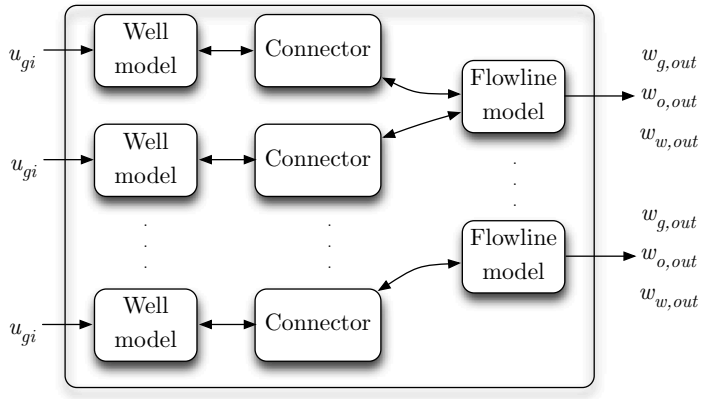


Figure 4.11: Multiple wells and flowlines interact through several connectors. Each one connecting a well with the corresponding flowline.

4.7 Test simulation

A previous well-flowline model consisting of 5 wells and 2 flowlines has been implemented in Matlab [25]. Here, a comparison of ODE solvers from Matlab and JModelica.org is given in Table 4.3. The ODE solver included in JModelica.org is CVODE, using *Backward Differentiation Formula* (BDF) or *Adams-Moulton* (AM) multistep methods.

In Matlab, the semi-explicit DAE is solved by first obtaining a solution of the algebraic equations for a given set of states and control inputs, and then solving the differential equations. In this way, it is possible to use one of the built-in ODE solvers. In JModelica.org, when importing a model as a FMU for simulation, the DAE is automatically converted into an equivalent ODE enabling use of CVODE.

Matlab

Solver	Abstol	Reltol	Time [s]	Steps	Function evaluations	Remark
ODE45	1.E-06	1.E-06	4.098	1194	7702	
ODE15s	1.E-06	1.E-06	0.3995	343	693	
ODE23t	1.E-06	1.E-06	0.861	733	1291	
ODE23s	1.E-06	1.E-06	5.2891	492	14770	
ODE113	1.E-06	1.E-06	2.776	2582	5459	

JModelica.org

Solver	Abstol	Reltol	Time [s]	Steps	Function evaluations	Remark
CVODE BDF	1.E-06	1.E-06	0.8146	481	669	
	1.E-05	1.E-05	0.5905	341	495	
	1.E-04	1.E-04	0.3652	199	279	
	1.E-03	1.E-03	0.2633	135	190	Unstable
CVODE AM	1.E-06	1.E-06	1.1494	696	971	
	1.E-05	1.E-05	0.7713	481	631	
	1.E-04	1.E-04	0.5701	339	479	Unstable
	1.E-03	1.E-03	0.4261	234	359	Unstable

Table 4.3: Comparison between ODE solvers in Matlab and JModelica.org. Abstol & Reltol are absolute and relative tolerance respectively. Time horizon is set to 5 hours.

Simulations are given in Figure D.2-D.5. Here, ODE23t from Matlab and CVODE with BDF is compared, each with absolute & relative tolerance set to 1.E-06.

Chapter 5

Formulation of optimal control problem

In this chapter, an optimal control problem is proposed. This is based on the well-flowline model in Chapter 4.

In Section 5.1, two different objective functions are given: maximization of production and reference tracking. The first seeks to maximize oil production over a given time horizon. For reference tracking, a constant unreachable reference value is used. In this case, the objective aims to minimize deviation between production and the reference. Both objective functions include a cost of lift-gas usage. A more general objective may include a profit function formulation to reflect income from oil and gas, and a cost of water treatment [14].

Further, a description of bounds and total lift-gas constraint are presented in Section 5.2. The OCP consists of all wells and flowlines, coupled by a total gas capacity constraint. Without this constraint, the full problem can be divided into subproblems depending on routing configuration, treating each set of wells connected to a single flowline as a subproblem.

The overall OCP is cast as a DAE-constrained Lagrange problem in Section 5.3. Formulation into a Mayer problem and DAE to ODE transformation is described in Section 5.4.

5.1 Objective

An objective function for the well-flowline network is formulated to reflect income from production and cost of using lift-gas injection. Here, short-term production planning is considered with two types of objective functions, i.e. maximize oil production and reference tracking as illustrated by Figure 5.1. In addition, a cost of total lift-gas usage is included. For a set of wells and flowlines with indices according to Table 4.1, the objective functions may be defined as:

$$\Phi_{max}(y,u) = \int_{t_0}^{t_f} -\sum_{l\in\mathcal{L}} q w_{o,out,l}^2 + \sum_{j\in\mathcal{J}} r u_{gi,j}^2 dt, \tag{5.1}$$

$$\Phi_{ref}(y,u) = \int_{t_0}^{t_f} \sum_{l\in\mathcal{L}} q(w_{o,out,l} - w_{o,out}^r)^2 + \sum_{j\in\mathcal{J}} r u_{gi,j}^2 dt, \tag{5.2}$$

where (5.1) aims to maximize oil production and (5.2) seeks to track a reference. The weights q and r represent the relative importance between production and lift-gas usage. In this thesis, a reference is not known a priori, such that a constant high value of $w_{o,out}^r$ is used, which in practice leads to an unreachable reference.

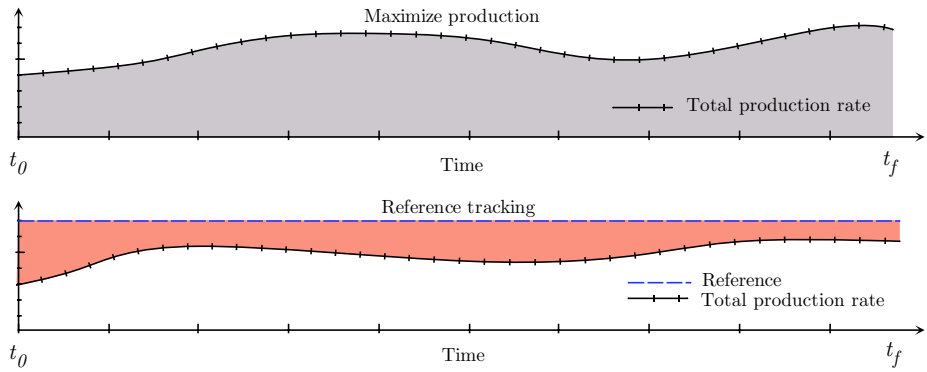


Figure 5.1: Maximize production and tracking objectives. The first seeks to maximize accumulated production as illustrated by the grey area. For tracking, the objective attempts to minimize the accumulated deviation between an unreachable reference and production as illustrated by the red area. Each objective includes a cost of total lift-gas usage.

5.2 Constraints

For multiple wells, there are constraints related to the total lift-gas usage and bounds on lift-gas injection for each well. These are written as:

$$u_{min} \leq u_{gi,j} \leq u_{max} \ , j \in \mathcal{J}, \tag{5.3}$$

$$U_{tot,min} \leq \sum_{j \in \mathcal{J}} u_{gi,j} \leq U_{tot,max}, \tag{5.4}$$

where (5.3) are bounds on lift-gas injection for each well and (5.4) is a constraint on total lift-gas usage.

From an optimization point of view, (5.4) is the only constraint coupling all wells together, thus forming a full optimization problem consisting of all wells and flowlines. Without this constraint, it would be possible and probably more efficient to divide the full problem into subproblems, with each subproblem consisting of a set of wells connected to a flowline, as illustrated by Figure 5.2. This is possible due to the assumption of a constant separator pressure p_s.

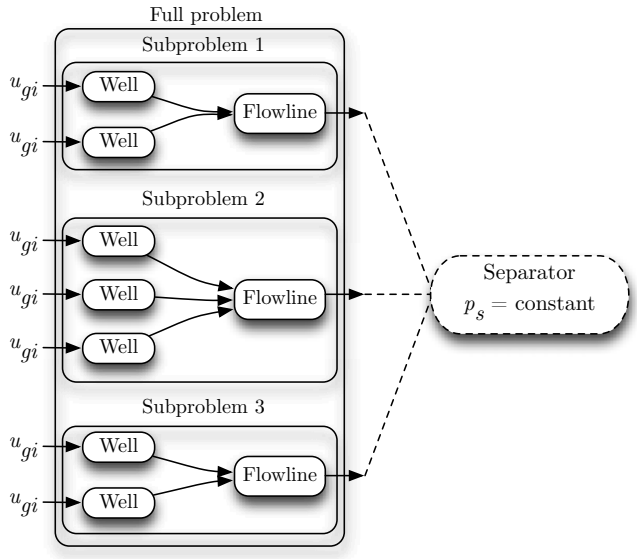


Figure 5.2: The full problem consists of subproblems linked by the total lift-gas usage constraint. Subproblems can be formed according to the routing configuration.

5.3 Overall optimal control problem

The optimal control problem for a well-flowline network can be formulated as:

$$\begin{aligned}
\min_{z,y,u} \quad & \Phi(y,u) && \text{(5.5a)}\\
\text{s.t.} \quad & \dot{z} = f(z,y,u), && \text{(5.5b)}\\
& z(t_0) = z_0, && \text{(5.5c)}\\
& g(z,y,u) = 0, && \text{(5.5d)}\\
& U_{tot,min} \leq \sum_{j\in\mathcal{J}} u_{gi,j} \leq U_{tot,max}, && \text{(5.5e)}\\
& u_{min} \leq u_{gi,j} \leq u_{max} \quad , j \in \mathcal{J}, && \text{(5.5f)}\\
& t \ \in \ [t_0, t_f],
\end{aligned}$$

where the well-flowline model is given as a semi-explicit DAE in (5.5b)-(5.5d). Control inputs being lift-gas injection rates are constrained by total lift-gas usage and bounds for each well in (5.5e)-(5.5f).

One of two different objectives can replace (5.5a), that is:

$$\Phi_{max}(y,u) = \int_{t_0}^{t_f} -\sum_{l\in\mathcal{L}} q w_{o,out,l}^2 + \sum_{j\in\mathcal{J}} r u_{gi,j}^2 dt, \tag{5.6}$$

$$\Phi_{ref}(y,u) = \int_{t_0}^{t_f} \sum_{l\in\mathcal{L}} q(w_{o,out,l} - w_{o,out}^r)^2 + \sum_{j\in\mathcal{J}} r u_{gi,j}^2 dt, \tag{5.7}$$

where (5.6) seeks to maximize oil production during the time horizon and (5.7) represents a tracking problem using an unreachable reference.

As (5.5) is given as a Lagrange problem, a reformulation into a Mayer problem is done in Section 5.4. For the performance assessment, all methods use the Mayer formulation. In addition, as shooting methods depend on the ODE integrator CVODES, it is also necessary to transform the DAE system equations to equivalent ODEs; see Section 5.4.

5.4 Alternative OCP formulations

The well-flowline OCP (5.5) is reformulated into a Mayer problem by introducing the cost state z_c and following the procedure in Section 2.1 yielding:

$$\begin{aligned}
\min_{z,y,u} \quad & z_c(t_f) && \text{(5.8a)}\\
\text{s.t.} \quad & \dot{z} = f(z,y,u), && \text{(5.8b)}\\
& z(t_0) = z_0, && \text{(5.8c)}\\
& g(z,y,u) = 0, && \text{(5.8d)}\\
& \dot{z}_c = \dot{\Phi}(y,u), && \text{(5.8e)}\\
& z_c(t_0) = 0, && \text{(5.8f)}\\
& U_{tot,min} \leq \sum_{j\in\mathcal{J}} u_{gi,j} \leq U_{tot,max}, && \text{(5.8g)}\\
& u_{min} \leq u_{gi,j} \leq u_{max} \quad , j \in \mathcal{J}, && \text{(5.8h)}\\
& t \ \in \ [t_0, t_f],
\end{aligned}$$

where (5.8b)-(5.8d) is the well-flowline DAE, while (5.8e) is the additional cost state including the objective function.

Problem (5.8) is suited for the direct collocation algorithm, which handles DAE-constrained OCPs. For shooting methods using the ODE integrator CVODES, the DAE system needs to be transformed into equivalent ODEs. This transformation is non-trivial and is performed by CasADi functions located in the `SymbolicOCP` class. These eliminate all algebraic variables and transform (5.8) into:

$$\begin{aligned}
\min_{z,u} \quad & z_c(t_f) && \text{(5.9a)}\\
\text{s.t.} \quad & \dot{z} = \overline{f}(z,u), && \text{(5.9b)}\\
& z(t_0) = z_0, && \text{(5.9c)}\\
& \dot{z}_c = \overline{\dot{\Phi}}(z,u), && \text{(5.9d)}\\
& z_c(t_0) = 0, && \text{(5.9e)}\\
& U_{tot,min} \leq \sum_{j\in\mathcal{J}} u_{gi,j} \leq U_{tot,max}, && \text{(5.9f)}\\
& u_{min} \leq u_{gi,j} \leq u_{max} \quad , j \in \mathcal{J}, && \text{(5.9g)}\\
& t \ \in \ [t_0, t_f],
\end{aligned}$$

where the algebraic variables y have been eliminated, as these are functions of states z and control inputs u. System equations $\overline{f}(z,u)$ and objective $\overline{\Phi}(z,u)$ are now functions of z and u only. The transformation is possible due to the invertibility of $g(z,y,u)$ in (5.8d).

Chapter 6

Performance assessment

The well-flowline OCP presented in Chapter 5 is solved by means of direct optimization methods: *direct collocation* (DC), *single shooting* (SS) and *multiple shooting* (MS). In this chapter, a description of the performance assessment is given. This consists of:

- Algorithmic performance: A comparison of running times, number of iterations and robustness in terms of varying algorithm parameters. The parameters that will be varied are: number of elements, number of interpolation points within each element (polynomial approximation degree), integrator tolerance and control input discretization.
- Production optimization: Assessment of productivity increase by applying dynamic optimization. Scenarios for a diversity of constraints are presented.

Complete simulation and optimization results are presented in Appendix D.

6.1 Assessment description

The methods to be assessed are:

- **Direct collocation (DC)** : JModelica.org's algorithm for direct collocation using CasADi for automatic differentiation (`LocalDAECollocationAlg`).
- **Single shooting (SS)** : The single shooting implementation presented in Section 3.5.2.
- **Multiple shooting (MS)** : The multiple shooting implementation presented in Section 3.5.3.

These methods are applied to the well-flowline system presented in Chapter 4 using $J = 5$ wells, $L = 2$ flowlines and a constant routing configuration according to Table 6.1. Model parameters are given in Table C.1-C.2.

Flowline 1	Well 1	Well 2	
Flowline 2	Well 3	Well 4	Well 5

Table 6.1: Routing configuration used for all OCPs and simulations.

DC solves the DAE-constrained Mayer problem given in (5.8). SS and MS solve the ODE-constrained Mayer problem (5.9). As both problems are formulated in Mayer form, an additional cost state is added, resulting in $3J + 6L + 1$ differential states and equations. According to Table 4.2, this leads to a number of variables and equations given in Table 6.2. For SS and MS methods, DAEs are transformed into ODEs, eliminating all algebraic variables y.

Differential states	28
Algebraic variables	181
Control inputs	5
Differential equations	28
Algebraic equations	181

Table 6.2: Dimensions of system equations and variables for the well-flowline OCP.

Operating system	Windows 7
Processor	Intel Core i5 2500 3.30 GHz
RAM	8 GB
System type	64-bit

Table 6.3: Computer specifications for solving the optimization problem.

IPOPT tolerance (`tol`) is set to 0.1 and maximum iterations (`max_iter`) is set to 150. All other options are set to default values.

Initial conditions and guess

- **DC:** A random uniform number in the interval $[1, 10]$ as a constant profile for the whole time horizon for all control inputs is used to simulate an initial guess profile, which is then supplied as initial guess to the DC algorithm according to Section 3.6.
- **SS:** A random uniform number in the interval $[1, 10]$ as a constant profile for the whole time horizon for all control inputs is used as initial guess.
- **MS:** A random uniform number in the interval $[1, 10]$ as a constant profile for the whole time horizon for all control inputs and initial conditions for states at grid points are used as initial guess.

Initial conditions are obtained from the steady-state solution of the well-flowline DAE with $u_{gi,j} = 1\ , \forall j \in \mathcal{J}$. A weight of $q = 1$ and $r = 0.1$ is used by default.

- **Algorithmic performance**: Results for DC, SS and MS are given in Table D.4-D.6 for various algorithm parameters. Each row corresponds to average values of 10 runs for DC and SS, and 1 run for MS. For each run, a different initial guess is chosen. All running times and iterations for DC and SS are shown in Figure D.6-D.7. A comparison of optimized profiles for the methods is shown in Figure D.8-D.9. The time horizon is set to $6[h]$.
- **DC approximation:** A comparison between approximated and simulated profiles for various N and K in the DC algorithm is shown in Figure D.10. Corresponding profiles for control inputs are given in Figure D.11.
- **Reference tracking:** Production increase after applying SS with reference tracking for various $U_{tot,max}$ is shown in Figure D.12. The same $U_{tot,max}$ is used before and after applying optimized control inputs at $t = 6$ $[h]$.
- **Maximize production**: Figure D.13 shows SS applied with maximize production objective for various $U_{tot,max}$, comparing the cost state $z_c(t)$ throughout the time horizon (indicating productivity increase).
- **Unconstrained optimum**: Figure D.14 shows SS applied with maximize production objective with zero weighting ($r = 0$) on lift-gas usage for an increasing $U_{tot,max}$. The intention is to show that production rates reach an unconstrained optimum even if $U_{tot,max}$ increases further, i.e. the total lift-gas capacity constraint becomes inactive after a certain point.
- **Varying bounds**: Figure D.15-D.17 shows SS applied with maximize production objective for various bounds and total lift-gas capacities.
- **Time-varying total lift-gas capacity**: Figure D.18 shows SS applied with maximize production objective for an increasing $U_{tot,max}$.

Chapter 7

Discussion

In this chapter, main points from the performance assessment in Chapter 6 are given. Algorithmic performance of the three methods is treated in Section 7.1. This includes an analysis of running times, iterations and robustness in terms of varying algorithm parameters.

Section 7.2 provides an analysis from a production optimization view. Different scenarios are assessed; comparing production increase by applying optimized control inputs with the case of using constant control settings. For reference tracking, a steady-state behavior is obtained, providing a way of comparing production rates. The maximization objective is compared using the cost state $z_c(t)$, which includes the objective function. Using this, it is possible to measure the objective function value throughout the time horizon.

Finally, a qualitative comparison of the JModelica.org framework and Matlab platform is given in Section 7.3.

7.1 Algorithmic performance

Direct collocation

Solving the well-flowline OCP by means of DC is fast according to Table D.5. The vast amount of decision variables and constraints puts high demands on the underlying numerical packages, i.e. IPOPT and CasADi. The problem is efficiently solved using automatic differentiation and heuristics such as exploitation of sparsity structure of the NLP, which are all features of the DC algorithm. Decreasing K and N results in faster running times, which can also be seen in Figure D.6. This is expected as the NLP becomes smaller in terms of decision variables and constraints.

For the variable profile approximation of DC, Figure D.10-D.11 shows the effect of varying N and K. Decreasing the value of these parameters, leads to a worse approximation when comparing the approximated profiles to similar ones obtained from numerical integration. For K=2, algebraic variable profiles and control inputs become piecewise linear functions, i.e. Lagrange polynomials of degree K-1, as would be expected from the interpolation equations (2.36)-(2.37). From the results it can be argued that there is a trade-off between a good approximation and fast running time. For this comparison, a low capacity constraint is used to obtain oscillating production rates, which makes it easier to see the difference between approximated and simulated variable profiles.

DC is sensitive to initial guess in terms of running times and number of iterations; see Figure D.6. In particular, for this problem it is necessary to provide an initial guess profile. Without this, the algorithm fails at the first iteration. Further, scaling of model equations and variables improves running time. A comparison between optimization with and without scaling is given in Table D.3. Here, it can be seen that the running time for DC is approximately 3 times greater for the case without scaling, and 10 out of 10 runs terminate due to exceeding maximum iterations. However, scaling requires a priori knowledge of the nominal values for each variable to be scaled.

Single shooting

SS leads to the smallest NLP with decision variables being the control inputs only. For the parameters tested here, average running times are less than 11 seconds, with a corresponding low number of iterations; see Table D.6 and Figure D.7. A high integrator tolerance often leads to faster running times compared to a low tolerance. However, a too high tolerance can yield slow running times, e.g. reference tracking with Tol=1.E-02.

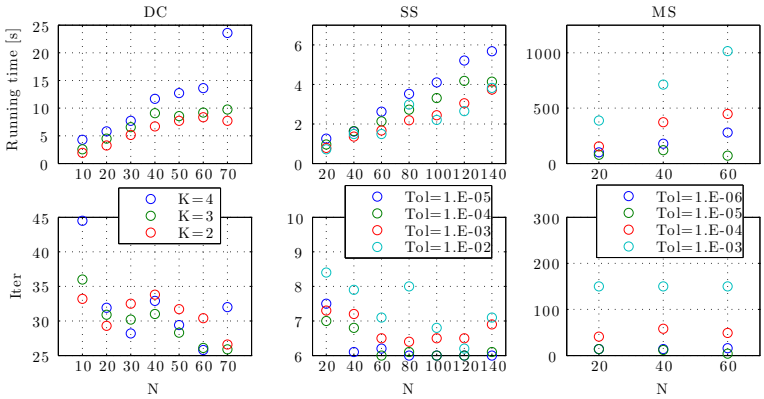


Figure 7.1: Average running times and iterations for DC, SS and MS with maximization objective.

Decreasing the control input parameterization N leads to faster running times, which is expected since the NLP becomes smaller in terms of decision variables. The number of iterations seems to be slightly unaffected by N, except for the case of reference tracking with Tol=1.E-02, which also leads to a somewhat different objective value.

SS is the most efficient algorithm and seems to be almost unaffected by the initial guess according to running times and number of iterations in Figure D.7, as each run uses a different guess. Further, SS leads to running times of approximately 17 times greater for the case without scaling; see Table D.3. Here, 9 out of 10 runs terminate due to exceeding the maximum iterations and one due to restoration failed.

Multiple shooting

MS leads to a larger NLP than SS, but smaller than DC. However, the running times do not indicate an efficient algorithm implementation; see Table D.4. For this reason, only 1 run is performed for each parameter combination. For higher tolerances, the algorithm terminates due to exceeding maximum iterations. It may be expected that MS lead to lower performance because of the increased NLP size and the need for more sensitivity information. MS is also sensitive to scaling as it terminates 10 out of 10 times with exitflag: Invalid number in NLP, for the case without scaling, as seen in Table D.3.

Solution profiles from DC, SS and MS indicate almost similar shapes as shown in Figure D.8-D.9. Table D.4-D.6 are also visualized in Figure 7.1.

7.2 Production optimization

Reference tracking

Figure D.12 shows production increase after applying optimized control inputs yielding an approximate steady-state response in total production rates. It can be seen that the increase in oil production rates is highest when the total lift-gas capacity $U_{tot,max}$ is low. Table 7.1 shows total oil production rates before and after applying optimized control with the corresponding percentage change. Reference tracking is used here because it leads to profiles resembling a steady-state behavior.

Total oil production rate

$U_{tot,max}[kg/s]$	Before $[kg/s]$	After $[kg/s]$	Percentage change [%]
5	21.58	22.26	3.15
6	22.32	22.98	2.96
7	22.88	23.54	2.88
8	23.37	23.84	2.01
9	23.59	23.89	1.27
10	23.77	23.89	0.50

Table 7.1: Total oil production rates at $t = 4[h]$ (before) and $t = 9[h]$ (after).

Varying bounds and total lift-gas capacity

Figure D.15-D.17 shows production rates and control inputs for diverse bounds u_{min} and total lift-gas capacity $U_{tot,max}$. From these it can be seen that low bounds and/or low gas capacity leads to oscillatory behavior in both production rates and control inputs. Figure D.18 shows similar behavior when keeping the bounds constant, while increasing capacity $U_{tot,max}$.

Unconstrained optimum

By letting $r = 0$ in the objective function for maximization, this corresponds to only maximizing oil production without adding a cost of lift-gas usage. SS using this objective function over a time horizon with an increasing $U_{tot,max}$ is shown in Figure D.14. Here, it is seen that for $U_{tot,max} \geq 13$ the total lift-gas constraint becomes inactive. That is, even if there is no cost of using lift-gas, production has reached an unconstrained optimum where further lift-gas injection will not increase production anymore.

Maximize production

Maximization of production is performed over a 12-hour time horizon for a range of $U_{tot,max}$ as shown in Figure D.13. For low lift-gas capacity, an oscillating production is observed. However, increasing capacity leads to stable production.

To measure productivity increase over a time horizon, the cost state $z_c(t)$ is used. For the maximization objective, this is written as

$$z_c(t) = \Phi_{max}(y, u) = \int_{t_0}^{t_f} -\sum_{l \in \mathcal{L}} q w_{o,out,l}^2 + \sum_{j \in \mathcal{J}} r u_{gi,j}^2 dt. \tag{7.1}$$

The optimization formulation used in this thesis is stated as a minimization problem, aiming to make the objective function value $z_c(t_f)$ as negative as possible.

Optimized control inputs are applied at $t = 2.4[h]$ and $U_{tot,max}$ is kept constant. The cost state $z_c(t)$ decreases more when applying optimized control inputs, compared to not doing it. For the latter case, control inputs are set to be the same as before. In Table 7.2 it is seen that $z_c(t_f)$ decreases with higher capacity $U_{tot,max}$, but the percentage change of $z_c(t_f)$ is greatest for a low capacity when comparing the objective values with and without optimal control.

Cost state: $z_c(t_f)$ for $t_f = 12[h]$

$U_{tot,max}$	Without OC	With OC	Percentage change [%]
5	-10,138,432	-11,654,389	14.95
6	-10,889,610	-11,544,917	6.02
7	-11,467,704	-11,939,498	4.11
8	-11,904,728	-12,310,983	3.41

Table 7.2: Comparison of $z_c(t_f)$ with and without using *optimal control* (OC). The objective is to maximize production over a 12-hour time horizon.

The result of applying optimal control leads to a potential gain when considering the objective function value. However, in practice, the oscillating behavior of control inputs and production rates are considered to be non-desirable solutions. The oscillating behavior seems to appear for low gas capacities and low bounds on lift-gas injection. Further constraints on production rates may stabilize production and eliminate the oscillations.

7.3 Platform comparison

Using the JModelica.org framework, optimal control problems can be solved efficiently through moderate implementation effort. There are several advantages of this approach:

- A state of the art algorithm for direct collocation solves OCPs with models formulated in Modelica and Optimica. Performance can be improved by supplying initial guess profiles, scaling of models and adjustment of algorithmic parameters.
- CasADi enables a quick way for implementing optimization-based algorithms with moderate effort, e.g. shooting methods. In particular, derivative information including sensitivity analysis is automatically supplied to IPOPT. Additional features include DAE to ODE transformation, model import from Modelica and scaling of models.
- Direct collocation, single shooting and multiple shooting methods use automatic differentiation with CasADi, which includes heuristics for improving efficiency of the AD computations.
- CVODES and IPOPT are well-established numerical packages for simulation and nonlinear programming respectively.
- The Modelica language is well suited for implementing large-scale models and includes a rich set of features that have been utilized in this thesis, e.g. object-orientation, scalability and acausal modeling.

Utilization of existing numerical packages can result in efficient algorithms compared to implementing these from scratch. In particular, nonlinear optimization in Matlab using the built-in function `fmincon` is by default using a finite difference approach for computing derivatives related to the NLP, which can greatly reduce performance and robustness. Additional disadvantages are the lack of methods for sensitivity analysis and automatic differentiation, which are not standard packages in the Matlab platform. These features are crucial for implementing efficient dynamic optimization solvers.

However, the Matlab platform is well known in the academic community and provides an easy user interface for debugging code. Implementation of direct collocation and shooting methods on this platform may require a tremendous effort due to the lack of underlying numerical packages. However, implementations may be easier to understand in terms of code complexity compared to JModelica.org, which is based on Python or C as standard languages.

Chapter 8

Conclusion

In this thesis, dynamic optimization is applied to an oil gathering system consisting of a well-flowline network. The model is written in the Modelica language by using an object-oriented approach, treating wells and flowlines as subsystems. A general model is constructed by connecting the subsystems, thus promoting scalability and flexibility for further usage.

An optimal control problem is defined to maximize production or tracking a reference with an additional cost related to the use of lift-gas injection. This optimization problem is constrained by continuous-time nonlinear differential algebraic equations representing the well-flowline network. The Modelica-extension Optimica is used to formulate the optimal control problem on a high abstraction level. To this end, three approaches have been explained, implemented and assessed. These are direct collocation, single shooting and multiple shooting.

JModelica.org's built-in algorithm for direct collocation solves optimal control problems constrained by differential algebraic equations. The shooting methods solve optimal control problems constrained by ordinary differential equations, using the CVODES integrator. For the latter case, the differential algebraic equations are transformed into an equivalent set of ordinary differential equations. This is a non-trivial task, and CasADi-functions are used to perform this transformation.

Direct collocation using automatic differentiation with CasADi has proven to be efficient, leading to running times from 1.6-23.6 seconds, depending on algorithm parameters such as number of elements and interpolation points. However, it is sensitive to scaling and requires an initial guess profile.

Single and multiple shooting methods are implemented in Python using CasADi for automatic differentiation, CVODES for integration and IPOPT for solving the NLP. Single shooting is the fastest method with running times from 0.7-10.7 seconds, depending on control input discretization and integrator tolerance. Multiple shooting is significantly slower with running times from 71-1015 seconds, depending on control input and segment discretization as well as integrator tolerance. Both shooting methods are sensitive to scaling.

An increase of $0.5 - 3.2\%$ in total oil production rates is achievable by applying optimized control inputs with an objective of reference tracking, leading to solutions resembling a steady-state behavior. Further, maximization of production over a 12-hour time horizon has shown to increase the objective function value by $3.4-15\%$. It is also observed that for low lift-gas capacities and bounds, an oscillating production rate is obtained. Finally, an unconstrained optimum for production has been indicated, where injecting more lift-gas does not produce more oil.

Chapter 9

Further work

This thesis explores the JModelica.org framework for implementing optimization-based algorithms with models formulated in the Modelica language. Single and multiple shooting methods use the CVODES integrator for solving ODEs and IPOPT for solving NLPs. Here, the DAE system is transformed into an equivalent ODE system. Another approach may be to use the DAE integrator IDAS from Sundials. The NLP can also be solved by KNITRO, or a SQP-type algorithm.

The multiple shooting method implemented here is not particularly efficient in terms of running time. Further exploration of how to implement this method is suggested, in particular parallelization of the algorithm.

The well-flowline model used here is based on previous work and should be considered as a simplified model of a subsea production plant. Further work may include using more advanced models for multiphase flow that describe phenomena such as slug flow. In particular, the built-in component library provided with Modelica should be considered as an opportunity to construct realistic physical models. A comparison against a high-fidelity dynamic multiphase flow simulator or physical measurements may also increase credibility of such a model.

Appendix A

Nomenclature

General

z	Differential state
y	Algebraic variable
u	Control input
n_z	Number of differential states in DAE system
n_y	Number of algebraic variables in DAE system
n_u	Number of control inputs in DAE system
n_I	Number of path inequalities in OCP
N	DC: Number of elements SS: Control input discretization MS: Control input and segment discretization
K	Polynomial approximation degree for states in DC.

Well-Flowline OCP

u_{min}	Minimum lift-gas injection rate for a well	$[\frac{kg}{s}]$
u_{max}	Maximum lift-gas injection rate for a well	$[\frac{kg}{s}]$
$U_{tot,min}$	Minimum total lift-gas capacity rate	$[\frac{kg}{s}]$
$U_{tot,max}$	Maximum total lift-gas capacity rate	$[\frac{kg}{s}]$
q	Weight for oil production rate from a flowline (income)	$[-]$
r	Weight for lift-gas usage in a well (cost)	$[-]$
$w^r_{o,out}$	Reference value for oil production rate	$[\frac{kg}{s}]$
z_0	Steady-state values for well-flowline DAE	$[kg]$

Well

m_{ga}	Mass of gas in annulus	$[kg]$
m_{gt}	Mass of gas in tubing	$[kg]$
m_{lt}	Mass of liquid in tubing	$[kg]$
u_{gi}	Lift-gas injection rate (control input)	$[\frac{kg}{s}]$

Flowline

m_{gp}	Mass of gas in pipeline	$[kg]$
m_{op}	Mass of oil in pipeline	$[kg]$
m_{wp}	Mass of water in pipeline	$[kg]$
m_{gr}	Mass of gas in riser	$[kg]$
m_{or}	Mass of oil in riser	$[kg]$
m_{wr}	Mass of water in riser	$[kg]$
$w_{g,in}$	Mass flow of gas at pipeline inlet	$[\frac{kg}{s}]$
$w_{o,in}$	Mass flow of oil at pipeline inlet	$[\frac{kg}{s}]$
$w_{w,in}$	Mass flow of water at pipeline inlet	$[\frac{kg}{s}]$
$w_{g,lp}$	Mass flow of gas at low point	$[\frac{kg}{s}]$
$w_{o,lp}$	Mass flow of oil at low point	$[\frac{kg}{s}]$
$w_{w,lp}$	Mass flow of water at low point	$[\frac{kg}{s}]$
$w_{g,out}$	Mass flow of gas at riser outlet	$[\frac{kg}{s}]$
$w_{o,out}$	Mass flow of oil at riser outlet	$[\frac{kg}{s}]$
$w_{w,out}$	Mass flow of water at riser outlet	$[\frac{kg}{s}]$
p_s	Separator pressure	$[Pa]$

Appendix B

Model equations

B.1 Well

Differential state equations: Mass balances

$$\dot{m}_{ga} = w_{gl} - w_{gi} \tag{B.1}$$

$$\dot{m}_{gt} = w_{gr} + w_{gi} - w_{gp} \tag{B.2}$$

$$\dot{m}_{lt} = w_{lr} - w_{lp} \tag{B.3}$$

Algebraic equations:

Mass flows:

$$m_t = m_{gt} + m_{lt} \tag{B.4}$$
$$w_{gl} = u_{gi} \tag{B.5}$$
$$w_{gi} = C_{iv}\sqrt{\rho_{gi}\max\{0, p_{ai} - p_{ti}\}} \tag{B.6}$$
$$w_p = C_{pc}\sqrt{\rho_p \max\{0, p_p - p_m\}}u_{pc} \tag{B.7}$$
$$w_{gp} = \frac{m_{gt}}{m_t}w_p \tag{B.8}$$
$$w_{lp} = \frac{m_{lt}}{m_t}w_p \tag{B.9}$$
$$w_{wp} = r_{wc}w_{lp} \tag{B.10}$$
$$w_{op} = (1 - r_{wc})w_{lp} \tag{B.11}$$
$$w_{lr} = \rho_l Q_{max}\left(1 - (1 - C)\left(\frac{p_{bh}}{p_r}\right) - C\left(\frac{p_{bh}}{p_r}\right)^2\right) \tag{B.12}$$
$$w_{gr} = r_{glr}w_{lr} \tag{B.13}$$
$$r_{glr} = (1 - r_{wc})r_{gor} \tag{B.14}$$

Densities:

$$\rho_{gi} = \frac{M_g}{RT_a}p_{ai} \tag{B.15}$$
$$\rho_p = \frac{\rho_l M_g p_p m_t}{\rho_l R T_t m_{lt} + M_g p_p m_{gt}} \tag{B.16}$$
$$\rho_l = r_{wc}\rho_w + (1 - r_{wc})\rho_o \tag{B.17}$$

Pressures:

$$p_{ai} = \left(\frac{RT_a}{V_a M_g} + \frac{g}{2A_a}\right)m_{ga} \tag{B.18}$$
$$p_p = \frac{RT_t m_{gt}}{M_g V_t - M_g \nu_l m_{lt}} - \frac{g m_t}{2A_t} \tag{B.19}$$
$$p_{ti} = p_p + \frac{g m_t}{A_t} \tag{B.20}$$
$$p_{bh} = \frac{\left(1 + r_{glr} + \frac{r_{glr}M_g g L_w}{2RT_t}\right)p_{ti} + \rho_l g L_w}{1 + r_{glr} - \frac{r_{glr}M_g g L_w}{2RT_t}} \tag{B.21}$$

B.2 Flowline

Differential state equations: Mass balances

$$\dot{m}_{g,p} = w_{g,in} - w_{g,lp} \tag{B.22}$$
$$\dot{m}_{o,p} = w_{o,in} - w_{o,lp} \tag{B.23}$$
$$\dot{m}_{w,p} = w_{w,in} - w_{w,lp} \tag{B.24}$$

$$\dot{m}_{g,r} = w_{g,lp} - w_{g,out} \tag{B.25}$$
$$\dot{m}_{o,r} = w_{o,lp} - w_{o,out} \tag{B.26}$$
$$\dot{m}_{w,r} = w_{w,lp} - w_{w,out} \tag{B.27}$$

Algebraic equations:

Horizontal pipeline:

$$V_{g,p} = V_p - \frac{m_{op}}{\rho_o} - \frac{m_{wp}}{\rho_w} \tag{B.28}$$
$$\rho_{g,p} = \frac{m_{gp}}{V_{g,p}} \tag{B.29}$$
$$\rho_{l,p} = \frac{\rho_o \rho_w}{\rho_w m_{op} + \rho_o m_{wp}}(m_{op} + m_{wp}) \tag{B.30}$$
$$p_m = \frac{\rho_{g,p} R T_p}{M_g} \tag{B.31}$$
$$\alpha_{l,p} = \frac{\rho_w m_{op} + \rho_o m_{wp}}{\rho_o \rho_w V_p} \tag{B.32}$$
$$U_{s,l,p} = \frac{\rho_w w_{o,in} + \rho_o w_{w,in}}{\rho_o \rho_w \pi r_p^2} \tag{B.33}$$
$$\mu_{l,p} = \frac{m_{op}}{m_{op} + m_{wp}}\mu_o + \frac{m_{wp}}{m_{op} + m_{wp}}\mu_w \tag{B.34}$$
$$Re_p = \frac{2\rho_{l,p} U_{s,l,p} r_p}{\mu_{l,p}} \tag{B.35}$$
$$f_p^{1/2} = \frac{1}{-1.8\log[(\frac{\epsilon_p}{3.7D_p})^{1.11} + \frac{6.9}{Re_p}]} \tag{B.36}$$
$$\Delta P_{f,p} = \frac{\alpha_{l,p} L_p \rho_{l,p} f_p U_{s,l,p}^2}{4 r_p} \tag{B.37}$$

Riser:

$$V_{g,r} = V_r - \frac{m_{or}}{\rho_o} - \frac{m_{wr}}{\rho_w} \tag{B.38}$$

$$\rho_{g,r} = \frac{m_{gr}}{V_{g,r}} \tag{B.39}$$

$$\rho_{l,r} = \frac{\rho_w \rho_o}{\rho_w m_{or} + \rho_o m_{wr}}(m_{or} + m_{wr}) \tag{B.40}$$

$$\rho_{m,r} = \frac{m_{gr} + m_{or} + m_{wr}}{V_r} \tag{B.41}$$

$$\rho_t = \alpha_{l,t}\rho_{l,r} + (1 - \alpha_{l,t})\rho_{g,r} \tag{B.42}$$

$$p_{r,t} = \frac{\rho_{g,r} R T_r}{M_g} \tag{B.43}$$

$$\alpha_{l,r} = \frac{\rho_w m_{or} + \rho_o m_{wr}}{\rho_o \rho_w V_r} \tag{B.44}$$

$$\alpha_{l,t} = \frac{2(m_{or} + m_{wr})}{V_r \rho_{l,r}} - \frac{A_{L,p}}{\pi r_p^2} \tag{B.45}$$

$$U_{s,l,r} = \frac{\rho_w w_{o,in} + \rho_o w_{w,in}}{\rho_w \rho_o \pi r_r^2} \tag{B.46}$$

$$U_{s,g,r} = \frac{w_{g,in}}{\rho_{g,r} \pi r_r^2} \tag{B.47}$$

$$U_{m,r} = U_{s,l,r} + U_{s,g,r} \tag{B.48}$$

$$\mu_{l,r} = \frac{m_{or}}{m_{or} + m_{wr}}\mu_o + \frac{m_{wr}}{m_{or} + m_{wr}}\mu_w \tag{B.49}$$

$$Re_r = \frac{2\rho_{m,r} U_{m,r} r_r}{\mu_{l,r}} \tag{B.50}$$

$$f_r^{1/2} = \frac{1}{-1.8 \log[(\frac{\epsilon_r}{3.7 D_r})^{1.11} + \frac{6.9}{Re_r}]} \tag{B.51}$$

$$\Delta P_{f,r} = \frac{\alpha_{l,r} f_r \rho_{m,r} U_{m,r}^2 L_r}{4 r_r} \tag{B.52}$$

Connection between pipeline and riser:

$$p_{lp,p} = p_m - \Delta P_{f,p} \tag{B.53}$$
$$p_{lp,r} = p_{r,t} + \Delta P_{f,r} + \rho_{m,r} g L_r \tag{B.54}$$

$$A_{G,p} = \frac{V_{g,p}}{V_p} A_p \tag{B.55}$$
$$A_{L,p} = A_p - A_{G,p} \tag{B.56}$$

$$z_{o,p} = \frac{m_{op}}{m_{gp} + m_{op} + m_{wp}} \tag{B.57}$$
$$z_{w,p} = \frac{m_{wp}}{m_{gp} + m_{op} + m_{wp}} \tag{B.58}$$
$$z_{g,r} = \frac{m_{gr}}{m_{gr} + m_{or} + m_{wr}} \tag{B.59}$$
$$z_{o,r} = \frac{m_{or}}{m_{gr} + m_{or} + m_{wr}} \tag{B.60}$$
$$z_{w,r} = \frac{m_{wr}}{m_{gr} + m_{or} + m_{wr}} \tag{B.61}$$

$$w_{g,lp} = K_{G,p} A_{G,p} \sqrt{\rho_{g,p} \max\{0, p_{lp,p} - p_{lp,r}\}} \tag{B.62}$$
$$w_{o,lp} = K_{L,p} A_{L,p} \sqrt{\rho_o \max\{0, p_{lp,p} - p_{lp,r}\}} z_{o,p} \tag{B.63}$$
$$w_{w,lp} = K_{L,p} A_{L,p} \sqrt{\rho_w \max\{0, p_{lp,p} - p_{lp,r}\}} z_{w,p} \tag{B.64}$$
$$w_{g,out} = K_{G,r} \sqrt{\rho_t \max\{0, p_{r,t} - p_s\}} z_{g,r} \tag{B.65}$$
$$w_{o,out} = K_{L,r} \sqrt{\rho_t \max\{0, p_{r,t} - p_s\}} z_{o,r} \tag{B.66}$$
$$w_{w,out} = K_{L,r} \sqrt{\rho_t \max\{0, p_{r,t} - p_s\}} z_{w,r} \tag{B.67}$$

B.3 Single shooting NLP expressions

Decision variables are contained in $U \in \mathbb{R}^{Nn_u}$ on the following form:

$$U = \begin{bmatrix} u^1 \\ \vdots \\ u^N \end{bmatrix}. \tag{B.68}$$

State constraints on z_1 and z_2 are stacked in g with their corresponding bounds:

$$g = \begin{bmatrix} z_1(t_1) \\ \vdots \\ z_1(t_N) \\ z_2(t_1) \\ \vdots \\ z_2(t_N) \end{bmatrix}, \; g_L = \begin{bmatrix} z_{1,min} \\ \vdots \\ z_{1,min} \\ z_{2,min} \\ \vdots \\ z_{2,min} \end{bmatrix}, \; g_U = \begin{bmatrix} z_{1,max} \\ \vdots \\ z_{1,max} \\ z_{2,max} \\ \vdots \\ z_{2,max} \end{bmatrix}. \tag{B.69}$$

The objective function is given by

$$f = z_c(t_N) = z_c(t_f). \tag{B.70}$$

Variable bounds are given as:

$$u_L = \begin{bmatrix} u_{min} \\ \vdots \\ u_{min} \end{bmatrix}, \; u_U = \begin{bmatrix} u_{max} \\ \vdots \\ u_{max} \end{bmatrix}. \tag{B.71}$$

Local expressions for state variables are defined to enable construction of g. These contain state variables at time points, which are obtained by the integrator calls.

$$\text{Z1} = \begin{bmatrix} z_1(t_1) \\ \vdots \\ z_1(t_N) \end{bmatrix}, \; \text{Z2} = \begin{bmatrix} z_2(t_1) \\ \vdots \\ z_2(t_N) \end{bmatrix}. \tag{B.72}$$

B.4 Multiple shooting NLP expressions

Decision variables are contained in $V \in \mathbb{R}^{N(n_z+n_u)}$ on the following form:

$$V = \begin{bmatrix} u^1 \\ \vdots \\ u^N \\ z_0^1 \\ \vdots \\ z_0^N \end{bmatrix}, \tag{B.73}$$

where $z_0^i = \begin{bmatrix} z_{c,0}^i & z_{1,0}^i & z_{2,0}^i \end{bmatrix}^T$, $i = 1, ..., N$.

Initial conditions and continuity constraints are stacked in g:

$$g = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_N \end{bmatrix} = \begin{bmatrix} z_0^1 - z_0 \\ z_0^2 - z^1(t_1) \\ \vdots \\ z_0^N - z^{N-1}(t_{N-1}) \end{bmatrix}, \; g_L = g_U = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{B.74}$$

The objective function f is given by

$$f = z_c^N(t_N) = z_c^N(t_f). \tag{B.75}$$

Variable bounds are given as:

$$v_L = \begin{bmatrix} u_{min}^T & \dots & u_{min}^T & z_{min}^T & \dots & z_{min}^T \end{bmatrix}^T, \tag{B.76}$$

$$v_U = \begin{bmatrix} u_{max}^T & \dots & u_{max}^T & z_{max}^T & \dots & z_{max}^T \end{bmatrix}^T, \tag{B.77}$$

where $z_{min} = \begin{bmatrix} z_{c,min} & z_{1,min} & z_{2,min} \end{bmatrix}^T$ and $z_{max} = \begin{bmatrix} z_{c,max} & z_{1,max} & z_{2,max} \end{bmatrix}^T$.

Local expressions for control inputs and states in the example are defined to make formulation of continuity constraints easier. These are given by:

$$U = \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^N \end{bmatrix}, \; ZC = \begin{bmatrix} z_{c,0}^1 \\ z_{c,0}^2 \\ \vdots \\ z_{c,0}^N \end{bmatrix}, \; Z1 = \begin{bmatrix} z_{1,0}^1 \\ z_{1,0}^2 \\ \vdots \\ z_{1,0}^N \end{bmatrix}, \; Z2 = \begin{bmatrix} z_{2,0}^1 \\ z_{2,0}^2 \\ \vdots \\ z_{2,0}^N \end{bmatrix}. \tag{B.78}$$

Appendix C

Parameters

Well		
Parameter	Value	Unit
C_{iv}	0.00016	$[m^2]$
C_{pc}	0.0014	$[m^2]$
R	8.3145	$[\frac{J}{molK}]$
T_a	350	[K]
T_t	350	[K]
M_g	0.0195	$[\frac{kg}{mol}]$
g	9.81	$[\frac{m}{s^2}]$
ρ_o	930	$[\frac{kg}{m^3}]$
ρ_w	1030	$[\frac{kg}{m^3}]$
A_a	0.02	$[m^2]$
A_t	0.012	$[m^2]$
C	0.8	[-]
p_r	250	[bara]
ε	0.1	[-]

Flowline		
Parameter	Value	Unit
L_p	13 000	$[m]$
r_p	0.1	$[m]$
T_p	330	[K]
T_r	330	[K]
μ_w	$8.94 \cdot 10^{-4}$	$[Pa \cdot s]$
μ_o	$1 \cdot 10^{-4}$	$[Pa \cdot s]$
p_s	50	[bara]
L_r	600	$[m]$
r_r	0.1	$[m]$
ϵ	$2.8 \cdot 10^{-5}$	$[m^2]$
$K_{G,p}$	0.03	[-]
$K_{L,p}$	1	[-]
$K_{G,r}$	0.0034	$[m^2]$
$K_{L,r}$	0.0014	$[m^2]$

Table C.1: Well and flowline general parameters.

Parameter	Well 1	Well 2	Well 3	Well 4	Well 5	Unit
V_a	30	30	25	30	25	$[m^3]$
V_t	18	18	15	18	15	$[m^3]$
L_w	400	400	800	600	500	$[m]$
r_{wc}	0.4	0.7	0.2	0.5	0.5	[-]
r_{gor}	0.08	0.07	0.07	0.09	0.06	[-]
Q_{max}	0.025	0.05	0.035	0.10	0.02	$[\frac{m^3}{s}]$

Table C.2: Well specific parameters.

Appendix D

Optimization and simulation results

D.1 Table entries

Entry	Description
	DC: Number of elements.
N	SS: Control input discretization.
	MS: Control input and segment discretization.
K	Polynomial approximation degree for states in DC.
Tol	Integrator absolute and relative tolerance.
DV	Total number of decision variables in NLP.
Eq	Total number of equality constraints in NLP.
Ineq	Total number of inequality constraints in NLP.
Running time	Time spent in NLP solver including function evaluations.
Iter	Number of iterations.
Obj.Val	Unscaled objective function value at optimal point.
Max.Iter	IPOPT termination with exitflag: Maximum iterations exceeded
Rest.Failed	IPOPT termination with exitflag: Restoration failed
Inv.Number	IPOPT termination with exitflag: Invalid number in NLP

Table D.1: Description of table entries for optimization results.

D.2 Coupled tanks example

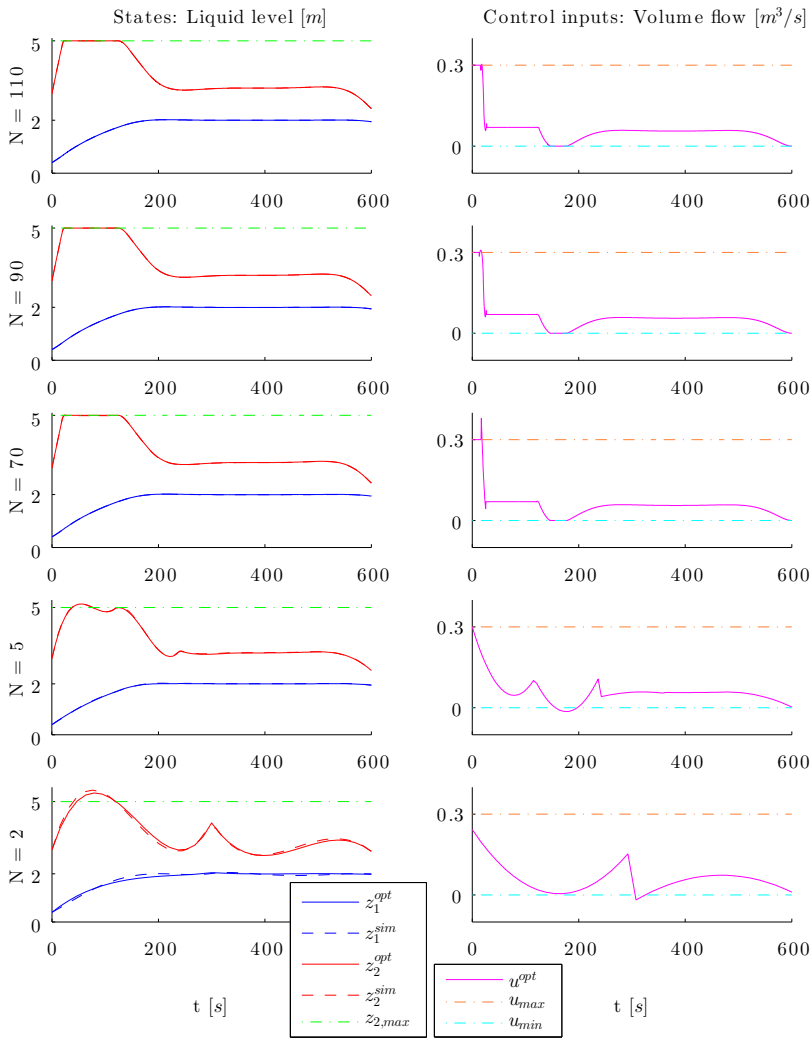


Figure D.1: Simulated profiles compared to approximated profiles from the DC algorithm when varying the number of elements N, with K=3. Corresponding control inputs are given to the right.

D.3 Test simulation of well-flowline model

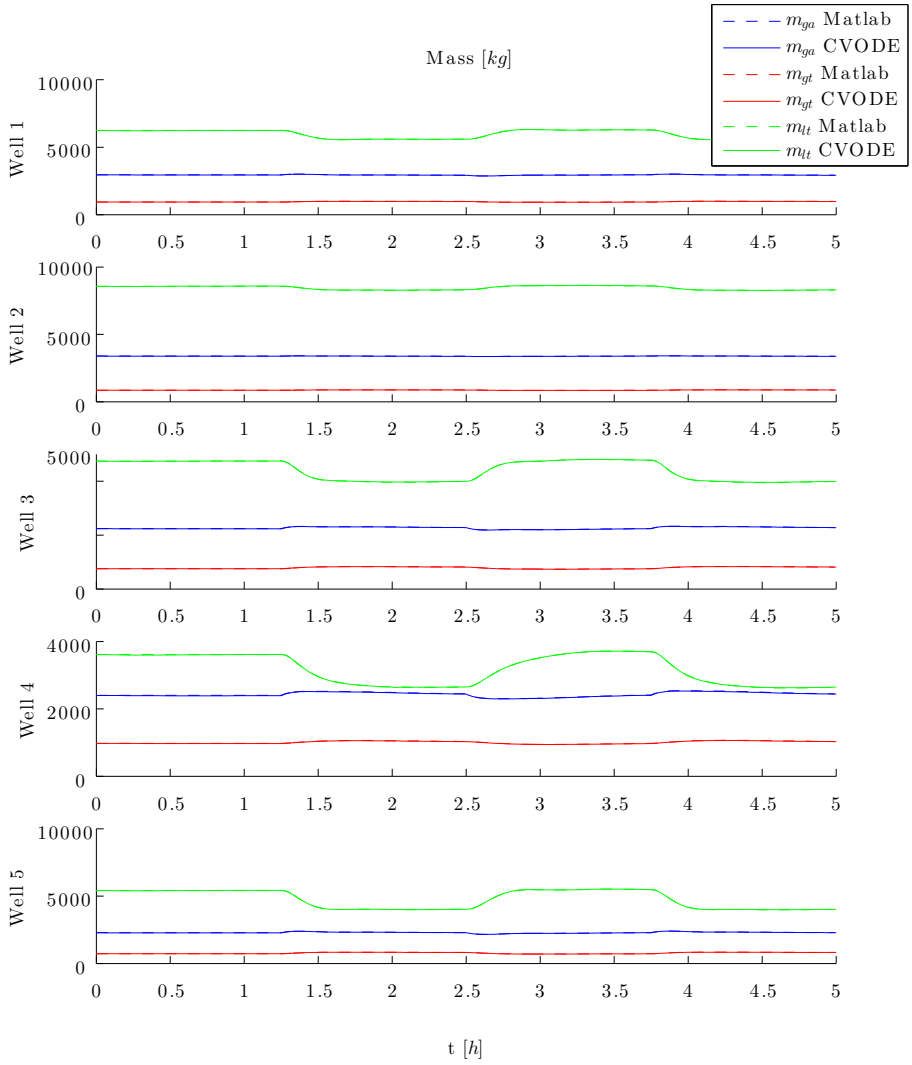


Figure D.2: Well states from test simulation of well-flowline model.

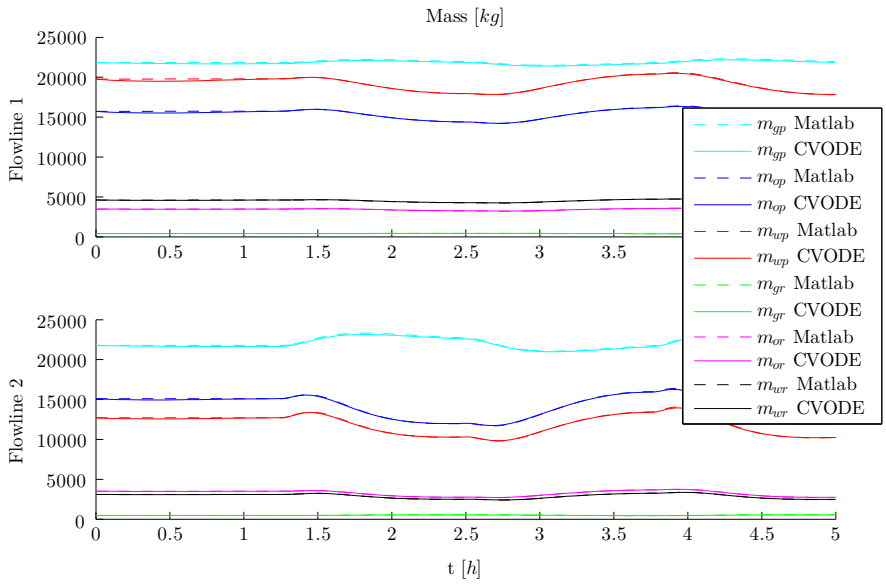


Figure D.3: Flowline states from test simulation of well-flowline model.

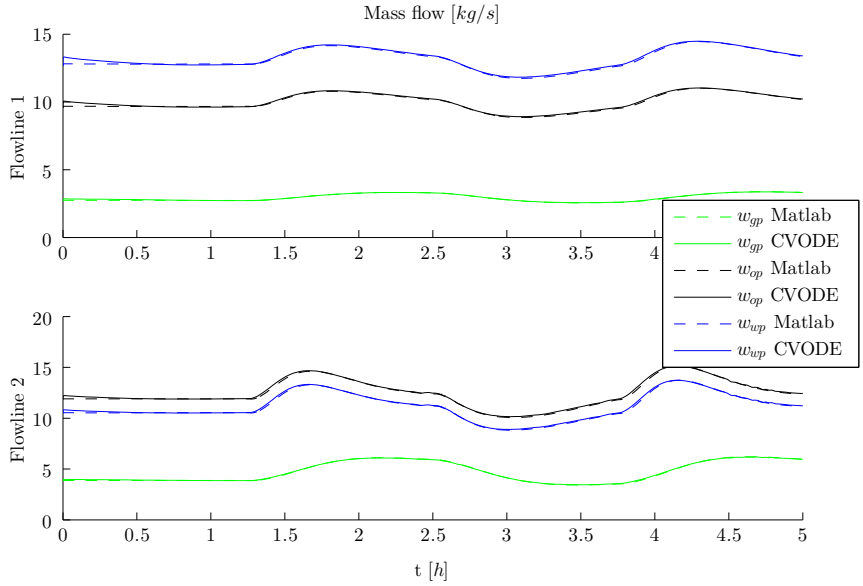


Figure D.4: Flowline outlet flows (algebraic variables) from test simulation of well-flowline model.

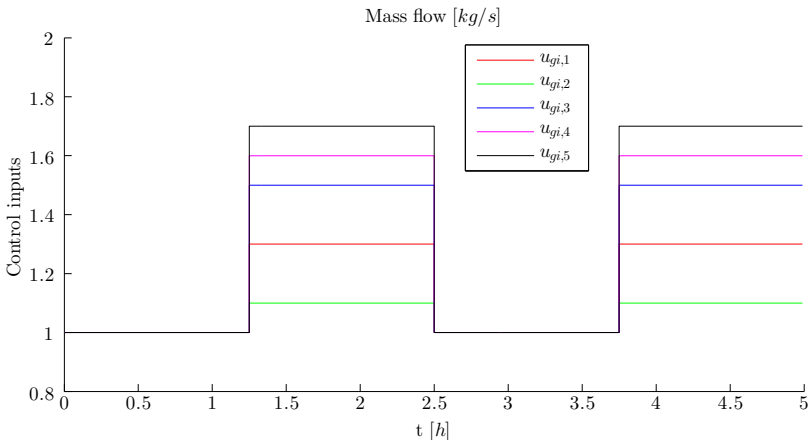


Figure D.5: Control input profile from test simulation of well-flowline model.

D.4 Optimization results

Description	u_{min}	u_{max}	$U_{tot,min}$	$U_{tot,max}$	q	r	$t_f[h]$
Value	1	2	1	7	1	0.1	6

Table D.2: Time horizon, constraint and weighting values used for algorithmic performance assessment in Appendix D.4.

	DC (N=50, K=3)		**SS** (N=60, Tol=1.E-05)		**MS** (N=60, Tol=1.E-05)	
	Scaled	Unscaled	Scaled	Unscaled	Scaled	Unscaled
Running time [s]	11.5	32.6	2.6	45.2	144.4	300.2
Iter	36.1	50	6.2	49.0	17.0	15.0
Obj.Val	-6.06	-5.05E+06	-6.06	-6.06E+06	-6.33	N/A
Max.Iter	0	10	0	9	0	0
Rest.Failed	0	0	0	1	0	0
Inv.Number	0	0	0	0	0	10

Table D.3: Performance of DC, SS and MS methods when using scaled and unscaled DAE system. Running time, Iter and Obj.Val corresponds to average values of 10 runs. Maximum iterations are set to 50. The maximize production objective is used in this comparison.

MS										
					Maximize production			**Reference tracking**		
N	Tol	DV	Eq	Ineq	Running time [s]	Iter	Obj.Val	Running time [s]	Iter	Obj.Val
60	1.E-06	1980	1680	60	280.0	16.0	-6.06	334.8	17.0	14.34
	1.E-05	1980	1680	60	71.0	4.0	-5.69	286.8	21.0	14.34
	1.E-04	1980	1680	60	447.8	49.0	-6.06	593.0	69.0	14.34
	1.E-03	1980	1680	60	1015.1	150.0	-6.06	1005.5	150.0	14.34
40	1.E-06	1320	1120	40	179.0	14.0	-6.06	177.0	14.0	14.34
	1.E-05	1320	1120	40	119.5	12.0	-6.06	170.2	19.0	14.34
	1.E-04	1320	1120	40	373.1	58.0	-6.06	906.3	150.0	14.34
	1.E-03	1320	1120	40	713.3	150.0	-6.06	698.5	150.0	14.35
20	1.E-06	660	560	20	100.2	14.0	-6.06	104.0	15.0	14.34
	1.E-05	660	560	20	78.8	14.0	-6.06	101.1	20.0	14.34
	1.E-04	660	560	20	154.4	41.0	-6.06	507.7	150.0	14.34
	1.E-03	660	560	20	387.3	150.0	-6.06	384.0	150.0	14.35

Table D.4: MS optimization results. Values of 1 run.

DC

					Maximize production			**Reference tracking**		
K	N	DV	Eq	Ineq	Running time [s]	Iter	Obj.Val	Running time [s]	Iter	Obj.Val
4	70	69934	68534	3372	23.6	32.0	-6.05	19.4	30.5	14.35
	60	59974	58774	2892	13.6	25.8	-6.06	18.2	31.6	14.35
	50	50014	49014	2412	12.7	29.4	-6.06	12.8	30.0	14.35
	40	40054	39254	1932	11.7	32.9	-6.06	10.5	30.2	14.35
	30	30094	29494	1452	7.7	28.2	-6.06	7.1	26.2	14.35
	20	20134	19734	972	5.8	31.9	-6.06	7.1	37.4	14.35
	10	10174	9974	492	4.3	44.5	-6.06	3.0	32.7	14.34
3	70	52994	51944	2532	9.7	25.9	-6.06	10.5	27.5	14.35
	60	45454	44554	2172	9.2	26.1	-6.06	9.1	26.9	14.35
	50	37914	37164	1812	8.6	28.3	-6.06	9.5	29.0	14.35
	40	30374	29774	1452	9.1	31.0	-6.06	9.1	32.9	14.35
	30	22834	22384	1092	6.6	30.2	-6.06	5.0	24.4	14.35
	20	15294	14994	732	4.5	30.9	-6.06	4.2	29.3	14.34
	10	7754	7604	372	2.6	36.0	-6.07	1.6	26.0	14.34
2	70	36054	35354	1692	7.7	26.6	-6.06	7.7	25.5	14.35
	60	30934	30334	1452	8.4	30.4	-6.06	6.1	25.0	14.35
	50	25814	25314	1212	7.7	31.7	-6.06	9.6	36.7	14.35
	40	20694	20294	972	6.7	33.8	-6.06	6.3	31.6	14.35
	30	15574	15274	732	5.2	32.5	-6.06	3.7	25.7	14.34
	20	10454	10254	492	3.2	29.3	-6.06	2.7	28.0	14.34
	10	5334	5234	252	1.9	33.2	-6.06	2.0	40.0	14.34

Table D.5: DC optimization results. Average values of 10 runs.

SS

					Maximize production			**Reference tracking**		
N	Tol	DV	Eq	Ineq	Running time [s]	Iter	Obj.Val	Running time [s]	Iter	Obj.Val
140	1.E-05	700	0	140	5.7	6.0	-6.04	5.6	6.0	14.33
	1.E-04	700	0	140	4.1	6.1	-6.04	4.3	6.0	14.34
	1.E-03	700	0	140	3.7	6.9	-6.01	3.5	6.7	14.34
	1.E-02	700	0	140	3.8	7.1	-5.94	5.5	8.6	14.45
120	1.E-05	600	0	120	5.2	6.0	-6.06	4.9	6.0	14.36
	1.E-04	600	0	120	4.2	6.0	-6.05	3.8	6.2	14.36
	1.E-03	600	0	120	3.0	6.5	-6.03	2.9	6.7	14.37
	1.E-02	600	0	120	2.7	6.2	-5.98	10.7	10.2	14.54
100	1.E-05	500	0	100	4.1	6.0	-6.06	4.0	6.0	14.35
	1.E-04	500	0	100	3.3	6.0	-6.05	3.2	6.1	14.36
	1.E-03	500	0	100	2.4	6.5	-6.04	2.7	6.8	14.36
	1.E-02	500	0	100	2.2	6.8	-6.01	4.1	8.8	14.43
80	1.E-05	400	0	80	3.5	6.0	-6.06	3.5	6.0	14.35
	1.E-04	400	0	80	2.7	6.1	-6.05	3.1	7.0	14.35
	1.E-03	400	0	80	2.2	6.4	-6.05	2.5	7.2	14.35
	1.E-02	400	0	80	3.0	8.0	-5.95	3.6	9.6	14.42
60	1.E-05	300	0	60	2.6	6.2	-6.06	2.7	6.7	14.35
	1.E-04	300	0	60	2.1	6.0	-6.06	2.1	6.8	14.35
	1.E-03	300	0	60	1.7	6.5	-6.05	1.8	7.0	14.35
	1.E-02	300	0	60	1.5	7.1	-5.99	4.7	10.1	14.55
40	1.E-05	200	0	40	1.6	6.1	-6.06	1.9	7.0	14.35
	1.E-04	200	0	40	1.6	6.8	-6.06	1.5	7.0	14.35
	1.E-03	200	0	40	1.4	7.2	-6.06	1.3	7.3	14.35
	1.E-02	200	0	40	1.5	7.9	-6.02	3.4	13.0	14.40
20	1.E-05	100	0	20	1.3	7.5	-6.06	1.2	7.8	14.35
	1.E-04	100	0	20	1.0	7.0	-6.06	0.9	7.0	14.35
	1.E-03	100	0	20	0.8	7.3	-6.06	1.1	9.5	14.35
	1.E-02	100	0	20	0.7	8.4	-6.03	1.5	12.2	14.37

Table D.6: SS optimization results. Average values of 10 runs.

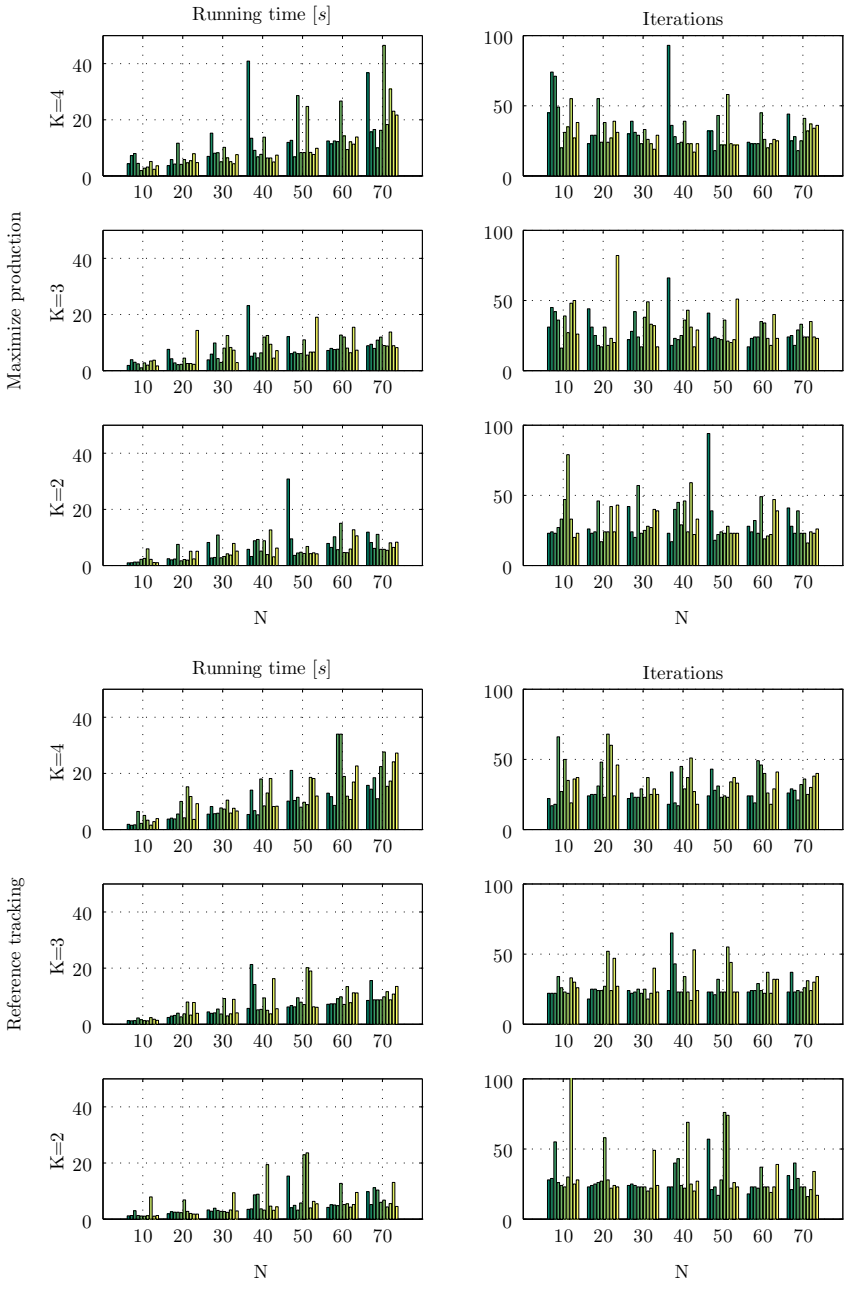


Figure D.6: All running times and iterations for the DC method.

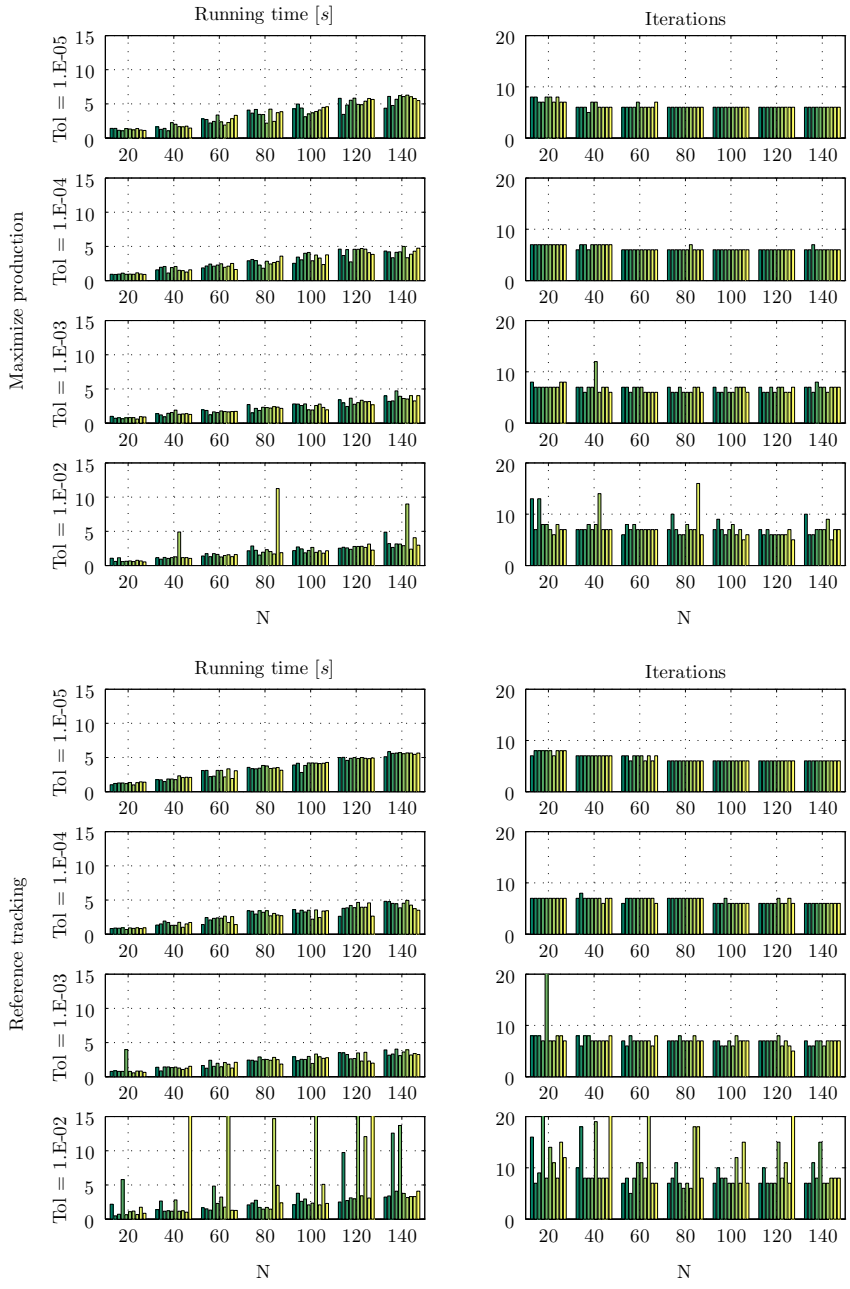


Figure D.7: All running times and iterations for the SS method.

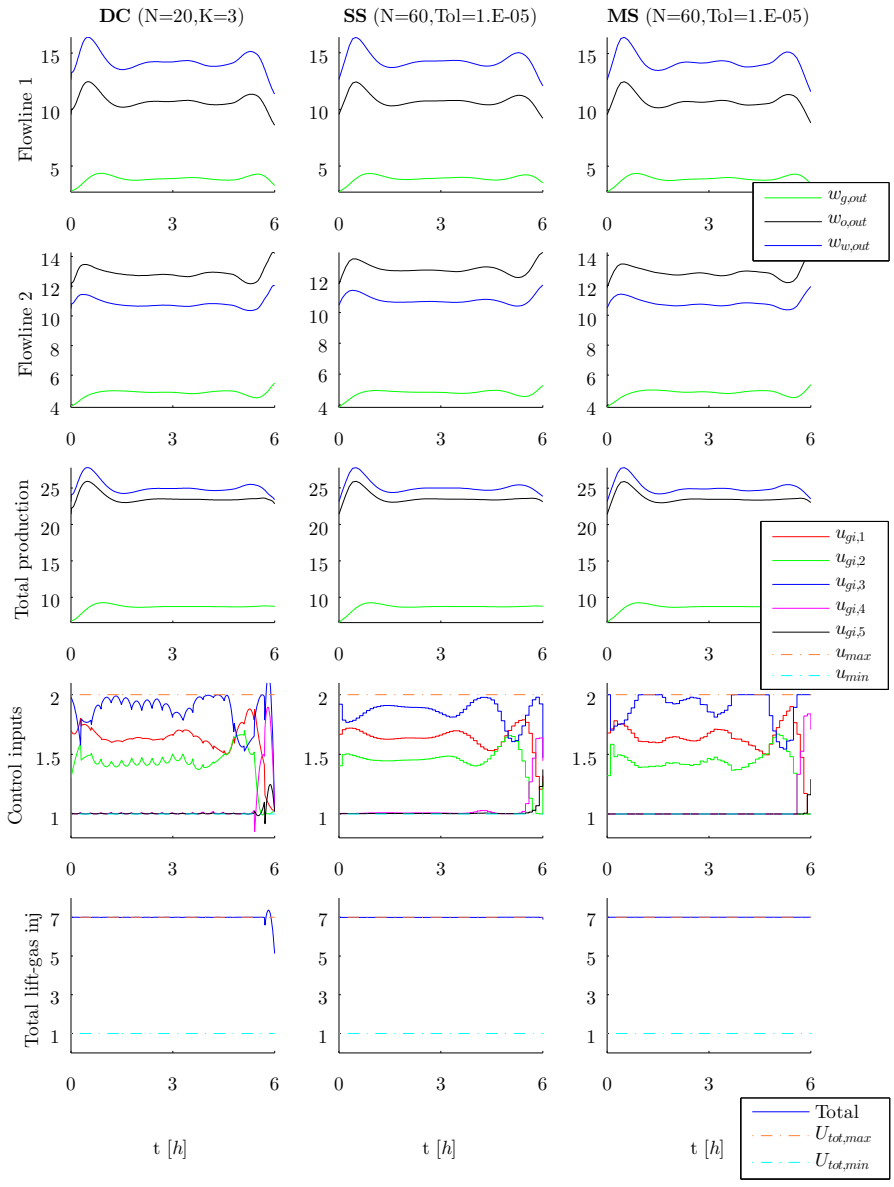


Figure D.8: Comparison of optimized control input profiles from DC, SS and MS with their corresponding simulated variable profiles. Here, the objective is to maximize production.

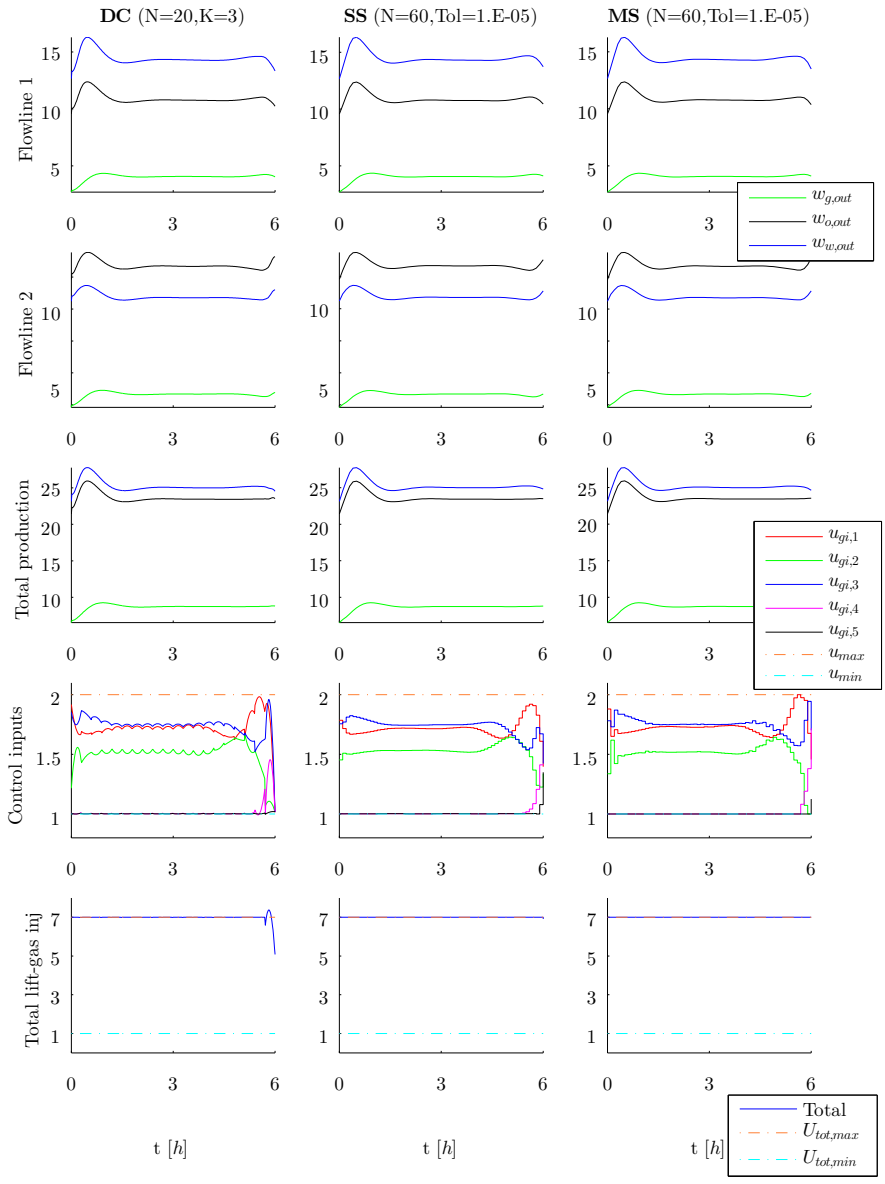


Figure D.9: Comparison of optimized control input profiles from DC, SS and MS with their corresponding simulated variable profiles. Here, the objective is reference tracking with $w^r_{o,out} = 30$.

D.5 Well-Flowline approximation

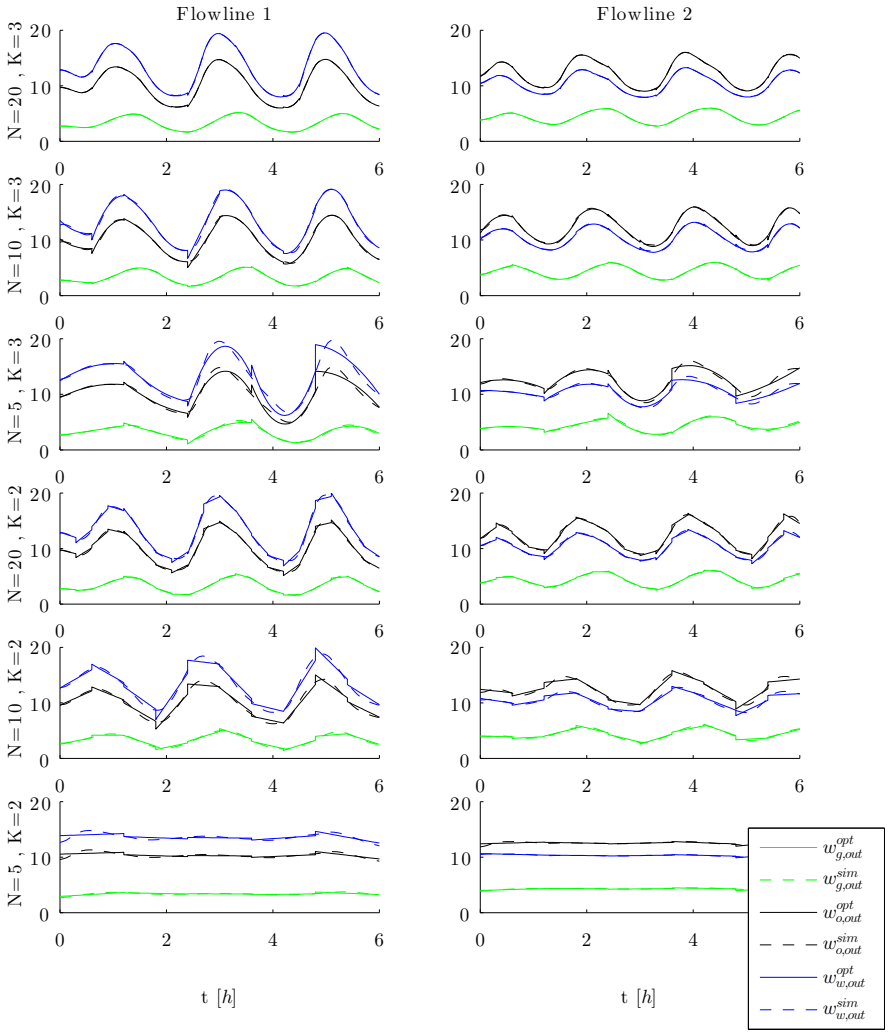


Figure D.10: Simulated profiles (dashed lines) compared to approximated profiles (solid lines) from the DC algorithm when varying N and K. Here, the objective is to maximize production.

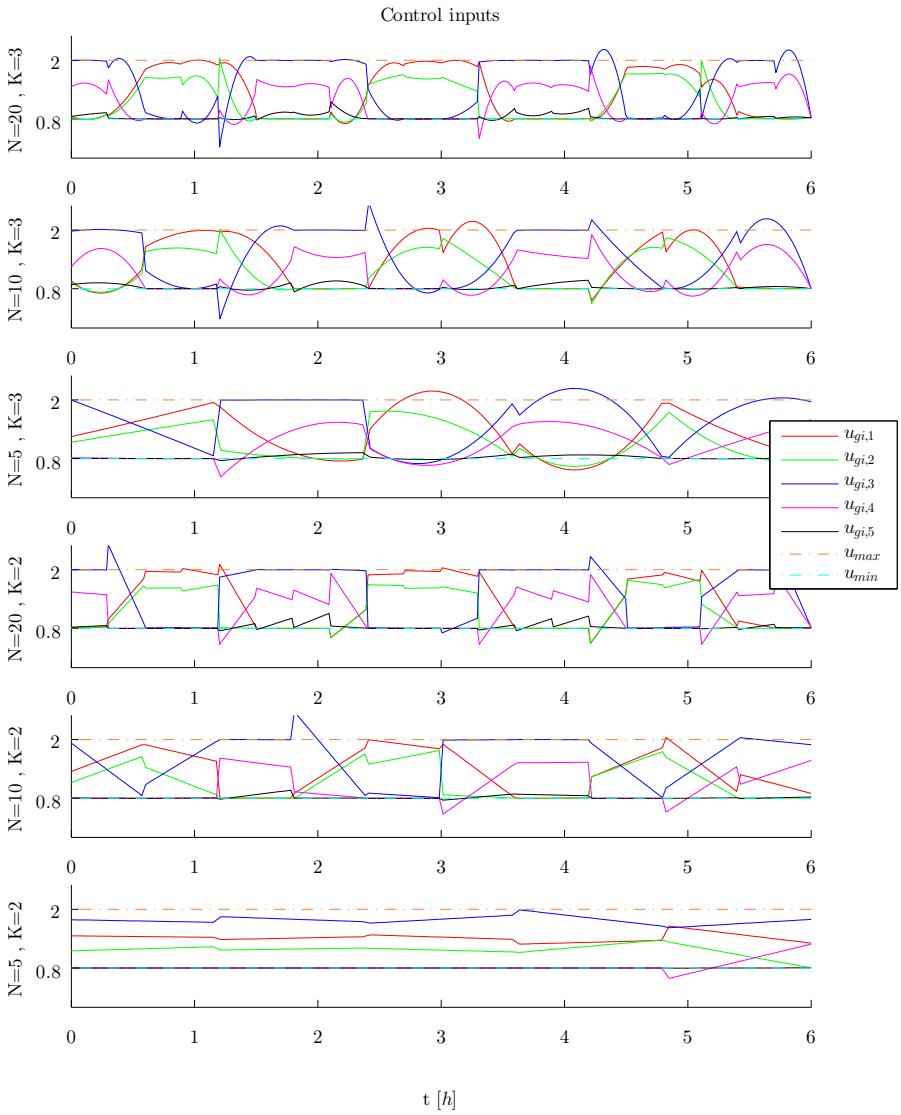


Figure D.11: Approximated profiles for control inputs from the DC algorithm when varying N and K, together with upper and lower bounds. Here, the objective is to maximize production. The following values are used for capacity and bounds: $U_{tot,min} = 1$, $U_{tot,max} = 6$, $u_{min} = 0.8$ and $u_{max} = 2$.

D.6 Production optimization

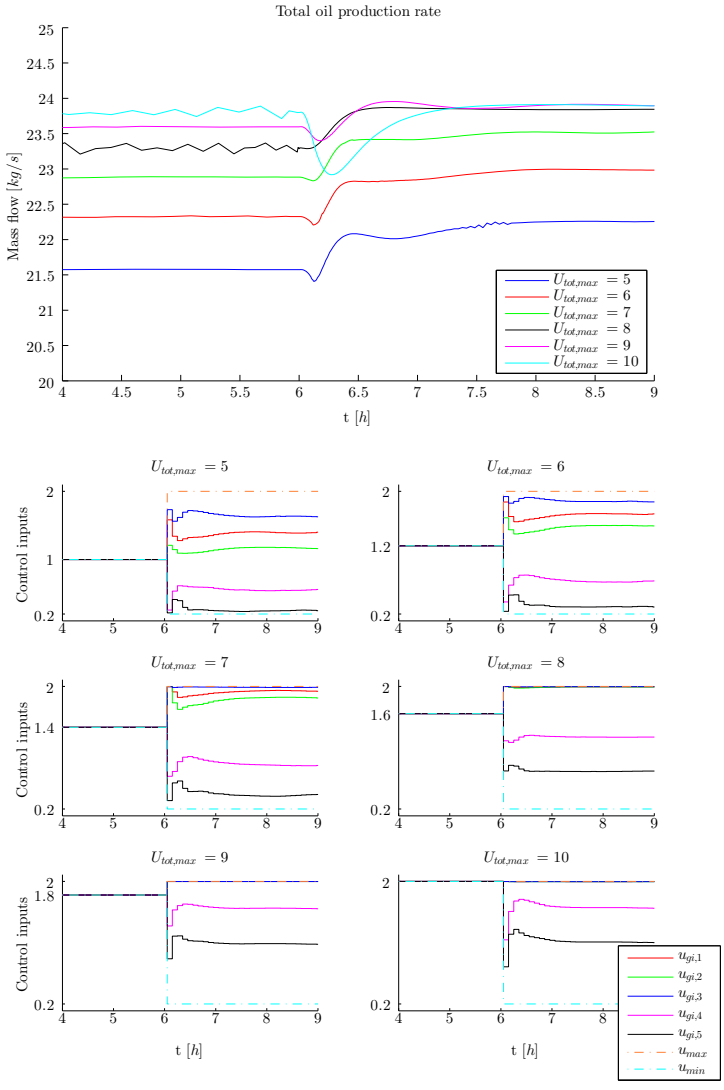


Figure D.12: Total oil production rates before and after applying optimized control inputs for different $U_{tot,max}$ that are kept constant throughout the time horizon. Corresponding control input profiles are given with their upper and lower bounds. Here, the objective is reference tracking with $w^r_{o,out} = 30$. SS (N=120, Tol=1.E-06).

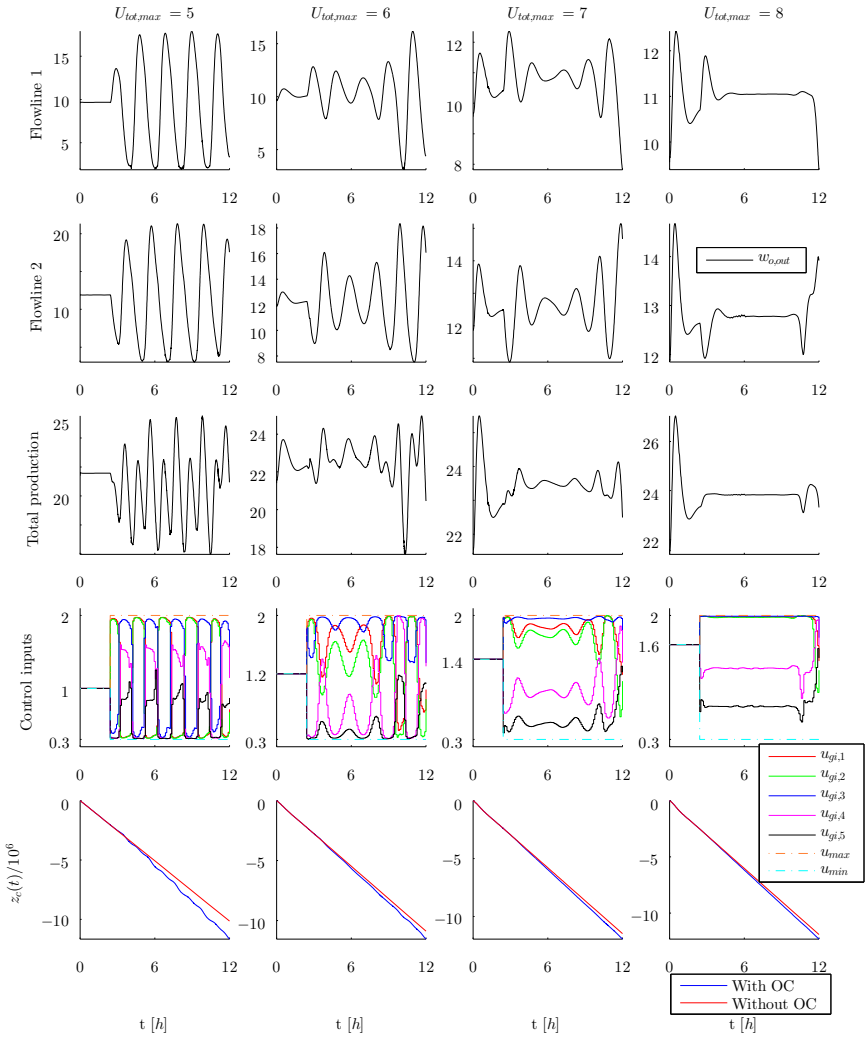


Figure D.13: Production rates, control inputs with bounds and cost state profile for various $U_{tot,max}$. Here the objective is to maximize production. With OC refers to using optimized control inputs. Without OC refers to using: $u_{gi,j} = \frac{U_{tot,max}}{5}, \forall j \in \mathcal{J}$ for $U_{tot,max} = \{5, 6, 7, 8\}$. SS (N=120, Tol=1.E-06).

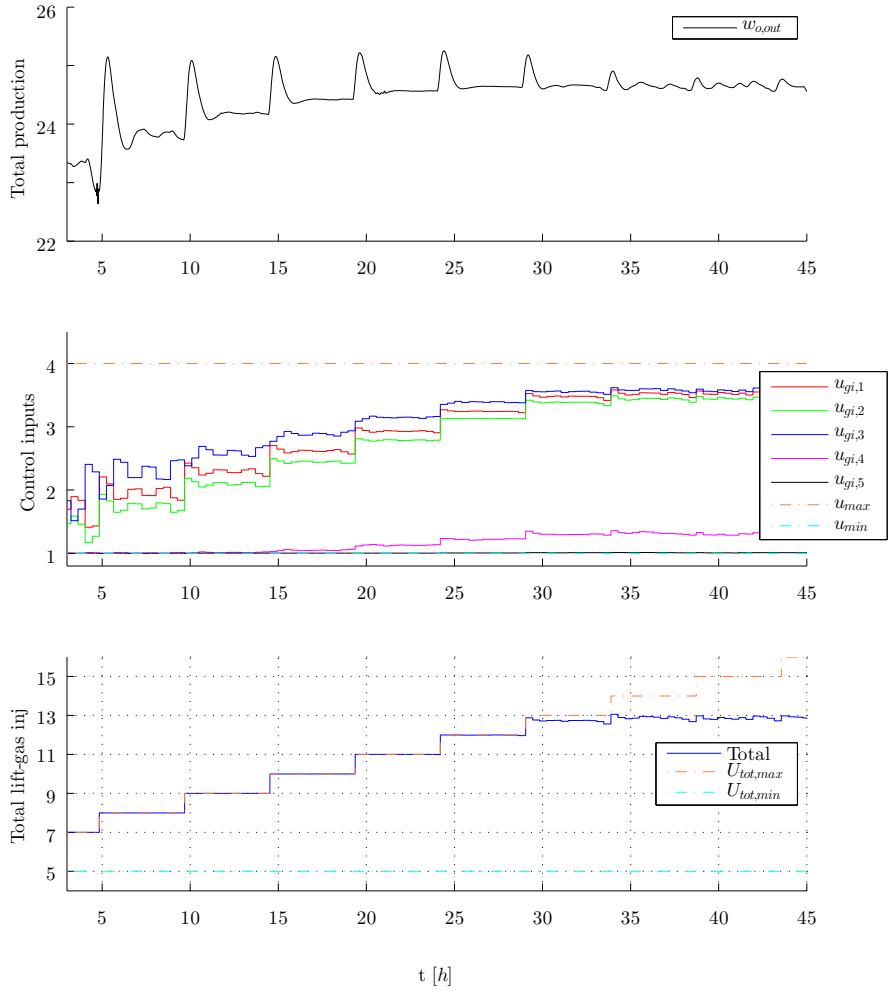


Figure D.14: Total production rates, control inputs and total lift-gas injection when increasing $U_{tot,max}$ from 7 to 16. Here, the objective is to maximize production without adding a cost of lift-gas usage, i.e. $r = 0$. SS (N=120, Tol=1.E-06).

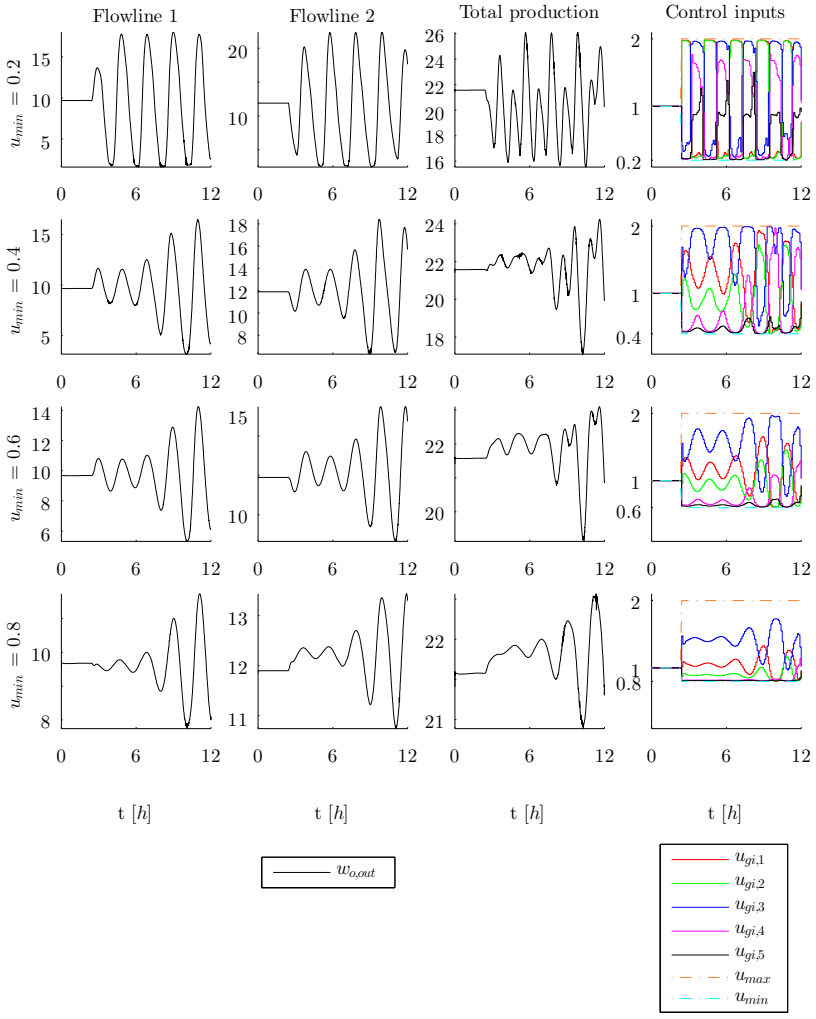


Figure D.15: Production rates and control inputs for various u_{min}. A constant capacity of $U_{tot,max} = 5$ is used. Here, the objective is to maximize production. SS (N=120, Tol=1.E-06).

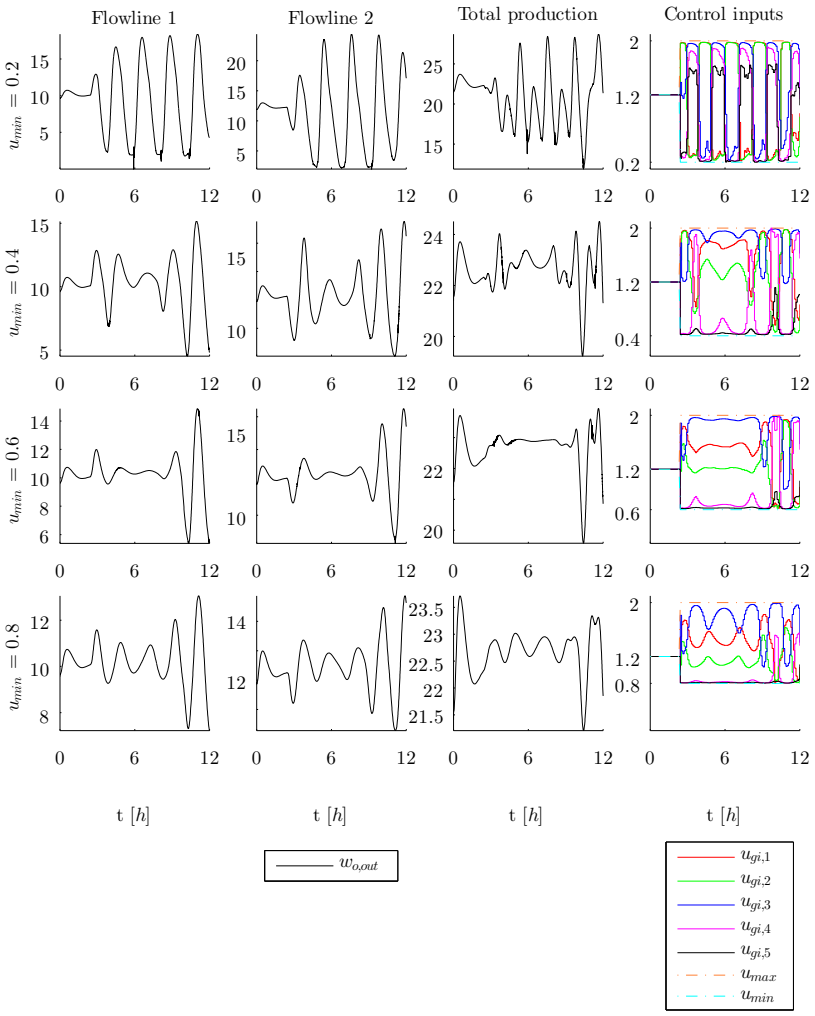


Figure D.16: Production rates and control inputs for various u_{min}. A constant capacity of $U_{tot,max} = 6$ is used. Here, the objective is to maximize production. SS (N=120, Tol=1.E-06).

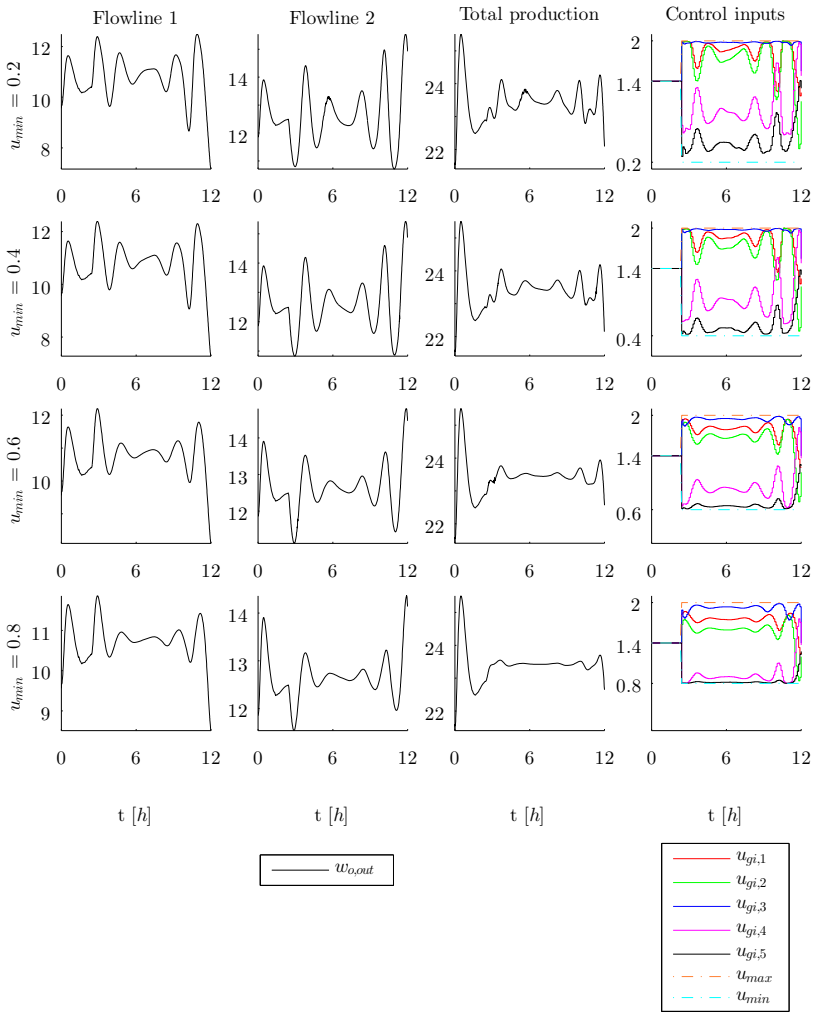


Figure D.17: Production rates and control inputs for various u_{min}. A constant capacity of $U_{tot,max} = 7$ is used. Here, the objective is to maximize production. SS (N=120, Tol=1.E-06).

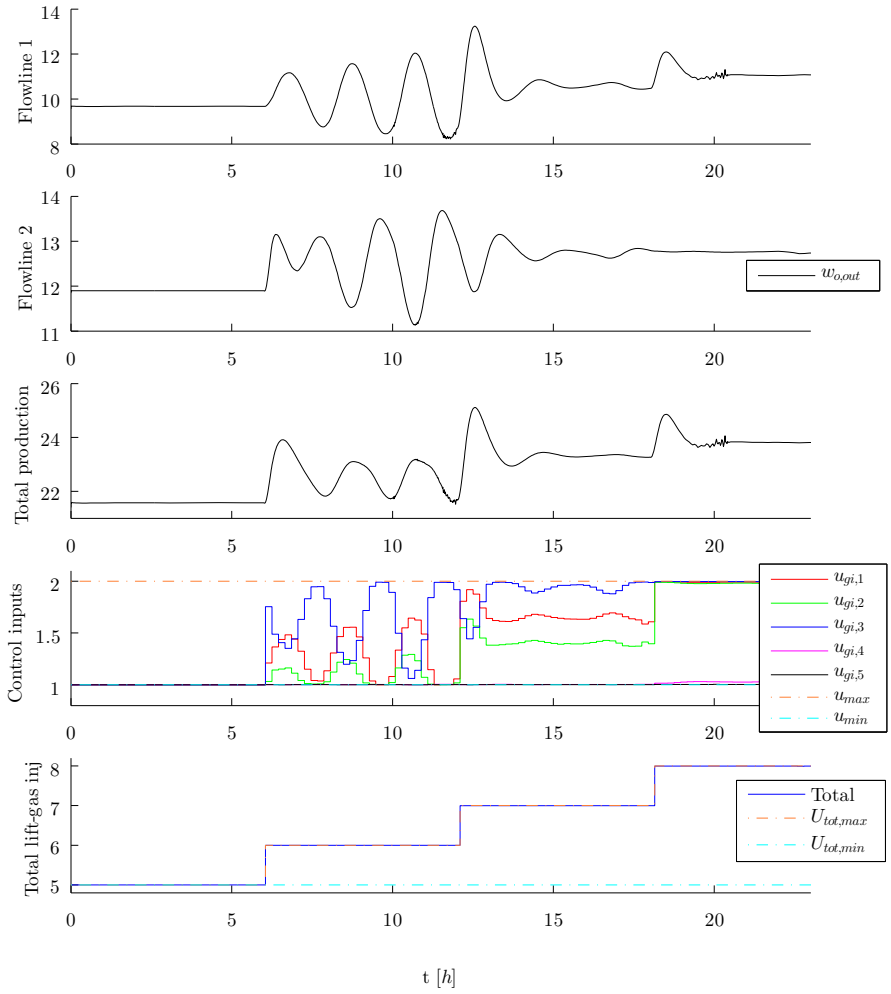


Figure D.18: Production rates and control inputs for a time-varying $U_{tot,max}$ with bounds: $u_{min} = 1$ and $u_{max} = 2$. Here, the objective is to maximize production. SS (N=120, Tol=1.E-06).

Bibliography

[1] JModelica.org User Guide Version 1.9.1. Modelon AB, 2013.

[2] Åkesson, J. *Languages and Tools for Optimization of Large-Scale Systems.* PhD thesis, Department of Automatic Control, Lund University, 2007.

[3] Åkesson, J., Årzén, K. E., Gäfvert, M., Bergdahl, T., and Tummescheit, H. Modeling and optimization with Optimica and JModelica.org - Languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering 34* (2010), 1737–1749.

[4] Åkesson, J., Ekman, T., and Hedin, G. Implementation of a Modelica compiler using JastAdd attribute grammars. *Science of Computer Programming 75* (2010), 21–38.

[5] Andersson, J., Åkesson, J., Casella, F., and Diehl, M. Integration of CasADi and JModelica.org. In *8th International Modelica Conference* (2011), Modelica Association, pp. 218–231.

[6] Andersson, J., Åkesson, J., and Diehl, M. CasADi: A Symbolic Package for Automatic Differentiation and Optimal Control. In *Recent Advances in Algorithmic Differentiation*, vol. 87 of *Lecture Notes in Computational Science and Engineering*. Springer Berlin Heidelberg, 2012, pp. 297–307.

[7] Andersson, J., Åkesson, J., and Diehl, M. Dynamic optimization with CasADi. In *51st IEEE Conference on Decision and Control* (2012), pp. 681–686.

[8] Andersson, J., Houska, B., and Diehl, M. Towards a Computer Algebra System with Automatic Differentiation for use with Object-Oriented modeling languages. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, October 2010, Oslo, Norway* (2010), pp. 99–105.

[9] Biegler, L. T. *Nonlinear Programming: Concepts, Algorithms and Applications to Chemical Processes*. MOS-SIAM Series on Optimization. Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.

[10] Bieker, H. P., Slupphaug, O., and Johansen, T. A. Real Time Production Optimization of Offshore Oil and Gas Production Systems: A Technology Survey. *SPE Intelligent Energy Conference and Exhibition* (2006).

[11] Binder, B. J. T. Production Optimization in a Cluster of Gas-Lift Wells. Master's thesis, Norwegian University of Science and Technology, 2012.

[12] Bischof, C., and Bücker, M. Computing Derivatives of Computer Programs. In *Modern Methods and Algorithms of Quantum Chemistry* (2000), John von Neumann Institute for Computing.

[13] Cervantes, A., and Biegler, L. T. Optimization Strategies for Dynamic Systems. Chemical Engineering Department, Carnegie Mellon University, 2000.

[14] Codas, A., and Camponogara, E. Mixed-integer linear optimization for optimal lift-gas allocation with well-separator routing. *European Journal of Operational Research 217* (2012), 222–231.

[15] Devold, H. *Oil and Gas Production Handbook*. ABB, 2006.

[16] Diehl, M., Andersson, J., Gillis, J., and Åkesson, J. CasADi: A Tool for Automatic Differentiation and Simulation-Based Nonlinear Programming, 2012.

[17] Eikrem, G. O., Aamo, O. M., and Foss, B. A. On Instability in Gas Lift Wells and Schemes for Stabilization by Automatic Control. *SPE Production & Operations 23* (2008), 268–279.

[18] Foss, B. Process control in conventional oil and gas fields - Challenges and opportunities. *Control Engineering Practice 20* (2012), 1058–1064.

[19] Foss, B., Gunnerud, V., and Díez, M. D. Lagrangian Decomposition of Oil-Production Optimization Applied to the Troll West Oil Rim. *SPE 14* (2009), 646–652.

[20] Gunnerud, V., and Foss, B. Oil production optimization - A piecewise linear model, solved with two decomposition strategies. *Computers & Chemical Engineering 34* (2010), 1803–1812.

[21] Gunnerud, V., Foss, B. A., McKinnon, K. I. M., and Nygreen, B. Oil production optimization solved by piecewise linearization in a Branch & Price framework. *Computers & Operations Research 39* (2012), 2469–2477.

[22] Imsland, L., Kittilsen, P., and Schei, T. S. Model-based optimizing control and estimation using Modelica models. *Modeling, Identification and Control: A Norwegian Research Bulletin 31* (2010), 107–121.

[23] Jahanshahi, E., and Skogestad, S. Simplified Dynamical Models for Control of Severe Slugging in Multiphase Risers. In *18th IFAC World Congress* (2011), pp. 1634–1639.

[24] Magnusson, F., and Åkesson, J. Collocation Methods for Optimization in a Modelica Environment. In *9th International Modelica Conference, Munich, Germany* (2012), pp. 649–658.

[25] Nalum, K. Model Predictive Control of Well-Pipeline Systems. Project report , Norwegian University of Science and Technology, 2012.

[26] Nocedal, J., and Wright, S. J. *Numerical Optimization*, 2 ed. Springer, 2006.

[27] Rantil, J., Åkesson, J., Führer, C., and Gäfvert, M. Multiple-Shooting Optimization using the JModelica.org Platform. In *Proceedings of the 7th Modelica Conference, Como, Italy* (2009), pp. 757–764.

[28] Serban, R., and Hindmarsh, A. C. CVODES: An ODE Solver with Sensitivity Analysis Capabilities. ACM Transactions on Mathematical Software, 2003.

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