Solving a Network Flow Decision Problem with Sampled Nonlinearities

Anders Thoresen Sandnes

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

Master's Thesis Assignment:

Solving a Network Flow Decision Problem with Sampled Nonlinearities

Department of Engineering Cybernetics
NTNU, Norway

January 30, 2013

Candidate:	Anders T. Sandnes
Supervisor:	Prof. Bjarne A. Foss
Co-supervisor:	Bjarne A. Grimstad
Start date:	14.01.2013
End date:	14.06.2013

1 Problem Description

As shown in [2], the network flow decision problem can be formulated as the following Mixed-Integer Nonlinear Programming Problem:

$$\begin{aligned}
\max_{x,y}.\quad & c^T x \\
\text{s.t.}\quad & A_{eq}[x\ y]^T = b_{eq} \\
& A[x\ y]^T \leq b \\
& h_i(x) = 0, \quad i = 1 \dots p \\
& x_{lb} \leq x \leq x_{ub} \\
& x \in \mathbb{R}^n, \quad y \in \{0,1\}^m
\end{aligned} \tag{P1}$$

where x is a vector of continuous variables and y is a vector of binary decision variables. Here, the structural constraints of the network, such as the mass balance equations, are given by a polyhedron (linear constraints). The nonlinear equality constraints, $h_i(x) = 0$, represent the simulators and nonlinear maps of individual components in the network. This can for instance be a pressure loss correlation over a pipeline or the nonlinear relation between GOR and flow rates of a well. Note that each $h_i(x)$ is considered to be an unknown and non-factorable function. Furthermore, to comply with all multi-phase simulators it is reasonable to assume that only the function value of $h_i(x)$ can be evaluated and that no derivative information is available.

When solving P1 each simulator must be called to evaluate the corresponding $h_i(x)$. In a local search it is sufficient to evaluate $h_i(x)$ at each iteration, but if a *deterministic* global search is attempted the full domain of the simulators must be sampled before optimizing. The rationale is that to perform decisions in a global sense the optimization algorithm must have at least one global bound on each $h_i(x)$.[1]

Problem P1 can be approximated by pre-sampling the nonlinear functions $h_i(x)$ and replacing them with calls to an interpolation procedure. The optimization algorithm can then work independently of the simulators – promoting speed at the cost of accuracy. If upper and lower bounds on the interpolated functions are known the approximated problem can be solved by a global optimizer. The resulting solution can then be used as the starting

[1]A *Stochastic* global optimization may not require bounds on the nonlinear functions, only their domains.

point in a local search that utilizes the simulators directly and maintains feasibility of P1. This is similar to the optimization procedure proposed in [1].

Previous studies [2, 3] show that linear interpolation is unsuited for optimization purposes due to its non-smooth nature. It is therefore of interest to look for other interpolation methods that may be suitable for an optimization approach like the one outlined above.

2 Assignments

The candidate is given the following assignments:

- Perform a brief literature study on global optimization of MINLP problems, convex and non-convex. The study should be focused on sub topics relevant for solving the network flow problem (e.g. Branch-and-Bound).
- Write a survey on interpolation methods for regular and irregular grids (scattered data). The survey should include only interpolation methods that produce continuous first-order derivatives. The closely related topic of efficient sampling techniques should be covered, albeit to a lesser extent. Discuss applicability of sampling and interpolation, as surrogates for nonfactorable (black-box) functions, in the context of mathematical programming. Utilization of interpolation in global optimization, where generation of upper and lower bounds are required, should be included in the discussion. Interpolation methods of interest are to be compared in terms of suitability in optimization and computational efficiency. The survey should produce an advisory summery of the selected interpolation methods.
- Implement and perform a computational study of the suggested algorithm, with the preferred interpolation method, on a network flow problem. The algorithm should be compared against a local optimization approach in terms of speed and robustness.

References

[1] Clifford A. Meyer, Christodoulos A. Floudas, and Arnold Neumaier. Global optimization with nonfactorable constraints. *Ind. Eng. Chem. Res.*, 2002.

[2] Anders Sandnes. Final Year Project Report: Formulation and Optimization of a Network Flow Problem with Sampled Nonlinearities. *NTNU*, 2012.

[3] Sheri S. Shamlou and Stine Ursin-Holm. Final Year Project Report: Comparative Study of Optimization Solution Techniques – Applied to Petroleum Production Problems. *NTNU*, 2012.

Abstract

The tensor product B-spline is applied in global solution of approximated mixed integer nonlinear programs, exploiting the special structure of the B-spline to create convex relaxations in a Branch-and-Bound framework. The main application considered is the short-term oil production optimization problem, were one of the fundamental challenges is to replace the nonlinear simulator models with more conveniently manipulated approximations. Representing the network structure analytically allows for a decoupling of large simulator models into smaller, manageable components that can be replaced individually. The suggested method is applied to a real production case and a classic academic problem. The results presented are of both theoretical and practical interest and indicate that the method has potential and is worthy of further investigation.

Sammendrag

Tensor produkt B-spline er anvendt i global løsning av tilnærmede ulineære heltallsprogrammer, hvor den spesielle strukturen i B-spline representasjonen er utyttet til å lage konvekse relakseringer til bruk i et Branch-and-Bound rammeverk. Hovedapplikasjonen som utforskes er optimering av oljeproduksjon over en kort tidshorisont. En av hovedutfordringene her er å erstatte simulatorene som er brukt til å modellere systemet med tilnærminger som er enklere å manipulere. Analytisk representasjon av nettverkstrukturen åpner får å dekoble store simulatormodeller i mindre, håndterbare komponenter som kan erstattes enkeltvis. Metoden er anvendt på et virkelig produksjonsproblem og på et klassisk akademisk problem. Resultatene er av både teoretisk og praktisk interesse, og antyder at metoden har potensial for videre utvikling.

Contents

Nomenclature

KKT	Karush-Kuhn-Tucker
LICQ	Linear Independence Constraint Qualification
LP	Linear Program
MILP	Mixed Integer Linear Program
MINLP	Mixed Integer Non-Linear Program
NLP	Non-Linear Program
QP	Quadratic Program
SQP	Sequential Quadratic Programming

List of Figures

List of Tables

List of Algorithms

Preface

This masters thesis is written under the Department of Engineering Cybernetics at The Norwegian University of Technology and Science. The topic of this thesis—numerical optimization of oil production networks—is one that I have been working with for almost a year now. The work has been done in cooperation with the Center for Integrated Operations in the Petroleum Industry and its partners. It started with a summer-student position, then continued in the final year project last semester and is now taken one step further with this thesis. In this past year I have been working closely with my co-supervisor Bjarne Grimstad and this work will be continued after this thesis and will hopefully lead to publishable material.

Acknowledgements

First I wish to thank my co-supervisor Bjarne Grimstad for great discussions, feedback and advice throughout the making of this thesis. I would also like to thank my supervisor Professor Bjarne Foss and Postdoctoral research fellow Vidar Gunnerud for their guidance in the past year; fellow master student Stine Ursin-Holm for helping me with the test case used in this thesis; fellow master student Sheri Shamlou for the cooperation during the BP-workshop; research partner Petrobras for providing data for the test case and research partner BP for sharing knowledge and insight from the real-life application of the topics investigated in this thesis. Finally I wish to thank my family for being awesome. Thank you.

Anders Thoresen Sandnes
Trondheim

Chapter 1

Introduction

As with everything else in life, oil production can be formulated as a really difficult mathematical optimization problem. In this thesis we take a look at one particular class of oil production problems, namely, the short-term production optimization problem. This chapter will introduce the topic of oil production optimization and go through the content of this thesis.

1.1 Background and Earlier Work

Optimization is performed at many levels in the oil production management hierarchy. The different levels consider different aspects of the production problem and the problems are defined over different time horizons [Foss, 2012]. The type of problem considered in this thesis is the optimization of flow through the production network over a short time horizon. Short time will typically be days or weeks, depending on the individual field.

Quickly summarized, oil is produced by drilling wells into a reservoir, connecting the wells to pipelines, potentially connecting the pipelines together to form a large pipe network, then have the pipelines rise up to the surface where the oil is received and processed before it is transported to its final destinations. Oil production optimization is an attempt to optimize the flow of oil from the wells, through the pipe network, and up to the topside facilities. In this thesis the discussion will assume that optimize is synonymous with maximization of oil per time, but this is by no means a requirement. It could also be to produce at a target rate while minimizing the usage of gas lift resources or the wear caused by sand particles.

A significant amount of research has been dedicated to oil production optimization over the past years. Several different approaches have been suggested and tested. This thesis is concerned with one particular family of methods known as *short-term production optimization*. These methods attempt to find the best production strategy by only looking at the system state as it is right now in a intermediate time scale, ignoring future changes and assuming steady state throughout the pipeline network. With this simplification there are two major degrees of freedom that can be optimized. The first is deciding on the routing configuration, which dictates how the flows can move through the network. The second is how much a individual well should produce. The production rate can be controlled by either choking the flow back by closing a valve or boosting the flow by injecting gas to raise the flow potential. The later being referred to as *gas lift*. The flow is pressure driven and the relationship between flow, pressure and pressure drop is modelled in multi-phase flow simulators. The input-output relationships of these simulators can be quite complex and it is difficult for a solver to work directly with a black-box function like this. A key step towards making the problem easier to solve is to break down the large black-box representing the entire network into smaller pieces that represent individual network components and then tie these together using explicit mass and momentum balance constraints. Examples of such formulations can be found in [Gunnerud et al., 2012] and [Kosmidis et al., 2005]. The next step is then to replace the now small simulator components by a approximating functions, that captures the essence of the simulator input-output relationship while allowing for more control in terms of extending the function domain and enforcing properties that are beneficial for a numerical solver. This is done by sampling the simulator, creating datasets covering the relevant variable values, and feeding the data to a interpolation or approximation scheme. The simulator is then replaced by the approximation in the actual problem formulation. The solution found by the solver is then taken back to the original simulators and adjusted to account for approximation errors. A optimization problem formulated this way is a *network flow decision problem with sampled nonlinearities*, which also happens to be the title of this thesis. A broad range of approximation methods has been tried and tested for solving this type of problem, some successful others less so, and there is still uncertainty related to how the simulators should be replaced. The replacement strategy will typically depend on the solution strategy (or the other way around). Different strategies lead to different types of solutions and there is a natural trade-off between solution time, solution quality and the solver complexity. Results from a selection of different algorithms can be found [Gunnerud & Foss, 2010] and [Shamlou & Ursin-Holm, 2012].

This thesis will attempt to shed some new light on how the simulators can be replaced with both local and global optimization in mind.

1.2 Problem Assignment and Interpretation

This section will go through the problem description and give my interpretation of the items involved. This will be the basis for the rest of the thesis.

The first problem assignment is:

> Perform a brief literature study on global optimization of MINLP problems, convex and non-convex. The study should be focused on sub topics relevant for solving the network flow problem (e.g. Branch-and-Bound).

This is a standard literature study assignment that will be covered in the theory sections of this thesis. The topics that will be given attention are the ones that will put the work done here into a broader theoretical context and aid the discussions later in the thesis. The write up on these topics will for the most part be rather short. The exception being convex relaxations, which will be introduced in some detail as it is quite central for the entire thesis.

The second problem assignment is:

> Write a survey on interpolation methods for regular and irregular grids (scattered data). The survey should include only interpolation methods that produce continuous first-order derivatives. The closely related topic of efficient sampling techniques should be covered, albeit to a

> lesser extent. Discuss applicability of sampling and interpolation, as surrogates for nonfactorable (black-box) functions, in the context of mathematical programming. Utilization of interpolation in global optimization, where generation of upper and lower bounds are required, should be included in the discussion. Interpolation methods of interest are to be compared in terms of suitability in optimization and computational efficiency. The survey should produce an advisory summery of the selected interpolation methods.

The motivation behind this assignment is that the problems of interest are described by black-box simulators that act as oracles, meaning that we can only query for function values at points. This query procedure can be both time consuming and unpredictable. Time consuming because it is a complex simulator that must be invoked. Unpredictable because the simulators are only calibrated for a limited range of input values and the numerical solvers may want to venture outside these regions while iterating. It has therefore been established that it is desirable to replace the simulators by approximations during the actual optimization steps. Various methods have been applied here, but they tend fall in one of two categories: easy to implement and difficult to solve, or tedious to implement and easy to solve. The ultimate goal is thus to find a method that, hopefully, gets the best of both worlds.

The assignment calls for a survey on interpolation methods. Seeing that interpolation is an amazingly diverse topic, it would not be possible to cover all individual methods in detail. In this thesis I will attempt to address this assignment by first introducing the general function approximation problem and point out the main challenges. Then proceed with a look at the most common assumptions that can be made and introduce the families of techniques that these assumptions lead to. Focus will be on two extremes on the assumption scale, namely scattered and grid structured data sets. Examples will be provided of the most relevant techniques to illustrate their properties. The discussion will be kept in the light of oil production optimization.

The assignment also calls for a discussion on sampling techniques. Two questions are relevant here. The first is *how* the data samples are structured and this is covered when discussing the various methods. The second question is *when* the samples are generated. This is basically about whether all the samples are generated upfront, before the optimization begins, or during the optimization, updating the approximation locally as the solver iterates. This will be discussed briefly in the context of local and global optimization.

The third problem assignment is:

> Implement and perform a computational study of the suggested algorithm, with the preferred interpolation method, on a network flow problem. The algorithm should be compared against a local optimization approach in terms of speed and robustness.

This part of the thesis will test the most promising method form the previous assignment and discuss it in terms of local and global optimization.

1.3 Structure of the Thesis

This thesis has 10 chapters, starting of with this introductory chapter which covers earlier work, background and motivation. It then continuous with some theoretical chapters, dealing with numerical optimization in Chapter 2, general function approximation in Chapter 3, specifics on the suggested interpolation method in Chapter 4, and the network flow problem in Chapter 5. The branch and bound solver developed and used in this thesis it then described in Chapter 6 and this serves as the method chapter of the thesis. This is followed by two chapters presenting the results; with Chapter 7 covering the function approximation results and Chapter 8 covering the optimization results. Finally, a discussion is given in Chapter 9 and a conclusion in Chapter 10.

Chapter 2

Numerical Optimization

Numerical optimization is the art of solving a certain type of mathematical programming problem using a computer. This is different from solving them by hand, because a computer has a very limited set of mathematical operations at its disposal. One of the few things it can do is to solve a set of linear equations $Ax = b$. A numerical solver will typically generate and solve such equation sets until enough information has been gathered to draw a conclusion. The time it takes for a computer to solve a problem depends, roughly speaking, on two things: the size of the problems and the number of problems. The ultimate goal obviously being to strike the perfect balance between the complexity of each problem and the number of problems. Some solvers aim to take few, but significant steps, while others intend to use many cheap steps instead. The best strategy is problem dependent. This chapter will go through some of the basics of numerical optimization that will be useful for the discussions later. Most of the information given here can be found in both *Numerical Optimization* by Nocedal and Wright[Nocedal & Wright, 2006] and in *Convex Optimization* by Boyd and Vandenberghe [Boyd & Vandenberghe, 2004], if not, other sources will be provided.

2.1 Mathematical Formulation and Notation

In this thesis optimization problems will be stated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & c_i(x) \leq 0, i \in I \\ & c_i(x) = 0, i \in E \\ & x_i \in \mathbf{Z}, \; i \in Z \\ & x \in X, \end{aligned} \tag{2.1}$$

with I, E and Z being sets of indices. In this formulation X is a vector space where the optimization problem is defined and x is the *optimization variable*. In this thesis X will always be a subset of $\mathbf{R}^n$. The goal is to minimize the *objective function* $f : X \mapsto \mathbf{R}$ while satisfying the *constraint functions* $c_i : X \mapsto \mathbf{R}$ and the *integer constraints* $x_i \in \mathbf{Z}$. A vector $x \in X$ is called *feasible* if it satisfies all the constraints. The set of feasible points is called the *feasible region* and is defined as

$$X_F = \{x \,|\, x \in X \wedge c_i(x) \leq 0, i \in I \wedge c_i(x) = 0, i \in E \wedge x_i \in \mathbf{Z}, \; i \in Z\}. \tag{2.2}$$

A vector x is called *infeasible* if $x \notin X_F$. A vector $x^* \in X$ is called *globally optimal* if $f(x^*) \leq f(x)$ for all $x \in X_F$ and *locally optimal* if $f(x^*) \leq f(x)$ for all $x \in X_N \bigcap X_F$ where X_N is a neighbourhood of x^*. For the problem to be well defined the search space X must be a subset of the domain of all functions involved in the constraints and the objective, that is

$$X \subseteq \mathbf{dom}\, f, \quad X \subseteq \mathbf{dom}\, c_i, i \in E \cap I. \tag{2.3}$$

and will in all cases relevant cases be given as *box constraints* on the variables:

$$X = \left[\underline{x}_1, \overline{x}_1\right] \times \left[\underline{x}_2, \overline{x}_2\right] \times \cdots \times \left[\underline{x}_n, \overline{x}_n\right], \tag{2.4}$$

where $\underline{x}_i$ is the lower bound and $\overline{x}_i$ the upper bound describing the valid interval for variable x_i. This is often stated as $\underline{x} \leq x \leq \overline{x}$, with the inequalities interpreted elementwise.

The problem in 2.1 is called a *Mixed Integer Non-Linear Program* (MINLP). In its most general form it is a non-convex optimization problem. This typically means it is hard to solve. A interesting and useful class of problems are the convex optimization problem. Problem 2.1 is convex if the objective and all the inequality constraints are convex, the equality constraints are linear (affine) and no variables are integer constrained. In other words;

$$\nabla^2 f i(x) \geq 0, \tag{2.5}$$
$$\nabla^2 c_i(x) \geq 0, i \in I, \tag{2.6}$$
$$\nabla^2 c_i(x) = 0, i \in E, \tag{2.7}$$
$$Z = \emptyset. \tag{2.8}$$

Informally speaking convex optimization is the optimization of a convex objective function over a convex feasible region. Convex problems are nice, because they can be solved fast and reliably. They are also nice because their solutions arrive accompanied by strong statements, such as, if the solution is locally optimal it is globally optimal also. The same goes for infeasibility, if a problem appears to be locally infeasible, it is actually infeasible and the problem must either be discarded or reformulated. These features makes the convex problems very attractive to solve. Two special cases that are worth mentioning are the convex *Quadratic Programs* (QP) and the *Linear Programs* (LP). A convex QP is a problem where all constraints are linear functions and the objective is a convex quadratic function. If the objective function is linear as well the problem is an LP. The rest of this chapter will assume the objective function is convex. This can be done without loss of generality, because a non-convex objective can always be changed into a linear objective by introducing a new variable t and a constraint $c(x) = f(x) - t \leq 0$ and minimizing t. This is known as the *epigraph form* of a optimization problem.

There are two ways a optimization problem above could be presented to a solver. One is the *parameter* description, which would give explicit descriptions of all functions involved through a predefined parameter list (*e.g.* a matrix). The other is the *oracle* , or *black-box*, description, which only provides the functions as subroutines that can be evaluated accompanied by some additional information such as the domain and possibly properties like convexity. A parameter description can obviously be changed into a oracle description, but a oracle cannot necessaries be represented parametrically. In this thesis a central problem is to change a oracle description into a parameter description by approximation of the oracle.

2.2 Local Solution

Optimality Conditions for a Local Minimum

The easiest way to define optimality conditions is to assume the functions involved are twice differentiable and that first and second derivative information is available (or estimated). This can be used to establish local minimums through basic calculus techniques (*e.g.* vanishing gradient for stationary point, positive curvature for a minimum) and solvers can be made to search for points with these properties. The following will give a brief summary of the conditions used to determine if a point is a local minimum. The first step towards these conditions is to introduce a vector $\lambda = [\lambda_1, \dots, \lambda_n]$ (n equal to the number of constraints) and define the Lagrangian function:

$$L(x, \lambda) = f(x) + \sum_{i \in I \cup E} \lambda_i c_i(x). \tag{2.9}$$

The Lagrangian function is simply a way to incorporate the constraints into the calculus arguments on vanishing gradients and curvature. It can then be shown that a point x^* is locally optimal if there exist a λ^* such that

$$\begin{aligned}
c_i(x^*) &\leq 0 \qquad i \in I && (2.10)\\
c_i(x^*) &= 0 \qquad i \in E && (2.11)\\
\lambda^* &\geq 0 \qquad i \in I && (2.12)\\
\lambda^* c_i(x^*) &= 0 \qquad i \in I && (2.13)\\
\nabla_x L(x^*, \lambda^*) &= 0. && (2.14)
\end{aligned}$$

and

$$w^T \nabla^2_{xx} L\left(x^*, \lambda^*\right) w > 0, w \in X_F \cap X_N. \tag{2.15}$$

It is also necessary to make some additional assumptions on the constraints to make sure the equations actually have a well defined solution. These are known as *constraint qualifications* and the most famous here is the *Linear Independence Constraint Qualification* (LICQ) that requires the linearised active constraints to be linearly independent.

Equation 2.10-2.14 are the *first-order necessary conditions* and are often called the *Karush-Kuhn-Tucker (KKT) conditions.* The first two are just the constraints and only say that a optimal solution must be a feasible point. The third condition says the Lagrangian multipliers for the inequality constraints must be positive. Equality constraints do not have this requirement since they could easily be multiplied by -1 without changing the feasible regions and thus gradient direction will not play the same role in the final condition. The fourth condition says that only the inequality constraints that are at their boundary can be allowed to play a role in the final condition. With the third and fourth condition in place the final condition says: there is no direction we can move that would reduce the objective function value without also *immediately* taking us outside the feasible region. Equation 2.15 is the *Second-Order Sufficient Condition.* It ensures that the curvature is positive in all feasible directions.

If no assumption is made on differentiability the KKT conditions cannot be applied and it becomes harder to verify a possible solution. The number of reliable solvers drops significantly when the problems are not differentiable and they are typically sub-gradient schemes with slow convergence properties. These solvers are avoided if possible. As an example, last years project attempted to use an interior point solver, which assumes twice differentiable functions, to solve problems with piecewise linear functions. This did not turn out so well and the results were unpredictable. Instead of keeping the piecewise linear problem and change the solver to a sub-gradient method, we chose to change the problem and keep the solver. Non-differentiable problems will therefore not be discussed further in this thesis.

Local Solvers

Local solvers attempt to find a point where the KKT conditions are satisfied. The difficulty of this task will naturally depend on how many of the KKT-conditions that must be considered. The easiest scenario is when there are no constraints. A classic example of unconstrained optimization problem is the minimization of a sum of squared error terms. This is known as the least squares problem

$$\text{minimize} \quad \| Ax - b \|_2 \tag{2.16}$$

and its solution can be found by taking the pseudo-inverse of A and computing x as $x = A^{\dagger}b$. This is about as easy as it gets in the numerical optimization world, since its solution can be found after a single set of linear equations. The next step is to extend this procedure to account for constraints. The easiest constraints to deal with would be linear equality constraints, because they can be removed by a change of variables and running the unconstrained procedure in the null space of the constraints. Adding linear inequalities is significantly worse, since they cannot be removed by elimination and as stated in the KKT conditions, only *active* inequalities (the inequality is equal to zero) can be used in the optimality argument. This causes problems and there are various ways to deal with it. One solution is to simply guess which inequalities that would hold and treat these as if they were equalities while ignoring the rest. The modified problem is then dealt with as equality constrained QP problems. Iterations are preformed to find the correct active set. This is known as *active set methods* and they are very popular for QP problems. If the objective and constraints are non-linear it gets even worse and a classic approach is to introduce yet another layer of iterations by first creating approximations of the constraints and objective using derivative information and solve these approximated problems to create the steps. This leads to methods such as *Sequential Quadratic Programming* (SQP). A slightly different angle of attack is to work on the KKT equations directly. This is done by introducing slack variables on the inequalities and weight the slack in the objective functions. The modified system is then solved as if it was a large unconstrained or equality constrained problem by minimizing the error in the KKT system. These methods are know as *Interior Point Methods.* It is such a method that is used to solve problems in this thesis.

2.3 Global Solution

This section will go through some central aspects of global optimization, but first introduce some useful lower bound problems, namely, the *Dual Problem* and the *Relaxed Problem.*

The Dual Problem

A special lower bound problem is produced by taking the infimum over x of the Lagrangian function,

$$g(\lambda) = \inf_x L(x,\lambda) = \inf_x \left(f(x) + \sum_{i \in I \cup E} \lambda_i c_i(x) \right), \tag{2.17}$$

in combination with the third component of the KKT conditions

$$\lambda \geq 0,\, i \in I. \tag{2.18}$$

For any fixed λ this infimum will produce a value that is less than or equal to that the optimal solution of the original problem. This can be seen by realizing that a valid point for the infimum is at the optimal solution x^* and since this point has to be feasible we get

$$\sum_{i \in I \cup E} \lambda_i c_i(x) \leq 0 \tag{2.19}$$

because of equation 2.18 and the way the constraints are defined. This gives g a value equal to the optimal solution pluss a negative number. If the infimum occurs anywhere else than at the optimal solution, the result is simply even lower than this option. Thus for any valid λ the expression in 2.17 will produce a lower bound on the global minimum. Many of these lower bounds will however simply end up being $-\infty$, so it would be desirable to know the largest possible lower bound that can be produced this way. This is formulated as a new optimization problem

$$\begin{aligned} \text{minimize} \quad & -g(\lambda) \\ \text{subject to} \quad & \lambda \geq 0, i \in I. \end{aligned} \tag{2.20}$$

Finally, the feasible region is restricted to points where g take on a finite value. This is done by introducing the set

$$F = \{\lambda | g(\lambda) > -\infty\} \tag{2.21}$$

and augmenting the problem to become

$$\begin{aligned} \text{minimize} \quad & -g(\lambda) \\ \text{subject to} \quad & \lambda \geq 0, i \in I \\ & \lambda \in F. \end{aligned} \tag{2.22}$$

This is know as the *dual problem* and has many useful properties that can be exploited by the numerical solver. When discussing duality the original problem is referred to as the *primal problem.* The difference between the primal and dual solution is called the duality gap. If the problem is convex (and some constraint qualifier holds, for instance Slaters condition) the duality gap is zero. This means the solution of the primal and dual are the same. For LP problems the dual problem can be found analytically and it is also an LP. It is often beneficial to solve the dual instead of the primal, because of the hot-start possibilities of the dual problem. Analytical manipulation of the dual is however limited to problems using a parameter representation. For oracle representations the Primal-Dual interior point method is an example of an algorithm that incorporates duality in the solution procedure.

The Relaxed Problem

For non-convex problems the dual problem is not sufficient. It is not computationally efficient and the duality gap is non-zero, so other methods must be used to produce lower bounds. This is done by creating a relaxation of the original problem. There are a myriad of ways to produce such relaxed problems and the only rule that really applies is that the solution to the relaxed problem must be a lower bound on the solution to the original problem. It would also be desirable that the relaxed problem is significantly easier to solve than the original problem, which typically means it has to be convex. It is therefore often called a *convex relaxation*.

An intuitive way to create a relaxed problem is to go through the constraints one by one and deal with the non-convex inequalities and the non-linear equalities individually. The most trivial way to deal with a constraint is to simply remove it. This is usually not that useful, but for the integer restrictions $x_i \in \mathbf{Z}$ this is actually the best way to create a relaxation. However, in most continuous cases this will not suffice and the constraint is either replaced by a new set of constraints or modified so it becomes convex. There are multiple ways to formulate these relaxations. Here some of the general relaxation strategies will be explained using the constraint

$$c(x) = a_0 + a_1 x + \cdots + a x_i x_j + \cdots + f(x) = 0 \tag{2.23}$$

as an example. This constraint is made up of a series of terms of varying complexity. The constant term a_0 and the linear term $a_1 x$ are trivial since they don't contribute to non-convexity and can be ignored. The bilinear term $x_i x_j$ and the generic term $f(x)$ however will have to be relaxed. This can be done by for instance introducing new variables to replace the non-linear terms:

$$c(x) = a_0 + a_1 x + \cdots + a z_1 + \cdots + z_2 = 0 \tag{2.24}$$

and then create new convex constraints to place restrictions on these new variables. There are various ways to produce these new constraints. The ideal situation is if the relaxed term has a special structure for which an explicit convex relaxation is known. The bilinear term is an example of such a term and the relaxation is given by four linear constraints that can be proven to be the tightest possible relaxation (it is equivalent to the convex hull of the points on the surface). They are given by:

$$z_1 \geq \underline{x}_i x_j + \underline{x}_j x_i - \underline{x}_i \underline{x}_j \tag{2.25}$$
$$z_1 \geq \overline{x}_i x_j + \overline{x}_j x_i - \overline{x}_i \overline{x}_j \tag{2.26}$$
$$z_1 \leq \overline{x}_i x_j + \underline{x}_j x_i - \overline{x}_i \underline{x}_j \tag{2.27}$$
$$z_1 \leq \underline{x}_i x_j + \overline{x}_j x_i - \underline{x}_i \overline{x}_j. \tag{2.28}$$

where $\underline{x}_i$ and $\overline{x}_i$ are the domain bounds [Androulakis et al., 1995]. This relaxation is illustrated in figure 2.1. Similar replacements are known for a variety of fractional and monomial terms. The number of new constraints will naturally increase with the complexity of the replaced term. For instance, a trilinear term $x_i x_j x_k$ can be replaced by a new variable with eight linear constraint associated with it.

If the replaced term has a general structure without any redeeming properties the typical relaxation strategy is to augment it by overpowering terms. In the example constraint this is the $f(x)$ term. The relaxation is done by first pretending the new variable is simply equal to the replaced term

$$z_2 = f(x), \tag{2.29}$$

then split this into the equivalent inequalities

$$z_2 \geq f(x) \tag{2.30}$$
$$z_2 \leq f(x) \tag{2.31}$$

and augment these inequalities with new terms

$$z_2 \geq f(x) + g_1(x) \tag{2.32}$$
$$z_2 \leq f(x) - g_2(x). \tag{2.33}$$

The augmentation terms are always positive inside the variable bounds, which means they can only increase the feasible region when applied this way. These constraints will be convex if $f(x) + g_1(x)$ is concave and $f(x) - g_2(x)$ is convex. This is happens when

$$\nabla^2 \left(-f(x) - g_1(x)\right) \geq 0 \tag{2.34}$$
$$\nabla^2 \left(f(x) - g_2(x)\right) \geq 0. \tag{2.35}$$

Quadratic terms are often suggested as augmentations because of their simplicity during the derivation, since it basically boils down to adding large values along the diagonal of the Hessian matrix. In this case the augmentation term is given by

$$g(x) = \sum_{j=1}^{n} \alpha_j (x_j - \underline{x}_j)(\overline{x}_j - x_j) \tag{2.36}$$

and each term in this sum is a quadratic that crosses zero at the upper and lower bound for the corresponding variable. This approach is taken in the αBB algorithm, which is thoroughly described in [Adjiman et al., 2000]. The drawback with this method is the fact that the relaxation can potentially be quite loose. Another problem is to find suitable values for α_i, which should be as small as possible, ideally equal to half of the most negative eigenvalue of the Hessian. This value can however be difficult to find (since the cases where an analytical expression for this value is know, it is likely that a good relaxation is know as well) and must often be approximated, which can be quite computationally demanding [Adjiman et al., 2000]. An illustration of the difference between the quadratic augmentation strategy and the exact replacement for the bilinear term is given in figure 2.1.

When all the constraints have been dealt with using these methods the new problem will be a convex relaxation that can be used to produce a lower bound on the original problem.

Optimality Conditions for a Global Minimum

Local solutions could be identified by local information and there was a well defined set of conditions that had to be satisfied. Finding a global minimum requires a global perspective, but there is no global equivalent to the KKT-conditions from section 2.2. Finding the global solution is therefore done by finding a local solution and then proving that even if there are other local solutions to the problem, non of them can have a objective value lower than the one suggested. One way to do this is to take the original optimization space X and divide it into n smaller sub-spaces X_i so

$$X_i \subseteq X,\ i = 1, \ldots, n \tag{2.37}$$

and

$$X = \bigcup_{i=1}^{n} X_i. \tag{2.38}$$

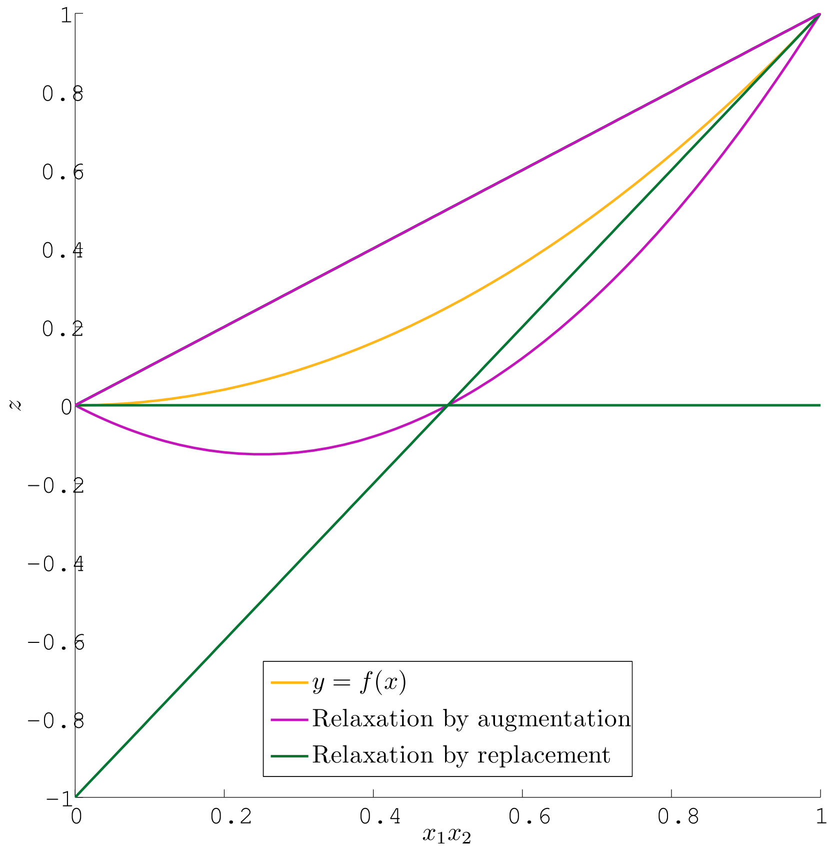

Figure 2.1: Cross section of the bilinear function $y = x_1x_2 = f(x)$ at $x_1 = x_2$ defined over $[0,1] \times [0,1]$, together with the nonlinear αBB relaxation from equations 2.32-2.33 and the linear exact relaxation from equations 2.25-2.28. All three upper bounds intersect at $x_1 = x_2$ and are therefore displayed on top of each other (in the same way they all cross each other at $x_1x_2 = 0.5$, just viewed from a different angle).

The original problem is now restricted to these sub-spaces by rewriting problem 2.1 to

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & c_i(x) \le 0, i \in I \\ & c_i(x) = 0, i \in E \\ & x_i \in \mathbf{Z}, i \in Z \\ & x \in X_i \end{aligned} \tag{2.39}$$

and have corresponding convex lower bound problems created with the same domain reductions, using for instance the relaxation techniques mentioned in the previous section. These local problems are solved to get a local solution and a lower bound inside each sub-space. We define f_i to be the objective function value returned from solving sub-problem i to a local solution $x_i^* \in X_i$. If no feasible solution is found, $f_i = \infty$. This does not imply that the problem is actually infeasible inside X_i, only that the solver converged to a point of local infeasibility and stopped there. The solution to a lower bound sub-problem is denoted $\bar{f}_i$ and if no feasible solution is found to the lower bound problem we have $\bar{f}_i = \infty$. If $\bar{f}_i = \infty$, then there is no feasible point inside X_i for the original problem. A global solution $x^* = x_i^*$ can now be identified by using the lowest lower bound as a global lower bound,

$$\bar{f} = \min\{\bar{f}_1, \ldots, \bar{f}_n\}, \tag{2.40}$$

and the best local solution as the suggested global solution,

$$f^* = \min\{f_1^*, \ldots, f_n^*\}, \tag{2.41}$$

and having

$$f^* \le \bar{f}. \tag{2.42}$$

For this to be possible, it must be required that the lower bound problems becomes tighter and closer to the original problem when the size of the solution space is reduced. In some situations this can be guaranteed to happen with a finite number of partitions, while in others it will theoretically happen at $n = \infty$. It is therefore common to relax the condition in 2.42 to

$$f^* = f_i \le \bar{f} + \epsilon \tag{2.43}$$

and label the solution as *ϵ-suboptimal*. If the problem is convex, this entire procedure is trivial since the dual problem can be used as a lower bound certificate on the solution since it is guaranteed to give the same solution as the original (primal) problem. For non-convex problems it is an entirely different story and it can easily end up begin computationally unreasonable to verify the global solution.

Global Solvers

Just at local solvers can be constructed by adding layers of iterations on top of each other, each layer solving a simpler problem that the one above, a global solver is constructed by adding yet another layer of iterations. These iterations will be to solve a sequence of optimization problems to local solutions using the local solvers. These sub-problems serves two purposes. One is to generate a decreasing sequence of local solutions. The other is to generate a sequence of non-decreasing lower bounds. The global solvers differ in how they produce these sub-problems, but tree structured search methods seems to dominate. A well known method is the *Branch-and-Bound* framework. The branch and bound algorithm progress has a three structure and iterations are referred to as nodes in the tree. It is the leaf nodes in the Branch-and-Bound three that play the key role in the algorithm, all other nodes are not really of interest since they must have been completely processed before their child nodes were produced and their information content turned out to be insufficient. Combined the child nodes will always contain all information from the parent (and hopefully some new information too). Each node is related to a sub-space $X_i \subseteq X$, with the root node having $X_r = X$. Given a parent node with sub-space X_p with m child nodes with sub-spaces $X_{c,j}$, $j = 1, \ldots, m$, the following will hold:

$$X_p = \bigcup_{j=1}^{m} X_{c,j}. \tag{2.44}$$

Enumerating the leaf nodes from left to right with $l = 1, \ldots, n$ will then give

$$X = \bigcup_{l=1}^{n} X_l \tag{2.45}$$

because of the property in equation 2.44. The global lower bound from equation 2.40 is made of the lower bounds on the leaf nodes. The algorithm will continue to divide a promising leaf node into two or more child nodes (which then becomes new leafs) until equation 2.43 is satisfied or some other termination criteria is fulfilled (*e.g.* reached maximum number of iterations). Different variations on the branch and bound is available and they differ in the types of problems they accept. A selection of these will be briefly introduced in the following paragraphs.

A special class of MINLP problems are the problems where the only source of non-convexity is the integer restrictions. That is, if the integer requirement is dropped, the problem becomes convex. These are referred to as *convex MINLPs*. Solvers such as COIN-ORs BONMIN is made for such problems [Bonami et al., 2008]. The convex MINLPs are special because they are trivial to relax, and the solvers for these problems are able to find global solution while working solely with a oracle description of the problem. If all the functions involved are linear this problems becomes a Mixed Integer Linear Program(MILP), which has several tailored algorithms related to it [Grossmann, 2002].

Another class of MINLP problems is the problems with special structure, that is, they are formulated using only special terms that the solver expects. These terms typically have known relaxations that can be exploited efficiently. The advantage of restricting the terms that can be used to build functions is that the problem can be stated explicitly in a parametric representation. The *polyhedral branch and cut* implemented in BARON is a solver tailored for problems with a special structure [Tawarmalani & Sahinidis, 2005].

A slightly broader class of problems are those where general terms are allowed, but under the assumption that it is possible to generate bounds on the second derivatives (or Hessian eigenvalues) for the non-convex terms. The α-Branch-and-Bound is an algorithm aimed at these kinds of problems and it will overpower the nonconvexities of general structure with augmenting terms as discussed in the previous section [Adjiman et al., 2000].

The final class of MINLPs that will be mentioned is the collection of all remaining problems that don't possess any redeeming properties. These cover non-factorable functions and black-box descriptions were few assumptions can be made. The solution in this case is to approximate the problem by a slightly better behaved problem whose solution will be close to the original problem. Two strategies are dominant here. The first is to sample the functions upfront, create an approximation and then ignore the original function for the rest of the optimization. A prime example of such a scheme is *piecewise linearisation*, which creates linear interpolating functions on the data and use binary variables to switch between the linear pieces. A constraint structure known as *special ordered set* is used to aid this procedure. This leads to a MILP formulation, which can then be solved using the dedicated MILP solvers. The drawback here is that the number of variables will grow quickly with function domain dimensions and they can quickly become too large to solve. Piecewise linearisation was applied to the oil production problem in [Gunnerud & Foss, 2010]. Other, less know, methods based on upfront sampling is the smooth blending scheme with a know underestimator discussed in [Meyer et al., 2002] and the B-spline hypervolume scheme in [Park, 2012]. The latter only considering unconstrained optimization. This is similar to the approach that will be used in this thesis. The other type of sampling strategy is to sample during the optimization steps to locally refine the approximation as the solver iterates. These methods seem to dominate on problems where the function evaluations are *extremely* expensive (*e.g.* 20 hour car crash simulations), making it impossible to sample the entire domain upfront. An overview of such methods, and their pitfalls, is given in [Jones, 2001]. Many of these are based on *Kriging* or *Thin-plate Spline* approximation schemes, which can be derived as stochastic models. Correlations and uncertainties are used as a base for the decisions made by the solver. This is sometimes called *Bayesian Optimization*, see [Jones et al., 1998] for details.

2.4 Chapter Summary

A local solution to a optimization problem can be established based on a local arguments without making too many assumption about the problem. A global solution on the other hand is more difficult to verify. A central part of proving a global solution is to construct lower bounds on the objective function value. The difficulty of creating these bounds will depend on the problem structure. Specific problem structures can be dealt with efficiently by tailored algorithms, while more general problems may not be reasonable to solve. The most difficult type of problem (of those mentioned here) would be the MINLP where the nonlinearities are given as oracle functions where no favourable assumptions can be made. These will typically be dealt with by approximation techniques. The oil production problem belong in this category.

Chapter 3

Approximation of Functions

The title of this thesis is *Solving a Network Flow Decision Problem with Sampled Nonlinearities*. This chapter will shed some light on the *sampled nonlinearities* part. The goal is to establish how nonlinearities should be sampled and how the samples should be used to construct a reasonable replacement for the original function. Function approximation is a surprisingly diverse topic and it can be studied with pure theoretical interest or as a means to solve practical problems. The practical aspect is the main motivator for this thesis and the material will be presented with this in mind. Function approximation is easily related to the optimization topics in the previous chapter and this chapter will try to follow the tone set there. For the theoretically inclined, the book *Best Approximation in Inner Product Spaces* [Deutsch, 2001] provides a very solid—and very abstract—mathematical treatment of the general approximation problem, but that level cannot be followed here. Most of the information given here is taken from *Curve and Surface Fitting, An Introduction* [Lancaster & Salkauskas, 1988] and *A Practical Guide to Splines* [De Boor, 1978].

The function approximation problem arise when we have a function f that we wish to manipulate or evaluate but for some reason do not have a convenient way to interact with. In our situation this function is a model of reality, given as a complex simulator, that is supposed to be part of a optimization problem. It is, however, cumbersome to have the solver interacting directly with the simulator, which is why we seek to replace it by an analytical approximation $\hat{f}$.

3.1 The Fundamentals of Function Approximation

Problem Formulation

In this chapter $f : X \mapsto \mathbf{R}$ will represent the function to be approximated. It is assumed to be continuous, that is $f \in \mathscr{C}$ ($\mathscr{C}$ being the space of continuous functions). The domain of f has dimension d and the codomain dimension 1 (d is occasionally used to denote metric functions, but the distinction will be clear from the context). In the optimization setting the variables are assumed to be restricted to a box constrained space and it is important that all functions involved are well defined inside this region. Because of this we will assume that the function domain is described by such a box, or atleast can be restricted to one, and the notation

$$X = \left[\underline{x}_1, \overline{x}_1\right] \times \cdots \times \left[\underline{x}_d, \overline{x}_d\right], \tag{3.1}$$

which was introduced in chapter 2, is used to describe this box domain. Apart from this, our knowledge of f is limited to a set of m samples given as pairs $\left(y_i, x_i\right)$ where $f(x_i) = y_i$. The set of x_is is referred to as the *data sites*. The function $\hat{f} : X \mapsto \mathbf{R}$ will be the approximation of f and it will take the form

$$\hat{f} = \sum_{i=1}^{n} c_i b_i(x) \tag{3.2}$$

where $b_i : X \mapsto \mathbf{R}$ are called *basis functions* and c_i *coefficients*. A basis function is said to have *global support* if it is supported over the entire domain X and *compact (local) support* if its support only covers a subset of X.

Formally we may state the approximation problem by letting S and $\hat{S}$ be function spaces such that $\hat{S} \subseteq S \subseteq \mathscr{C}$ and defining $d : S \times S \mapsto \mathbf{R}$ to be a metric on this space. The task is then to find the closest point $\hat{f} \in \hat{S}$ to the point $f \in S$. This can be stated in various ways, but seeing that the theme of this thesis is optimization we will state is as a optimization problem by writing

$$\begin{aligned} \text{minimize} \quad & d(f, \hat{f}) \\ \text{subject to} \quad & \hat{f} \in \hat{S}. \end{aligned} \tag{3.3}$$

A approximation is said to be convergent if the approximation converges to the sampled function as the number of samples increases. Formally this is stated by letting

$\hat{f}_n$ be the approximation based on n samples and having

$$\lim_{n\to\infty} \hat{f}_n = f. \tag{3.4}$$

A approximation is said to be *shape preserving* if it captures certain properties of the data set. This typically means that if the underlying function is positive ($f(x) \geq 0$), monotone ($\nabla f(x) \geq 0$) or convex ($\nabla^2 f(x) \geq 0$), then the approximation has the same property. For our discussions shape preservation is in it self not that important, as long as the approximations does not introduce artificial oscillations in between the samples. Shape preservation is typically enforced via constraints on the coefficients. For instance if the basis functions span a space of nth order polynomials, this space will, generally, contain both convex and non-convex functions. Putting restrictions on the coefficients makes it possible to ensure that the solution to the approximation problem lies within a subspace that only contains convex polynomials [Magnani et al., 2005]. Other ways to introduce shape preservation is to estimate gradients at the data sites, based on the neighbouring data points, and add these to the coefficient constraints.

It may not actually be possible, or reasonable, to compute the solution to the problem in 3.3. In our case the function f is unknown, so computing the distance $d(f,\hat{f})$ might not be possible. In fact, the only information we have at our disposal is a set of samples $\left(y_i, x_i\right)$, where $y_i = f(x_i)$, so the best we can achieve is to minimize a metric on the distances $y_i - \hat{f}(x_i)$, for instance by using the sum of squares,

$$d(f,\hat{f}) = \sum_{i=1}^{n} \left(f(x_i) - \hat{f}(x_i)\right)^2 = \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i)\right)^2. \tag{3.5}$$

As an example: given a dataset with $m = 10$ samples of a function $f : \mathbf{R} \mapsto \mathbf{R}$, let $\hat{S}$ be the space of polynomials of order three. A basis for $\hat{S}$ has $n = 3$ functions and can for instance be given as

$$b_k(x) = x^{k-1},\ k = 1,2,3. \tag{3.6}$$

The problem in 3.3 can now be written as

$$\text{minimize} \quad \| \bar{B}c - y \|_2 \tag{3.7}$$

with

$$\bar{B} = \begin{bmatrix} b_1(x_1) & \dots & b_n(x_1) \\ \vdots & \ddots & \vdots \\ b_1(x_m) & \dots & b_n(x_m) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ & \vdots & \\ 1 & x_{10} & x_{10}^2 \end{bmatrix}, \tag{3.8}$$

$$c = \left[c_1, \dots, c_n\right]^T, \tag{3.9}$$

$$y = \left[y_1, \dots, y_m\right]^T. \tag{3.10}$$

This is the same as the least squares problem in equation 2.16, which is convex and was labelled as "easy" to solve. This is fortunate, since all schemes of interest will have their coefficients produced this way. There will however be differences in how the $\bar{B}$ matrix is structured, ranging from completely filled dense matrices to the trivial identity matrix. For the special case where $d(f, \hat{f}) = 0$, the approximation is said to *interpolate* the data, otherwise it is *smoothing* the data. There are an infinite number of possible functions $\hat{f}$ that are able to reduce the metric in 3.5 to zero and therefore the solution may not be unique. This means that the selection of subspace $\hat{S}$ must be done carefully to avoid uniqueness issues.

3.2 Dataset Structure and Basis Construction

In the previous chapter the efficiency of a numerical solver could be related to how much the solver assumed about the optimization problem. More assumptions leads to better tailored solvers. The function approximation equivalent is the assumptions made on the data samples and how the basis functions will be tailored to these assumptions. The more assumptions that is made on the data structure the better the basis functions will be at constructing pleasing surfaces for the data.

For this thesis exact interpolation is set as a goal, but it should not be taken for granted that this is the best option. If the simulators have logical statements or unreliable computations for certain variable ranges, it may be better to allow some smoothing of the data. The discussion in chapter 9 will attempt to address the issue. The discussion in the remainder of this section will be based on the assumption that the approximating function *should* interpolate the data samples. To achieve interpolation the coefficients in equation 3.9 must be found by solving

$$Bc = y \tag{3.11}$$

were B is generated from the basis functions and data samples. In this thesis B will be equal to $\bar{B}$ from equation 3.8, but it could contain all types of information, such as derivatives if that is available. Since equation 3.11 must have an exact solution (as opposed to the least squares solution in the smoothing case), B must be invertible. This will put some restrictions on the combination of basis functions and data sites. In the univariate case it is fairly easy to make B invertible and the only issue is to strike a balance between having enough data to construct a good enough approximation and being able to generate and handle the amount of data efficiently. For multivariate functions sampling has more complexity to it. In this case it is not only the distance between individual points that must be chosen, but the positioning as well. A small example (modified from [Lancaster & Salkauskas, 1988]) of how this can cause problems can be made with the bilinear interpolation of four data points. The four basis functions are

$$b_1(x) = 1 \tag{3.12}$$
$$b_2(x) = x_1 \tag{3.13}$$
$$b_3(x) = x_2 \tag{3.14}$$
$$b_4(x) = x_1 x_2. \tag{3.15}$$

Let the data samples be taken at the corners of a square: $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$. With these data sites, equation 3.11 is given with

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.16}$$

This can be solved trivially. However, if the data site pattern is tilted 45° to make a diamond shape it does not work out so well. Taking the samples at $(-1,0)$, $(0,-1)$, $(1,0)$ and $(0,1)$ the left hand side is changed to

$$B = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \tag{3.17}$$

which clearly cannot be solved unless $y_1 + y_3 = y_2 + y_4$. The matrix B only has rank 3. If the B matrix is studied as the samples are rotated from the square pattern to the diamond pattern one would notice that the matrix condition number would go from good to bad to singular as the rotation goes from 0° to 45°. For a scheme to be successful it must avoid these situations, either by assuming a specific structure to the data sites that suits the basis functions or by adapting the basis to the data sites. The following sections will go through some of the sampling strategies and the types of approximation schemes they typically lead to. Three types of sampling will be mentioned here: scattered, cross-section and grid. They are illustrated in figure 3.1. The distinction between these are only relevant in the multivariate case, as they are all equal when limited to one dimension.

Scattered Data Sites

Scattered data has full freedom in all directions when choosing data sites. This is the most difficult data to deal with and existence of a solution to 3.11 is a major issue. There are two dominating ways to deal with the existence issue. One is to avoid the computation by setting the coefficients be equal to the data values. The other is to triangulate the data sites and design an interpolation scheme inside the triangles.

Fixing the coefficients to $c_i = y_i$ is obviously restrictive and leaves limited choices for the basis functions. The first observation is that for such a scheme to be interpolating the basis functions must satisfy

$$b_i(x_j) = \begin{cases} 1, \, i = j \\ 0, \, i \neq j. \end{cases} \tag{3.18}$$

These basis functions are often built around a distance function(metric) and the upside is that they do not directly rely on how other data samples are positioned, which makes it perfect for scattered interpolation. The downside is that they are often non-smooth or look "bad". Typical example of a non-smooth scheme would be *Nearest Neighbour* which divides the function domain into patches P_i and gives one patch to each data sample and the patches have the property that a patch P_i contain the points that are closer to x_i than any other data sample. With this the basis functions are just

$$b_i(x) = \begin{cases} 1, & \text{if } x \in P_i \\ 0, & \text{otherwise .} \end{cases} \tag{3.19}$$

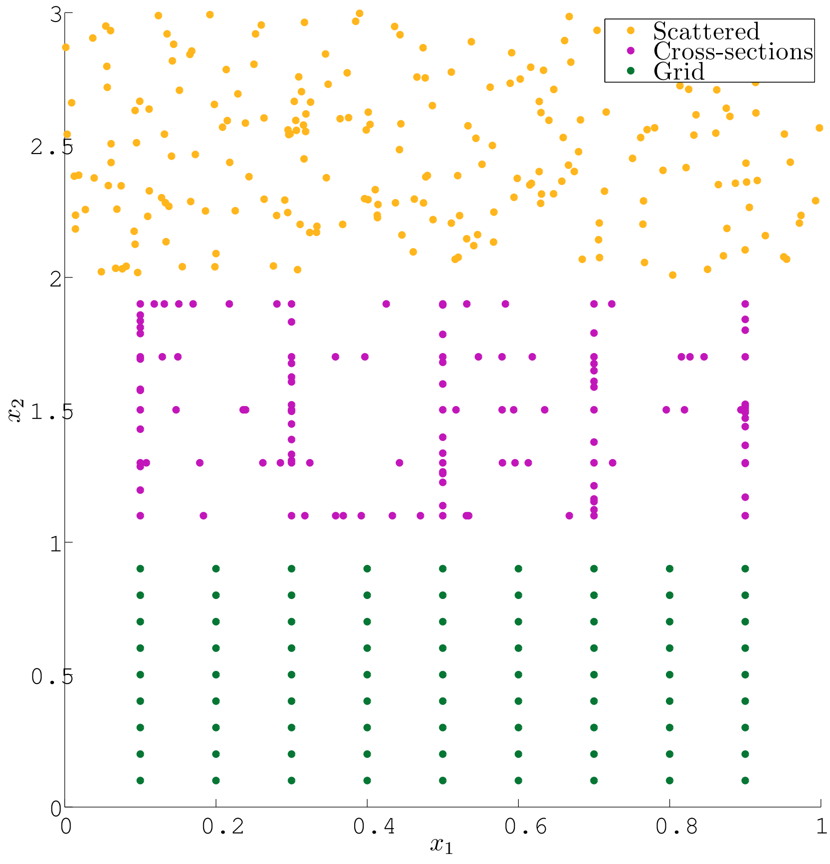

Figure 3.1: Three different sampling methods with varying degree of freedom. Scattered is completely free to choose data sites. Cross section is free to chose data sites along certain lines (the lines could been taken diagonally). Grid has no choice, since all data samples must lie on the intersections in a imagined grid that is stretched over the function domain.

In the univariate case this would produce a function that looks like a series of steps. The smoother schemes of this type are often referred to as *Inverse Distance Weighting* or *Interpolating Moving Average*. These are based on weighting functions built around the inverse of a metric function. The most basic example here is *Shepard's Interpolation* where the weights are given as

$$w_i(x) = \frac{1}{d(x, x_i)^p}, \tag{3.20}$$

where p is a chosen parameter that will influence the function properties. The interpolation scheme is then given as

$$\hat{f}(x) = \frac{\sum_{i=1}^n y_i w_i(x)}{\sum_{j=1}^n w_j(x)} = \sum_{i=1}^n y_i b_i(x), \quad b_i(x) = \frac{w_i(x)}{\sum_{j=1}^n w_j(x)} \tag{3.21}$$

The idea is fairly simple, but the results are often surprisingly disappointing. An illustration of Shepard's method is given in Figure 3.2 with two different values of p. There are plenty of variations on this method. A more involved scheme that made use of derivative information in the weighting process was applied successfully in a global optimization context in [Meyer et al., 2002] It must be said that this scheme also produces "wobbly" surfaces, but not as bad as the ones seen in Figure 3.2. It is not without reason that these schemes never get suggested as candidates for interpolation in the univariate case. They are just not that good. Their strength lies in the simplicity and the minimal amount of assumptions made.

An alternative to the distance based schemes is to triangulate the data sites and focus on one triangle at the time. The triangles are mapped to a *standard triangle* where all the work is done (much like mapping any interval on the real line to $[0,1]$ for simpler analysis). A set of basis functions can be tailored to the standard triangle so an interpolating solution is guaranteed to exist (since the geometry of the relevant data sites are now known and located at the corners of the triangle). When computations are done, the answers are mapped back to the original triangle and used. Methods based on these techniques are know as *Finite Element Methods* and are most common in two or three dimensional space. They do, to my knowledge, not scale that gracefully and as a consequence the sufficiently smooth Finite Element methods are restricted to functions of few dimensions, but in these dimensions they seem to perform well. Finite Elements are not restricted to triangles, any pattern that is able to tessellate the function domain would do, but the more exotic geometries may require more structure in the data sites.

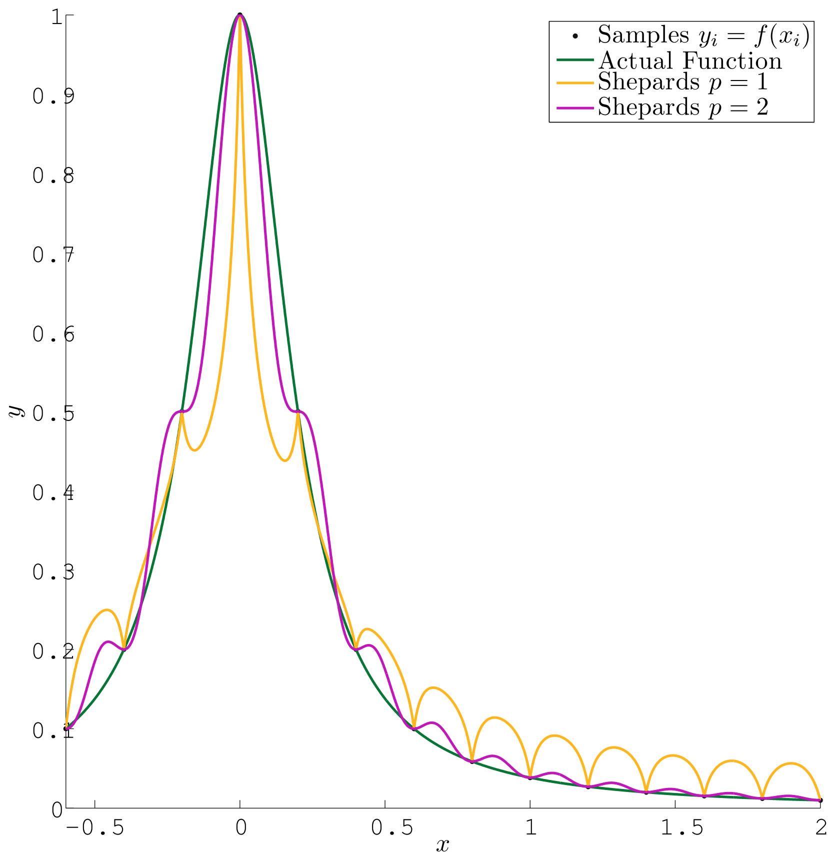

Figure 3.2: Shepard's interpolation with $p = 1$ and $p = 2$.

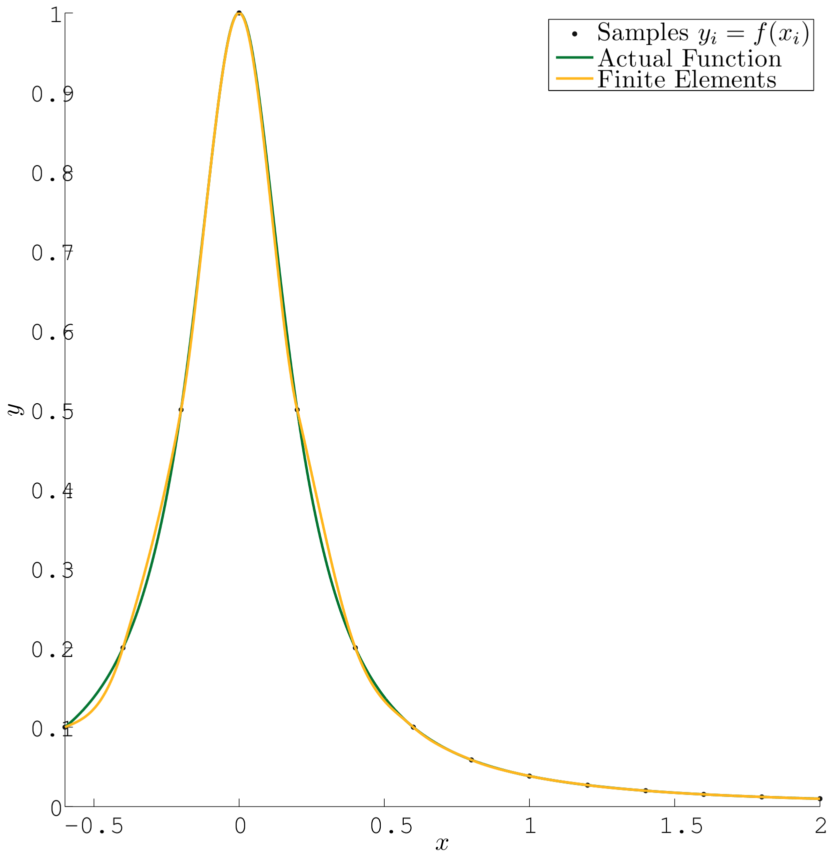

Figure 3.3: Cross-section of finite element interpolation with cubic elements.

Other alternatives for scattered data approximation worth mentioning are *radial basis function methods* and *Kriging*. Radial basis functions are functions that peak at some point and gets smoothed out as the distance to the point is increased. This is similar to the weighing schemes, except that the radial basis functions do not take other points into account, and the basis coefficients must be computed. Thin-plate spline is a method based on this technique. Kriging is a stochastic model that seems to be used mostly in geostatistical communities. These methods do, as far as I can understand, not interpolate the datasets in the general multivariate case.

Cross-section Data Sites

The methods based on sampling cross-sections of the function domain typically comes up in engineering practices that deals with the construction of metal structures like car bodies and ships. Here it is natural to deal with cross-sections of the structure and then leave it to a interpolation scheme to fill in the gaps between these cross-sections. The idea is to create mono-variable interpolating functions along each cross-section and then blend these together. This type of interpolation is referred to as *Blending Methods*. This blending procedure would in the general case be subject to the same difficulties as the scattered schemes, but when the cross-sections are chosen in a neat pattern as in Figure 3.1 it is fairly easy to create a controlled blend of only the functions closest to the point being evaluated.

These schemes can get quite involved when the number of dimensions is increased and during the literature study for this thesis they only came up explicitly stated in the two dimensional case. They were however not pursued very far and I cannot see any reason for why they should not generalize to higher dimensions.

Grid Data Sites

Grid, sometimes also referred to as mesh, is a sample structure where the data sites are neatly laid out in a grid formation and there is no freedom involved. Formally it is described by letting $t_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}\}$ be a ordered set with n_i elements for each $i = 1, 2, \ldots, d$ (that is, one set for each variable x_i). These sets have the property that $t_{i,j} < t_{i,j+1}, i = 1, 2, \ldots, n_i - 1$ and $t_{i,j} \in \left[\underline{x}_i, \overline{x}_i\right], i = 1, 2, \ldots, n_i$. This is basically taking n_i different values inside the boundaries of the variable. Using t_i to create a new set by applying the Cartesian product to each set in turn we get

$$T = t_1 \times t_2 \times \cdots \times t_d. \tag{3.22}$$

A data set is said to have a grid structure if there exists such a set T that is equal to the set of data sites. The grid based basis functions seem to outperform all other schemes in the situations where they can be applied. A nice feature of the grid based schemes is that they have a notation that "scales" with the domain dimension, making them quite convenient to work with once this notation has been established. The idea here is to create mono-variable basis functions that *could* have been used to interpolate along a lattice in the grid, but instead use them to create new multivariate basis functions by multiplying them together in a pattern similar to the construction of the grid. These methods are called *Tensor Product Methods.* They are constructed by first considering the grid components $t_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}\}$ which dictates where the samples are located regarding variable x_i. A set of univariate basis functions $b_{i,j}(x_i)$, $j = 1, \ldots, n_i$ are constructed and this basis have the ability to interpolate at the data sites in t_i. Let this set be labelled

$$b^i = \{b_{i,1}, \ldots, b_{i,n_i}\}. \tag{3.23}$$

With one such set of univariate basis functions for each variable the multivariate basis is given as the tensor product of these:

$$b = \bigotimes_{i=1}^{d} b^i. \tag{3.24}$$

Reusing the example with bilinear interpolation of the four data samples positioned as a square: $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$. This is a grid structure with

$$t_1 = \{t_{1,1}, t_{1,2}\} = \{0,1\} \tag{3.25}$$

and

$$t_2 = \{t_{2,1}, t_{2,2}\} = \{0,1\} \tag{3.26}$$

which would give data sites at

$$t_1 \times t_2 = \{(t_{1,1}, t_{2,1}), (t_{1,1}, t_{2,2}), (t_{1,2}, t_{2,1}), (t_{1,2}, t_{2,2})\} = \{(0,0), (1,0), (0,1), (1,1)\}. \tag{3.27}$$

With two data sites in each direction, simple linear functions will do as univariate basis functions:

$$b^1 = \{1, x_1\} \tag{3.28}$$

$$b^2 = \{1, x_2\}. \tag{3.29}$$

This gives the tensor product basis

$$b = \bigotimes_{j=1}^{2} b^j = b^1 \otimes b^2 = \{1 * 1,\, 1 * x_2,\, x_1 * 1,\, x_1 * x_2\}, \tag{3.30}$$

which simplifies to

$$b = \{1,\, x_2,\, x_1,\, x_1 x_2\}. \tag{3.31}$$

This is recognized as the same basis suggested by equations 3.12-3.15. The tensor product schemes scales well with dimension as it is only a matter of deriving sufficient univariate basis functions for all dimensions and them applying equation 3.24. The multivariate basis functions will inherit the smoothness of the univariate functions.

The tensor product schemes are usually based on piecewise polynomial functions, but there are no actual restrictions here. With cubic piecewise polynomials the tensor product scheme could be made to produce the same cross section as the finite element method in figure 3.3.

Choice of Basis Functions

So far the discussion has been about various ways to make the basis functions and data sites co-operate nicely and not so much about what the individual basis functions should look like. The central question here is whether the basis should consist solely of polynomial terms or not. Schemes that chose to introduce non-polynomial terms usually does so because they want to replicate certain shapes or effects that can be described by fewer symbols if terms with logarithms or exponentials are allowed. This happens in some of the statistics based schemes and some weighting and radial basis schemes. In most cases however it is low order polynomial that dominate the basis function construction. Part of the reason being that computers cannot really work with anything else than polynomials anyway and all other terms will have to be approximated by a polynomial based expression. As an example take the two sets of univariate basis functions

$$A = \{x^0, x^1, \ldots, x^n\} \tag{3.32}$$

and

$$B = \{x^0, x^1, \ldots, x^{n-1}, e^x\} \tag{3.33}$$

and $n+1$ samples in the interval $[0,1]$:

$$x_i \in \{0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}, 1\}. \tag{3.34}$$

The matrix $\bar{B}$ from equation 3.8 is formed for each set and labelled $\bar{B}_A$ and $\bar{B}_B$. The rank of these matrices as n goes from 2 to 30 is visualized in figure 3.4. When the number of samples become large enough the basis with e^x will lose rank since the representation of this function is based on the monomials $x^k, k = 1, \dots, n-1$(or atleast numerically indistinguishable from these terms), thus making it linearly dependent on the other terms. In theory this should not happen, but in practice it does. It can also be seen that when the number of samples is increased even further the pure monomial basis will also lose rank. This is because the numerical precision is unable to capture the numbers produced by such high powers, so eventually $0 \approx \left(\frac{1}{n}\right)^n$ and the matrix will lose rank when evaluated numerically. The matrix $\bar{B}_B$ is know as the *Vandermonde matrix* and is know to be ill conditioned. Fortunately one does not have to deal with the Vandermonde matrix when constructing interpolating polynomials. For instance if the basis is changed to the *Lagrange polynomial* basis,

$$b_i(x) = \prod_{j=1}^{n} a_i(j), \quad a_i(j) = \begin{cases} \dfrac{x - x_j}{x_i - x_j}, \, i \neq j \\ 1, \, i = j \end{cases}, \tag{3.35}$$

the coefficients can be computed trivially, since $\bar{B}$ becomes the identity matrix (the basis has the property in equation 3.18). These are just two ways of representing the same space, one using a trivial basis with difficult coefficient computation and the other using a difficult basis with trivial coefficient computation. There are many other famous basis constructs for a space of polynomials and the best choice depends on the task at hand. However, regardless of representation, high order polynomials do not possess the best approximation properties and often go by the name of *oscillating polynomials* in the interpolation context. Figure 3.5 illustrates why this is. This is known as the *Runge phenomenon.* The oscillatory polynomials will not generally converge to the underlying function as the number of samples are increased. There are ways to mitigate the Runge phenomenon, for instance sampling at smaller intervals towards the end points.

The basis function discussed so far has been based on globally supported terms. This has turned out to scale poorly. The alternative is to define locally supported functions, where the local pieces have lower complexity, typically low order polynomials. These are known as *piecewise polynomials.* The piecewise polynomials have

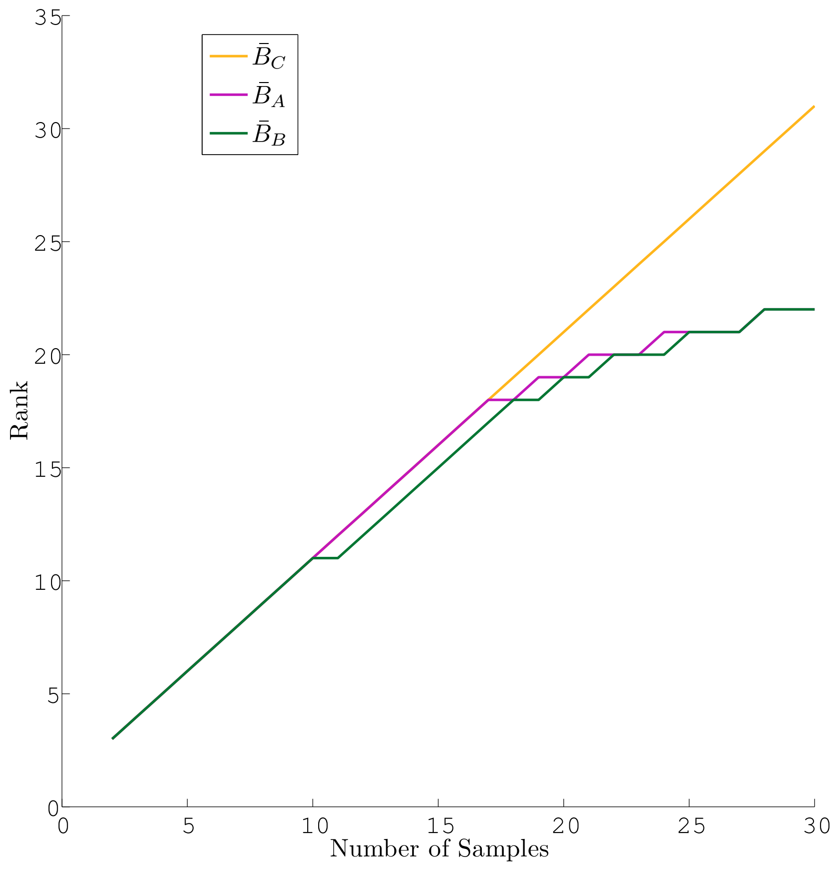

Figure 3.4: Comparison of the rank of $\bar{B}$ from equation 3.8 for three different sets of basis functions when evaluated numerically for an increasing number of samples in the interval $[0,1]$. Global monomials given as $\bar{B}_A$, global monomials with one exponential term given as $\bar{B}_B$ and local piecewise monomials given as $\bar{B}_C$.

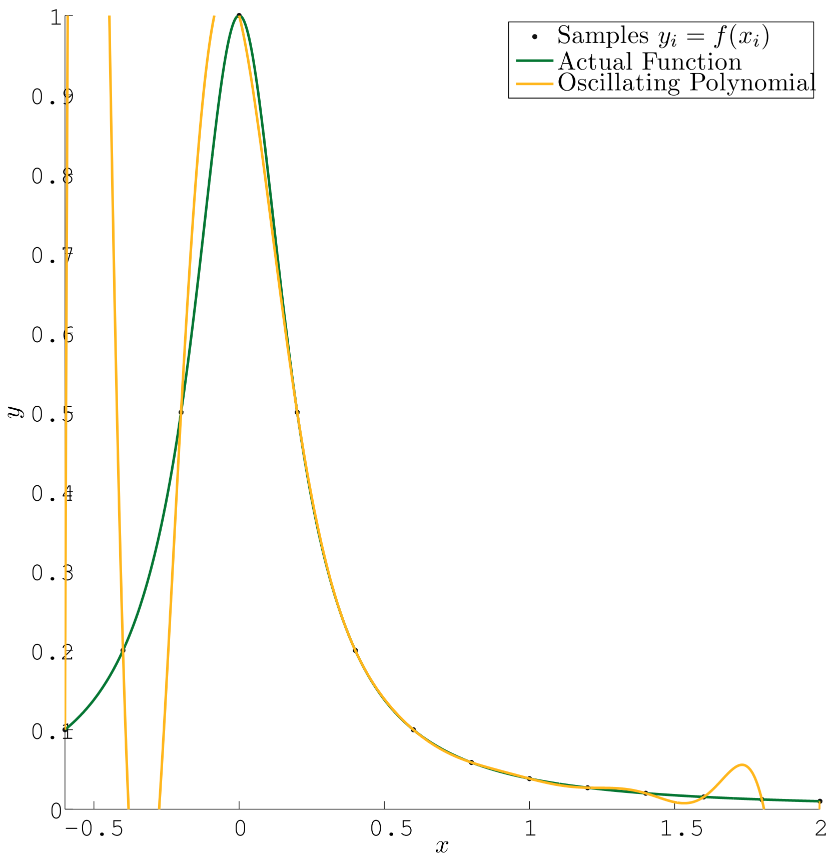

Figure 3.5: Interpolation using a single polynomial. It is obvious why this method is referred to as oscillatory polynomials.

much better convergence properties, but it comes at the price of increased complexity in the basis construction. The idea is simple; the function domain will be divided into box-shaped regions and each region is assigned a polynomial. The difficult part is to make smooth transitions between the regions. It is common to use cubic polynomials inside the regions because this gives enough flexibility to make the transitions twice differentiable while keeping the complexity as low as possible. The basis functions could for instance be written as

$$b_{i,j}(x) = \begin{cases} x^j, & \text{if } x \in P_i \\ 0, & \text{otherwise .} \end{cases} \tag{3.36}$$

where the P_js are the regions that tessellate the domain. The univariate interpolation scheme could then be found as

$$\hat{f}(x) = \sum_{i=1}^{M} \sum_{j=0}^{k} a_{i,j} b_{i,j}(x) \tag{3.37}$$

were M is the number of regions and k is the order of the polynomials. With a bit of re-indexing this could be cast into the familiar shape

$$\hat{f}(x) = \sum_{i=1}^{M*k} a_i b_i(x). \tag{3.38}$$

The $\bar{B}$ matrix for piecewise polynomials is well behaved because it is block diagonal(one block for each local piece) where each block has the same rank, thus the rank of the complete matrix grows gracefully without causing too much trouble. It can be seen in figure 3.4 labelled as $\bar{B}_C$.

Piecewise polynomial basis functions have great approximating power. They can be made to be convergent if used sensibly.

3.3 Discussion

As stated in chapter 2, the numerical solvers of interest assumes the functions involved are twice continuously differentiable. The approximation scheme must therefore have this property. Of the schemes mentioned in the sections above, only the tensor product schemes and the inverse distance weighting schemes seem to allow for sufficient smoothness in the general multivariate case without growing overly complex. Seeing that the scattered schemes produce surfaces that are unnecessarily "wobbly" and the tensor schemes don't, it is not really a difficult choice to make, considering that the sampling structure can be chosen freely. For most simulators it would even be easier to generate a grid based data sets anyway. It should however be pointed out that both scattered and grid interpolation schemes are found in the global optimization literature. Scattered interpolation methods using a Shepard's type of interpolation was used in [Meyer et al., 2002] with some success, but it was concluded that the required number of samples grew too quickly with dimension. The scheme had a suggested sampling algorithm that was based on refining a grid structure until the approximation error was tolerable. Even if the suggested scheme used a grid layout to generate data, the actual interpolation method could, to my understanding, have accepted scattered data, but it would have complicated the tuning of the schemes internal parameters. This is however not the most interesting part. The main lesson to be learned here is that the number of data samples does grow exponentially with dimension if the samples are taken with a grid structure with the same resolution in all variables. This will obviously reach a limit where the computational load becomes too high. Assuming that the function dimension cannot be reduced, the only ways of lowering the number of samples is either to increase the threshold of approximation error or to increase the approximating power of the interpolation scheme. Of all the interpolation methods that have been seen throughout the literature study of this thesis, the piecewise polynomials seem to be a clear winner when it comes to approximative power. Combine this with the generally attractive properties of the tensor product schemes and the resulting function is quite an interpolating powerhouse. Tensor product piecewise polynomials are called splines. Splines was used in unconstrained global optimization in [Park, 2012] with success, using a representation known as B-spline form.

Having singled out splines as the most promising technique, the next question is how should the coefficients be computed. The choices are related to what kind of data should be used and what kind of influence should the data have. We only have function values, so the first question is rather easy, but in the case where derivative information is available, it would be beneficial to add this to the equation sets as well. The next question is about influence. With locally supported basis functions it is possible to have the coefficients computed only based on the data samples inside the support of each individual function. Global influence will make all samples affect all basis functions. Global influence will result in a global set of equations, but it would also make it easier to maintain smoothness where the pieces are joined together. Local influence will make the computations easier since it can be reduced to many small sets of equations rather than one large. The drawback being that special care must be taken to match the local pieces together with sufficient smoothness. This is typically done by estimating a gradient at the boundary between two pieces and include this in the computations on both sides. Piecewise polynomials where derivatives are used to match independent pieces together are called *Hermite polynomials*. In this thesis a simple global influence scheme was selected, because it is the easiest method to deal with in multiple dimensions and the more complex methods could not really be justified. Details on the computation will be given in Chapter 4. The computations used are not shape preserving. There are methods to make splines shape preserving, but these where not studied in detail in this thesis. It might be interesting to note that in [Neamtu, 1991] it was concluded that a local method cannot be guaranteed to preserve the convexity of a convex data set.

Representing splines can be done in various ways and any polynomial basis could be used for this task. Some choices are however better that others and the B-spline form will be used here, since it has a solid literature behind it and it has a structure that can be exploited when creating convex relaxations.

In addition to neat approximation features, splines also posses other features that can be exploited in the branch and bound setting. Since their basis functions have local support, some of the basis functions will be left unsupported as the algorithm iterates. This means that parts of the basis can be removed when a node is divided into new child nodes, which simplifies the approximating function, making computations faster. Exactly how this will be used is explained in chapter 6.

3.4 Chapter Summary

A selection of approximation methods has been presented. The tensor product piecewise polynomial has been singled out as the most promising method. It requires the data sets to be structured with a grid layout, but this is acceptable since the simulators being used can generate data in this format. The scattered interpolation techniques did not hold up as good as the tensor product methods and will be left as a last resort if the available data for some reason fails to have the necessary structure. Scattered interpolation could then be used to re-sample the data to fill in the missing pieces and then apply a tensor product scheme. The selected spline representation is the B-spline form and this will be explained in detail in chapter 4.

Chapter 4

The B-spline

Having concluded in Chapter 3 that a tensor product scheme with piecewise polynomials seems to be the best alternative, the next step is to find a suitable basis for this space. The naive monomial basis is not useful for most applications apart from being easy to describe. This chapter will introduce the B-spline, which is convenient representation of piecewise polynomial functions. The B-spline has a solid literature related to it and quite a few neat algorithms and features that can be exploited in global optimization. This chapter will go through the anatomy of a B-spline and it is assumed that the goal is to interpolate a set of data with a grid structure as it is described in section 3.2. A lot of well know ground will be covered here and several works could be used as a reference for most of the topics. This thesis has relied on the famous book by DeBoor [De Boor, 1978] and the slightly more up to date writings of Schumaker [Schumaker, 2007]. During the literature study the initial source of B-spline information was a book draft by Lyche and Mørken [Lyche & Morken, 2004] and this is, in my opinion, the most user friendly introduction to the topic in a self study setting. However, Schumakers book currently has the broader scope and will be the default source of information. Additional references will be given when necessary.

Reading the B-spline literature I came across a few different ways to represent B-splines. The differences stem from whether the indexing used represents the degree or the order of the polynomials. This can be quite confusing when switching between different sources on information. So to clear up the possible confusion, in this thesis the *order* will be used to generate indexes. Order is equal to the degree plus one. That is, a linear polynomial is degree 1 and order 2, and a cubic polynomial is degree 3 and order 4.

4.1 The Knot Sequence

A knot is the point where two polynomial pieces are tied together. The knot sequence is a partitioning of the function domain and will be used to generate the basis functions. One knot sequence is required for each function variable, *e.g.* if the function is $f : \mathbf{R}^d \mapsto \mathbf{R}$, d knot sequences is required. A knot sequence consists of $n_j + k_j$ numbers, where n_j is the number of univariate functions and k_j is the order of the univariate functions for the variable x_j. The knot sequence for x_j is denoted

$$T_j = \{t_{j,i}\}_{i=1}^{n_j+k_j}. \tag{4.1}$$

In the case of a one dimensional function domain the subscripts j can be removed, leaving only

$$T = \{t_i\}_{i=1}^{n+k}, \tag{4.2}$$

and we will use this as the default notation, only including the multivariate subscripts when they are needed. For a knot sequence to be acceptable its elements must satisfy

$$t_{i-1} \leq t_i < t_{i+k} \, \forall \, i \in \{1, \ldots, n_j\}. \tag{4.3}$$

This basically means that the knot sequence must be increasing and that the same value only can be repeated up to k times. A second property which is nice, but not necessary, is that the first k elements are equal and the last k elements are equal. Such a knot sequence is called *regular* and it gives the basis functions some nice properties that will be mentioned in the next section. The knot sequence can be chosen quite freely, but some choices are safer than others. For interpolation purposes it is common to let the knot sequence reflect the data sample sites. Let

$$\tau = \{\tau_1, \ldots, \tau_{n+k}\} \tag{4.4}$$

be the one of the grid partitions (grids were explained in section 3.2). To make the knot sequence from equation 4.2 regular, it would have $t_1 = t_2 = \cdots = t_k = \tau_1$ and $t_{n+1} = \cdots = t_{n+k} = \tau_n$. For the values in between these k first and last points various strategies exist and they depend on the order of the polynomials involved (because the order dictates the number of repetitions of the first and last value). If the order is two (piecewise linear basis functions) the knot sequence is of length $n+2$ and since the first and last value is repeated twice, the knot sequence can simply be taken as identical to the grid values with the first and last repeated once:

$$t_i = \begin{cases} \tau_1, & \text{if } i = 1 \\ \tau_n, & \text{if } i = n+2 \\ \tau_{i-1}, & \text{otherwise.} \end{cases} \tag{4.5}$$

If the basis functions are fourth order, the knot sequence needs $n+4$ elements and the first and last value is repeated four times each. If each value in between should be used as in the linear case this would give $n+6$ elements in the sequence, hence all data sites cannot be present if a regular sequence is desired. It is therefore common to skip the second sample from both sides (τ_2 and τ_{n-1}) and then proceed as in the linear case. This gives

$$t_i = \begin{cases} \tau_1, & \text{if } i = 1,2,3,4 \\ \tau_n, & \text{if } i = n+1, n+2, n+3, n+4 \\ \tau_{i-2}, & \text{otherwise.} \end{cases} \tag{4.6}$$

If the polynomial order is an odd number (*e.g.* quadratic polynomials), this type of approach would create asymmetry, so instead the knots would be taken to lie in between all the grid values. In the general case the knot sequence is typically generated so the knots are evenly spaced out among the data samples to make sure there is a match between the number of supported basis functions and the number of samples in all regions. In this thesis all approximations are done using cubic B-splines with knot sequences generated by equation 4.6.

The knot sequences suggested above is by no means the only useful alternative. If, for instance, a Hermite interpolation scheme is desired it would be necessary to repeat all knot values twice. This is to allow for enough flexibility to specify both function value and function derivative at each data sample. In this thesis only function values are considered so the simple regular sequences will suffice.

4.2 The Basis Functions

Starting with the one dimensional case and a knot sequence T as defined in equation 4.2. This knot sequence specifies n basis functions $B_i^k(x), i = 1, \ldots, n$. The superscript k being the polynomial order of the basis function. Using the approximating function syntax from chapter 3, the final B-spline function defined this way is

$$y = \sum_{i=1}^{n} c_i B_i^k(x) = \sum_{i=1}^{n} c_i b_i(x) \tag{4.7}$$

The basis functions are defined by the *Cox-de Boor* recursion

$$B_i^k = \frac{x - t_i}{t_{i+k-1} - t_i} B_i^{k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1}^{k-1}(x),$$
$$B_i^1(x) = \begin{cases} 1, & \text{if } t_i \le x < t_{i+1} \\ 0, & \text{otherwise} \end{cases}.$$

This means a basis function of order k is given by a normalized convex combination of two basis functions of order $k-1$. Basis functions have support inside half open intervals because of the way B_i^1 is defined. It is desirable to close the last of these intervals so the complete function has support inside the closed interval $\left[\underline{x}, \overline{x}\right] = [t_1, t_{n+1}]$. This can for instance be done by defining the basis function values at $B_i^k(ub)$ as the limit

$$B_i^k(\overline{x}) = \lim_{x \to \overline{x}} B_i^k(x). \tag{4.8}$$

This is surprisingly often not mentioned in the literature and is just something to be aware of. The reason probably being that it is quite annoying to keep bringing up this special case in every discussion, so from now on it will just be assumed that special care is taken at the last interval to make sure the function is properly defined inside the closed box domain.

The basis defined above is sometimes referred to as a normalized B-spline basis and it has the property that

$$0 \le B_i^k(x) \le 1 \quad \forall x \in \left[\underline{x}, \overline{x}\right]. \tag{4.9}$$

Without normalization the values would depend on the positioning of the knots involved in the recursion. When a regular knot sequence is assumed the basis functions become a partition of unity, which means that they always sum to one:

$$\sum_{i=1}^{n} B_i^k(x) = 1 \quad \forall x \in \left[\underline{x}, \overline{x}\right] \tag{4.10}$$

The basis functions have local support. For any i the basis function $B_i^k(x)$ is nonzero only for $x \in [t_i, t_{i+k})$. As a consequence of this, at most k basis functions will be nonzero at any time and equation 4.10 can be restricted to only the nonzero functions

$$\sum_{i=I-k}^{I} B_i^k(x) = 1 \quad \forall x \in [t_I, t_{I+1}) \tag{4.11}$$

for any given interval $[t_I, t_{I+1})$.

An interesting aspect of the B-spline basis functions is that each function covers multiple polynomial pieces, and the functions overlap each other. A kth order basis function is stretched over k pieces and up to k basis functions will be non-zero in the same interval. To get familiar with the basis functions, their development is illustrated up to the forth order in figure 4.1 using the knot sequence

$$t = [t_1\ t_2\ t_3\ t_4\ t_5] = [1\,2\,3\,4\,5], \tag{4.12}$$

which contains the number of knots needed to produce a single cubic basis function. In addition to one cubic basis function this knot vector defines four constant, three linear and two quadratic basis functions, all which are used during the construction of the cubic function. The overlapping of basis functions is illustrated in figure 4.2. This has the interesting effect that each piece in the piecewise polynomial is not described by k monomials of increasing order with support limited to the interval, but by k polynomials of order k with support covering multiple intervals. This is good because is makes it easy to guarantee smooth transitions between the intervals (but one could also argue that it is bad because it makes the scheme quite difficult to grasp at first glance). There are four degrees of freedom inside the interval(one coefficient for each basis function), but the basis functions are defined over more than one interval. This means that the basis coefficients will affect neighbouring intervals as well and causes a chain reaction that makes all coefficients dependant on each other and they must therefore be computed by a global (banded) set of equations if the knot sequence has the layout suggested in the previous section.

In the multivariate case the basis functions are created by first constructing sets of univariate basis functions using the procedure described above and then take the tensor product of these sets to produce the multivariate basis. This is done by letting

$$b^j = \{B_{j,1}^{k_j}(x_j), \ldots, B_{j,n_j}^{k_j}(x_j)\} \quad j = 1, \ldots, d \tag{4.13}$$

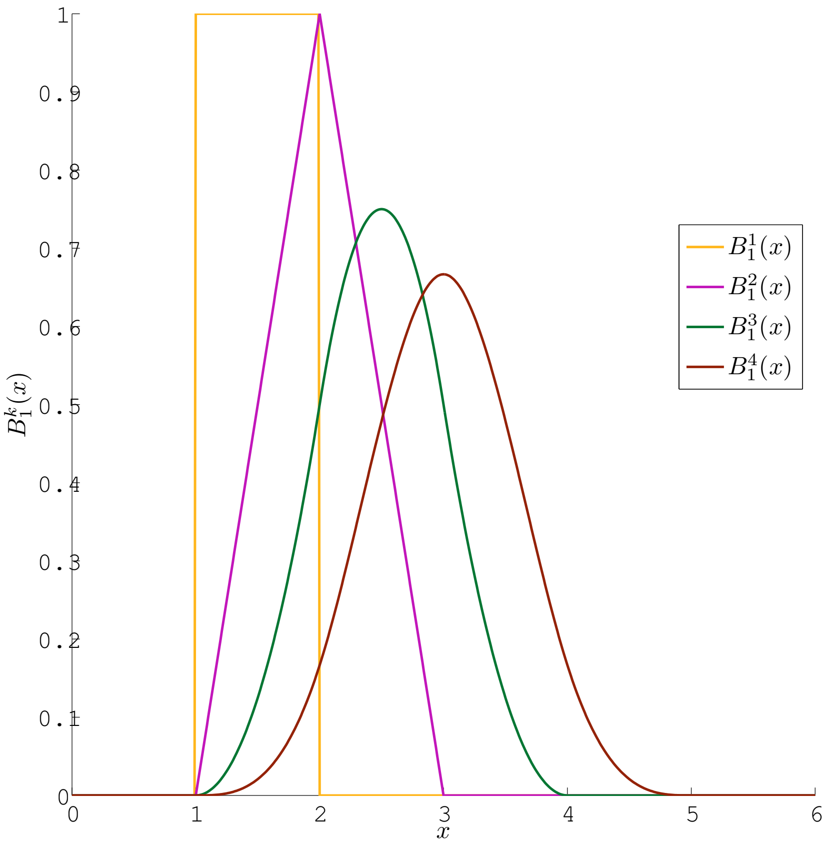

Figure 4.1: Four B-spline basis functions of increasing order defined on the knot sequence in equation 4.12. As the order increases the basis functions gets smoother and stretched over an increasing number of partitions.

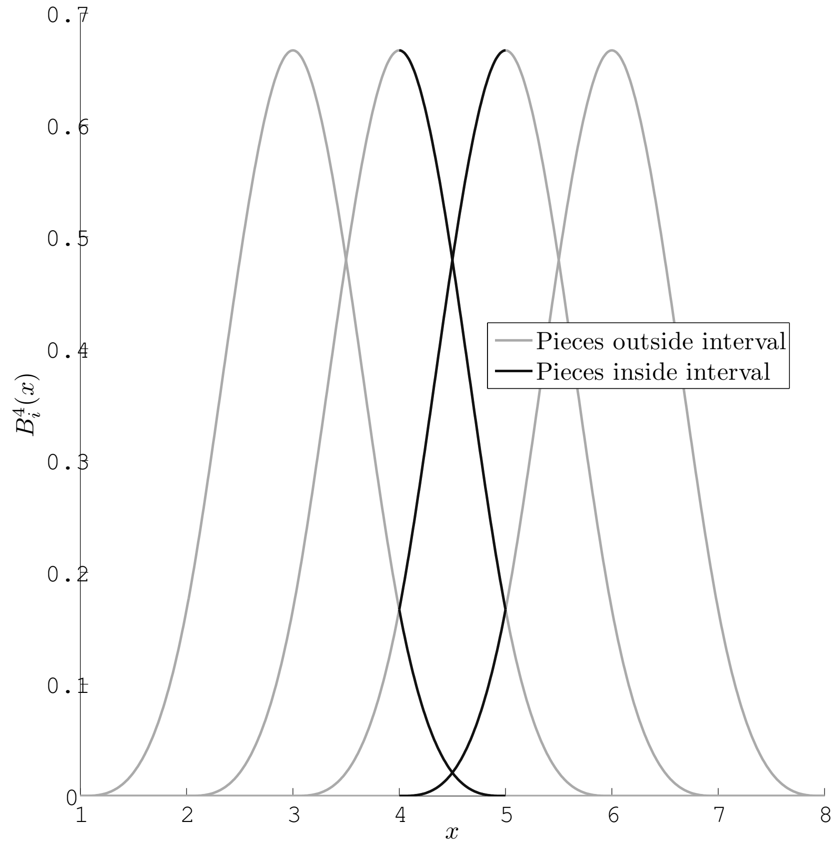

Figure 4.2: Four cubic basis functions, all active inside the interval [4,5). The parts of each function inside this interval is coloured black, the rest gray. The four pieces inside the interval makes up all the individual pieces of a single basis function.

be the sets of univariate functions and create the multivariate basis by tensor product:

$$b = b^1 \otimes b^2 \otimes \cdots \otimes b^d = \bigotimes_{i=1}^{d} b^j. \tag{4.14}$$

The elements of b can now be used as any other basis:

$$y = \sum_{i=1}^{n} c_i b_i(x). \tag{4.15}$$

Here, n is the final number of basis functions and is given by

$$n = \prod_{j=1}^{d} n_j. \tag{4.16}$$

The final form of the basis is

$$b_i(x) = B_{i_1,1}^{k_1}(x_1) B_{i_2,2}^{k_2}(x_2) \dots B_{i_d,d}^{k_d}(x_d) \tag{4.17}$$

and can be described either by the one dimensional index i or the d-dimensional index $[i_1, i_2, \dots, i_d]$. Using multiple sums and the multidimensional indexing the tensor product B-spline can be written as

$$y = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} c_{i_1,i_2,\dots,i_d} B_{i_1,1}^{k_1}(x_1) B_{i_2,2}^{k_2}(x_2) \dots B_{i_d,d}^{k_d}(x_d). \tag{4.18}$$

4.3 Parametric Curve Representation

The function $f(x) = y$ represent pairs (x, y) by calculating y as a function of x, $(x, y) = (x, f(x))$. The same pairs can be represented by parametric curve by introducing a new variable t and compute x and y as functions of t. The pairs can then be written as $(x, y) = (x(t), y(t))$. For a B-spline

$$y = \sum_{i=1}^{n} c_i b_i(x) \tag{4.19}$$

the parametric curve functions are

$$y(t) = \sum_{i=1}^{n} c_i b_i(t) \tag{4.20}$$

and

$$x(t) = \sum_{i=1}^{n} t_i^* b_i(t). \tag{4.21}$$

These expressions both use the same basis functions as the original B-spline function. In addition, $y(t)$ is also using the original coefficients, which means this is simply the original function just using a new variable. The coefficients for $x(t)$ are computed so $x(t) = t$. The complete parametric representation is given by pairing the two sets of coefficients as $\mathbf{c}_i = (t_i^*, c_i)$ and write

$$(x, y) = \sum_{i=1}^{n} \mathbf{c}_i b_i(t). \tag{4.22}$$

Because of the properties from equation 4.9 and 4.10, the pair (x, y) is a convex combination of the points $\mathbf{c}_i$ and thus completely contained inside the convex hull of these. This will be useful when creating convex relaxations and will be discussed further in chapter 6.

4.4 Control Points

The coefficients in the B-spline are often called *control points* in the B-spline literature. The distinction between coefficients and control points is a bit blurry. It seems that the term control point is used when the surface is described as a parametric curve. In this thesis the term coefficient will be used when referring to the c_i in a function representation (equation 4.19) and the term control point when referring to the $\mathbf{c}_i$ in the parametric representation (equation 4.22) and they are related by

$$\mathbf{c}_i = \left[t_i^{*T}, c_i\right]^T. \tag{4.23}$$

The t_i^*s are vectors with the same dimension as the function domain,

$$t_i^* = \left[t_{i,1}^*, \ldots, t_{i,d}^*\right]^T, \tag{4.24}$$

where the components are completely determined by the knot sequences,

$$t_{i,j}^* = \frac{t_{i,j+1} + \cdots + t_{i,j+k-1}}{k-1}. \tag{4.25}$$

This is sometimes referred to as *knot averages*. The coefficients c_i are found by a set of equations in the same way it was described in chapter 3. Using the simple, regular knot sequences described above, the coefficients are given by

$$\bar{B}c = y \tag{4.26}$$

where the components are built using equation 3.8, 3.9 and 3.10. The t_i^* can be found using the same set of equation, but replacing the right hand side with the matrix

$$X = \left[x_1, x_2, \ldots, x_n\right]^T, \tag{4.27}$$

which is built similarly to the vector y except by using the x_i component of the data samples rather than the y_i component (this would produce the same values as equation 4.25). The cubic spline with knots and coefficients found this way is know as the *natural spline*. They are the interpolating function with smallest second derivative, in the sense that they minimize the integral of the square of the second derivative [Lyche & Morken, 2004].

When a knot sequence has k equal knots, $t_I = \cdots = t_{I+k-1} = a$, the basis function whose support begins at a will evaluate to one at a. That is, $B_I^k(a) = 1$ and all other basis function will be zero. In addition to this, the t_I^* from equation 4.25 will be equal to a. This means the function interpolates one of its control points at the points where a knot is repeated k times. With regular knot sequences this means the surface interpolates the control points at the boundary of the function support.

4.5 B-spline Derivatives

The smoothness of a B-spline is governed by the order of the basis functions and the multiplicity of the elements in the knot sequences. If m is the number of times a value x occurs in the knot sequence, then the B-spline is differentiable $k-m$ times at x. Given a univariate k-th order B-spline the r-th derivative is given by a new B-spline with basis functions of order $k-r$; defined on the same knot sequence, but excluding the r first and last knots:

$$\frac{d^r}{dx^r} y = \sum_{i=r}^{n} \mathbf{c}_i^{(r)} B_i^{k-r}(x), \tag{4.28}$$

with the control points $c_i^{(r)}$ defined by:

$$c_i^{(j)} = \begin{cases} (m-j+1)\frac{c_i^{(j-1)}-c_{i-1}^{(j-1)}}{\left(t_{i+k-j+1}-t_i\right)}, & \text{if } \left(t_{i+k-j+1}-t_i\right) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.29}$$

$$c_i^{(0)} = c_i. \tag{4.30}$$

For multivariate B-splines the derivatives are computed by the same scheme applied to one variable at the time fixing all other indexes in turn.

4.6 Knot Insertion

Knot insertion is a way to refine the existing knot sequence by adding new knots to it—and by that adding new basis functions to the B-spline. This will not alter the function, only its representation. The piecewise polynomial is the same, but represented in a new basis which spans a larger space and has the original space as a subspace. When inserting a new knot, the control points near the new knot has to be recalculated. This recalculation has the neat property that the adjusted control points will be closer to the function surface. How the control points converge to the surface is discussed in [Cohen & Schumaker, 1985]. The distance between the control points and the surface is related to the distance between the knots and the second derivatives.

There are several algorithms available for knot insertion and they have various pros and cons. The interested reader is referred to [Lyche et al., 1985] for a comparison of the most common algorithms. For this project the Boehm knot insertion scheme is used because of its simplicity. Given a knot sequence T and a new knot value $\hat{t}$ such that $t_I < \hat{t} \leq t_{I+1}$, the new knot sequence $\hat{T}$ will have elements given as

$$\hat{t}_i = \begin{cases} t_i, & \text{if } i \leq I \\ \hat{t}, & \text{if } i = I+1 \\ t_{i-1}, & \text{if } i > I+1 \end{cases} \tag{4.31}$$

which is basically just inserting the new value in sorted order. The new control points are then calculated as

$$\hat{c}_i = \begin{cases} c_i, & \text{if } i \leq I - k + 1 \\ \frac{\hat{t}-t_i}{t_{i+k-1}-t_i} c_i + \left(1 - \frac{\hat{t}-t_i}{t_{i+k-1}-t_i}\right) c_{i-1}, & \text{if } I - k + 2 \leq i \leq I \\ c_{i-1}, & \text{if } I + 1 \leq i \end{cases} \tag{4.32}$$

For more details about the Boehm scheme see [Boehm, 1980]. For the multivariate case the univariate procedure is applied in each dimension by fixing the indexes of the other variables in turn.

4.7 Convex B-splines

As with all polynomials there is no trivial test that will tell whether a general B-spline is convex or not. Unfortunately no efficient convexity test was found for general multivariate B-splines during the literature study. The ones suggested are very restrictive and generate a lot of false negatives. In one or two dimensions there are some criteria that are sufficient to prove convexity and some of these are "asymptotically necessary", meaning that they aim to generate a set of inequalities that has to be satisfied, and the number of inequalities can be increased to reduce the number of false negatives towards zero.

The problem with the B-spline convexity tests are that the second derivatives that make up the Hessian are represented by a set of B-splines, all using sightly different basis functions and they are not directly comparable. In the one dimensional case with cubic splines the Hessian is simply a piecewise linear spline and it is trivial to argue for convexity by inspecting the coefficients and see that they are all non-negative. In the two dimensional case the articles found on the subject attempts to rewrite the equations to a form where the Hessian components can be related to each other, see for instance [Floater, 1993] for details on this procedure. But, as said, no general B-spline method was found that seemed worth perusing further with the limited time available. It would, of course, be possible to apply more general methods to test for convexity as a last resort.

4.8 Implementation

Tensor product B-spline functions has been implemented in C++ and is now considered a part of the code project that was developed as part of my project last semester. The implementation includes all functionality needed to read data from a specified file, construct a suitable set of knot sequences and compute the control points for a natural tensor B-spline automatically. The B-spline objects can evaluate function values and derivatives. Knot insertion is done with the Boehm algorithm mentioned above. The B-spline part of the code project consist of approximately 1000 lines of code. The B-spline implementation has been tested for linear and cubic basis functions with the knot sequence layouts described in section 4.1.

It should be mentioned that initially the plan was to find a open source library for tensor produce B-splines and just use this directly. However, after spending quite some time searching for such codes without finding anything that supported the desired functionality, it was concluded that it would be easier to just implement it ourself.

4.9 Chapter Summary

The tensor product B-spline approximation methods has been introduced. It is based around piecewise polynomials and is represented by well behaved basis functions. The tensor product B-spline has all the approximating power and flexibility needed to approximate the black-box functions in our problem. This comes at the price of assuming a strict grid structure on the data sets, which, for the time being, is a price we are willing to pay.

Basic B-spline functionality was successfully implemented in C++. It seems to perform well in terms of representing the underlying functions, but it should be noted that the natural B-spline is not shape preserving and will not remain strictly positive even if all data samples are positive, as will become evident when looking at the results of some of the test problems. In this thesis this is not really going to cause any headache, since the interesting part here is really just to see if it is possible to use B-splines in a global optimization setting. Putting shape preserving properties on the coefficients could be done afterwards if that is desired. Chapter 7 will show some results on B-splines paired with simulator data.

Chapter 5

Flow Network Optimization

This chapter will go through the basics of flow network optimization from a oil production point of view. First the classic flow network problem will be introduced. Then this will be gradually augmented until we arrive at the oil production problem. The final model is supposed to be a sufficient but simple representation of oil and gas flow networks. The model scope begins were fluid flows leaves the oil-well accompanied by a certain pressure and ends when the flow reaches the separator level top side. Between the wells and separators we find a network of pipelines and connections. It is this we refer to as the flow network. Models for the oil production problem has been developed in several papers, see for instance [Kosmidis et al., 2005] or [Gunnerud & Foss, 2010].

There are many variables that will affect how fluids travel through the network. The goal here is to develop a model that allows us to decide how to maximize the amount of oil that is able to pass though the network over a short future time horizon. To achieve this maximization it is necessary to model mass transfers through the system, and the model is thus based around the first principle of mass conservation. The mass balance equations developed are of the simplest kind and no phase transitions are allowed. The driving force for mass transfer is pressure, so a momentum balance is therefore required as well. All other effects (*e.g.* temperature and energy balance) are neglected and assumed to be constant at this point. All modelling is done assuming standardized conditions for flow rates ($15°C$ and $101.325 kPa$).

The system is viewed in a intermediate time scale were non of the system components display any dynamic behaviour. The reservoirs and separators appear as constant based on the time scale argument that they have a large capacity relative to the flow rates and will therefore not exhibit any detectable changes over short time periods. The pipes and manifolds are assumed to be at "steady state" based on the time scale argument that they have small capacities relative to the mass transfer per time unit, and will there for quickly reach equilibrium and display event dynamics.

5.1 Introduction to Flow Networks

The Classic Flow Network Problem

The classic case of flow network optimization is a famous problem where the goal is to find the maximum flow from a source to a sink node when all the connections between them have capacity constraints. An example of a flow network is given in figure 5.1. Formally the classic flow network problem is stated as: let $N = (V, E)$ be a network graph with edges E and vertices V. Let $s \in V$ be the source and $t \in V$ be the sink. All edges $(u, v) \in E$ has a non-negative capacity $c_{u|v}$ and if the nodes u and v are not connected we have $c_{u|v} = 0$. The flow $f_{u|v}$ across an edge must be less than the capacity of the edge. Flows have directions, which means $f_{u|v} = -f_{v|u}$. Flow must be conserved trough the network, which means the flow in and out of all vertices must be equal. The only exception being the sink and source nodes. This is stated as

$$\sum_{v \in V} f_{u|v} = 0 \quad \forall u \in V/\{s, t\} \tag{5.1}$$

and is referred to as the mass balance equations. The goal is to maximize the flow through the network, which for instance can be measured as the flow entering the sink,

$$\sum_{u \in V} f_{u|t}. \tag{5.2}$$

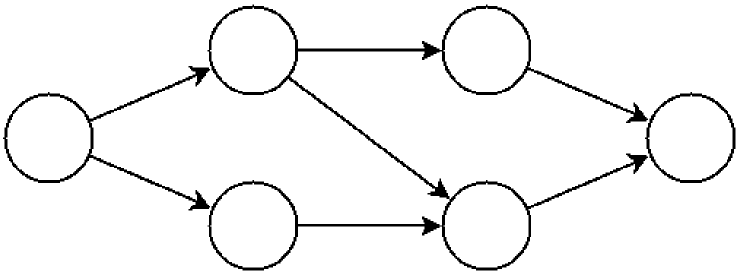

Figure 5.1: A small flow network with six vertices and seven edges.

The problem can now be stated as

$$
\begin{aligned}
\text{maximize} \quad & \sum_{u \in V} f_{u|t} \\
\text{subject to} \quad & f_{u|v} \leq c_{u|v} \quad \forall (u,v) \in E \\
& f_{u|v} = -f_{v|u} \quad \forall (u,v) \in E \\
& \sum_{v \in V} f_{u|v} = 0 \quad \forall u \in V / \{s,t\},
\end{aligned}
\tag{5.3}
$$

which could be cast into the standard optimization problem. It would turn out to be an LP problem. There are algorithms tailored to solve this problem in polynomial time by exploiting the specific problem structure, an example being *Edmonds-Karp* [Cormen et al., 2009].

Unfortunately the oil production problems does not simplify all the way down to this classic case. It does for instance include routing decisions and non-linear pressure-flow relations. Since the final problem cannot be solved using a solver that assumes a single source and sink in the graph, the formulation will be extended to allow multiple source and sink vertices (this is technically not necessary, but is simplifies the rest of the modelling procedure). Labelling the set of source vertices as $V_s \subset V$ and the set of sink vertices as $V_t \subset V$, the mass balance is reformulated to

$$\sum_{v \in V} f_{u|v} = 0 \quad \forall u \in V / V_s \cap V_t \tag{5.4}$$

and the objective to

$$\sum_{t \in V_t} \sum_{v \in V} f(v,t). \tag{5.5}$$

We will also take the liberty to remove the skew-symmetric flows ($f_{u|v} = -f_{v|u}$), since our solver will not need those. The negative flows are only used to state the mass balances as a single sum for each node. Introducing the sets $V_U(u)$ and $V_D(u)$ which are sets representing the vertices that are connected to the vertex u upstream and downstream respectively. They are defined by

$$V_D(u) = \{v \mid v \in V \wedge c_{u|v} > 0\}, \tag{5.6}$$

$$V_U(u) = \{v \mid v \in V \wedge c_{v|u} > 0\}. \tag{5.7}$$

With these sets, the mass balance can be rewritten to

$$\sum_{v \in V_U(u)} f_{u|v} - \sum_{u \in V_D(u)} f_{u|v} = 0 \quad \forall u \in V / V_s \cap V_t. \tag{5.8}$$

This intuitively reads: the sum of upstream flows must be equal to the sum of downstream flows. We now arrive at at slightly modified optimization problem that is equivalent to the original flow network problem, but on a format more suitable for our solver methods:

$$\begin{aligned} \text{minimize} \quad & -\sum_{v \in V_t} \sum_{u \in V} f_{u|v} & \\ \text{subject to} \quad & f_{u|v} \le c_{u|v} & \forall (u,v) \in E \\ & \sum_{v \in V_U(u)} f_{v|u} - \sum_{v \in V_D(u)} f_{u|v} = 0 & \forall u \in V / V_s \cap V_t. \end{aligned} \tag{5.9}$$

The Flow Network Decision Problem

In the classic formulation of the flow network problem there is no restrictions on how the flow is distributed among the edges. In the decision problem an edged has to be selected before it can have a non-zero flow. The selection is denoted $s_{u|v}$ and this is equal to one if the edge (u,v) is chosen and zero if not. Flow is now restricted by

$$f_{u|v} \le c_{u|v} s_{u|v}, \tag{5.10}$$

which says that an edge must both have a non-zero capacity and be selected before the flow can be non-zero. There are restrictions on the selection procedure. Some choices can be mutually exclusive. In our case this mutual exclusion is that for vertices with multiple leaving edges, only *one* can be selected. That is,

$$\sum_{v \in V_D(u)} s_{u|v} \le 1, \quad u \in V. \tag{5.11}$$

A valid combination of selections is one that turns the flow network into a tree (or forest) structure with flows flowing from the leaf nodes towards the root node(s). The flow network decision problem could be cast into a MILP-problem where the goal is to find the tree structure within the flow network that provides the maximum flow. This is still not sufficient to describe the oil production problem. The last step is to introduce a driving force that drives the flow from the source and towards the sink. This driving force is in our case unfortunately non-linear.

The Potential Driven Flow Network Decision Problem

Adding a driving force to the flow problem is done by introducing a non-negative potential at each vertex. The potential at vertex u is labelled p_u. Each edge is assigned a function $\Delta p_{u|v}\left(f_{u|v}\right)$ that dictates how much potential is lost over an edge as a function of the flow rate. The potential must be conserved so the loss over an edge must be equal to the difference in potential at the vertices it is connected to. This is written as

$$p_u - p_v = \Delta p_{u|v}\left(f_{u|v}\right). \tag{5.12}$$

The conservation is only relevant when the edge is selected making the final equation

$$\left(p_u - p_v - \Delta p_{u|v}\left(f_{u|v}\right)\right) s_{u|v} = 0, \quad \forall (u,v) \in E. \tag{5.13}$$

The potential at a source can either be fixed or related to the leaving flow. Here it will be modelled so the leaving flow is a function of the source potential by introducing a function $f_u(p_u)$ for each source. A flow can only be non-zero if the edge is selected, so this function must be multiplied by the corresponding selection variable,

$$f_{u|v} = f_u(p_u) s_{u|v} \, \forall v \in V_D(u), \, u \in V_s. \tag{5.14}$$

To summarize, the complete problem in now given as

$$
\begin{aligned}
\text{minimize} \quad & -\sum_{v\in V_t}\sum_{u\in V} f_{u|v} & & \\
\text{subject to} \quad & f_{u|v} \le c_{u|v} s_{u|v} & & \forall (u,v)\in E \\
& \sum_{v\in V_U(u)} f_{v|u} - \sum_{v\in V_D(u)} f_{u|v} = 0 & & \forall u \in V / V_s \cap V_t \\
& f_{u|v} = f_u(p_u) s_{u|v} & & \forall v \in V_D(u),\, u\in V_s \\
& \left(p_u - p_v - \Delta p_{u|v}\left(f_{u|v}\right)\right) s_{u|v} = 0 & & \forall (u,v)\in E \\
& \sum_{v\in V_D(u)} s_{u|v} \le 1 & & u \in V.
\end{aligned}
\tag{5.15}
$$

This problem will serve as a frame for the oil production problem that is formulated in the next section.

5.2 Oil Production Flow Networks

This section will formulate the oil production problem using the nonlinear flow network framework. This will be done using the Marlim test case as an example. The Marlim problem is visualized in figure 5.2. It is a case with twelve wells and two riser pipelines. Six wells are topside wells, which means they are connected directly to the separator. The other six are subsea wells and they are connected to a manifold. This manifold has the ability to route each well individually to one of two riser pipelines. This model was used by fellow students Stine Ursin-Holm and Sheri Shamlou in their project last semester and the solution to this problem is known [Shamlou & Ursin-Holm, 2012]. This thesis will use the same model and data sets.

Before the actual modelling begins, it should be pointed out that the network flow problem as it is described in the previous section defines a lot of unnecessary variables and constraints. Each vertex is theoretically connected to all other vertices, but only a sparse subset of these connections will have non-zero capacities. This means that many equations will end up as trivial $0 = 0$ constraints and many variables will be forced to be constants. These are obviously ignored when the problem is solved. The reason the problem is formulated this way is that it is really difficult to sufficiently describe all possible sparse topologies without introducing a lot of different index sets to cover all the special cases.

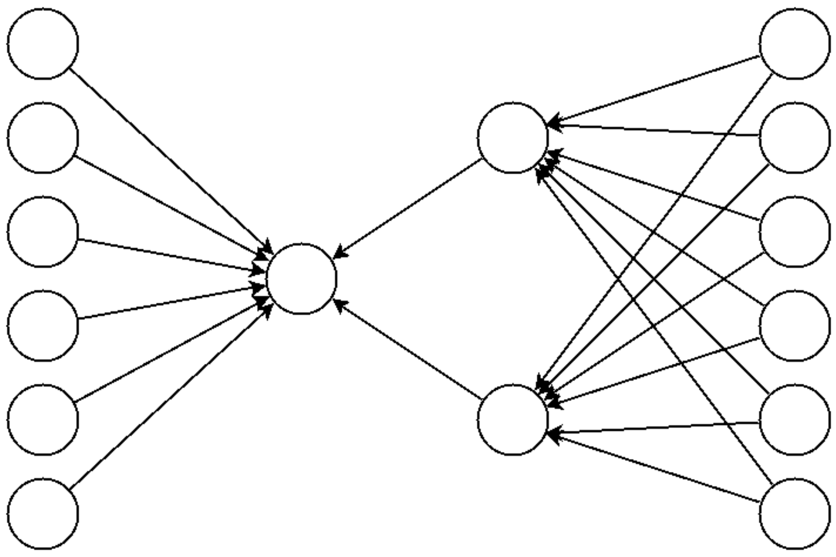

Figure 5.2: The Marlim flow network. There are 12 wells , 2 pipelines and 1 separator. The graph has 15 vertices and 20 edges. The six left hand side wells are the topside wells and they are connected directly to the separator. The six right hand side wells are the subsea wells and they are connected to a manifold and the manifold is connected to two riser pipelines. The manifold has been divided into two vertices, one for each pipeline. Each subsea well is connected to both manifold vertices, but only one of these connections can be selected at a time.

Model Topology

The flow network problem is based around a graph representation of the topology. The topology of the Marlim field is given in figure 5.2. The graph has 15 vertices and 20 edges. Of these, 18 edges have routing and two have pressure drop equations.

In addition to the notation introduced in the previous section, it will be useful to have a set containing only the edges connected to a source vertex and a set containing only the edges connected to a sink vertex. Starting with the source edges, denote the set by E_s and let it be given by

$$E_s = \{(u,v) \mid (u,v) \in E \wedge u \in V_s\}. \tag{5.16}$$

The sink edges are denoted by E_t and is given by

$$E_t = \{(u,v) \mid (u,v) \in E \wedge v \in V_t\}. \tag{5.17}$$

Mass Balance and Flow Equation

The mass balance is for the most part identical to the one stated in the framework above. The changes made here is to introduce multiphase flow and gas lift and to simplify the relationships between flow and pressure at the wells.

The fluid flowing in the oil production network is made up of several different phases. In the Marlim model the phases are oil, gas and water, which are labelled q^o, q^g and q^w respectively. They are related to the network edge flows by

$$f_{u|v} = \left[q^o_{u|v},\ q^g_{u|v},\ q^w_{u|v} \right]^T. \tag{5.18}$$

In the flow network all flows $f_{u|v}$ are non-negative and this serves as the lower bound on the variables in the optimization problem. The edge capacity $c_{u|v}$ will be the upper bound. The actual values for the capacities are taken as the maximum values indicated by the simulator data sets. If no such maximum exists, the bound is set high enough to not interfere with the solution while still being as low as possible (only to limit the optimization search space). It should be noted that the riser pipelines in the Marlim case have a non-zero lower bound on their flow rates.

In the model above the flow rate leaving a source was given as a function of pressure. In the oil production case it is possible to inject additional gas into the well to increase the flow. The gas lift rate for well u is q_u^{gl}. The pressure-flow relationship is augmented to include the gas lift,

$$f_{u|v} = f_u\left(p_u, q_u^{gl}\right) s_{u|v}, \forall\,(u,v) \in E_s. \tag{5.19}$$

The gas and water rates are linear in the oil rate, so this expression can be separated to three simplified equations. These equations are

$$q_{u|v}^{o} = f_u^{o}\left(p_u, q_u^{gl}\right) s_{u|v} \qquad \forall\,(u,v) \in E_s, \tag{5.20}$$

$$q_{u|v}^{g} = \left(k_u^{g} q_{u|v}^{o} + g_u^{gl}\right) s_{u|v} \qquad \forall\,(u,v) \in E_s, \tag{5.21}$$

$$q_{u|v}^{w} = \left(k_u^{w} q_{u|v}^{o}\right) s_{u|v} \qquad \forall\,(u,v) \in E_s, \tag{5.22}$$

where k_s^g and k_s^w are given constants for each well. Multiplying the water equation with the selection variable is redundant since the oil flow will always be zero when the selection variable is zero. For the gas equation is it required because the gas lift is possibly non-zero, as it is shared among all routing alternatives.

There is a limit on the total gas lift capacity. Denoting this limit by C_{GL}, the limitation written as

$$\sum_{(u,v)\in E_s} q_u^{gl} s_{u|v} \leq C_{GL}. \tag{5.23}$$

This formulation will in our case include some gas lift variables twice, but atleast one of these will be multiplied by zero. The reason for this formulation is that some wells have a non-zero lower bound on their gas lift usage, meaning that if the well is on it must also use a certain amount of gas lift to function properly. If the well is turned of, the optimization variable will remain non-zero, but by multiplying with the selection variable it will not be penalised in the constraint above.

For the Marlim case these are also limitations on the total gas and water production at the separator level. Let C_W be the water capacity and C_G be the gas capacity. The restrictions are then given as

$$\sum_{(u,v)\in E_t} q_{u|v}^{g} \leq C_G \tag{5.24}$$

for the gas production and

$$\sum_{(u,v)\in E_t} q_{u|v}^{w} \leq C_W \tag{5.25}$$

for the water production.

In the original flow problem the objective is to maximize all flow, since it only operates on single phase flows. The oil production problem has three different phases and only one is interesting to maximize. The objective is altered slightly to accommodate this,

$$\sum_{(u,v)\in E_t} q^o_{u|v}. \tag{5.26}$$

Routing Options

The decisions problem stated above had the ability to turn all edges on and off. In the oil production case only a few of these choices actually makes sense. In the Marlim case only the edges that are connected to a well has valid routing options. All other edges will have their selection fixed to 1. This is done by setting both the upper and lower variable bound to 1. A variable that is restricted to only one value is treated as a constant in the optimization problem.

As the Branch-and-Bound solver iterates, the other selection variables will eventually be fixed in the same manner, but then to either 0 or 1 depending on the branch. In the Marlim case there are 18 on-off selections. The six topside wells have two possible settings each and the six subsea wells have four possible settings each. This gives a total of $2^6 4^6 = 2^{18} = 262144$ different routing combinations. Only $2^6 3^6 = 46656$ of these are actually routing feasible, the others failing to satisfy equation 5.11. Since the pipelines are modelled to require a minimum flow rate, many of these combinations will also end up being infeasible. In all there are 38528 feasible routing combinations.

Momentum Balance

The momentum balance is almost the same as the driving force equations outlined in section 5.1. The difference is that the edges connected to a well do not have pressure drop as a function of flow, but given as a non-negative choke variable. This choke variable is basically there to allow for additional pressure drop if this is desired.

For the pipeline edges we keep the original pressure drop equation:

$$\left(p_u - p_v - \Delta p_{u|v}\left(f_{u|v}\right)\right) s_{u|v} = 0, \quad \forall\,(u,v) \in E/E_s. \tag{5.27}$$

For the source edges the original equation is altered slightly. A choke variable $\Delta p^c_{u|v}$ is introduced and it takes the place of the original pressure drop term,

$$\left(p_u - p_v - \Delta p^c_{u|v}\right) s_{u|v} = 0, \quad \forall\,(u,v) \in E_s, \tag{5.28}$$

combined with the non-negativity restriction $\Delta p^c_{u|v} \geq 0$.

For the source vertices, bounds on the pressure variables are taken equal to the variable range used in the data sets. For the other vertices the bounds are taken to be the lowest lower bound and highest upper bound from the sources.

Simulators and Approximations

The nonlinear flow-pressure relationships for the wells and pipelines are given by simulators. Data sets from the simulators are provided, but the simulators themselves have not been available. The data sets are used to create interpolating tensor product B-splines that will replace the simulator functions in the problem formulation. A selection of approximation results is given in Chapter 7.

The well data is given in a 12×12 grid structure. The variable ranges vary with the individual wells. The size of these data sets are small enough to be comfortably dealt with. No reduction of sample size was attempted, but for some wells it would have been possible to remove almost half of the samples since the function surfaces have large areas with almost no curvature. Some wells are able to produce at zero flow, but most wells have a non-zero minimum flow rate they must produce at if the well is active. Some wells also have a non-zero minimum gas lift usage that is required if the well is active.

The pipeline data is given in a $30 \times 30 \times 30$ grid structure. Both pipelines using the same data sites. These data sets contain 27000 samples. This is not impossible to manage, but it is definitely a noticeable preprocessing job when the amount of data reach these numbers. Some samples were removed to see the effect this had on the approximations. This did not degrade the approximation significantly in the regions of interest. The final data sets used a $18 \times 18 \times 18$ subset of the original data set, but additional samples could easily have been removed if we where willing to sacrifice some accuracy in the regions with large second derivatives (this is typically on the boarder of the function domain where the simulator produces some oscillating results that are impossible to capture without all the samples).

5.3 Chapter Summary

The complete oil production model was derived by gradually augmenting and adjusting the classic optimization problem of maximizing network flow. The Marlim case is used as a test problem for this thesis. The model and problem components are summarized below. The index sets used are summarized in table 5.1 for quick reference.

$$
\begin{array}{lll}
\text{minimize} & -\sum\limits_{(u,v)\in E_t} q^o_{u|v} & \\
\text{subject to} & f_{u|v} \le c_{u|v} s_{u|v} & \forall (u,v)\in E \\
& \sum\limits_{v\in V_U(u)} f_{v|u} - \sum\limits_{v\in V_D(u)} f_{u|v} = 0 & \forall u \in V / V_s \cap V_t \\
& q^o_{u|v} = f^o_u\left(p_u, q^{gl}_u\right) s_{u|v} & \forall\,(u,v)\in E_s, \\
& q^g_{u|v} = \left(k^g_u q^o_{u|v} + g^{gl}_u\right) s_{u|v} & \forall\,(u,v)\in E_s, \\
& q^w_{u|v} = \left(k^w_u q^o_{u|v}\right) s_{u|v} & \forall\,(u,v)\in E_s, \\
& \left(p_u - p_v - \Delta p^c_{u|v}\right) s_{u|v} = 0, & \forall\,(u,v)\in E_s \\
& \left(p_u - p_v - \Delta p_{u|v}\left(f_{u|v}\right)\right) s_{u|v} = 0 & \forall (u,v)\in E/E_s \\
& \sum\limits_{v\in V_D(u)} s_{u|v} \le 1 & u \in V \\
& \sum\limits_{(u,v)\in E_s} q^{gl}_u s_{u|v} \le C_{GL} & \\
& \sum\limits_{(u,v)\in E_t} q^g_{u|v} \le C_G & \\
& \sum\limits_{(u,v)\in E_t} q^w_{u|v} \le C_W. &
\end{array}
\tag{5.29}
$$

Description	**Symbol**	**Definition**
All edges	E	-
All vertices	V	-
Vertices connected to u downstream	$V_D(u)$	$\{v \mid v \in V \wedge c_{u\mid v} > 0\}$
Vertices connected to u upstream	$V_U(u)$	$\{v \mid v \in V \wedge c_{u\mid v} > 0\}$
Source vertices	V_s	$\{v \mid v \in V \wedge V_U(u) = \emptyset\}$
Sink vertices	V_t	$\{v \mid v \in V \wedge V_D(u) = \emptyset\}$
All edges leaving a source	E_s	$\{(u,v) \mid (u,v) \in E \wedge u \in V_s\}$
All edges entering a sink	E_t	$\{(u,v) \mid (u,v) \in E \wedge v \in V_t\}$

Table 5.1: Index sets used in the Marlim model formulation. The topology graph is given in figure 5.2.

In order, the constraints represent: flows restricted by selection, mass balance at all internal vertices, oil-, gas- water- and momentum relations for each well, momentum equation for each pipeline, routing restrictions, gas lift capacity, gas production capacity and water production capacity.

Chapter 6

B-spline Branch and Bound

The idea behind the Branch-and-Bound was explained in Chapter 2 and this chapter will go into the details of the algorithm that was implemented and used for this thesis. The algorithm will accept problems on the form

$$
\begin{aligned}
\text{minimize} \quad & c^T x \\
\text{subject to} \quad & c_i(x) \leq 0, i \in I \\
& c_i(x) = 0, i \in E \\
& x_i \in \mathbf{Z}, i \in Z \\
& x \in X.
\end{aligned}
\tag{6.1}
$$

It is assumed that the problem has a linear objective. If the problem does not have a linear objective, it must be rewritten to the epigraph formulation by introducing a new variable and a new constraint as explained in Chapter 2. Assuming a convex objective like this is only to simplify the convex relaxation process, since only constraints have to be considered in the implementation. All functions are assumed to be twice differentiable. Affine functions are represented using matrices and vectors. Non-linear functions are either represented by special terms, such as bilinear, or by tensor product B-splines. The solver is a fairly standard branch and bound implementations, and the only unique element is the B-spline representation, which is a approximation technique that allows us to transfer the problems from a black-box (oracle) description to a parameter description that can be manipulated and relaxed conveniently.

6.1 Description of the Algorithm

Pseudo code for the Branch-and-Bound framework is given in algorithm 1. Recall from Chapter 2 that the Branch-and-Bound progress can be interpreted as a tree structure. The algorithm maintains a global best solution f^* and a global lower bound $\bar{f}$ and iterates until these are acceptably close to each other. Each iteration will first select a leaf node to process and find a lower bound $\bar{f}_i$ on the objective value for this node. If the lower bound is worse than the best known solution, the node is simply removed from the list of interesting nodes and this branch of the tree is considered completed. If the lower bound is better that the best known solution, a local solution f_i^* is found. If f_i^* coincide with $\bar{f}_i$ this branch of the has converged and no further exploration is required, otherwise the node is divided into two child nodes and the child nodes are placed in the list of leafs that needs to be processed. The key steps in the algorithm are explained in the following sections.

Select Leaf Node

Various heuristics exist for choosing the next leaf node to be processed. Classic three-traversing methods like *Depth-First* and *Breadth-First* are among the simplest methods. Other methods are variations of a *Best-First* type of scheme where "best" is measured by some property on the parent node *e.g.* lowest lower bound or lowest upper bound. Heuristic testing is not the topic for this thesis, so a simple best-first selection scheme is used based on the parent lower bound. This should give a steady progress for the lower bound iterates, without requiring too complicated implementations.

Algorithm 1 Branch-and-Bound Algorithm

```
Set $f^* \leftarrow \infty$
Set $\bar{f} \leftarrow -\infty$
Set leaf node indices $I \leftarrow 1$
repeat
    $N_i \leftarrow$ select leaf node
    $\bar{f}_i \leftarrow$ solve lower bound problem for $N_i$
    if $\bar{f}_i > f^*$ then
        Fathom $N_i$
    else
        $f_i^* \leftarrow$ solve upper bound problem for $N_i$
        if $f_i^* < f^*$ then
            $f^* \leftarrow f_i$
        end if
        if $f_i^* \leq \bar{f}_i + \epsilon$ then
            Fathom $N_i$
        else
            Branch and add child nodes to leaf node indices
        end if
    end if
    $I \leftarrow I / \{i\}$
    Set $\bar{f} \leftarrow \min\{\bar{f}_i, i \in I\}$
until $f^* \leq \bar{f} + \epsilon$
```

Solve Lower Bound Problem

The lower bound problem at node N_i is given by

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & \bar{c}^i_j(x) \leq 0, j \in I \\ & \bar{c}^i_j(x) = 0, j \in E \\ & x \in X_i. \end{aligned} \tag{6.2}$$

It is produced by creating relaxed versions of all the non-linear constraints and removing the integer constraints. All B-spline constraints are relaxed, since the algorithm currently don't check whether a B-spline inequality is convex or not. This is obviously a weakness in the algorithm, and will hurt the convergence, but due to limited time such features were ignored. The consequences of this is discussed later. Of the nonlinear functions with special structure, only bilinear terms have been used in this thesis. They are treated using the best possible relaxation, which was given in Chapter 2. B-spline constraints are relaxed using the convex hull of the control points. This is explained in Section 6.2. Other constraints are either linear or have special structure. The only special structure term used in this thesis is the bilinear terms in the network flow model. Bilinear terms are relaxed using the constraints from equations 2.25-2.28. It is interesting to note that if the bilinear term had been represented as a second order B-spline using a sample at each corner of the variable bounds, then the B-spline would be a perfect approximation (bilinear basis representing a bilinear function) and the relaxation of this B-spline would be identical to the exact relaxation. This is because the B-spline would interpolate the four control points and the relaxation is the convex hull of these. It would therefore be possible to achieve the same performance using only B-splines.

Solve Upper Bound Problem

The upper bound problem at node N_i is given by

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & c_j^i(x) \leq 0, j \in I \\ & c_j^i(x) = 0, j \in E \\ & x_j = k_j^i, j \in Z \\ & x \in X_i. \end{aligned} \tag{6.3}$$

It is produced by fixing the integer variables to a integer inside the problem domain, making them constants instead of variables in the problem. These constants could be chosen in various ways and many of these choices could easily lead to infeasible problems. In this implementation they are set based on the lower bound solution. When solving a oil production problem it would obviously be possible to exploit the routing structure of the flow network when selecting these constants, but such heuristics will not be explored here.

Branching

The main problem here to find a suitable rule for selecting the branching variable. The branching rules used in this implementation are rather simple. Integer variables are given priority and the variable violating the integer constraint the most is chosen among these. When no integer variables are left, the branching process moves on to the continuous variables. Among the continuous variables, the variable with the largest distance between the upper and lower bound is chosen. This is not a very sophisticated branching strategy and there are other strategies could have been applied. The motivation for using this strategy is that it is simple and predictable, which allows for a easier analysis of the results later. This is because the quality of the B-spline relaxation is closely connected to the distance between the knots in the knot sequences and this distance is directly linked to the variable range (this is illustrated in the next section).

6.2 Convex Relaxation of B-spline Constraints

Recall from Chapter 4 that a surface described by B-splines could be written as the parametric curve

$$(x, y) = \sum_{i=1}^{n} \mathbf{c}_i b_i(t) \tag{6.4}$$

and that the basis functions have the properties

$$1 = \sum_{i=1}^{n} b_i(t) \tag{6.5}$$

and

$$0 \le b_i(x) \le 1. \tag{6.6}$$

The pairs (y, x) is a convex combination of the control points $\mathbf{c}_i$ and will be contained inside the convex hull of these points. A convex relaxation can be made by allowing *all* convex combinations of $\mathbf{c}_i$. This can be done by introducing n new variables z_i, $i = 0, \ldots, n$ to replace the basis functions in all three of the equations above. This gives the constraint

$$(y, x) = \sum_{i=1}^{n} \mathbf{c}_i z_i. \tag{6.7}$$

In the original problem the B-spline basis has the property that they sum to one and are always non-negative, but these are not explicitly stated in the problem. In the relaxation it is necessary to include these as the constraints

$$1 = \sum_{i=1}^{n} z_i \tag{6.8}$$

and

$$z_i \in [0, 1], \quad i = 1, \ldots, n. \tag{6.9}$$

It would also be possible to compute the convex hull of the control points as a simplex and express this as a set of linear inequalities in x and y, thus avoiding the introduction of new variables, but this comes at the cost of a significantly larger set of constraints. The algorithm implemented here will use the method with new variables. The convex hull of a B-spline is illustrated in figure 6.1. For the relaxation to be useful it must be a reasonably good approximation of the actual convex hull of the surface and it must also converge to the actual convex hull as the variable range is reduced by branching. This is to make sure local solutions and the lower bounds eventually converge to each other. When parts of the domain of a B-spline is removed by splitting a variable range, some of the basis functions will be unsupported inside the new range. This is because the basis functions have local support. For the univariate basis functions the support is given by half open intervals:

$$\sup B_i^k(x) = [t_i, t_{i+k}). \tag{6.10}$$

For a basis function to remain relevant it must have

$$\sup B_i^k \cap [\underline{x}, \overline{x}] \neq \emptyset. \tag{6.11}$$

Illustrating this, let the monovariable B-spline be given as

$$y = \sum_{i=1}^{n} c_i B_i^k(x) \tag{6.12}$$

with knot sequence

$$T = \{t_1, t_2, \ldots, t_{n+k}\}. \tag{6.13}$$

If the lower bound is increased from $\underline{x} = t_1$ to $\underline{x}^* \in [t_I, t_{I+1})$ the basis functions with index lower than $i - k$ will become unsupported and the B-spline sum can be restricted to

$$y = \sum_{i=i-k}^{n} c_i B_i^k(x). \tag{6.14}$$

The relaxation of this expression will contain fewer control points than the original relaxation, and should thus shrink. But, this will not converge, since no matter how small the interval gets there will always be four nonzero basis functions inside it and the convex relaxation will therefore never get better that these four control points if the original control points are used. This is illustrated in figure 6.2. To guarantee proper convergence, the B-spline will be updated when the domain size is reduced by making use of knot insertion. Knots will be inserted to make sure the knot sequence remains regular, which means that up to k new knots will be added at the value where the domain was cut. By keeping the knot sequence regular it is ensured that the B-spline will interpolate the control points at the domain boundaries. This is shown in figure 6.3. In addition to keeping the knot sequence regular it will also be kept at a minimum number of knots by inserting additional single knots between the existing knots. This is referred to as *knot refinement.* With these strategies in place for updating the B-spline the development of the convex relaxation between parent and child nodes can be seen in figure 6.4.

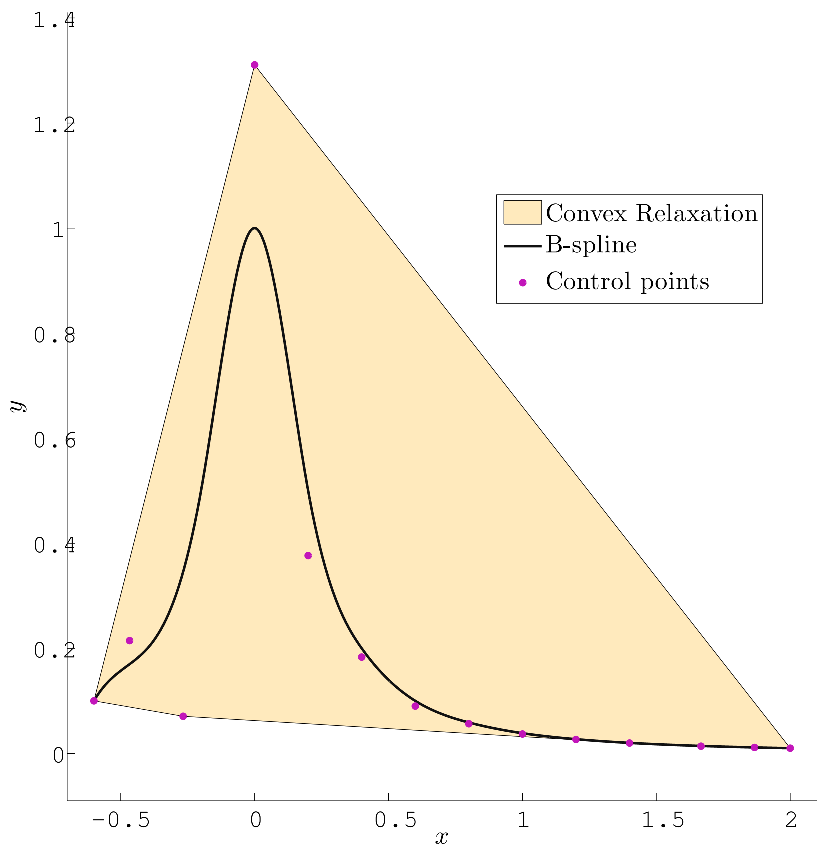

Figure 6.1: A B-spline and the convex hull of its control points. The distance between the curve and the control points is increased when the second derivative of the curve is large.

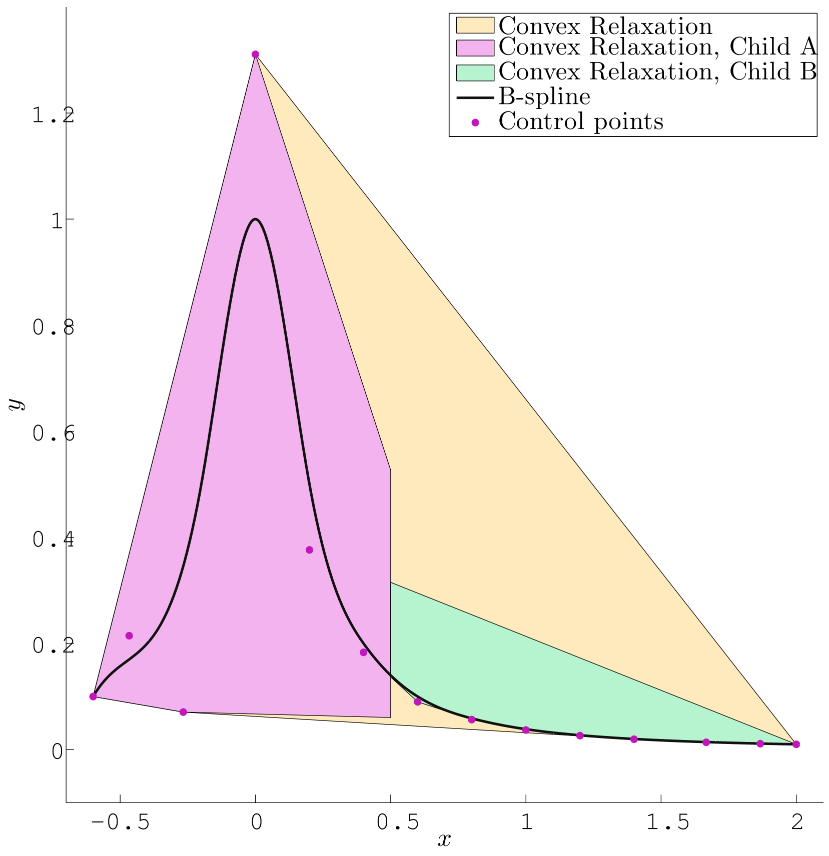

Figure 6.2: The child node relaxations after a cut at $x = 0.5$ when the B-spline representations remains unchanged after the cut. The relaxation is improved, but not significantly due to the fact that there will be a overlap of k control points which must be included in both child nodes. This makes the relaxations unable to interpolate at the endpoints and will eventually halt the convergence completely.

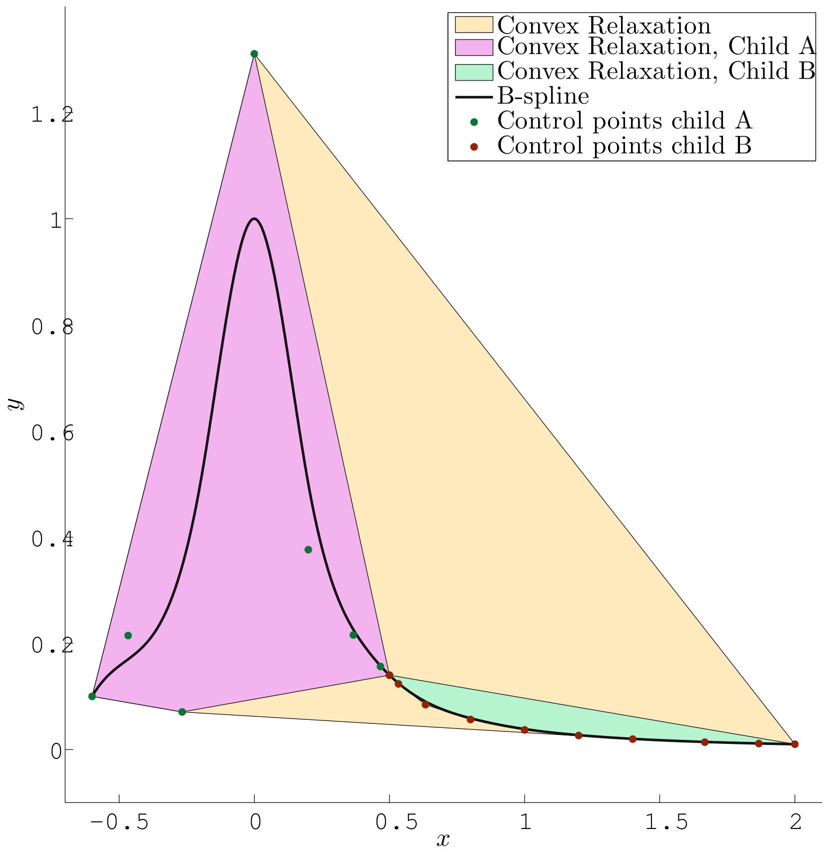

Figure 6.3: The child node relaxations after a cut at $x = 0.5$ when the B-spline representations are updated in the child nodes to have regular knot sequences. This allows relaxations to interpolate at the endpoints and there is no overlap of control points between the two child nodes.

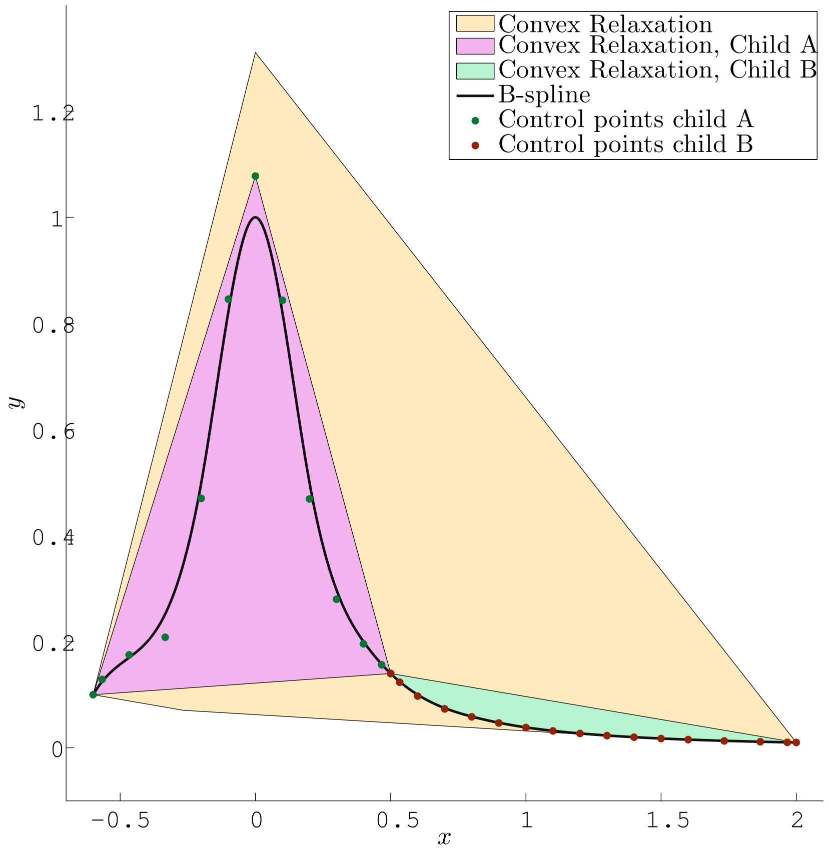

Figure 6.4: The child node relaxations after a cut at $x = 0.5$ when the B-spline representations are updated in the child nodes to have regular knot sequences and the knot sequences are refined to contain a minimum number of knots . The refinement improves the convex hull by drawing all the affected control points closer to the actual surface, but it is not always necessary and in this case only child node A actually benefits from the refinement. The benefit of the refinement is correlated with the magnitude of the second derivatives of the function and the distance between the knots in the original sequence.

In the multivariate case the regularization and refinement process is applied to all points in the grid by iterating trough and fixing the indexes of the unaffected variables while the changes are made as in the monovariable case. The complete domain restriction and update procedure is given in algorithm 2.

Algorithm 2 B-spline Domain Reduction and Knot Refinement

```
  Split box domain for variable $x_i$ at $s$ and refine corresponding knot sequence $T_i$.
  The new domain is either $[\underline{x}_i, s]$ or $[s, \overline{x}_i]$.
Require: $s \in (\underline{x}_i, \overline{x}_i)$
  repeat
    Insert $s$ in $T_i$ using Boehms knot insertion algorithm
  until $s$ occurs $k$ times in $T_i$
  Remove unsupported basis functions
  repeat
    $t_i \leftarrow$ first index of largest interval in $T_i$
    $\tau \leftarrow (t_i + t_{i+1})/2$
    Insert $\tau$ in $T_i$ using Boehms knot insertion algorithm
  until Number of knots in $T_i$ is equal the number of desired knots
```

6.3 Implementation Details

The algorithm is implemented in C++ and is a continuation of the code project that was started last semester. The complete code project is now roughly 10 000 lines of code, of which about 8000 have been relevant for this thesis. The code is written by Bjarne Grimstad (co-supervisor for the thesis) and myself, the work being split fairly even between the two of us. The local solver framework developed during the previous semester was kept mostly unchanged and has now been extended to become a Branch-and-Bound based global solver. This solver supports a selection of nonlinearities with special structure and general nonlinearities represented as tensor product B-splines. My contributions to the code-project during this thesis is mainly related to the B-spline functionality while Bjarne Grimstad has done most of the branch and bound code, but we have both been involved with all project components at some point.

The focus for the implementation has been to produce a solver that can illustrate the suggested algorithm and computational speed has only been a secondary goal. The code it therefore quite slow, mostly due to inefficient data structure manipulation. Because of this the discussion later will be focused on the solver progress at each iteration and not so much on the actual solution times.

Numerical Solver

The solver used in the Branch-and-Bound implementation is COIN-ORs open source interior point solver Ipopt. It is intended for large scale sparse non-linear optimization [Wächter & Biegler, 2005]. It works on a oracle description of the problem that can be asked for function values and potentially derivatives and the Hessian of the Lagrangian function. With the setting used here, Ipopt supplied with the derivatives and estimates the Hessian.

Ipopt is ran using mostly the standard settings suggested by the examples in the user manual [Wächter et al., 2009]. The only significant change to the settings is that the *bound_relax_factor* is set to 0. This is done because the B-spline functions are only supported inside the variable bounds, so it cannot tolerate that the solver relaxes these bounds and then evaluates outside the function support. This might introduce some convergence issues and in hindsight the B-spline support should have been extended slightly beyond the actual samples as a work-around to this problem.

6.4 Chapter Summary

A fairly standard Branch-and-Bound framework has been implemented. The unique feature being the support for B-spline approximation of general nonlinear functions. A relaxation strategy for B-spline constraints has been suggested. This strategy will be tested on a academic problem and on a oil production problem. The results of these test are given in Chapter 7 and discussed in Chapter 9.

Chapter 7

Approximation Results

This chapter will attempt to visualize the approximations produced by the tensor product B-spline. Since it is difficult to convey multidimensional surfaces in a two dimensional format the figures presented here will be plots of cross sections of the function surfaces where all except one variable has been fixed for each plot. Chapter 5 introduced the Marlim case which is used for all flow network related tests in this thesis. The case came with a collection of data sets generated by simulator models. These data sets will be used to to illustrate the most important properties of the suggested B-spline interpolation technique. The selected plots will attempt to cover the situations where the approximation is successful and the situations where it is likely to fail.

7.1 Approximating Marlim Well Data

The test case has twelve wells. Each well has a nonlinear relationship between pressure, gas lift allocation and oil flow rate, where the oil flow rate is found as a function of pressure and gas lift. This relationship is modelled by simulators and the simulators have been sampled to produce data sets for each well. The given datasets are grids with twelve points in each dimension. The area covered by the samples vary with each set. The data sets consist of only 144 data samples and all samples are used to create the approximations.

The figures presented in this section are made by fixing the gas lift allocation variable to each value it takes in the sample grid before evaluating the function over the pressure range. The actual amount of gas lift is not indicated since this would just clutter the figures, but, for clarity, the plots with higher production are the ones with a higher gas lift allocation. In addition to the B-spline approximation, a piecewise linear interpolation is given to make it easier to compare the shape of the B-spline surface to the shape of the data set.

Most of the production curves were easily approximated and gave nice spline surfaces. Figure 7.1 illustrates the most common behaviour. This type of surface was seen in most of the high production wells. The top five plots in this figure are almost identical. At a certain point the effect of additional gas lift diminish and the surface is easy to approximate. It would not have been a problem to remove some of the samples in this region and still provide solid approximations. In general the trend is that the approximation get more difficult with decreasing gas lift and oil rates. The sample density must therefore be higher in the regions with low flow rates. The data set for this well does not go all the way down to zero flow rate, which is why there are no real difficulties here.

For some of the wells with low production potential the data sets were less favourable and the lack of shape preservation in the approximation becomes apparent. Figure 7.2 gives an example of this. The problem is that the flow suddenly dies and the point where it dies changes with the amount of gas lift used. With the data provided it is not possible to tell if the surface should even be continuous at the point where this happens. Since the natural spline will attempt to minimize the second derivative it will oscillate in the neighbourhood of these abrupt changes. The worst behaviour among all the wells is given in figure 7.3 and here a different coefficient calculation is shown in addition to the natural spline. The alternative interpolation is a cubic hermite spline with the derivatives computed in a shape preserving manner. The shape preserving spline produces a pleasing surface without any oscillations.

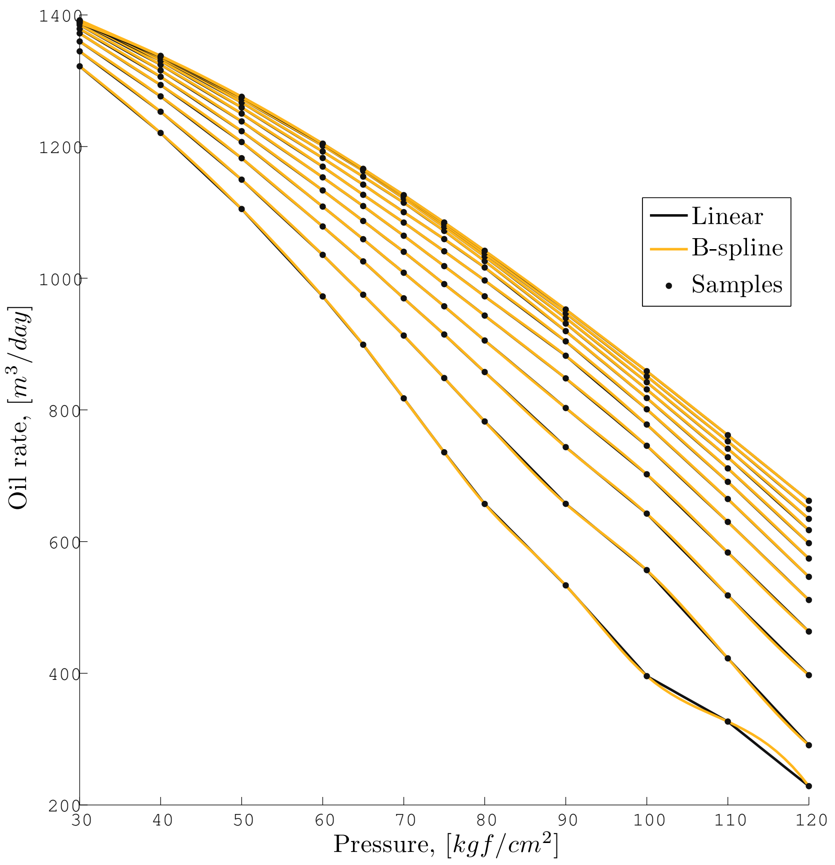

Figure 7.1: Cross sections of a well production curve. The cross sections are taken with the allocated gas lift fixed to each lattice in the sample grid. This is one of the wells with high production potential and the data sets for these wells are easy to approximate since the flow is stable over the entire grid.

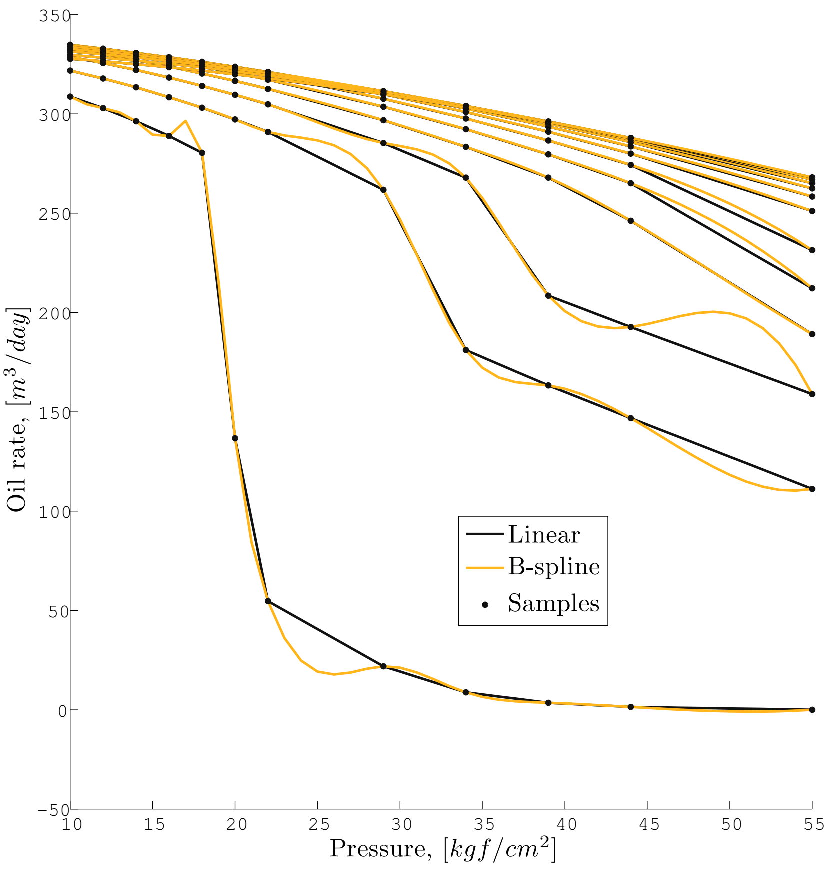

Figure 7.2: Cross sections of a well production curve. The cross sections are taken with the allocated gas lift fixed to each lattice in the sample grid. This is one of the wells with a lower production potential and these data sets are difficult to approximate near the regions where the flow dies.

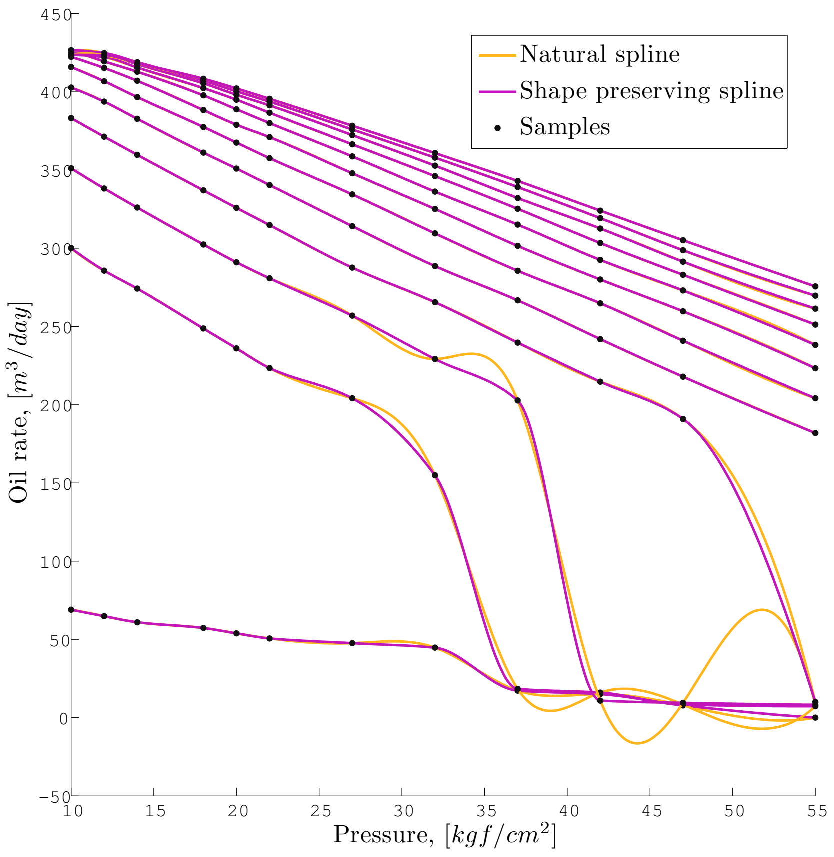

Figure 7.3: Cross sections of a well production curve. The cross sections are taken with the allocated gas lift fixed to each lattice in the sample grid. In this example the natural spline is compared to a shape preserving spline. It is clear that they are identical when the data is well behaved and that the shape preserving spline is superior when the underlying function exhibits step-like behaviour.

7.2 Approximating Marlim Pipeline Data

The test case has two pipelines. Each pipeline has a nonlinear relationship between pressure drop, gas, water and oil flow rate, where the pressure drop is calculated as a function of the flow rates. For both pipelines the data set is given as a $30\times30\times30$ grid of which a $18 \times 18 \times 18$ subset is used in the approximation. The samples are taken as every other sample except at the edges of the grid where additional samples are taken.

The figures presented in this sections are made by fixing the oil rate and the water rate to values in the sample grid before evaluating the function over the gas rate range. Each figure illustrate a single oil rate combined with a selection of water rates.

The pipeline functions are for the most part well behaved. Figure 7.4 illustrates the type of behaviour that dominates the part of the domain where the gas and liquid rates are fairly balanced. This is the part of the domain where the achievable flow rates are found. It can be seen that the plots are close to convex with respect to the gas rate and monotone in the liquid rates. This is fortunate for the solvers and is one of the reasons why they tend to find good solutions to the problem if given a reasonable starting point.

In the part of the domain where the gas rate dominate the liquid rates the data samples are less favourable. Figure 7.5 illustrates the typical behaviour in this region. The oil rate is fixed to one of the lowest values in the grid and the water rate ranges from the lowest to the highest water rate in the grid. When the water rate is high it is able to make up for the low oil rate and the surface is fine. But, as the water rate drops, the computation begins to falter and the surface seem to exhibit step like behaviour. The surface gets worse with higher gas-to-liquid ratios until it seems to get caught in a safety net that provides a reasonable computation for the most extreme cases. The natural spline is unable to capture the shape of the surface in this region, since it attempts to avoid steps as much as possible. The inability to recreate the surface is amplified by the fact that some of the samples are left out when creating the approximation.

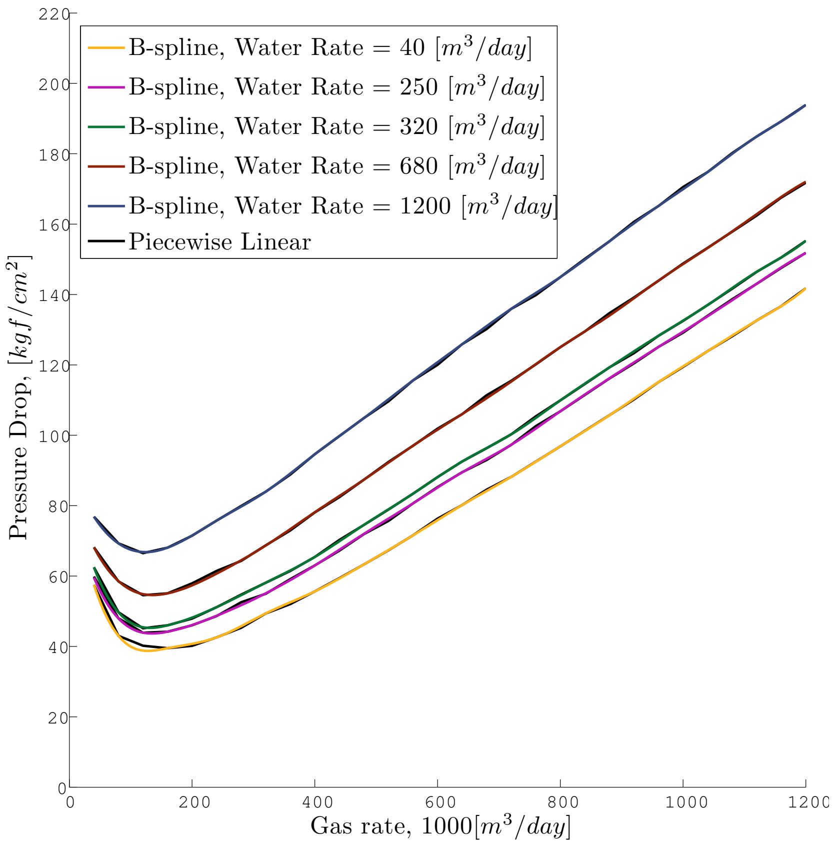

Figure 7.4: Cross-sections of a pipeline pressure drop function. The oil rate is fixed at 2200 $[m^3/day]$ for all plots. The water rate is fixed at different values for the individual plots. Gas rate is the free variable. This selection of plots captures the region of the pressure drop function where the total liquid rate is high.

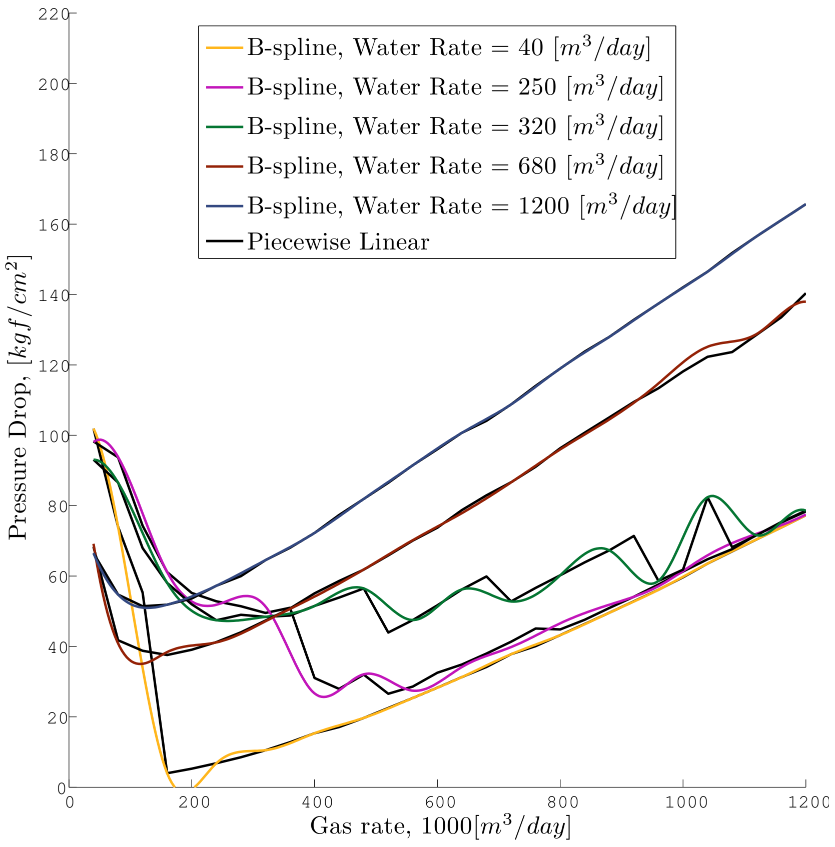

Figure 7.5: Cross-sections of a pipeline pressure drop function. The oil rate fixed at 200 $[m^3/day]$ for all plots. The water rate is fixed at different values for the individual plots. Gas rate is the free variable. This selection of plots captures the region of the pressure drop function where the total liquid rate is low.

7.3 Chapter Summary

The suggested B-spline seems to provide pleasing approximations in the regions where the simulators are smooth, but it will introduce oscillations when the data have step like behaviour. A small example of how shape preservation can be applied to combat this problem was provided. The discussion of these results will be given in Chapter 9.

Chapter 8

Optimization Results

This chapter will present the optimization results of this thesis. The problems investigated are aimed at revealing some of the strengths and weaknesses of the B-spline method, with focus being on the suggested relaxation strategy. The results will also attempt to shed some light on the possibilities of solving oil production problems to a global solution. The results will be generated using the Marlim test case that was introduced in Chapter 5 and a more academic test problem based on the Rosenbrock function.

8.1 The Rosenbrock Problem

Problem Formulation

The Rosenbrock function is a classic test function in unconstrained optimization. The n-dimensional version of the Rosenbrock function is

$$g(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} (1 - x_i)^2 + 100\left(x_{i+1} - x_i^2\right)^2 . \tag{8.1}$$

We will use an epigraph formulation of the unconstrained problem, except using an equality constraint. This is done by introducing a new variable x_{n+1} and pairing this with the Rosenbrock function to form the constraint function

$$c_1(x) = g(x_1, \dots, x_n) - x_{n+1} = 0 \tag{8.2}$$

and writing the objective function as

$$f(x) = x_{n+1}. \tag{8.3}$$

This problem is equivalent to the unconstrained problem. To complete the problem formulation in equation 2.1 we define the index sets to be $E = \{1\}$, $I = \emptyset$, $Z = \emptyset$ and limit the search space X to be

$$X = [0,2] \times \cdots \times [0,2] \times [0,200]. \tag{8.4}$$

Each dimension is sampled with the same partition, using seven uniformly spaced points. These are the used to produce cubic tensor product B-spline approximations $\hat{g}$, which are used in the actual optimization. The final problem formulation is:

$$\begin{aligned} \text{minimize} \quad & x_{n+1} \\ \text{subject to} \quad & \hat{g}(x_1, \dots, x_n) - x_{n+1} = 0 \\ & x \in X. \end{aligned} \tag{8.5}$$

The Rosenbrock problem is solved with $n = 2$, $n = 3$, $n = 4$ and $n = 5$.

Recall from Chapter 6 that increasing the number of knots in the sequence will draw the control points closer to the function surface and improve the convex relaxation, but it will also introduce more variables in the relaxed problem. The Rosenbrock function was sampled with 7 points in each dimension, which means the initial B-spline will have $7 + 4 = 11$ knots in each knot sequence. The algorithm is set to refine the knot sequences until the number of knots in each sequence reaches a predefined target. The problems are solved with the target number ranging from 11 to 23 for $n = 2, 3, 4$. For $n = 5$ the problem is only solved with the target knots up to 17. The symbol k will be used to represent the target number of knots. Just to clarify; all B-splines are first computed using the same 7 knots per sequence, then the splines that have a higher target number will refine their knot sequences until they reach the that number.

Tests and Results

The problem is solved to the global solution using the Branch-and-Bound algorithm described in Chapter 6 for the four different problem sizes listed above. The algorithm convergence is set to a 0.1 absolute difference between upper and lower bound . This is quite high, but the point here is to see how the algorithm behaves and this becomes clear regardless of how close the lower bound actually gets the optimal point. It should be noted that for the Rosenbrock problem the global solution is always found in the first iteration, so the only challenge for the solver is to prove that the solution is the global solution by generating sufficiently good lower bounds. It should also be noted that the global solution is not 0, but a value slightly below zero, because the approximation is not shape preserving.

The distance between the control points and the function surface was related to the distance between the knots in the knot sequence. Figures 8.1-8.4 attempt to illustrate the effect of halving the distance between the knots by showing the development of the global lower bound as the algorithm iterates for a selection of target knot numbers (one figure for each value of n). The target numbers visualised are 11, 15 and 23 (except for $n = 5$), because they will achieve this halving of the distances when the splines are cubic. To see this, remember that the first and last k knots are equal. This mean a knot sequence with 11 points has 3 knots between the endpoints. This makes four non-zero intervals. Increasing the number of knots to 15 allows one additional knot inside all these intervals. Increasing the number to 23 will divide all the intervals again. Remember also that the initial knot sequence did not necessarily have uniform distance between the intervals. With this in mind, looking at, for instance, the lower bounds created at the first iteration in each test run, it seems that going from 11 to 15 knots per sequence roughly cuts the lower bound in half. Going from 15 to 23 knots roughly cuts the lower bound in half again. There are of course other factors influencing the bounds also. These figures also illustrates the effect of the branching and how the relaxations develops as the algorithm iterates. The shape of the plots in these figures are almost identical, it is only the scaling of the axis that change when increasing the dimension.

The fact that methods based on sampling upfront will scale poorly with number of samples and dimension was brought up in Chapter 2. Figures 8.5-8.8 attempts to illustrate this. These figures show the number of iterations used to solve the problem and the average time spent per iteration, for all combination of n and k. Again with one figure for each value of n. These are all generated with the same number of samples, but with different number of knots. The number of samples only dictates the minimum number of knots that is required in the original problem. Due to the local support of the basis functions, the method is not required to have a target number of knots equal to or higher than the size of the grid. A lower number simply means that the algorithm must perform a certain number of cuts before the refinement will begin. The method therefore scales with the minimum number of knots. These figure show that the time spent processing a node in the Branch-and-Bound tree grows with both the number of knots and the dimension. The growth is exponential, with the number of knots per sequence as base and the dimension as exponent. The exponential growth with dimension is apparent when comparing the figures to each other. Looking at the individual figures in possible to recognise the polynomial grown when the dimension is fixed and the number of knots is increased. The figures also illustrates how the necessary number of iterations to solve the problems decreases with the number of knots used in the refinement.

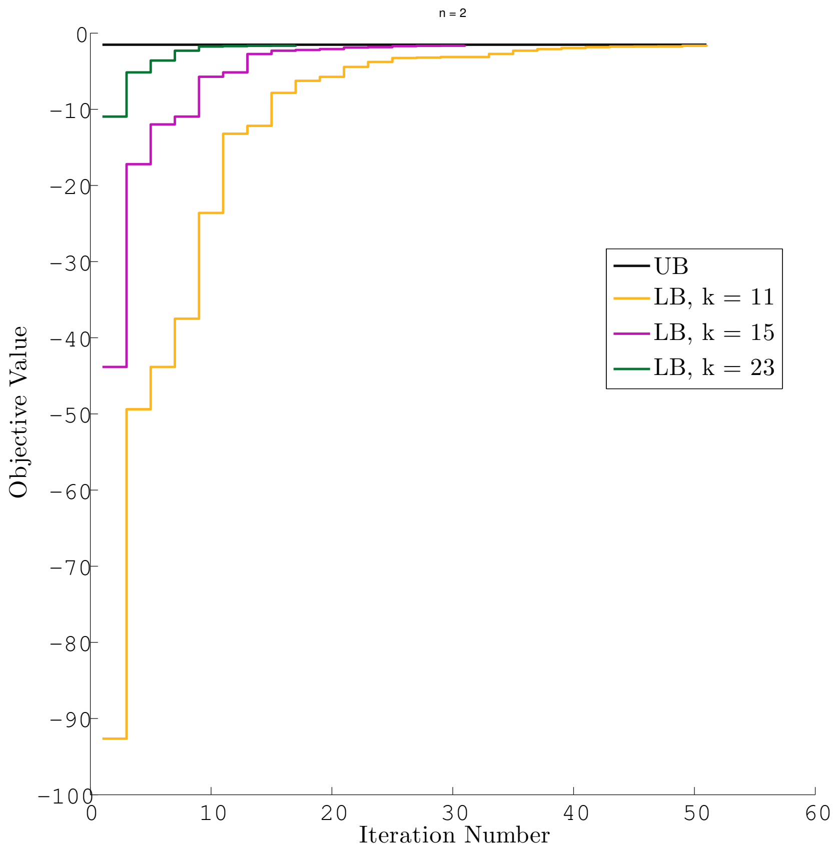

Figure 8.1: Development of the upper bound (UB) and lower bound (LB) for the Rosenbrock problem with $n = 2$. Lower bound shown for three different target numbers. Global solution is found as upper bound in the root node for all cases.

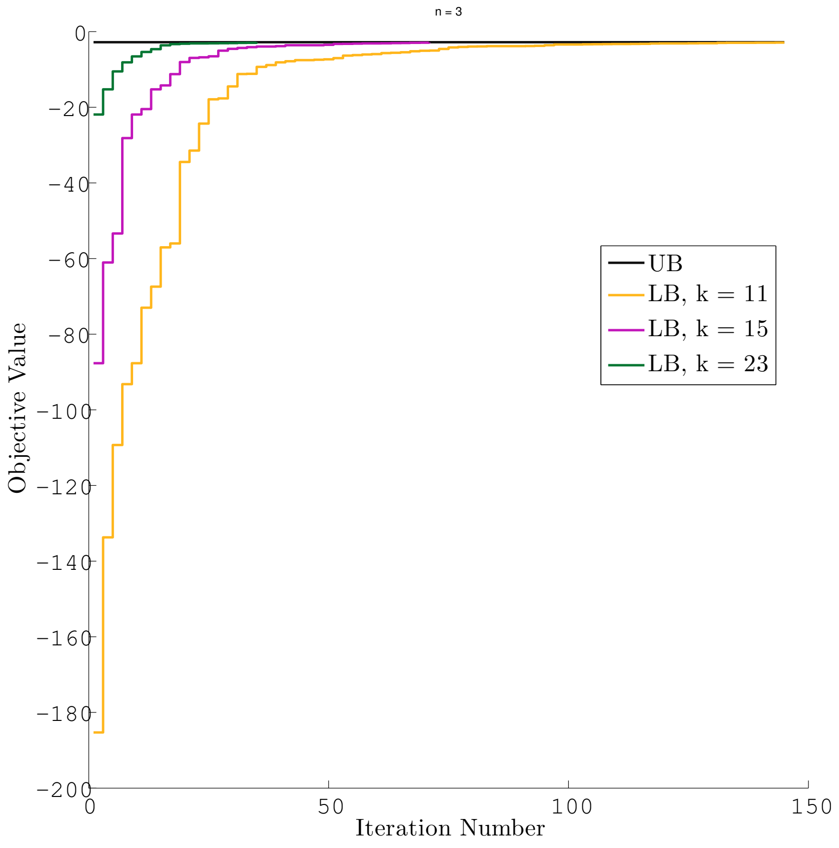

Figure 8.2: Development of the upper bound (UB) and lower bound (LB) for the Rosenbrock problem with $n = 3$. Lower bound shown for three different target numbers. Global solution is found as upper bound in the root node for all cases.

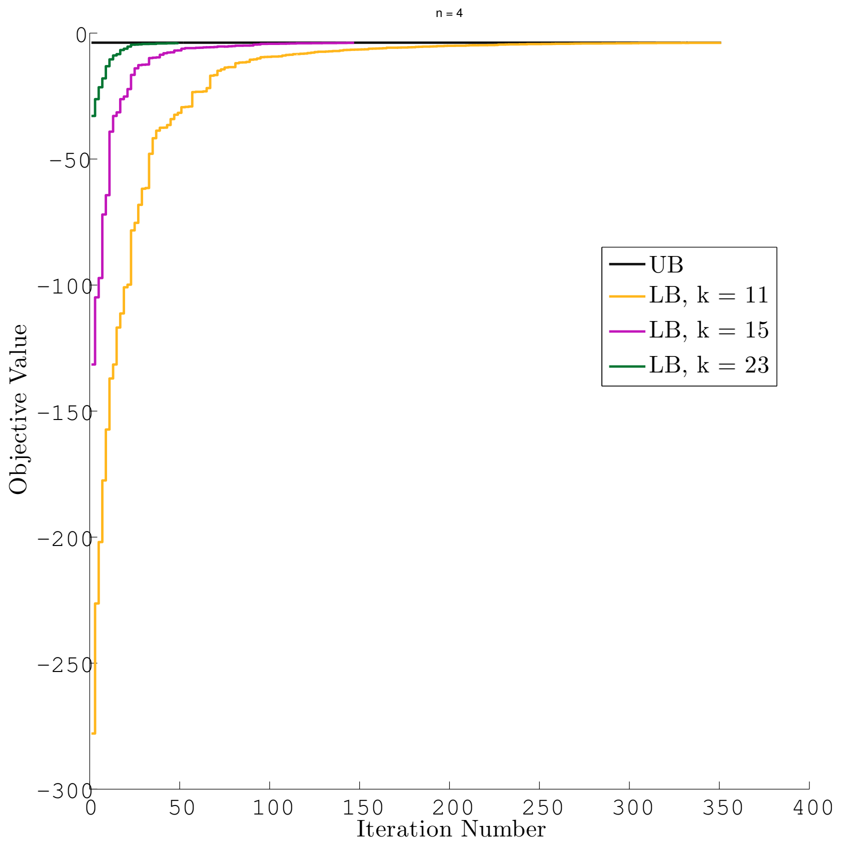

Figure 8.3: Development of the upper bound (UB) and lower bound (LB) for the Rosenbrock problem with $n = 4$.Lower bound shown for three different target numbers. Global solution is found as upper bound in the root node for all cases.

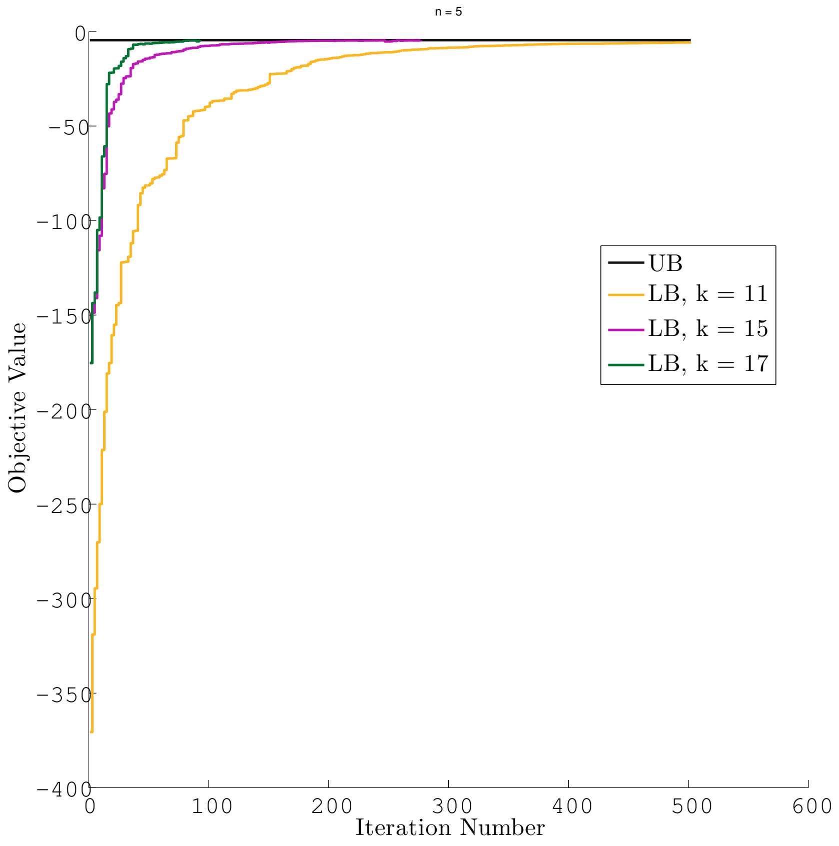

Figure 8.4: Development of the upper bound (UB) and lower bound (LB) for the Rosenbrock problem with $n = 5$. Lower bound shown for three different target numbers. Global solution is found as upper bound in the root node for all cases.

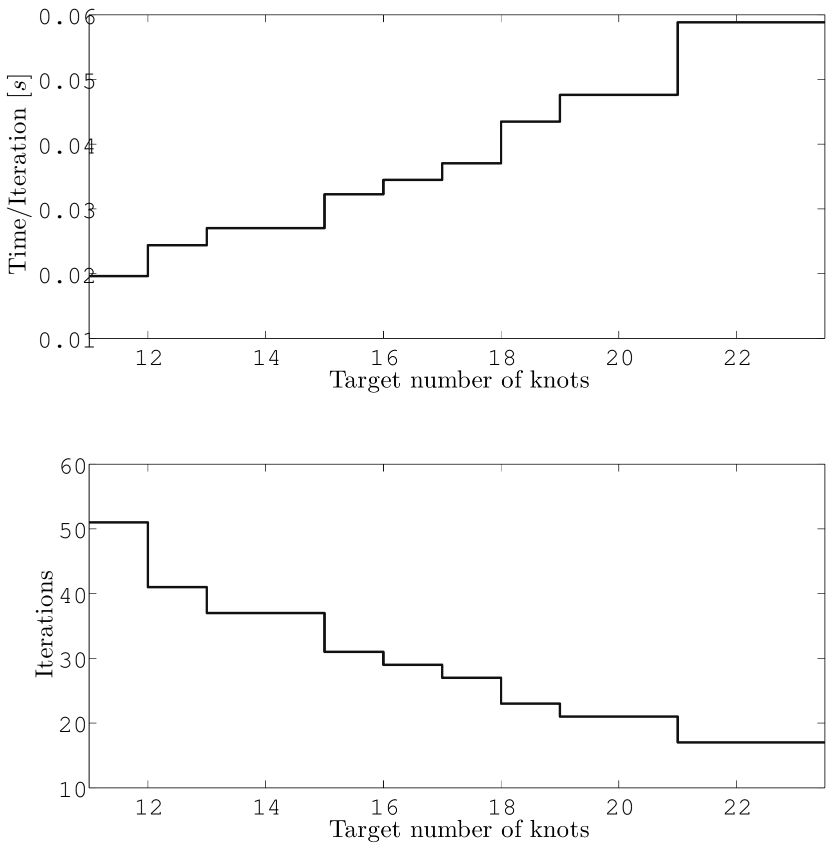

Figure 8.5: The number of iterations and the average time per iteration when solving the Rosenbrock problem with $n = 2$, using different targets for the number of knots.

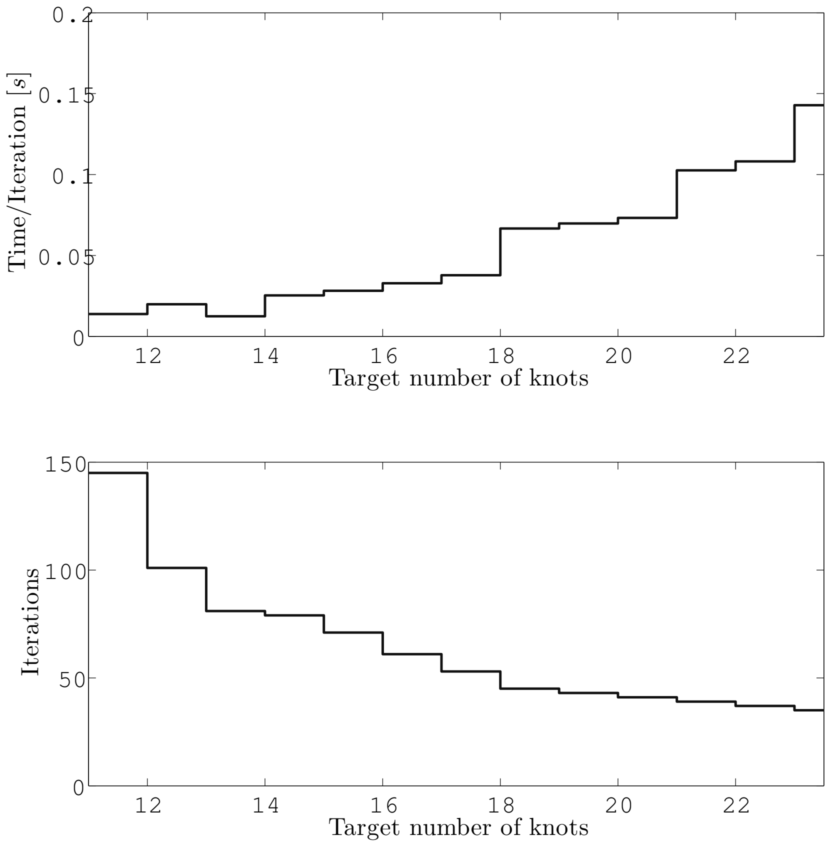

Figure 8.6: The number of iterations and the average time per iteration when solving the Rosenbrock problem with $n = 3$, using different targets for the number of knots.

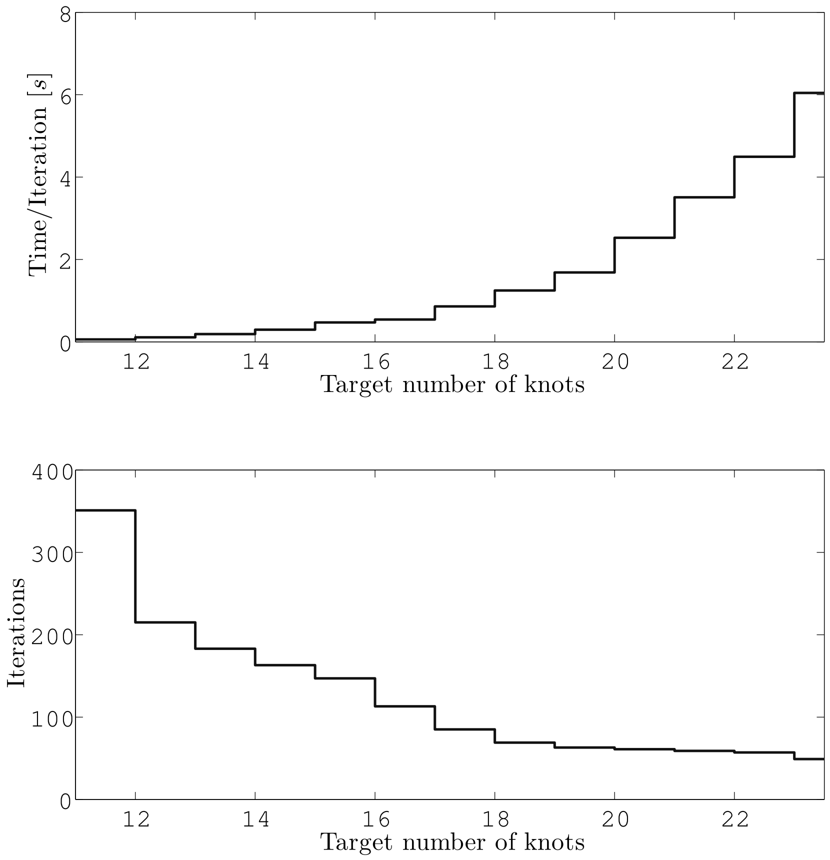

Figure 8.7: The number of iterations and the average time per iteration when solving the Rosenbrock problem with $n = 4$, using different targets for the number of knots.

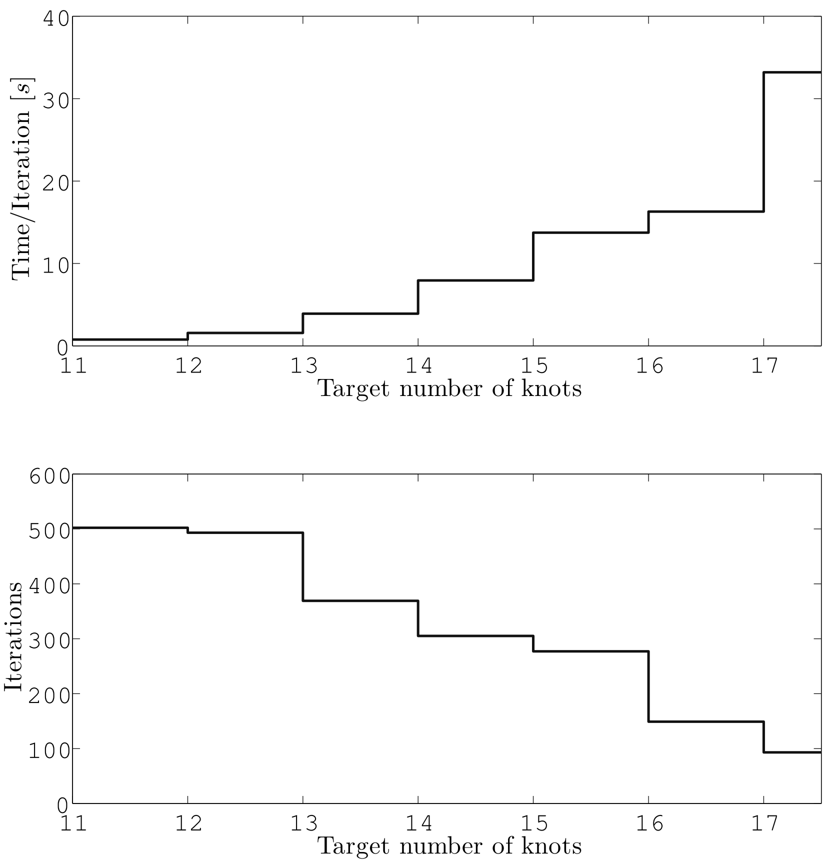

Figure 8.8: The number of iterations and the average time per iteration when solving the Rosenbrock problem with $n = 5$, using different targets for the number of knots.

8.2 The Marlim Flow Network

Problem Formulation

The Marlim problem formulation was given in Chapter 5. Some minor adjustments are made to the formulation. The bilinear terms that are part of large sums, *e.g.*

$$c_j(x) = \sum_{i=1}^{n} x_i y_i, \tag{8.6}$$

are rewritten to a sum of new variables

$$c_j(x) = \sum_{i=1}^{n} z_i, \tag{8.7}$$

combined with the simple bilinear constraints

$$c_k(x) = x_i y_i - z_i = 0,\ i = 1, \dots, n. \tag{8.8}$$

This is only done to make the implementation of the problem relaxations easier and will not play any significant role. The final problem has 154 variables(of which 18 are integer restricted), 112 equality constraints and 28 inequality constraints. There are 14 nonlinear equalities, one for each well and one for each pipeline. Remember that the objective is stated as a minimization of the negative of the oil flow.

The global solution to the problem is know to be in the region of -5838—obviously varying slightly with the approximation methods used—and it is restricted by separator pressure, water production and gas lift capacity[Shamlou & Ursin-Holm, 2012].

Tests and Results

The main results here are generated by first solving a single lower bound problem for all feasible routing combinations and then solving all the routing combinations with a lower bound lower than −5000 to a global solution. Before any numbers are stated, it should be noted that with the way the model is formulated there are multiple ways for some wells to *not* produce. This happens for wells whose pressure-flow characteristic go all the way down to zero flow rate, since they are then able to have zero flow rate independent of the routing variable. This is the case for some of less productive wells that are not allowed to flow at the optimal solution (for instance because they produce a lot of water and the optimal solution is restricted by water capacity). This basically means that multiple routing combinations will lead the same production profile. The best solution can be found with 12 different routing combinations.

As stated in Chapter 5 there are 38528 feasible routing options. All the feasible routing options where fixed in turn and a single lower bound problem solved. These problems were solved using Ipopt in the same way the Branch-and-Bound solver would when solving a lower bound sub-problem. The purpose of this is to gain some insight in how difficult the combinatorial part of this problem is. The results are collected in the histogram in figure 8.9, which illustrates how many routing combinations that have lower bounds in each interval. For comparison the same set of lower bound problems was solved using the relaxations that would arise with the piecewise linear approximation. The results are given in figure 8.10. This relaxation is interesting because it is the tightest possible lower bounds that can be created for any interpolating function using the Marlim data sets, because it is given as the convex hull of the data samples. This relaxation therefore serves as a nice benchmark for what it would be possible to achieve for any relaxation method.

With the suggested B-spline relaxation there are 19488 routing combinations with a lower bound better than -5000, these were solved to a global solution. Out of these problems a total of 8732 combination had a global solution better than -5000. The global solutions were found using the Branch-and-Bound solver described in Chapter 6 with an error tolerance of 5%. The solver is initialized with a starting point equal to the lower bound on the variables, which in general is an infeasible point. The distribution of solutions better than -5000 are illustrated in Figure 8.11. There are 411 solutions within 1% of the best solution and 863 solutions within 2% of the best solution (remember that many of these have the same production profile). On average the global solution is found after 8 iterations, but 15 problems failed to converge within 500 iterations (the maximum number of iterations). If the error tolerance had been lowered, the average number of iterations would have been increased dramatically. The distribution of iterations are given in Figure 8.13.

A different representation of the solutions to the best routing combinations is given in Figure 8.12. This figure illustrates how many routing combinations that are able to produce atleast the amount x for all x lower than -5000. This is interesting because it gives some insight to how likely it would be for a black-box search to find solutions within a certain limit of the global best solution (the derivative free black-box algorithm used in [Shamlou & Ursin-Holm, 2012] terminated with a solution of -4947).

The best routing combination was found to have a global solution of -5834. There are 8663 routing combinations with a initial lower bound better than the global solution using the B-spline relaxation. With the piecewise linear relaxation only 2578 routing combinations had an initial lower bound better than the global solution. This makes the number of solutions that needs further investigation 3.36 times higher with the B-spline relaxation. On average, the B-spline relaxation gives a lower bound that is 8% lower than the corresponding piecewise linear bound. The B-spline relaxation would be able to improve its performance if the target number of knots were raised, but this would also increase the solution times.

8.3 Chapter Summary

Two test problems have been investigated to gather information about how the suggested approximation scheme fares in terms of global optimization. The focus has been on the properties of the suggested relaxation method. The trade-off between a tight relaxation and the solution time of the relaxed problem was illustrated using the Rosenbrock problem. This problem also illustrated the progress of the lower bound improvements as the solver iterates. Interesting properties of the oil production problem was revealed by creating initial lower bounds on all routing combination by fixing them in turn and solving a single lower bound problem. This gave some insight to just how complicated the oil production problem can be when aiming for a global solution, by showing how sensitive it is to relaxation quality and by showing how many routing combinations that will look promising before starting spatial branching. The routing combinations with a lower bound better that -5000 was investigated further and solve to a global solution within that routing regime. The results will be discussed in the next chapter.

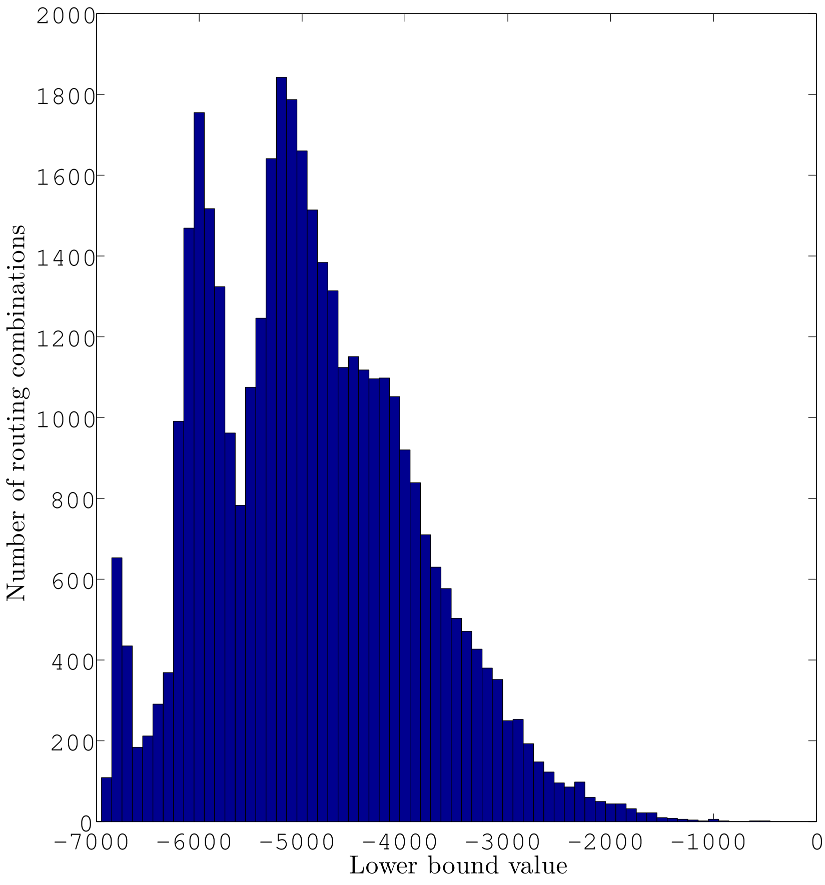

Figure 8.9: Histogram of the lower bounds for all feasible routing combinations using the cubic B-spline relaxation. Lower bounds are created by fixing the routing variables and solving a single lower bound problem. Histogram bin intervals are set to 100. A total of 38528 routing combinations are feasible, of which 8663 have a lower bound better than the global solution of the flow problem.

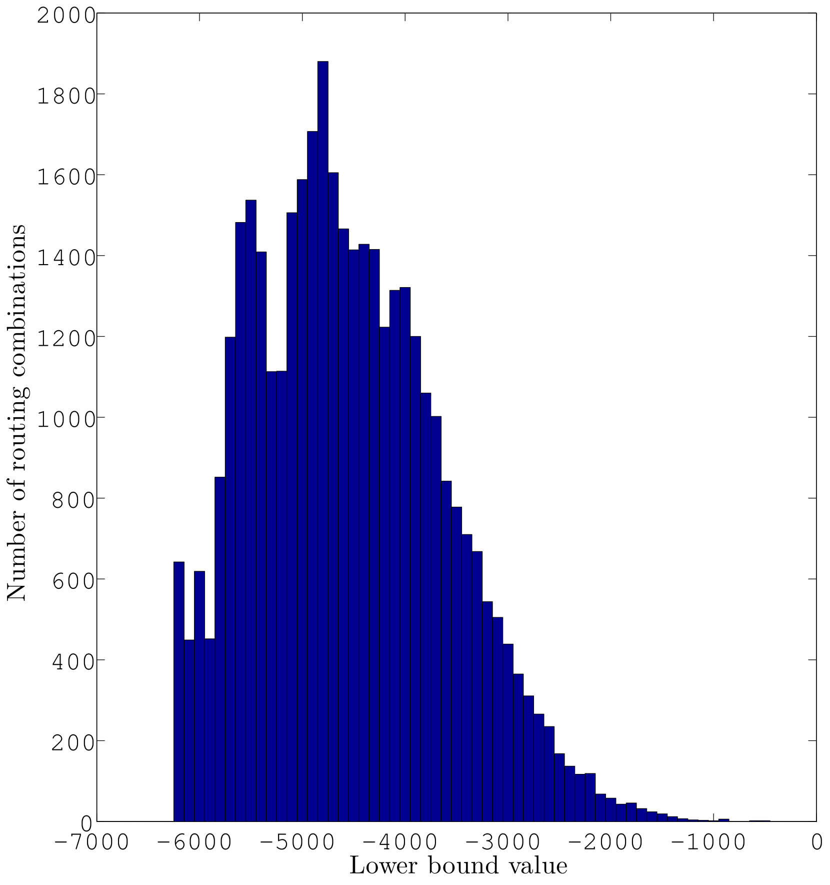

Figure 8.10: Histogram of the lower bounds for all feasible routing combinations using the piecewise linear relaxation. Lower bounds are created by fixing the routing variables and solving a single lower bound problem. Histogram bin intervals are set to 100. Only 2578 lower bounds are better than the global solution.

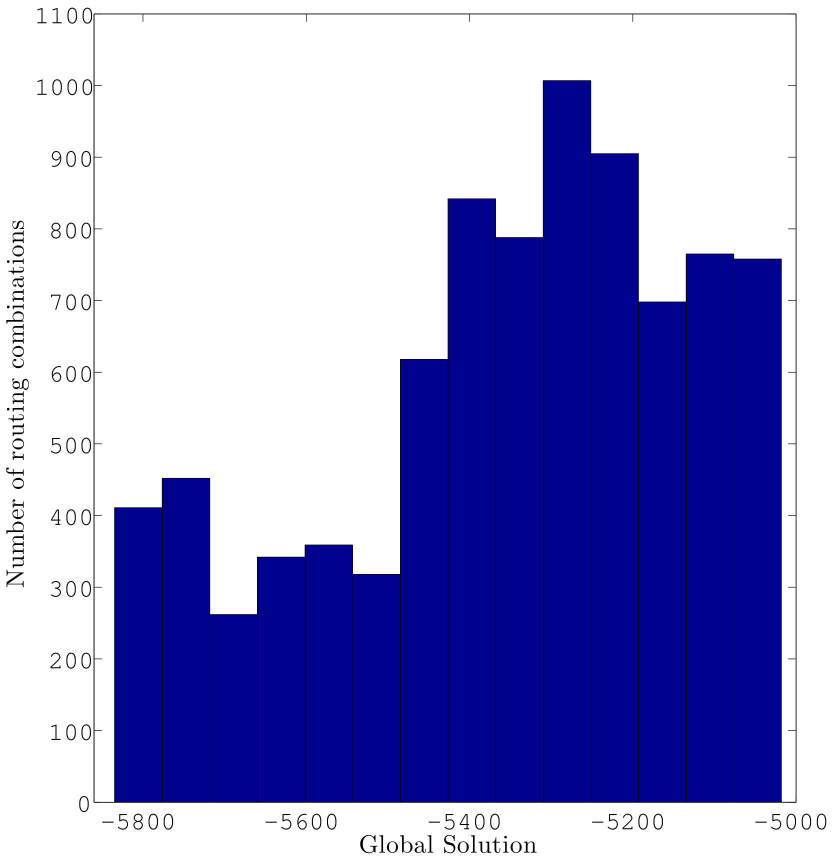

Figure 8.11: Histogram showing the distribution of global solutions better than 5000. The solutions were created by solving all routing combinations with a lower bound better than 5000 to a global solution. Histogram bin intervals are set to 1% of the best solution, which is 58. This means the first bin contains all solutions within 1% of the global best and the second all solutions between 1% and 2%, and so on.

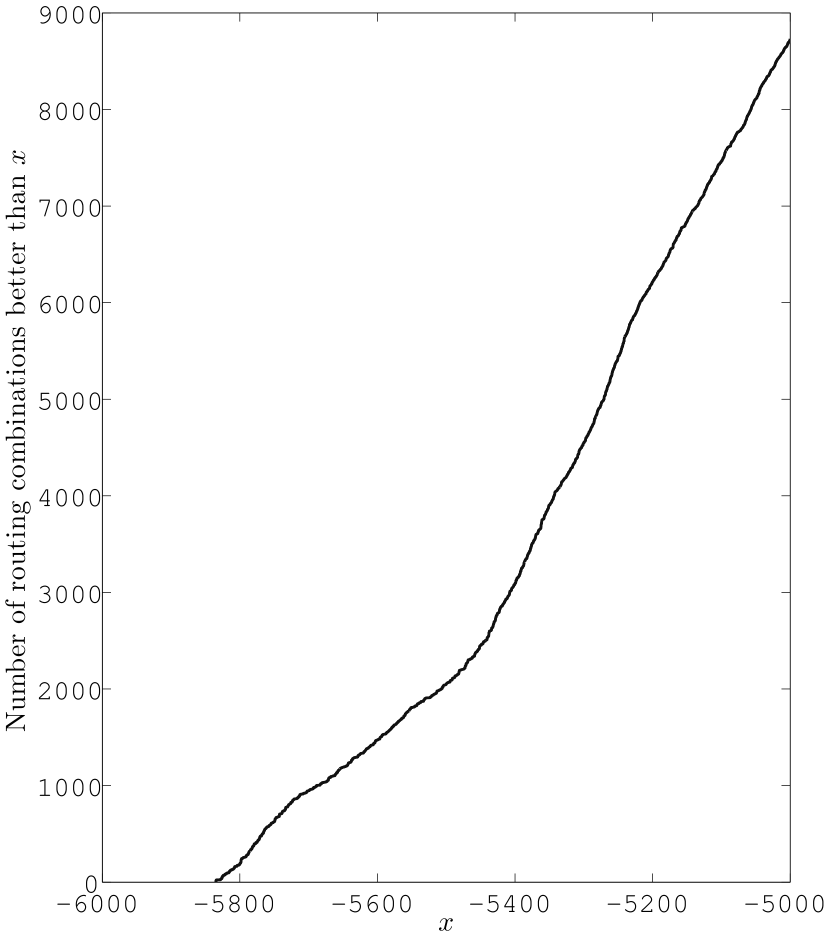

Figure 8.12: Illustration of how many routing combinations that are able to achieve a solution of x.

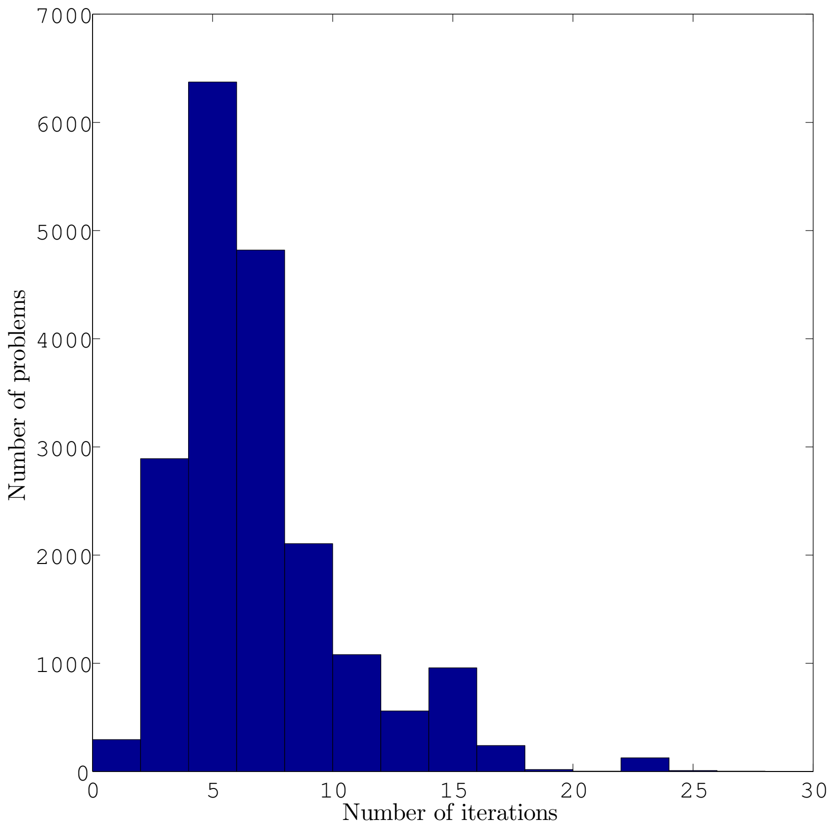

Figure 8.13: Histogram of the number of iterations used to solve the routing combinations with a initial lower bound to a global solution. The bin interval is 2. The figure goes up to 30 iterations, but there were 11 problems that used more than this. These are not shown in this figure as they are so sparsely scattered they would not be visible in the plot. Of the 19488 problems 15 did not converge within the limit of 500 iterations.

Chapter 9

Discussion

This chapter will discuss the results presented in Chapter 7 and Chapter 8. The first part of the discussion is about the B-spline as a simulator replacement. The second part is about how the approximation and its relaxation performed in optimization. The third part is about the oil production problem. Suggestions for further work will be given at the end of each part.

9.1 Approximating with B-splines

The suggested approximation method is the tensor product B-spline, with coefficients computed based only on function values in a way that minimizes the second derivatives of the surface.

B-spline from a User Perspective

The B-spline is incredibly flexible. And once familiar with it, it turns out to be quite elegant. There is however a bit of a learning curve associated with it and the theory behind it is not easily accessible. This is, of course, just my own opinion, but I have seen quite a few other thesis and reports that have searched for approximation techniques and concluded that piecewise polynomials are simply to complex and "beyond the scope". This is naturally because these methods are not emphasised in many field. They are, for instance, not mentioned in any standard curriculum at the Department of Engineering Cybernetics. However in communities that deal with computer-aided design, 3D modelling or computer graphics the B-spline and similar techniques are common (it is these communities that have driven much of the research on splines).

Since one of the main goals for this thesis was to find an approximation method that was both sufficiently smooth and easy to use, it is obviously a concern that the suggested method can, atleast in some fields, be considered as difficult. Luckily, the B-spline turns out to be easy to *use*, it is just difficult to *implement*. The reason being that all the necessary data structures can be generated automatically by information contained in the data set, which means that the users will not have to interact with the knot sequences or recursively defined basis functions or the tensor product operator unless they are looking for really specific behaviour. And, *if* the user is looking for really specific behaviour, the B-spline is probably the easiest way to get it.

During the implementation phase of this thesis I was unable to find solid open source codes supporting tensor product B-spline with the kind of functionality that was needed here. It seems that much of the software is either restricted to one dimensional interpolation or aimed at 3D modelling. MATLAB supports a variety of piecewise polynomial functionality and does actually include most of the B-spline functionality used in this thesis, but the interface to these B-spline objects are, in my opinion, a bit convoluted. The functionality needed was therefore implemented in C++ from scratch. This was quite time consuming—but also quite rewarding—and it is understandable why B-splines are often regarded as too complex to be applied in a single project when no easily interfaced software package is available. The code developed here will be used in future projects and it therefore seems like a reasonable time investment.

Bottom line here is that the B-spline can be difficult to understand and implement, but once implemented it is just as easy to use as any other approximation technique.

Sample Structure and Scaling Issues

The only significant draw back with the suggested method is its reliance on the data sample structure to guarantee the existence of an interpolating function. The data samples must be taken in a grid formation. Most simulators of interest can generate such data, if not, other interpolation techniques can be used to bring the data into this form. The problem with the grid structure is not the creation of the data sets. The main disadvantage with the grid structure is that it is not possible to locally refine the approximation by adding individual samples, since this breaks the grid structure. Take for instance a grid in 4 dimensions with 10 partitions in each dimension. This grid will produce $10^4 = 10000$ samples. If we wish to add a single sample to this set and the sample falls between all the grid lattices, then the grid partitions grow to 11 in each dimension. This gives $11^4 = 14641$ samples. Adding yet another point gives $12^4 = 20736$ samples in the grid. A grid based sample structure grows *extremely* fast. This is because the refinement procedure must update the entire grid. The cost of refining a grid grows exponentially with the domain dimension and this effectively restrict the application of grid based methods to functions in few variables.

The data sets that have been approximated in this thesis have all been well within the limits of what it is reasonable to deal with. The largest data sets belonging to the pipeline pressure drop functions. These pipelines have however been simplified to not take the inlet or outlet pressure into account. It would normally require one of these to perform the computations and the dimension of the domain would be increased by one. If, in addition, the flow had to be separated into additional phases, the grid sampling could quickly become too demanding.

Approximating Simulator Data

The suggested approximation requires a grid structure on the data sets and it was assumed this would be fine, since most simulator tools have the ability to produce such data sets. However, when taking a closer look at the provided data it is revealed that the simulators do not always produce nice and predictable behaviour over the entire grid. Both pipeline and well functions suffer from this problem. The short version of the story is simply that the optimization problem is formulated over a box-shaped region, while the simulators are tuned to match the reasonable operating range, which is a smaller, cone shaped region.

For most of the wells the flow-pressure relations were nice, predictable surfaces. The only problematic regions were close to the boundary of what the the well was able to produce before the flow dies. Most of the data sets were kept well within the regions of stable flow and the approximations of these were good. Figure 7.1 illustrates a typical example of this. The wells that caused problems were those that allowed production at very low flow rates, since these data sets covered the regions where the flow eventually dies. The rate where the flow dies is highly dependant on the amount of gas lift used. An examples of such a well is seen in Figure 7.2. The function surface looks like a cliff, with the cliff edge running diagonally across the domain. Such behaviour will not be reconstructed by the natural spline. A possible solution to this problem would be to use a shape preserving spline computation as illustrated in Figure 7.3. This clearly captures the behaviour better, but it comes at the price of a more complicated B-spline construction. The problematic regions are for the most part outside the interesting operational range, so it does not actually impact the optimal solution, only the solution time, for reasons that will be discussed later.

The two pipelines are very similar and they both have the same problems. The typical pipeline behaviour was illustrated in Figure 7.4 and Figure 7.5. The first showing the surface shape that is dominant over most of the feasible operational range of the domain and the latter showing the how the surface develops when the gas rates starts to dominate the liquid rates. This behaviour is not surprising. The simulators are tuned for a reasonable balance between gas and liquid, so when it is evaluated outside the intended variable range, the results are unpredictable. It can be seen that as the liquid rate gets lower and lower the pressure computation falters more and more, until it seems to get caught in a safety net that provides a reasonable computation for all flow compositions outside a certain limit. The surface occasionally exhibit step-like behaviour in these regions. As mentioned in Chapter 5, not all samples where used in the pipeline approximations. For the regions that behave nicely, it is barely noticeable when a sample is left out. The effect of removing a sample is greater in the unpredictable areas. Take for instance the steps seen in Figure 7.5 for the plot with a water rate of $320m^3/d$, it is obvious that when the step occur at a removed sample it is impossible to recreate it. Since there is no clear pattern to these steps it would be difficult to remove samples if the approximation had to capture these steps perfectly. The fact that the simulators exhibit such non-smooth behaviour is one of the reasons why letting a numerical solver evaluate the simulators directly is so difficult.

For both wells and pipelines we have seen that there are parts of the function domains that fall outside the reasonable operational conditions and the the simulator values can be unpredictable or just unfavourable in these regions. Without spending too much time dwelling on *why* these values are the way they are, it would not be surprising if some of this behaviour is induced by logic inside the simulator. This logic can cause sudden changes in the function behaviour. The suggested scheme is not able to capture such changes, but it could be modified to do so if that is desired. It should be noted that the shape preserving schemes require twice the amount of flexibility in each direction to be able to match both estimated derivatives and function values. This would have the same computational effect as doubling the size of the grid partitions, which, as discussed in the previous section, has quite dramatic effect on the size of the final data structures.

Since the only approximation problems seem to lie outside the regions that are interesting for the actual problem solutions, a possible solution is to not require interpolation of all data samples. An possible solution could be a smoothing B-spline that has the data sites weighted according to the simulator accuracy in each region. This would allow for a approximating function that interpolates at the "core" of the data sets and smooths out the wrinkles found at the boundaries, producing an overall neater shape to work with during the optimization without actually sacrificing any accuracy in the relevant regions of the function domain.

Further Work

The suggested method seems to work fine in the regions were the simulators work fine. The problems begin when the data samples begin to oscillate or have steps, because then the natural spline will amplify these effects. It would therefore be interesting to look at approximations that do not require interpolation of all the data points. A type of weighted smoothing B-spline would probably be a good candidate for future work.

9.2 Optimizing with B-splines

The B-spline is used in optimization to replace difficult nonlinearities. The replacements are generated upfront. The following sections will address some of the interesting topics concerning this strategy.

On Approximated Problems

The method in question is an approximation based solution method. The motivation for bringing in a approximation of a function is that it allows us to specify the representation of this function exactly the way we want it. This makes it easier to manipulate the function and to ensure that is satisfies the properties assumed by the solver, *e.g.* twice differentiable. The method suggested in this thesis have much in common with the piecewise linearisation method. The problem with these types of methods is they get computationally demanding when the number of samples needed to approximate the problem grows large. The method investigated here attempts to address this by applying the tensor product B-spline, which is a powerful approximation scheme that will be able to achieve the same approximation quality using fewer samples. This will hopefully increase the number of problems that can be approximated.

To achieve satisfying performance from a approximation based solver, it is necessary to have some knowledge of the functions involved. This is mainly to be able to find a reasonable sample resolution. Too few samples and the problem becomes irrelevant. Too many and it becomes impossible to solve. It is essential to strike a balance between these two extremes if the methods are to be successful. The problem of interest in this thesis has been the oil production problem and the nonlinearities here attempt to describe pressure-flow relationships. These functions are fairly predictable and an experienced engineer (which admittedly excludes me) would be able to predict in which areas the function is likely to change dramatically and increase the sample density in these regions. The unfortunate situation here is that areas where the sample density should be increased runs diagonally across the domain. With a grid structure, refining diagonally is equivalent to refining everywhere. This could actually be a problem and a possible solution would be to perform a change of coordinates, going from a oil rate and gas rate system to a oil rate and gas-to-oil ratio system, since a fixed gas-to-oil ratio would run diagonally in the original coordinate system.

A solution to the approximated problem cannot be guaranteed to also be a solution in the original problem, but assuming the the approximated functions are reasonable replacements the solution is likely to be close. The approximate solution will therefore make for a solid starting point if the original problem is solved afterwards. This can be seen as "adjusting" the approximate solution to fit the original problem and is a useful way to verify the solution. For instance for the Rosenbrock problem presented in Chapter 8 the approximated problem had a solution that was slightly negative and the optimal point was noticeably different from the known solution to the original problem. This is because the Rosenbrock problem was constructed with too few samples to let the natural spline capture the shape of the surface. It therefore introduced some oscillations between samples (similar to those seen in Figure 7.2 for the well curves) and the solution ends up being moved. It would however still have made a solid starting point for a adjustment using the original problem. With the Marlim case the approximated functions are unavailable, so it is not possible to verify these results properly, but they are similar to the results found in [Shamlou & Ursin-Holm, 2012].

On Upfront Sampling

The method investigated here is based on generating all the samples before the optimization begins. The samples are generates in a grid structure. As discussed earlier, there us a limit on the dimension of the functions that can be dealt with this way, because of the exponential growth of the data sets. In non-convex global optimization there will be a similar limit, because the computational effort grows exponentially with the function dimension here also. If it is reasonable to solve the problem with a global solver, it is probably also reasonable to sample the function upfront. In local optimization the situation is a bit different. These functions are not restricted by function dimension in the same way. The upfront sampling procedure will therefore be the main source of limitations in this case. A local solver will most likely not explore the entire function domain and since it does not need to generate the global lower bounds either, much of the work done in the upfront sampling stage will be unnecessary. This strategy is therefore less suited for local optimization, because of all the necessary work that is going into the sampling stage. Most local solvers would be happy to work directly on a black-box representation, so the approximation methods are applied for a different reason in local optimization. A common reason being that the black-box function is cumbersome to include in the optimization problem or that it is time consuming to evaluate. This is however problems specific issues that are difficult to judge on a general basis.

The alternative to upfront sampling is to continuously update the approximation during the solver iterates. In local optimization this would be possible to implement without having to change the solver, but in global optimization this would end up being a different class of methods entirely. These were briefly introduced in Chapter 2 as Bayesian optimization, but they have not been a major topic in this thesis. I have no real experience with these methods, but to my knowledge they have not been applied in global optimization of problems with size similar to the oil production problem.

B-spline Paired with a Local Solver

The B-spline seems to perform nicely when paired with the interior point algorithm. The progress at each iteration seems healthy, which indicates that the solver receives correct and useful derivative information and that the scheme in general is doing its job. This was of course expected, since the B-spline was set up to be twice differentiable, but it would still be possible to generate troublesome and sharp corners if the coefficients had be computed differently. For instance with an aggressive shape preserving scheme, some of the step like behaviour of the data would produce large second derivatives which would most likely cause trouble. The natural splines used here produces an interpolating function with a overall small second derivative and it should in this regard be the easiest interpolation function to deal with in terms of generating good search directions (because the surface will change direction as slow as possible).

Computationally the B-spline evaluation is quite cheap due to the compact support of its basis functions. It is also quite easy to differentiate, making it possible to supply the solver with both the Jacobian and the Hessian if that is desired.

The only real issue with the B-spline as it was implemented in this thesis is the fact that the B-spline support was made to match the data sets exactly, making the function drop to zero immediately outside the variable bounds. It turned out that the solver (Ipopt) often took the liberty to relax the variable bounds by a small value at some of its computational steps. This caused trouble, because the function would then be discontinuous at the bounds. The problem was fixed by applying a saturation function to the variables before evaluating the B-spline. This made the surface continuous, but not differentiable at the boundary. In addition to this, Ipopt was forced to not relax the variable bounds by adjusting the settings. This fixed most of the issues, but it is not an ideal solution and it should be addressed, by for instance extending the B-spline support slightly beyond the values in the data set.

Convex Relaxation of B-spline Constraints

One of the main reasons for choosing B-splines as the way of representing piecewise polynomials was the possibility of exploiting its special structure to generate convex relaxations. The convex relaxation is given as the convex hull of the control points. The difference between the convex hull of the control points and the convex hull of the function surface is related to the distance between the knots in the knot sequences and the second derivatives of the function. It is interesting to note that using the natural spline is beneficial here because it attempts to minimize the second derivative, but this is of course not a key point. The main tool for improving the relaxation was the knot refinement procedure. The effect of refining the knot sequences can be seen in Figures 8.1-8.4. Looking at the first iterations for the different levels of refinement it is clear that there is an improvement in the relaxations when additional knots are used. When the distances in the knot sequence is cut in half, the lower bound also seems to be cut in half. There are of course other effects influencing the relaxation, but it gives an indication on what can be expected when the knot sequence is refined.

The improvement associated with the refinement process will be subject to diminishing returns if measured by absolute value and remain fairly constant if measured relative to the distance between the knots. The unfortunate effect here is that the number of new knots that must be added to cut all the knot intervals in half will double each time. Since the computational effort grows fast with the number of knots, this additional refinement is only reasonable up to a certain point. In the experiments done in this thesis it seems that the benefit of adding new knots quickly dies out. This was attempted illustrated in Figures 8.5-8.8, where the number of iterations needed to solve a problem and the time spent per iteration given as functions of the number of knots used per knot sequence. It is clear that the impact of adding new knots dies out, while the computational time explodes. This is especially noticeable for the problems with higher dimensions. The overall message here is that the number of knots should be kept low and instead solve additional problems.

The way the relaxation is implemented here, all control points are used in the relaxation. There are ways to avoid this. Often only a subset of the control points will actually be needed to define the convex hull. See for instance the convex hull in Figure 6.4 for child A. Only three of the 13 points are needed to describe the convex hull. If this simplex is constructed the relaxation would get away with only adding three linear constraints as opposed to adding 13 new variables. This would be good when the convex hull could be reused for several iterations, as the initial cost of computing it would be spread out over many iterations. This happens when there are many constraints using different variables, because then most functions will not be affected when branching on a variable. There is however the possibility that all the control points will be needed to describe the convex hull. This is the case with child B in Figure 6.4. When this happens it is likely that the constraint is close to convex (or concave) and if a test was in place to detect this it would not be necessary to use a replacement based relaxation anyway.

Overall the relaxation strategy provides a good convex relaxation. It is comforting to know that with increasing refinement of the knot sequence the relaxation is asymptotically perfect, but unfortunately it is not computationally feasible to perform such extensive refinements. The main source of convergence is therefore left to the branching, which will end up having the same effect by using the same number of knots per sequence, but applied to a shorter interval each iteration. It must of course be noted that more conventional relaxation methods, such as the quadratic augmentation strategy, could still be applied to B-splines. The B-spline relaxation suggested here is simply an alternative and it seems to show great potential.

Properties of the Global Solver

Figures 8.1-8.4 illustrates how the lower bounds converged towards the global solution as the solver iterated. The general trend among all test runs here was that the lower bound converged, but the improvement per iteration seemed to die out. This is not unexpected and the reasoning behind this is the same effect that was mentioned in the previous section with knot refinement. Improving the lower bound by a fixed amount will require more and more effort the closer it gets to the global solution. This is unfortunate and not easily dealt with. The ideal solution would be to have a convexity test that was able to tell it the function is locally convex (or concave) and exploit this in the relaxation.

The suggested algorithm seems fairly robust, but there is only a limited amount of data to back this up. There is some comfort in the fact that the relaxed problems are LP problems, which is a well known type of problem and there are solvers that can deal with these easily. In [Tawarmalani & Sahinidis, 2005] it is argued that LP relaxations are highly favourable because of this. It is also advantageous that the B-spline algorithms are almost entirely based on convex combinations. This makes the knot insertion resilient to numerical errors that could potentially arise from repeated manipulation of the data structures during the branching procedure. This numerical robustness is in fact one of the reasons why B-splines are popular in general.

The algorithm displayed the expected development in all the test problems solved, with the exception of the 15 oil production cases with fixed routing that failed to converge. These seem to have triggered either a bug or very unfavourable behaviour in the solver. It was observed that in these situation Ipopt was sometimes unable to converge for the upper bound problem. It is therefore not unlikely that the problem is related to the poor approximation in the regions with low flow rates combined with an unfortunate routing combination. If only low potential wells such as the one seen in Figure 7.2 is routed to a pipeline and the solver is given an unfortunate starting point in the low flow rate region, it may be difficult to find the search direction that would lead out of this region (remember that the pipelines have a non-zero minimum flow rate, so this region in not necessarily feasible). It is therefore likely that these convergence problems are, at least partially, due to the oscillations introduced by the natural spline, since they make the surface gradients constantly change direction.

The computational times were rather high for all test problems. The reason is partially because of a slow implementation, but also because the solver choices were sub-optimal. It would most likely have been better to use a SQP solver for the upper bound problem due to the hot-start potential of these solvers. For the lower bound problem the solution times would have been improved dramatically if a dedicated LP solver had been used, for instance one that would exploit the dual problem which is common in Branch-and-Bound implementations. The solving sub-problems is however not the only time consuming element in the Branch-and-Bound algorithm. Creating and maintaining the convex relaxations contribute to a significant potion of the computational time and the main time sink here is the knot refinement algorithm. The algorithm used here is one of the simplest knot refinement alternatives, so additional speed up would be possible here as well. There is however no way around the fact that creating convex relaxations for functions of general structure is expensive, regardless of relaxation strategy.

Further Work

The suggested relaxation method is in dire need of some supporting heuristics to ease the computational load. Some suggestions have been made above, but there are many possibilities to explore. It is also necessary to find a test for convexity of B-splines that scales reasonably with dimension. As mentioned in Chapter 4 no satisfying test was found during the literature study of this thesis.

9.3 Short-term Oil Production Optimization

Questions specifically related to the oil production problem will be addressed in this section. The problem solved in this thesis was the Marlim test case. There are two main challenges in the oil production problem. One is the combinatorial problem that arise from the routing options. The other it the non-convex NLP that arise from the flow-pressure relations. In this thesis the focus has been on the NLP part of this problem, but the two are obviously linked. The overall goal is to solve the entire problem to a global solution. One of the difficulties is that the combinatorial problem is extremely large, so large that a brute force attack is not really a realistic solution method. It is therefore important to have a relaxation method that is tight enough to weed out many combinations at the same time. The discussion will therefore start with the lower bound problem and some of the issues discovered here, before moving on to the global solution and the value of finding the globally best operational state.

The Lower Bound Problem

The lower bound problem is crucial to the success of a global solver. The purpose of a lower bound is to hopefully rule out many of the possible combinations at the same time. With the oil production problem however, this turned out to be more difficult than initially assumed. The relaxed problems roughly fall into three categories. The first category is problems with a routing combination that is simply infeasible because it attempts to route a single well to multiple pipelines at the same time. These can trivially be discarded. The second category is problems with a routing combination where too many of the wells are shut off (typically more than half). These are also quite easily discarded as sub-optimal, since the sum of maximum flow out of the active wells are unable to beat the best solution. They are therefore discarded regardless of how good or bad the relaxations are. The final type of problem are the ones where enough wells are active to produce a respectable amount and they are distributed reasonably among the routing alternatives. It is for these problems that the quality of the pressure constraint relaxations becomes important.

It was seen that with the suggested relaxation a total of 8663 routing combinations had a relaxed solution better than the global solution. The distribution of these where given in Figure 8.9. Notice how the density of relaxed solutions seem to have a peak around -6000, which is just below the actual global solution of -5834. This is one of the reasons why the number of relaxed solutions that would need further investigation is so high. If the relaxation had just been a few percent better, the number would have dropped significantly. This was illustrated by using the tightest possible relaxation of the data sets. The results of this was given in Figure 8.10. As can be seen here, the peak has now been shifted to -5500, which is just above the global solution. The consequence is the the suggested method would have over three times as many candidate solutions that would need further investigation than the piecewise linear formulation, even though the relaxations only differ by 8% on average.

The suggested relaxation method is sensitive to the second derivatives of the function. Parts of the data sets have a series of step like changes, as was discussed earlier. These steps contribute to make the second derivative of the approximation unnecessarily large. In the case of the pipelines, these areas are also the only regions where the function is significantly non-convex. The convex hull of the piecewise linear approximation is not affected by these steps, but the B-spline control points will be driven far away from the function surface in these areas. This will worsen the overall relaxation and generate looser bounds. This could possibly be improved by applying some of the remedies suggested in the approximation discussion. Another possibility would be to improve the relaxation by additional refinement of the knot sequences, but this will quickly get expensive since the knot sequences are already quite large.

The Global Solution

The global solution of the problem was found and it was the same as it has always been for this test case, so the number itself is not the interesting part. The interesting part is the fact that there are so many routing combinations that are almost equally good. The distribution of achievable production for the best routing combinations was given in Figure 8.11. Within 1% of the global solution we find over 400 other routing combinations and within 2% there are more than 800 combinations. Remember though, that many of these combinations led to the same production profile.

With so many similar solutions and so many relaxations that look promising, finding the global solution will require significant effort. A general MINLP will always be difficult, but there are problem instances that are more forgiving than others. It turns out the the oil production problem cannot be expected to be one of these. Solving this problem is therefore likely to take quite some time, no matter how good the relaxations are. However, hope may lie in the fact that the constraints are tied together by a specific structure and parallel to the writing of this thesis fellow students students Sheri Shamlou and Stine Ursin-Holm are writing their Master Thesis on Branch-and-Bound heuristics for this exact problem. This will hopefully lead to some tailored heuristics that can be combined with the methods discussed here and produce a solver that is dedicated to short-term production optimization.

When the routing is fixed, the algorithm is able to find the global solution quite fast. Figure 8.13 illustrated the number of iterations required. For most of the problems the optimal solution within a routing regime was confirmed in less than 15 iterations. This is in itself quite good. Being able to, for instance, confirm the optimality of the gas lift allocation within a routing configuration is valuable. Admittedly the convergence tolerance was set a bit high and if the tolerance is lowered the number of iterations required will grow fast. This is because there are many variables that would potentially need branching to improve the relaxations enough and as seen in the Rosenbrock case, the number of iterations it takes to reduce the error by a fixed amount grows fast as the error gets smaller. Regardless of error tolerance, the method is able to quickly identify good solutions within a routing configuration. Another important fact here is that among the 19488 investigated routing combinations, the algorithm found a solution unassisted in all but 15 cases. Of course it remains to see if this statistic holds for other production fields too, but it is definitely a good start.

The fact that there are so many local solutions with a production close to the global solution makes the global solution less attractive. This is because re-routing a well is not free, and generally it is desirable to only make a few routing changes. If the global solution calls for a significant number of changes, but the second best and almost identical solution only requires a few, it is likely that the solution with less re-routing would be implemented. This motivates a problem statement the takes the current routing configuration into account. It would for instance be possible to formulate a problem the seeks the best one or two re-routing choices. This would also be a smaller combinatorial challenge, which could potentially make the global solution feasible to attain within a short time. When the new routing strategy has been successfully implemented, the problem could be repeated if desired. This would lead to a series of re-routing suggestion that eventually leads to a routing combination where it is impossible to improve the production further without making drastic changes to the configuration. The entire process would resemble the way a local solver iterates, gradually finding better and better solutions until there are no promising steps left to take. In such a scenario it would be beneficial to take the absolute best steps each time. The suggested solver method would be able to do this.

The global solver will be significantly slower than a local solution strategy, simply because of the extra branching required (due to the difficulty of creating solid lower bounds). It will however be more reliable in terms of finding a solution at all, for instance within a fixed routing configuration, and the global solver is less dependant on the input provided by the user (such as the initial point).

Further Work

Investigating the benefits of using a more refined relaxation would be interesting, because the current relaxation produces a significant amount of lower bounds that are just barely better that the global solution. The most important point for further work is however probably to apply the methods to other test cases to see if the phenomena observed in the Marlim case apply in general.

9.4 Chapter Summary

The tensor product B-spline approximations were satisfying for the most part, but its lack of shape preservation was revealed in the regions where the data sets displayed step like behaviour. In these regions the natural spline will begin to oscillate as an attempt to minimize the second derivative. Possible ways to address this has been suggested, with a weighted smoothing being my own favourite.

In the optimization setting the B-spine performed well. There are no issues that can be directly tied to the B-spline in the local optimization case. In the global optimization case the suggested relaxation schemes seems to work, but the improvements to the convex relaxations as subject to diminishing returns, which leads to a need for additional support heuristics for the algorithm to be satisfying.

The replacement strategy based on sampling and replacement upfront has a limit on the dimensions it is feasible to work with, due to the exponential growth of samples that must be created. This is not a major issue in the global optimization case, as the same exponential growth is found there also, but for local optimization this is a major problem.

The oil production problem was solved, but it is reavealed that the combinatorial part of the problem is difficult because there are many local solutions with very similar objective values, which means the algorithm has to branch a significant number of time for each alternative before it can draw conclusions.

Chapter 10

Conclusion

This thesis has investigated approximated MINLPs, with the oil production problem as the main application. The approximation method was tensor product B-splines and it have been evaluated in terms of its ability to approximate the relevant types of data sets and its performance in local and global optimization.

The B-spline method provides a suitable replacements for the simulators. The only issues discovered were related to fact that the simulator data is not reliable, nor relevant, in parts of the function domain. These areas turned out to be a liability for both the approximation quality and the optimization progress. It is therefore recommended to avoid interpolation over the entire domain and use a weighted smoothing scheme instead to find the B-spline coefficients. A downside with the method is that sampling in a grid structure is computationally expensive when the domain dimension of the approximated function is large.

The B-spline method is sufficiently smooth and cooperates well with the interior point solver used in this thesis and it seems suitable as a function replacement in local optimization. In global optimization the methods has a special structure that can be exploited to generate convex relaxations and the solver implemented was successful in doing so. The relaxation technique did however display poor convergence and requires some additional support heuristics to be efficient.

A oil production problem was investigated by applying the above-mentioned method. The results revealed that this particular case is difficult to solve globally because there are so many routing combinations with objective values close to the global optimum. This makes it difficult to discard solutions based on the relaxed problem and the combinatorial challenge becomes significant. With a fixed routing combination the suggested solver is able to find the best operating state within a reasonable amount of time and it seems to be able to do so reliably.

Bibliography

[Adjiman et al., 2000] Adjiman, C. S., Androulakis, I. P., & Floudas, C. A. (2000). Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46(9), 1769–1797.

[Androulakis et al., 1995] Androulakis, I. P., Maranas, C. D., & Floudas, C. A. (1995). aBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4), 337–363.

[Boehm, 1980] Boehm, W. (1980). Inserting new knots into B-spline curves. *Computer-Aided Design*, 12(4), 199–201.

[Bonami et al., 2008] Bonami, P., Biegler, L. T., Conn, A. R., Cornuéjols, G., Grossmann, I. E., Laird, C. D., Lee, J., Lodi, A., Margot, F., Sawaya, N., & Wächter, A. (2008). An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2), 186–204.

[Boyd & Vandenberghe, 2004] Boyd, S. & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

[Cohen & Schumaker, 1985] Cohen, E. & Schumaker, L. L. (1985). Rates of convergence of control polygons. *Computer Aided Geometric Design*, 2(1-3), 229–235.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction To Algorithms*. The MIT Press, 3. edition.

[De Boor, 1978] De Boor, C. (1978). *A practical guide to splines*. Springer-Verlag.

[Deutsch, 2001] Deutsch, F. R. (2001). *Best Approximation in Inner Product Spaces*. Springer.

[Floater, 1993] Floater, M. S. (1993). A weak condition for the convexity of tensor-product B ´ ezier and B-spline surfaces. (pp. 1–12).

[Foss, 2012] Foss, B. (2012). Process control in conventional oil and gas fields—Challenges and opportunities. *Control Engineering Practice*, 20(10), 1058–1064.

[Grossmann, 2002] Grossmann, I. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, (pp. 227–252).

[Gunnerud & Foss, 2010] Gunnerud, V. & Foss, B. (2010). Oil production optimization—A piecewise linear model, solved with two decomposition strategies. *Computers & Chemical Engineering*, 34(11), 1803–1812.

[Gunnerud et al., 2012] Gunnerud, V., Foss, B. A., McKinnon, K. I. M., & Nygreen, B. (2012). Oil production optimization solved by piecewise linearization in a Branch & Price framework. *Computers & Operations Research*, 39(11), 2469–2477.

[Jones, 2001] Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, (pp. 345–383).

[Jones et al., 1998] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, (pp. 455–492).

[Kosmidis et al., 2005] Kosmidis, V. D., Perkins, J. D., & Pistikopoulos, E. N. (2005). A mixed integer optimization formulation for the well scheduling problem on petroleum fields. *Computers & Chemical Engineering*, 29(7), 1523–1541.

[Lancaster & Salkauskas, 1988] Lancaster, P. & Salkauskas, K. (1988). *Curve and Surface Fitting: An Introduction.* Academic Press.

[Lyche et al., 1985] Lyche, T., Cohen, E., & Mø rken, K. (1985). Knot line refinement algorithms for tensor product B-spline surfaces. *Computer Aided Geometric Design*, 2(1-3), 133–139.

[Lyche & Morken, 2004] Lyche, T. & Morken, K. (2004). Spline methods draft. *Department of Informatics, University of Oslo, ...*

[Magnani et al., 2005] Magnani, A., Lall, S., & Boyd, S. (2005). Tractable fitting with convex polynomials via sum-of-squares. *Proceedings of the 44th IEEE Conference on Decision and Control*, (pp. 1672–1677).

[Meyer et al., 2002] Meyer, C. A., Floudas, C. A., & Neumaier, A. (2002). Global Optimization with Nonfactorable Constraints. *Industrial & Engineering Chemistry Research*, 41(25), 6413–6424.

[Neamtu, 1991] Neamtu, M. (1991). *A contribution to the theory and practive of multivariate splines*. PhD thesis, University of Twente.

[Nocedal & Wright, 2006] Nocedal, J. & Wright, S. J. (2006). *Numerical optimization*. Springer, second edition.

[Park, 2012] Park, S. (2012). Approximate branch-and-bound global optimization using B-spline hypervolumes. *Advances in Engineering Software*, 45(1), 11–20.

[Schumaker, 2007] Schumaker, L. (2007). *Spline functions: basic theory*.

[Shamlou & Ursin-Holm, 2012] Shamlou, S. S. & Ursin-Holm, S. (2012). Comparative Study of Optimization Solution Techniques.

[Tawarmalani & Sahinidis, 2005] Tawarmalani, M. & Sahinidis, N. V. (2005). A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2), 225–249.

[Wächter & Biegler, 2005] Wächter, A. & Biegler, L. T. (2005). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25–57.

[Wächter et al., 2009] Wächter, A., Laird, C. D., Margot, F., & Kawajir, Y. (2009). Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT.