NTNU
Norwegian University of
Science and Technology

Development of constraint handling techniques for well placement optimization in Petroleum Field Development

Problem formulation and implementation for FieldOpt software including well index calculation for deviated wells

Hilmar Magnusson

Master of Science in Physics and Mathematics
Submission date: April 2016
Supervisor: Markus Grasmair, MATH
Co-supervisor: Jon Kleppe, IPT
Bellout Mathias C., IPT

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Well placement optimization is an important part of Petroleum Field Development. However, in order to improve the optimization procedures, it can be important to incorporate considerations like knowledge about the geology of the reservoir or about existing or planned well paths. This leads to additional constraints that have to be satisfied during the optimization. In this thesis we concentrate in particular on constraints on the well lengths and the distance between the wells.

We suggest an alternating projections method to deal with both constraints at the same time, and develop an efficient numerical method for the solution. Although we cannot prove that the method is convergent, numerical results of our implementation indicate that the approach works as intended.

An additional important contribution from this work is the implementation of a well index calculator. In reservoir simulation, the well index relates the flow rate and pressure of the wellbore to the pressure solution of the subsurface fluid flow system, and is therefore an essential part in computing resulting production volumes.

We also implement an algorithm that, given a slanted well and the physical state of a reservoir, calculates the well indices for the well blocks that are intersected by the well. In particular the well index calculation for deviated wells is a nontrivial task that is important for well placement optimization research. This task is already handled by some industry reservoir simulators, but the implementation is hidden from the end-user.

All of the implementations are meant to be an addition to FieldOpt, a petroleum field development optimization framework that is currently under development by the Petroleum Cybernetics Group at NTNU.

Preface

This thesis was conducted as a part of the Master's degree in Physics & Mathematics at the Department of Mathematical Sciences at the Norwegian University of Science and Technology, NTNU. It was written during the spring semester of 2016 under the supervision of Assoc. Prof. Markus Grasmair, and co-supervised by Prof. Jon Kleppe and Postdoc Mathias C. Bellout. The work was done in close collaboration with the Petroleum Cybernetics Group at NTNU.

Acknowledgments

I wish to thank Markus Grasmair for helping me with formulating and solving the problems in this thesis, and for his excellent feedback on the written work. I also want to thank Mathias Bellout for his guidance and help with structuring my work during the last 6 months, and Einar Baumann, whose technical input has saved me more hours than I want to admit.
Lastly I would like to thank my family and friends for supporting me through all these years. I would also like to mention that the Pretty Little Liars are fabulous. You know who you are.

Table of contents

Chapter 1

Introduction

This chapter gives the reader an overview of our goals and the purpose of the work. We wish to contribute to the work in petroleum field development by implementing constraint handling routines and a well index calculator as an aid to current well placement optimization methods. These implementation are made so that in the future they can be integrated in FieldOpt [1], a petroleum field development optimization framework that will aid in the operations of producing hydrocarbons from the subsurface. The problem of placing wells is a substantial part of petroleum field development, and because of its importance we should use optimization procedures to augment the well placement decision-making. In this effort it is important that we have a way to measure the oil field production and its related costs, and that we are able to define good constraints, in the sense that they are proper representations of petroleum engineering knowledge. After we have defined the task as a mathematical problem with associated constraints, we should try to develop and implement efficient methods to deal with these constraints while maximizing income.

What is Petroleum Field Development(PFD). Petroleum Field Development is mainly concerned with maximizing the return of the financial investment. We can gain financial revenue by increasing the recovery of oil from the reservoir or by reducing the costs associated, e.g., drilling, labor, injection and the production of water.

1.1 The general well placement problem

What is the objective of well placement optimization. The problem of well placement optimization is the following: Given some physical information about a reservoir we wish to place one or several wells in such a way that an objective is reached. The objective is usually to maximize the net present value, which again is achieved by maintaining a high oil recovery while minimizing costs

at the same time. This problem is one that has been studied extensively and many methods for optimizing the placement of the well have been proposed [2], both derivative based methods and derivative-free methods.

How wells are parametrized. In the specific problem we study in this thesis, $\mathbf{x}$ represents the position of all wells in the system, but generally it may also include other variables such as control settings, bottom hole pressure or shape coefficients in the case of curved wells. A single straight well can be defined by the coordinates of the heel and toe of the well. As an example, four wells in three dimensional space can be defined by a vector $\mathbf{x}$ containing $N = N_{wells} \cdot N_{heelandtoe} \cdot N_{dimension} = 4 \cdot 2 \cdot 3 = 24$ real numbers, i.e., $\mathbf{x} \in \mathbb{R}^{24}$.
The general problem can be formulated as an unconstrained optimization problem in the following way

$$\min_{\mathbf{x} \in \mathbb{R}^N} J(\mathbf{x}), \tag{1.1}$$

where the objective function $J : \mathbb{R}^N \to \mathbb{R}$ maps a point $\mathbf{x} \in \mathbb{R}^N$ to a real number. J determines how well the objective is reached in the point $\mathbf{x}$. In a scenario of oil production J is typically be defined in such a way that oil production is maximized and the related costs (e.g., the cost of drilling, well equipment etc.) are minimized.

How is the objective computed, what is a reservoir model/simulation. In order to evaluate the objective function in a point $\mathbf{x}$ a simulation is needed to determine the pressure distribution in the reservoir system. This is typically done by providing a reservoir simulator, such as Eclipse [3], the physical state of the system which might include well block pressure, permeability and well indices.

What is the well index. The well index relates wellbore flow rate and pressure to well block quantities [4], which is important for computing oil recovery. The well index of a well block is uniquely determined by the well placement coordinates and is either left as a job to the reservoir simulator or computed and given as input by the user. The reservoir simulator then computes the pressure distribution in the reservoir by numerically solving a set of partial differential equations (PDEs) which then implicitly determines the oil production rate.
The wells in the well placement problem are treated as straight line segments or as continuous chains of straight line segments inside a reservoir of blocks. The blocks of the reservoir have six planar faces and every face of a block is either shared with the face of another block or lies on the boundary of the reservoir domain. It is possible to extend the model to not only consider the placement of wells. One could also include things such as time dependent well control variables. This would result in a more complex variation of the original problem (1.1), where oil production is treated over a longer time span instead of being instantaneous. Including the new variables could look like the following:

$$\min_{\substack{\mathbf{x} \in \mathbb{R}^N \\ \mathbf{y} \in \mathbb{R}^k}} \sum_t J(\mathbf{x}, \mathbf{y}, t), \tag{1.2}$$

where $\mathbf{x}$ are the well placement coordinates, $\mathbf{y}$ are the well control variables and t is a time variable.

How we solve for the well placement problem. Gradients of the objective function with respect to well placement variables are not readily available and are likely to be be discontinuous. As noted by Bellout et al. [2] the well placement problem does not appear to be as amenable using gradient-based methods because these approaches can get trapped in local minima. An alternative is to treat it using derivative-free approaches such as genetic algorithms, stochastic perturbation methods and particle swarm optimization.

Well placement constraints. It is important that the search for well configurations is constrained by realistic petroleum engineering considerations for how best to develop the field, e.g., knowledge about the geology and flow properties of the reservoir and information about existing or planned well paths and facilities. It is crucial for an efficient well placement optimization effort to articulate this type of information into a properly defined objective function with constraints that can be treated using mathematical programming. In order for our current well placement model to be practically useful there are several limitations to the placement of wells. These include, but are not restricted to, constraints on the length of a well (well length constraint), how close two wells can be to each other (inter-well distance constraint) and where a well is allowed to reside (well domain constraint). All of these restrictions on the wells result in a number of constraints on the well coordinates.

- Well length restrictions require that wells should not be too short but also not too long. We require that the heel $\mathbf{x}^h$ and toe $\mathbf{x}^t$ of every well should be separated by at least a distance $L_{\min}$ but not more than $L_{\max}$.
- The restriction on how close wells can be to each other results in the inter-well distance constraint. Essentially wells either interfere if they are placed too close to each other or it makes drilling either dangerous or impossible to perform. For all pairs of wells we require that every point of one well is at least a distance d away from every point of the other well.
- A well location restriction gives a domain constraint, which demands that a given well position be in some predefined feasible domain Ω_{wd}. Although this constraint looks simple enough, a domain Ω_{wd} may be arbitrarily defined and needs not be simple or convex, making projections on it difficult to find.

How do we solve for the constrained well placement problem. This thesis aims to contribute to the well placement problem by taking well coordinates from a single unconstrained optimization step and developing a way to project coordinates that violate constraints back into feasible space in such way that the coordinates of the wells are moved as little as possible. This process is

then continued iteratively so that after every unconstrained optimization step we project wells so that all constraints are satisfied. In addition an algorithm to calculate the well index for deviated wells (i.e., not strictly vertical) described by Shu in [5] is implemented. This calculation is currently not handled by FieldOpt itself but by the reservoir simulators.

Tasks for this thesis

- Determine and implement constraints that are physically reasonable.
- Implement a routine to deal with the constraint handling as an optimization problem. The routine projects non-feasible coordinates onto a feasible space.
- Implement well index calculation for deviated (i.e., not perfectly horizontal or vertical) wells.

In Chapter 2 we will introduce the well placement problem and formulate the well constraints in a more detailed manner, in addition we will explain some ideas for how to handle multiple constraints. In Chapter 3 we will solve the individual constraint problems and in Chapter 4 a method for computing the well index for blocks is outlined.

Chapter 2

Problem formulation

This chapter introduces the overall well placement optimization problem and how we represent wells. Moreover, we introduce the different constraints more clearly and indicate how we intend to handle several constraints at the same time. For the rest of this thesis we will assume that a well connects its heel and toe in a straight line. We describe well i with the coordinates of its heel and toe $(\mathbf{x}_i^h, \mathbf{x}_i^t) \in \mathbb{R}^3 \times \mathbb{R}^3$. If we have multiple wells we collect all N wells in a single variable $\mathbf{x} \in (\mathbb{R}^3 \times \mathbb{R}^3)^N$.

2.1 Well problem formulation

We define the overall well placement optimization problem as

$$\begin{aligned} &\min_{\mathbf{x} \in (\mathbb{R}^3 \times \mathbb{R}^3)^N} J(\mathbf{x}) \\ &\text{such that } \mathbf{x} \in \Omega, \end{aligned} \tag{2.1}$$

where $J : (\mathbb{R}^3 \times \mathbb{R}^3)^N \to \mathbb{R}$ is a user-defined objective function that maps the current well positions to a real number. The choice of J should maximize oil production or the net present value while minimizing various costs such as well drilling costs, well length costs and other factors. For now the exact definition of J is left open, as we concentrate on satisfying the constraints.

The domain Ω is the set of all well coordinates $\mathbf{x}$ that satisfy a set of linear and nonlinear constraints that enforce certain restrictions which we shall define below. We need to select, develop and implement constraints with the physical objective of drilling in mind. This means constraint types for the positioning of wells in a reservoir should be reasonable representations of engineering restrictions and priorities for how a petroleum field should be developed. To restrain overall well configuration in field development, in this work we define three types of constraints: a well length, an inter-well distance and a reservoir boundary constraint.

2.1.1 Well length constraint

First we must define a metric or distance function $g : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ which takes two points in three dimensional space and maps them to a real number. We use the most natural choice, namely the Euclidean distance. For two points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$ the distance between them is

$$g(\mathbf{p}, \mathbf{q}) := \|\mathbf{p} - \mathbf{q}\| = \sqrt{\sum_{i=1}^{3} (q_i - p_i)^2}. \tag{2.2}$$

A well should not be longer than $L_{\max}$ nor shorter than $L_{\min}$, or equivalently, the length should be in the interval $[L_{\min}, L_{\max}]$. Note that we will also require that $L_{\min} > 0$, i.e., we don't allow wells of zero length. The well length constraint can now be formulated as

$$\|\mathbf{x}_i^h - \mathbf{x}_i^t\| \leq L_{\max}, \tag{2.3}$$

$$\|\mathbf{x}_i^h - \mathbf{x}_i^t\| \geq L_{\min}, \tag{2.4}$$

$$\text{for all} \quad i = 1, 2, \ldots, N. \tag{2.5}$$

2.1.2 Inter-well distance constraint

Every pair of wells should be at least some minimum distance d apart. This means that every single point of one well needs to be at least a distance d from all points of every other well. If a well is the straight line between the heel and toe of the well then this is equivalent to requiring that

$$\|(\mathbf{x}_i^h + \lambda_1(\mathbf{x}_i^t - \mathbf{x}_i^h)) - (\mathbf{x}_j^h + \lambda_2(\mathbf{x}_j^t - \mathbf{x}_j^h))\| \geq d, \tag{2.6}$$

$$\lambda_1, \lambda_2 \in [0, 1], \tag{2.7}$$

$$\text{for all pairs } (i, j) \text{ of wells with } i \neq j. \tag{2.8}$$

2.1.3 Reservoir bound constraint

The reservoir is made up of grid blocks which are convex polyhedra, but the reservoir itself is not necessarily convex. All wells can be required to lie in a feasible domain, and heel and toe might have different feasible domains. Domain bounds should reflect the geological situation and it might also be natural to assume some restriction on heel position that is given by the drilling operator. If all wells lie in a feasible domain we say that $\mathbf{x}$ is feasible, or simply that $\mathbf{x} \in \Omega_{wd}$. Due to lack of time we were not able to define and solve a reservoir bound constraint, so for the rest of the thesis we will assume that all possible positions $\mathbf{x}$ satisfy the reservoir bound constraint.

Collecting all constraints we can rewrite equation (2.1) as

$$\min_{\mathbf{x}} J(\mathbf{x}) \tag{2.9}$$

such that

$$\|\mathbf{x}_i^h - \mathbf{x}_i^t\| \leq L_{\max}, \tag{2.10}$$

$$\|\mathbf{x}_i^h - \mathbf{x}_i^t\| \geq L_{\min}, \tag{2.11}$$

for all $i = 1, 2, \ldots, N$,

$$\|(\mathbf{x}_i^h + \lambda_1(\mathbf{x}_i^t - \mathbf{x}_i^h)) - (\mathbf{x}_j^h + \lambda_2(\mathbf{x}_j^t - \mathbf{x}_j^h))\| \geq d, \tag{2.12}$$

$$\lambda_1, \lambda_2 \in [0, 1], \tag{2.13}$$

for all pairs (i, j) of wells with $i \neq j,$.

2.2 Projection of multiple constraints

Given a set of well coordinates $\mathbf{x}_k$ and an objective function J, an unconstrained optimization step, O, is performed in order to achieve an improved position

$$O : \mathbf{x}_k \mapsto \tilde{\mathbf{x}}_{k+1}. \tag{2.14}$$

If a position $\mathbf{x}$ satisfies all constraints we say that $\mathbf{x} \in \Omega$. If the improved position does not satisfy all constraints then the coordinates need to be projected back to feasible space by some projection method $\mathcal{P}$. Ideally we want to find a projection method $\mathcal{P}$

$$\mathcal{P} : \tilde{\mathbf{x}}_{k+1} \mapsto \mathbf{x}_{k+1}, \tag{2.15}$$

that solves the problem

$$\min_{\mathbf{x}_{k+1}} \|\tilde{\mathbf{x}}_{k+1} - \mathbf{x}_{k+1}\|, \tag{2.16}$$

$$\text{such that} \quad \mathbf{x}_{k+1} \in \Omega, \tag{2.17}$$

i.e., a projection that moves a position back into feasible space by moving the points as little as possible.

2.3 Simultaneous constraint projection

Even if an analytical solution to (2.16) exists, i.e., solving the well length constraint and inter-well distance constraint at the same time, it is probably very difficult because both (2.11) and (2.12) are non-convex.

Using numerical solvers for constrained optimization, such as `fmincon()` in MATLAB [6], is problematic because of the implementation of the constraints. Especially the well distance constraint is pretty difficult to implement because of the piecewise definition of the closest points. Therefore we need to simplify our approach and look for a possibly (and probably) suboptimal solution of (2.16) if we wish find a working projection.

2.3.1 Method of alternating projections

Alternating projections is a standard approach for this kind of problem. If we know how to project onto the two sets C and D with the projections $\mathcal{P}_C$ and $\mathcal{P}_D$ respectively, then the alternating projection method is defined as

$$x_{k+1} = \mathcal{P}_C\left(\mathcal{P}_D(x_k)\right) \tag{2.18}$$

Moreover, if the sets C and D are convex and their intersection is non-empty, then the sequence (2.18) will converge to some point in this intersection.

Although we have no idea how to compute the whole projection analytically, we can still compute the individual projections onto the feasible sets for the well length constraint ($\mathcal{P}_l$), and the projection for the inter-well distance constraint for two wells ($\mathcal{P}_d$). The details will be discussed in Chapter 3.

Thus we can attempt to use the method of alternating projections to find a feasible point. Neither the feasible points for the well length constraints nor the feasible points for the inter-well distance constraint form convex sets, so we cannot guarantee any global convergence of the method.

However, in a result [7] by Lewis, Luke and Malick, [7, Thm. 5.16] states the following: If we have two sets, A and B, with A super-regular (see [7, Def. 4.3]) and B closed, and with non-opposing normal vectors to the sets at every point in their intersection, it then follows that the alternating projection converges locally R-linear to a point in $A \cup B$. From [7, Proposition 4.8] we have that amenability implies super regularity, and the remark one line earlier states that if A is defined by C^1 inequality constraints and the Mangasarian-Fromowitz constraint qualification [8, Def. 12.6] (or the stronger linear independence constraint qualification) holds, then A is amenable. By calculating the gradient of the well length constraint (2.11) we get that

$$\nabla\left(\frac{1}{2}\|x-y\|^2 - \frac{1}{2}L_{\min}^2\right) = \begin{bmatrix} x-y \\ y-x \end{bmatrix}, \tag{2.19}$$

which is non-zero for all $x \neq y$. Now since we require that wells have non-zero length, this implies that $x \neq y$. This means that the well length constriction satisfies the linear independence constraint condition which in turn implies that the set of feasible points for the well length constraint is super-regular. Therefore it follows that the alternating projection of the well length constraint and the inter-well distance constraint is locally convergent. Note, however, that this result for the projections $\mathcal{P}_l$ and $\mathcal{P}_d$ only holds for two wells.

2.3.2 Inter-well distance projection on more than two wells

The treatment of the inter-well distance constraint is particularly difficult for multiple wells because it puts requirements on every pair of wells, and thus it increases in complexity as the number of wells increases. In order to solve the inter-well distance constraint problem in a system with more than two wells, we apply the projection $\mathcal{P}_d$ to two wells at a time and hope that it eventually converges. Call this projection process $\mathcal{P}_m$.

2.3.3 Alternating projection pseudo code

Here we provide a pseudo code of the algorithm for the locally convergent alternating projection method for two wells. Note that by replacing $\mathcal{P}_d$ with $\mathcal{P}_m$ the code can also attempt to solve the projection problem for more than two wells.

Algorithm 1 Constraint handling

procedure PROJECT COORDINATES TO FEASIBLE SPACE
    Get initial coordinates $\mathbf{y} \leftarrow \mathbf{x}$

    while $\mathbf{x}_k$ not feasible **do**
        $\mathbf{y} \leftarrow \mathcal{P}_l(\mathbf{y})$
        $\mathbf{y} \leftarrow \mathcal{P}_d(\mathbf{y})$
    end while
    Return $\mathbf{y}$
end procedure

The complete iterative optimization process can be done by using the optimization step O and the projections $\mathcal{P}_l$ and $\mathcal{P}_d$. Note again that the process extends to handling multiple wells if we replace $\mathcal{P}_d$ with $\mathcal{P}_m$.

Algorithm 2 Iterative optimization method

procedure OPTIMIZE CONSTRAINED PROBLEM
    Get initial coordinates $\mathbf{x}_0$

    $k = 0$
    while $J(\mathbf{x}_k)$ not optimal **do**
        $\tilde{\mathbf{x}}_{k+1} \leftarrow O(\mathbf{x}_k)$
        while $\tilde{\mathbf{x}}_{k+1}$ not feasible **do**
            $\tilde{\mathbf{x}}_{k+1} \leftarrow \mathcal{P}_l(\tilde{\mathbf{x}}_{k+1})$
            $\tilde{\mathbf{x}}_{k+1} \leftarrow \mathcal{P}_d(\tilde{\mathbf{x}}_{k+1})$
        end while
        $\mathbf{x}_{k+1} \leftarrow \tilde{\mathbf{x}}_{k+1}$
        $k = k + 1$
    end while
    Return $\mathbf{x}_k$
end procedure

The solution for each individual projection will be handled in the next chapter.

Chapter 3

Constraint handling

Let $\mathbf{x}_{2k-1}$ and $\mathbf{x}_{2k}$ be the coordinates of heel and toe of well k respectively, and let $\boldsymbol{\xi}_{2k-1}$ and $\boldsymbol{\xi}_{2k-1}$ denote the initial coordinates (i.e., given as input) of heel and toe of well k. E.g., $\mathbf{x}_7$ is the heel of well number four and $\boldsymbol{\xi}_{20}$ are the initial coordinates of the toe of well number 10.

3.1 Well length constraint

Since the well length constraints for the different wells are independent of each other, we may compute their projections separately. Thus without loss of generality we may assume that $N = 1$, that is, we only deal with one well.

It is natural to assume that a well should have non-zero length and that the total length of one well should be allowed to vary. The distance between the heel and toe of a single well must be in the interval $[L_{\min}, L_{\max}]$. In other words they must be at least $L_{\min}$ apart but not further away from each other than $L_{\max}$. From the previous assumptions we get the constraints

$$\begin{aligned} \|\mathbf{x}_1 - \mathbf{x}_2\| &\geq L_{\min}, \\ \|\mathbf{x}_1 - \mathbf{x}_2\| &\leq L_{\max}, \end{aligned} \tag{3.1}$$

where the lengths $L_{\max}$ and $L_{\min}$ satisfy $L_{\max} > L_{\min} > 0$. If these conditions are not met by the initial input coordinates, $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ need to be projected back into feasible space by moving them as little as possible. This is done by solving

$$\min_{\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3} f(\mathbf{x}_1, \mathbf{x}_2) = \min_{\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3} \left(\frac{1}{2}\|\mathbf{x}_1 - \boldsymbol{\xi}_1\|^2 + \frac{1}{2}\|\mathbf{x}_2 - \boldsymbol{\xi}_2\|^2 \right) \tag{3.2}$$

subject to

$$h_1(\mathbf{x}_1, \mathbf{x}_2) = +\frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 - \frac{1}{2}L_{\min}^2 \geq 0, \tag{3.3}$$

$$h_2(\mathbf{x}_1, \mathbf{x}_2) = -\frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 + \frac{1}{2}L_{\max}^2 \geq 0, \tag{3.4}$$

where the constraints have been squared so they can be differentiated.
There are three kinds of solutions depending on the configuration of the initial positions of the heel and toe of the well. The initial positions $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ must either violate the $L_{\max}$ constraint, violate the $L_{\min}$ constraint or they satisfy both constraints. The solution for each initial configuration is given below and the proofs follow afterwards.

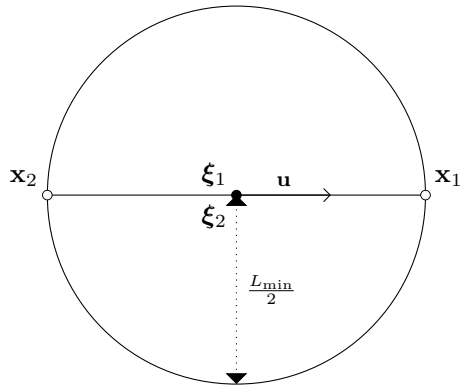


Figure 3.1: Minimum constraint violated. Initial points $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ are identical and solution points $\mathbf{x}$ and $\mathbf{y}$ lie on opposite sides of a circle with radius $\frac{L_{\min}}{2}$. Note that the solution shown in the figure is only one of the infinitely many solutions that exist for this case.

If $\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| < L_{\min}$ there are two solution types. If $\boldsymbol{\xi}_1 = \boldsymbol{\xi}_2$ then solutions are

$$\begin{aligned} \mathbf{x}_1 &= \boldsymbol{\xi}_1 + \frac{L_{\min}}{2}\mathbf{u}, \\ \mathbf{x}_1 &= \boldsymbol{\xi}_1 - \frac{L_{\min}}{2}\mathbf{u}, \end{aligned} \tag{3.5}$$

for any vector $\mathbf{u}$ of unit length. I.e., there is no unique solution and all solutions are pairs of points that lie on opposite sides of the circle with center $\boldsymbol{\xi}_1$ and radius $\frac{L_{\min}}{2}$. One such solution is shown in Figure 3.1.

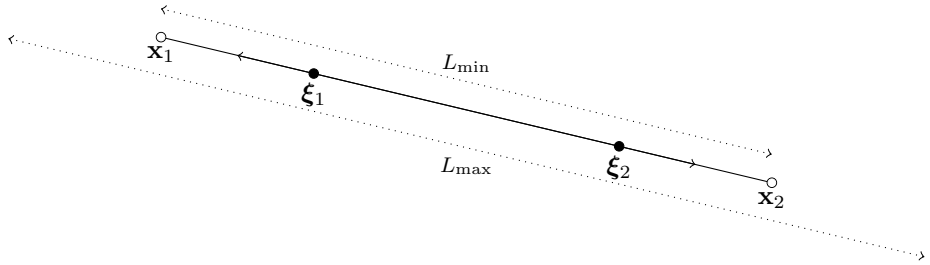


Figure 3.2: Minimum constraint violated. Initial points $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ are too close and the solution is to move the points away from each other along the line that passes through them in opposite directions until the distance between them is exactly $L_{\min}$.

If $\boldsymbol{\xi}_1 \neq \boldsymbol{\xi}_2$ then the solution is

$$\begin{aligned} \mathbf{x}_1 &= \boldsymbol{\xi}_1 + \frac{\lambda^*}{1-2\lambda^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \\ \mathbf{x}_2 &= \boldsymbol{\xi}_2 - \frac{\lambda^*}{1-2\lambda^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \end{aligned} \tag{3.6}$$

were $\lambda^* = \frac{1}{2}\left(1 - \frac{\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\|}{L_{\min}}\right)$. Here both points are moved away from each other an equal distance along the line that passes through $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ as shown in Figure 3.2.

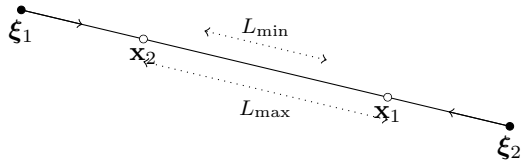


Figure 3.3: Maximum constraint violated. Initial points $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ are too far away from each other and the solution is to move the points closer along the line that passes through them until the distance between them is exactly $L_{\min}$.

If $\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| > L_{\max}$ the solution is given by

$$\begin{aligned} \mathbf{x}_1 &= \boldsymbol{\xi}_1 - \frac{\mu^*}{1+2\mu^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \\ \mathbf{x}_2 &= \boldsymbol{\xi}_2 + \frac{\mu^*}{1+2\mu^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \end{aligned} \tag{3.7}$$

where $\mu^* = \frac{1}{2}\left(\frac{\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\|}{L_{\max}} - 1\right)$. The geometric interpretation is that both points are moved closer to each other along the line that passes through $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ as

indicated in Figure 3.3.

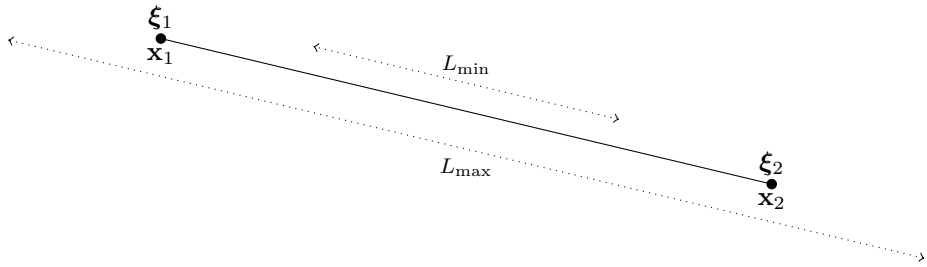


Figure 3.4: Constraints are satisfied by initial points and no movement of the points is needed.

If $L_{\min} \leq \|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| \leq L_{\max}$ the initial points satisfy both constraints and as shown in Figure 3.4 we need not move them. The solution in this case is

$$\begin{aligned} \mathbf{x}_1 &= \boldsymbol{\xi}_1, \\ \mathbf{x}_2 &= \boldsymbol{\xi}_2. \end{aligned} \tag{3.8}$$

To prove the formulae (3.5) – (3.8) consider the minimization problem (3.2) – (3.4), which can be solved by the method of Lagrange multipliers. Define the Lagrangian function

$$\begin{aligned} \mathcal{L}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \mu) = \frac{1}{2}\|\mathbf{x}_1 - \boldsymbol{\xi}_1\|^2 + \frac{1}{2}\|\mathbf{x}_2 - \boldsymbol{\xi}_2\|^2 \\ - \lambda\left(\frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 - \frac{1}{2}L_{\min}^2\right) - \mu\left(-\frac{1}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2 + \frac{1}{2}L_{\max}^2\right), \end{aligned} \tag{3.9}$$

where $\lambda, \mu \in \mathbb{R}$ are the Lagrange multipliers for the constraints. Note that only one of the constraints can be active at once, meaning that either λ or μ must be zero.
A necessary condition for a local minimum $\mathbf{x}_1^*, \mathbf{x}_2^*$ is that it satisfies the first order KKT conditions

$$\begin{aligned} \nabla_{\mathbf{x}_1,\mathbf{x}_2}\mathcal{L}(\mathbf{x}_1^*, \mathbf{x}_2^*, \lambda^*, \mu^*) &= 0, \\ \lambda h_1(\mathbf{x}_1^*, \mathbf{x}_2^*) &= 0, \\ \mu h_2(\mathbf{x}_1^*, \mathbf{x}_2^*) &= 0, \\ \lambda &\geq 0, \\ \mu &\geq 0, \\ h_i(\mathbf{x}_1^*, \mathbf{x}_2^*) &\geq 0, \qquad i = 1, 2. \end{aligned} \tag{3.10}$$

Differentiating the Lagrangian and setting the gradient to 0 we obtain the system

of equations

$$\begin{aligned} \nabla_{\mathbf{x}_1}\mathcal{L} &= \mathbf{x}_1 - \boldsymbol{\xi}_1 - \lambda(\mathbf{x}_1 - \mathbf{x}_2) + \mu(\mathbf{x}_1 - \mathbf{x}_2) = 0, \\ \nabla_{\mathbf{x}_2}\mathcal{L} &= \mathbf{x}_2 - \boldsymbol{\xi}_2 + \lambda(\mathbf{x}_1 - \mathbf{x}_2) - \mu(\mathbf{x}_1 - \mathbf{x}_2) = 0. \end{aligned} \tag{3.11}$$

The problem is solved by dividing the positions of the initial points, $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ into three different cases and then considering different candidate solutions $(\mathbf{x}_1^*, \mathbf{x}_2^*, \lambda^*, \mu^*)$. The three different categories for the initial position of the heel and toe of the well must either violate exactly one of the constraints, or violate none of them. The optimal solution of a configuration must satisfy the constraints, i.e., $L_{\min} \leq \|\mathbf{x}_1^* - \mathbf{x}_2^*\| \leq L_{\max}$. A solution must satisfy exactly one of the following equations

$$\begin{aligned} L_{\min} < \|\mathbf{x}_1^* - \mathbf{x}_2^*\| < L_{\max}, \\ \|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\min}, \\ \|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\max}. \end{aligned} \tag{3.12}$$

For each of the three cases for the initial positions of the well we will consider each of the three possibilities for the solution.

3.1.1 Case 1. Initial points feasible

Assume that $L_{\min} \leq \|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| \leq L_{\max}$. Then the initial points satisfy the distance constraint and no movement of the points is needed. Because the objective function f is non-negative and $f(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) = 0$ this solution is the global minimum.

3.1.2 Case 2. Initial points too close

Assume that $\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| < L_{\min}$. The distance between the initial points is too small and must be increased. According to (3.12) there are three different possibilities for the distance between the candidate solutions $\mathbf{x}_1^*, \mathbf{x}_2^*$. If

$$L_{\max} > \|\mathbf{x}_1^* - \mathbf{x}_2^*\| > L_{\min}, \tag{3.13}$$

i.e., the solution lies in the interior of both constraints. Then none of the constraints are active, which gives $\lambda^* = \mu^* = 0$ and therefore

$$\begin{aligned} \mathbf{x}_1^* &= \boldsymbol{\xi}_1, \\ \mathbf{x}_2^* &= \boldsymbol{\xi}_2. \end{aligned} \tag{3.14}$$

But this results in the contradiction $\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = \|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| \leq L_{\min}$, and thus the solution cannot satisfy (3.13).
If $\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\min}$, the maximum constraint is inactive which means that $\mu = 0$. Inserting this into equation (3.11) gives

$$\begin{aligned} \mathbf{x}_1 - \boldsymbol{\xi}_1 - \lambda(\mathbf{x}_1 - \mathbf{x}_2) = 0, \\ \mathbf{x}_2 - \boldsymbol{\xi}_2 + \lambda(\mathbf{x}_1 - \mathbf{x}_2) = 0, \end{aligned} \tag{3.15}$$

which is a linear system with respect to $\mathbf{x}_1$ and $\mathbf{x}_2$. Now if $\boldsymbol{\xi}_1 = \boldsymbol{\xi}_2$, which means the initial well has zero length, then it follows that $\lambda = \frac{1}{2}$ and we get

$$\mathbf{x}_1 + \mathbf{x}_2 = 2\boldsymbol{\xi}_1, \tag{3.16}$$
$$\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\min}. \tag{3.17}$$

This equation has solutions

$$\mathbf{x}_1 = \boldsymbol{\xi}_1 + \frac{L_{\min}}{2}\mathbf{u}, \tag{3.18}$$
$$\mathbf{x}_1 = \boldsymbol{\xi}_1 - \frac{L_{\min}}{2}\mathbf{u}, \tag{3.19}$$

for all unit vectors $\mathbf{u}$ and they are all KKT points. So if the initial well has zero length the solutions all lie on a circle with center $\boldsymbol{\xi}_2$ and radius $\frac{L_{\min}}{2}$. Now if $\boldsymbol{\xi}_1 \neq \boldsymbol{\xi}_2$, then solving the system (3.15) gives

$$\begin{aligned}
\mathbf{x}_1 &= \frac{1}{(1-\lambda)^2 - (\lambda)^2}\left((1-\lambda)\boldsymbol{\xi}_1 - \lambda\boldsymbol{\xi}_2\right) = \boldsymbol{\xi}_1 + \frac{\lambda}{1-2\lambda}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \\
\mathbf{x}_2 &= \frac{1}{(1-\lambda)^2 - (\lambda)^2}\left(-\lambda\boldsymbol{\xi}_1 + (1-\lambda)\boldsymbol{\xi}_2\right) = \boldsymbol{\xi}_2 - \frac{\lambda}{1-2\lambda}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2).
\end{aligned} \tag{3.20}$$

Combining this result with the condition that $\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\min}$ we obtain

$$\lambda^* = \frac{1}{2}\left(1 \pm \frac{\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\|}{L_{\min}}\right), \tag{3.21}$$

where both candidates for λ^* are positive and therefore yield KKT points. From (3.15) we have

$$f(\mathbf{x}_1, \mathbf{x}_2) = \lambda^2\|\mathbf{x}_1 - \mathbf{x}_2\|^2, \tag{3.22}$$

and thus the minimum is attained for the smaller candidate. The best KKT point is therefore given by

$$\begin{aligned}
\mathbf{x}_1^* &= \boldsymbol{\xi}_1 + \frac{\lambda^*}{1-2\lambda^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2), \\
\mathbf{x}_2^* &= \boldsymbol{\xi}_2 - \frac{\lambda^*}{1-2\lambda^*}(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2),
\end{aligned} \tag{3.23}$$

where $\lambda^* = \frac{1}{2}\left(1 - \frac{\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\|}{L_{\min}}\right)$.

The last possibility is that $L_{\max} = \|\mathbf{x}_1^* - \mathbf{x}_2^*\|$. The derivation is similar to the previous case. The maximum constraint is active so therefore the other constraint is inactive and hence $\lambda = 0$. If $\boldsymbol{\xi}_1 = \boldsymbol{\xi}_2$ then we get infinitely many solutions

$$\mathbf{x}_1 + \mathbf{x}_2 = 2\boldsymbol{\xi}_1, \tag{3.24}$$
$$\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\max}, \tag{3.25}$$

but since these all lie on a circle with center in $\boldsymbol{\xi}_1$ and radius $\frac{L_{\max}}{2}$ the solutions are all worse than the ones in (3.19) so none of them can be a global minimum.

$$\begin{aligned}\mathbf{x}_1 &= \frac{1}{(1+\mu)^2-(\mu)^2}\left((1+\mu)\boldsymbol{\xi}_1+\mu\boldsymbol{\xi}_2\right) = \boldsymbol{\xi}_1 - \frac{\mu}{1+2\mu}(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2),\\ \mathbf{x}_2 &= \frac{1}{(1+\mu)^2-(\mu)^2}\left(\mu\boldsymbol{\xi}_1+(1+\mu)\boldsymbol{\xi}_2\right) = \boldsymbol{\xi}_2 + \frac{\mu}{1+2\mu}(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2).\end{aligned} \tag{3.26}$$

Using the fact that $L_{\max} = \|\mathbf{x}_1^* - \mathbf{x}_2^*\|$ and solving for μ results in

$$\mu = -\frac{1}{2}\left(1 \pm \frac{\|\boldsymbol{\xi}_1-\boldsymbol{\xi}_2\|}{L_{\max}}\right) < 0, \tag{3.27}$$

which are not solutions since the Lagrange multiplier in both cases is negative. Now it follows that, since there are only two points that satisfy the KKT conditions, the better one has to be the global optimum, and thus the solution for Case 2 is given by (3.23)

3.1.3 Case 3. Initial points too distant

The initial condition is that $\|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2\| > L_{\max}$, which means that the initial points are too far away from each other and need to be moved closer to each other. The solutions for this case are analogous to Case 2 and the calculations will be omitted. The only valid solution is found when $\|\mathbf{x}_1^* - \mathbf{x}_2^*\| = L_{\max}$ which results in $\mu^* = \frac{1}{2}\left(-1 + \frac{\|\boldsymbol{\xi}_1-\boldsymbol{\xi}_2\|}{L_{\max}}\right)$ and $\lambda^* = 0$. This gives the solution

$$\begin{aligned}\mathbf{x}_1^* &= \boldsymbol{\xi}_1 - \frac{\mu^*}{1+2\mu^*}(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2),\\ \mathbf{x}_2^* &= \boldsymbol{\xi}_2 + \frac{\mu^*}{1+2\mu^*}(\boldsymbol{\xi}_1-\boldsymbol{\xi}_2).\end{aligned} \tag{3.28}$$

3.2 Inter-well distance constraint

Minimize the movement of the endpoints of two line segments such that every point on the first line segment is at least some distance d away from every point on the other line segment. Let the initial positions of the two line segments be defined by their endpoints $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3, \boldsymbol{\xi}_4 \in \mathbb{R}^3$ respectively. Minimizing the movement is the solution to the problem

$$\min_{\mathbf{x}_i \in \mathbb{R}^3} F(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \min_{\mathbf{x}_i \in \mathbb{R}^3} \sum_{i=1}^{4} \frac{1}{2}\|\mathbf{x}_i - \boldsymbol{\xi}_i\|^2 \tag{3.29}$$

under the conditions that

$$c(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \lambda_1, \lambda_2) \geq \frac{1}{2}d^2 \tag{3.30}$$

for all

$$\lambda_i \in [0,1], \quad i = 1,2, \tag{3.31}$$

where

$$c(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \lambda_1, \lambda_2) = \frac{1}{2}\|(\mathbf{x}_1 + \lambda_1(\mathbf{x}_2 - \mathbf{x}_1)) - (\mathbf{x}_3 + \lambda_2(\mathbf{x}_4 - \mathbf{x}_3))\|^2. \tag{3.32}$$

Call a solution in which k points are moved a k-point solution. We will first try to solve (3.29) – (3.31) by moving only two points, then by moving just three points, and lastly, if a solution is not yet found, move all four points. The idea is that, if the optimal solution for moving two or three points while temporarily ignoring the other point(s) satisfies all the constraints in (3.31), then this must be the optimal solution to (3.29).
To see why this is true, notice that in general if $\Omega_1 \subset \Omega_2$, then

$$\min_{x \in \Omega_1} f(x) \geq \min_{x \in \Omega_2} f(x) \tag{3.33}$$

holds for all real valued functions f. Thus if we have that

$$f(x^*) = \min_{x \in \Omega_2} f(x) \quad \text{and} \quad x^* \in \Omega_1, \tag{3.34}$$

then x^* is feasible for the first problem and

$$f(x^*) \leq f(x) \quad \forall x \in \Omega_1, \tag{3.35}$$

which means that x^* is the solution to both minimization problems, i.e.,

$$\min_{x \in \Omega_1} f(x) = \min_{x \in \Omega_2} f(x) = f(x^*). \tag{3.36}$$

Now ignoring one of the four points is the same as setting either λ_1 or λ_2 equal to 1 or 0 in (3.31). E.g., if we wish to only consider the points $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_4$ but ignore $\mathbf{x}_3$ this is done by letting $\lambda_2 = 1$. Call the set of feasible points for a k-point problem $\tilde{\Omega}_k$. Clearly we must have $\tilde{\Omega}_4 \subset \tilde{\Omega}_3 \subset \tilde{\Omega}_2 \subset \tilde{\Omega}_1$. Then by (3.33) – (3.36) we have that the k-point solution with the smallest value for k which is feasible in the four point problem, must also be the solution to the four point problem.

3.2.1 Number of points moved and form of solutions

A line segment that connects the two solution line segments, S_1, S_2, over the shortest distance will be called SD (Shortest Distance). If only one of the two line segments is needed in a figure then, without loss of generality, we will refer to this line segment as S_1 with endpoints $\mathbf{x}_1$ and $\mathbf{x}_2$. Divide the solutions of the problem into different categories depending on how many of the points have been moved. Denote the smallest angle between the SD and S_i by α_i. Note that SD is not unique if S_1 and S_2 are parallel, but the angles α_1 and α_2 are. We must

also have that $\alpha_i \geq 90°$. If one angle is below 90° then there exists a shorter distance between the line segments by moving the SD along the line segment in the direction of the angle as indicated by the arrow in Figure 3.5.

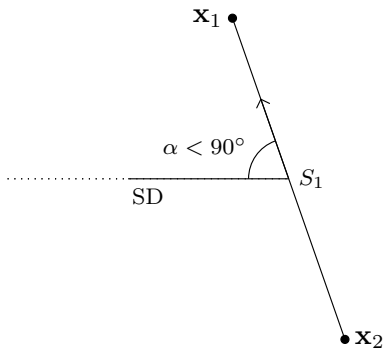


Figure 3.5: Candidate for shortest distance (SD). A shorter distance between the segments can be found by moving upwards along the line segment.

If one angle α is over 90 degrees, then the SD must connect to an endpoint of the corresponding line segment (or else SD could be improved by moving it along the line segment) as shown in Figure 3.6.

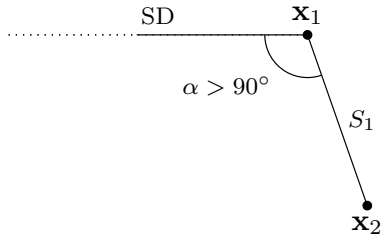


Figure 3.6: One angle over 90 degrees. Moving $\mathbf{x}_2$ does not change the length of SD.

Assume that this is the case. Without loss of generality we call this endpoint $\mathbf{x}_1$ and the other endpoint of the line segment $\mathbf{x}_2$. Then moving $\mathbf{x}_2$ a small distance does not change the shortest distance. It follows that $\mathbf{x}_2 = \boldsymbol{\xi}_2$ because no constraints are active for $\mathbf{x}_2$ and this cannot be a solution where all four points have been moved. Therefore, in a solution where all four points are moved, both of the angles must be 90 degrees. If we have a case where both angles are over 90 degrees as shown in Figure 3.7, then the SD connects two endpoints,

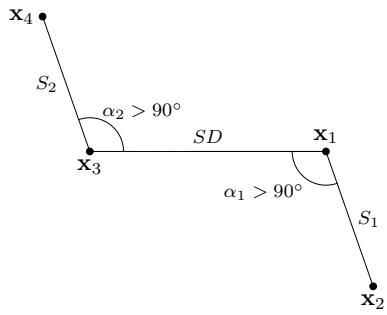


Figure 3.7: One angle over 90 degrees

and by applying the same argument as above to both line segments we see this is a solution where at most two points have been moved. Thus it cannot be a three-point solution. Therefore a three-point solution must have one angle α equal to 90° and the other one larger than 90°. If we are left with both angles larger than 90° then we must have moved either two points or no points.

3.2.2 Four-point solution

Denote the initial points $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3$ and $\boldsymbol{\xi}_4$ and the solution points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$. In the solution the SD will be orthogonal to both line segments. This means that in the solution the line segments will lie in two planes E^1 and E^2 that share the same normal vector (namely SD). This means that E^1 and E^2 are parallel and a distance d apart. Let

$$E^0 = E_{\mathbf{s},t} = \{\mathbf{x} : \langle \mathbf{s}, \mathbf{x} \rangle = t\} \tag{3.37}$$

be the plane that lies between the two solution planes with $\mathbf{s}$ the normalized SD vector (pointing towards the line segment with endpoints $\mathbf{x}_1, \mathbf{x}_2$) and $t \in \mathbb{R}$.

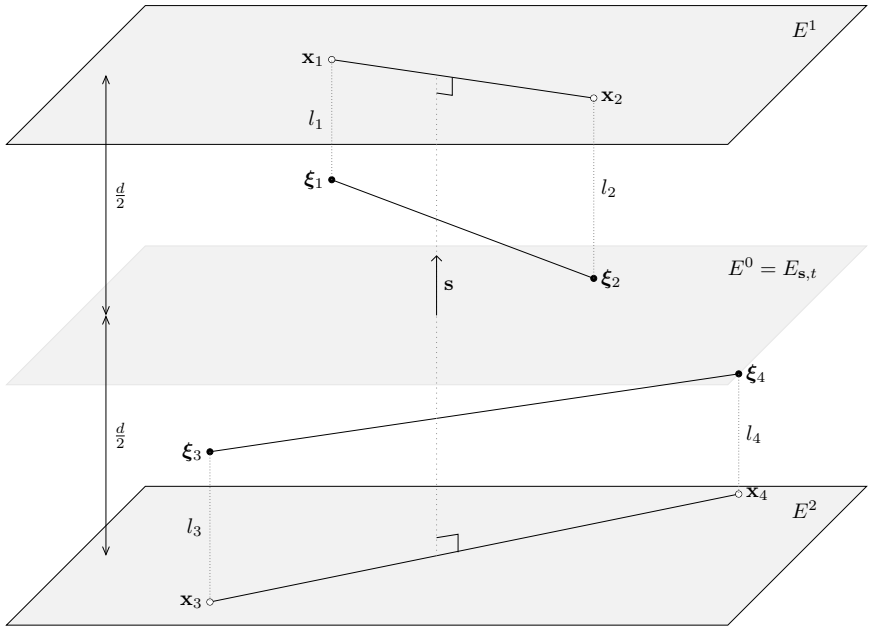


Figure 3.8: Four point solution. All four initial points have been moved and the resulting shortest distance line is orthogonal to both solutions. Project initial points down on planes to get the optimal solution.

If the two planes E^1 and E^2 are found, then the solution is simply the shortest distance from the initial points to the planes, which is found by projecting $\mathbf{x}_1$ and $\mathbf{x}_2$ onto $E^1 = E_{\mathbf{s},t+\frac{d}{2}}$ and by projecting $\mathbf{x}_3$ and $\mathbf{x}_4$ onto $E^2 = E_{\mathbf{s},t-\frac{d}{2}}$. Denote

$$\begin{aligned} l_1(\mathbf{s},t) &= \langle \mathbf{s}, \boldsymbol{\xi}_1 \rangle - t - \frac{d}{2}, \\ l_2(\mathbf{s},t) &= \langle \mathbf{s}, \boldsymbol{\xi}_2 \rangle - t - \frac{d}{2}, \\ l_3(\mathbf{s},t) &= \langle \mathbf{s}, \boldsymbol{\xi}_3 \rangle - t + \frac{d}{2}, \\ l_4(\mathbf{s},t) &= \langle \mathbf{s}, \boldsymbol{\xi}_4 \rangle - t + \frac{d}{2}, \end{aligned} \tag{3.38}$$

then $|l_i(\mathbf{s},t)| = \|\mathbf{x}_i - \boldsymbol{\xi}_i\|$. Thus we can rewrite the minimization problem (3.29) and solve for $\mathbf{s}$ and t by

$$\min_{\mathbf{x}_i \in \mathbb{R}^3} F(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \min_{\mathbf{s} \in \mathbb{S}^2, t \in \mathbb{R}} \frac{1}{2} \sum_{i=1}^{4} l_i(\mathbf{s},t)^2. \tag{3.39}$$

Holding $\mathbf{s}$ fixed and minimizing F with respect to t gives the condition

$$\begin{aligned}\frac{\partial F}{\partial t}(\mathbf{s},t) =&\langle \mathbf{s}, \boldsymbol{\xi}_1\rangle - t - \frac{d}{2} + \langle \mathbf{s}, \boldsymbol{\xi}_2\rangle - t - \frac{d}{2} && (3.40)\\ &+ \langle \mathbf{s}, \boldsymbol{\xi}_3\rangle - t + \frac{d}{2} + \langle \mathbf{s}, \boldsymbol{\xi}_4\rangle - t + \frac{d}{2} && (3.41)\\ =&\sum_{i=1}^{4}\langle \boldsymbol{\xi}_i, \mathbf{s}\rangle - 4t = 0, && (3.42)\end{aligned}$$

and therefore

$$t = \frac{1}{4}\sum_{i=1}^{4}\langle \boldsymbol{\xi}_i, \mathbf{s}\rangle. \tag{3.43}$$

We simplify the problem by introducing the shifted variables

$$\hat{\boldsymbol{\xi}}_i = \boldsymbol{\xi}_i - \frac{1}{4}\sum_{i=1}^{4}\boldsymbol{\xi}_i. \tag{3.44}$$

We then get the identity

$$\begin{aligned}\langle \mathbf{s}, \boldsymbol{\xi}_i\rangle &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_i\rangle + \langle \mathbf{s}, \boldsymbol{\xi}_i - \hat{\boldsymbol{\xi}}_i\rangle\\ &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_i\rangle + \Big\langle \mathbf{s}, \frac{1}{4}\sum_{i=1}^{4}\boldsymbol{\xi}_i\Big\rangle\\ &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_i\rangle + \frac{1}{4}\sum_{i=1}^{4}\langle \mathbf{s}, \boldsymbol{\xi}_i\rangle.\end{aligned} \tag{3.45}$$

For the shifted variables inserted into (3.38) the variable t is eliminated and equation (3.39) can be rewritten as

$$\min_{\mathbf{s}\in\mathbb{S}^2, t\in\mathbb{R}} F(\mathbf{s},t) = \min_{\mathbf{s}\in\mathbb{S}^2}\frac{1}{2}\sum_{i=1}^{4}\hat{l}_i(\mathbf{s})^2 \tag{3.46}$$

with

$$\begin{aligned}\hat{l}_1(\mathbf{s},t) &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_1\rangle - \frac{d}{2},\\ \hat{l}_2(\mathbf{s},t) &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_2\rangle - \frac{d}{2},\\ \hat{l}_3(\mathbf{s},t) &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_3\rangle + \frac{d}{2},\\ \hat{l}_4(\mathbf{s},t) &= \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_4\rangle + \frac{d}{2}.\end{aligned} \tag{3.47}$$

We differentiate F with respect to $\mathbf{s}$ to get

$$\frac{\partial F}{\partial \mathbf{s}}(\mathbf{s}) = \sum_{i=1}^{4}(\hat{\boldsymbol{\xi}}_i \otimes \hat{\boldsymbol{\xi}}_i)\mathbf{s} - \frac{d}{2}(\hat{\boldsymbol{\xi}}_1 + \hat{\boldsymbol{\xi}}_2) + \frac{d}{2}(\hat{\boldsymbol{\xi}}_3 + \hat{\boldsymbol{\xi}}_4). \tag{3.48}$$

Let

$$A = \sum_{i=1}^{4} \hat{\boldsymbol{\xi}}_i \otimes \hat{\boldsymbol{\xi}}_i \quad \text{and} \quad b = \frac{d}{2}(\hat{\boldsymbol{\xi}}_1 + \hat{\boldsymbol{\xi}}_2 - \hat{\boldsymbol{\xi}}_3 - \hat{\boldsymbol{\xi}}_4). \tag{3.49}$$

With the constraint $\|\mathbf{s}\|^2 = 1$ we get the necessary KKT conditions

$$\begin{aligned} (A - \mu I)\mathbf{s} &= b, \\ \|\mathbf{s}\|^2 &= 1, \end{aligned} \tag{3.50}$$

where $\mu \in \mathbb{R}$ is a Lagrange parameter. Either μ is an eigenvalue of A, or $A - \mu I$ is invertible. Assume first that μ is not an eigenvalue of A. This means that $A - \mu I$ is invertible and we can write

$$\mathbf{s} = (A - \mu I)^{-1} b. \tag{3.51}$$

Since A is symmetric it can be diagonalized and written as $A = QDQ^T$ with orthogonal matrix Q and a diagonal matrix D containing the eigenvalues of A. Inserting this into (3.50) gives

$$\mathbf{s} = (A - \mu I)^{-1} b = Q(D - \mu I)^{-1} Q^T b. \tag{3.52}$$

Take the norm (orthogonal matrices do not change the norm of a vector) of both sides to get

$$\|(D - \mu I)^{-1} Q^T b\|^2 = 1, \tag{3.53}$$

or equivalently

$$\sum_{i=1}^{3} \frac{1}{(D_i - \mu)^2} (Q^T b)_i^2 = 1. \tag{3.54}$$

The result is a sixth degree equation in μ which can have up to six distinct solutions, all of which satisfy the KKT conditions.
If μ is an eigenvalue of A then $A - \mu I$ is not invertible and

$$(A - \mu I)\mathbf{s} = b \tag{3.55}$$

has either no solutions or infinitely many solutions. If solutions exist assume that $\mathbf{s}_0$ solves (3.55). Then $\ker(A - \mu I) + \mathbf{s}_0$ is the space of all solutions to (3.55). So if there exists solutions to (3.55) we only need to find a single solution $\mathbf{s}_0$ and $\ker(A - \mu I)$. By requiring that

$$\|\mathbf{s}\|^2 = 1, \tag{3.56}$$

we obtain that the solutions space is the intersection of $\mathbb{S}^2$ with either a line, a plane or $\mathbb{R}^3$. The solution space depends on the multiplicity of the eigenvalues of $(A - \mu I)$ and the vector b. All solutions for μ are KKT points, but they need not all be local minima. The vector $\mathbf{s}$ is found by substituting μ back into equation (3.52), and then the best solution can be found by projecting the initial points to the planes as shown in Figure 3.8 and comparing different values of F for each configuration.

3.2.3 Three-point solution

Assume that the initial coordinate $\boldsymbol{\xi}_1$ belongs to line segment S_1 and that the coordinates $\boldsymbol{\xi}_2$ and $\boldsymbol{\xi}_2$ belong to the other line segment S_2. In the solution the SD will start in $\mathbf{x}_1$ and be orthogonal to the line segment defined by $\mathbf{x}_2$ and $\mathbf{x}_3$. The solution for this case is analogous to the four point case. Again we have the two planes E^1 and E^2, and we also have that $\mathbf{x}_1 \in E^1$ and $\mathbf{x}_2, \mathbf{x}_3 \in E^2$ as shown in Figure 3.9.

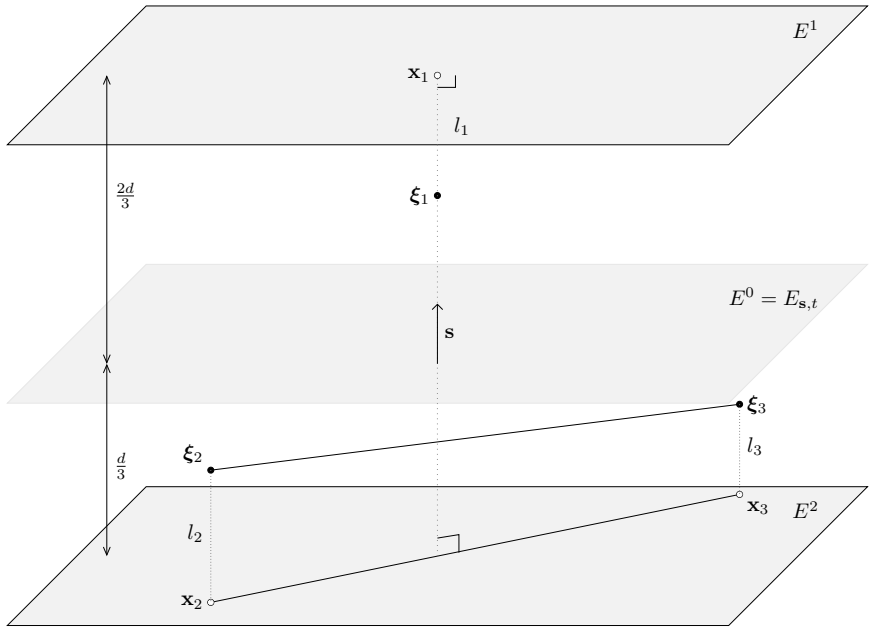


Figure 3.9: Three point problem

We solve

$$\min_{\mathbf{s}\in\mathbb{S}^2, t\in\mathbb{R}} F(\mathbf{s},t) = \min_{\mathbf{s}\in\mathbb{S}^2, t\in\mathbb{R}} \frac{1}{2}\sum_{i=1}^{3} l_i(\mathbf{s},t)^2, \tag{3.57}$$

where the lengths of the projections, l_i, are given by

$$l_1(\mathbf{s},t) = \langle \mathbf{s}, \boldsymbol{\xi}_1 \rangle - t - \frac{2d}{3}, \tag{3.58}$$

$$l_2(\mathbf{s},t) = \langle \mathbf{s}, \boldsymbol{\xi}_2 \rangle - t + \frac{d}{3}, \tag{3.59}$$

$$l_3(\mathbf{s},t) = \langle \mathbf{s}, \boldsymbol{\xi}_3 \rangle - t + \frac{d}{3}. \tag{3.60}$$

Again we minimize F with respect to t to get

$$t = \frac{1}{3}\sum_{i=1}^{3}\langle \boldsymbol{\xi}_i, \mathbf{s}\rangle. \tag{3.61}$$

Introducing the shifted variables

$$\hat{\boldsymbol{\xi}}_i = \boldsymbol{\xi}_i - \frac{1}{3}\sum_{i=1}^{3}\boldsymbol{\xi}_i \tag{3.62}$$

and using the value for t found in (3.61) and inserting these into (3.57) we obtain the simplified problem

$$\min_{\mathbf{s}\in\mathbb{S}^2, t\in\mathbb{R}} F(\mathbf{s}, t) = \min_{\mathbf{s}\in\mathbb{S}^2} \frac{1}{2}\sum_{i=1}^{3}\hat{l}_i(\mathbf{s})^2, \tag{3.63}$$

where

$$\hat{l}_1(\mathbf{s}, t) = \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_1\rangle - \frac{2d}{3}, \tag{3.64}$$

$$\hat{l}_2(\mathbf{s}, t) = \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_2\rangle + \frac{d}{3}, \tag{3.65}$$

$$\hat{l}_3(\mathbf{s}, t) = \langle \mathbf{s}, \hat{\boldsymbol{\xi}}_3\rangle + \frac{d}{3}. \tag{3.66}$$

We differentiate F with respect to $\mathbf{s}$ which gives

$$\frac{\partial F}{\partial \mathbf{s}}(\mathbf{s}) = \sum_{i=1}^{3}(\hat{\boldsymbol{\xi}}_i \otimes \hat{\boldsymbol{\xi}}_i)\mathbf{s} - \frac{2d}{3}\hat{\boldsymbol{\xi}}_1 + \frac{d}{3}(\hat{\boldsymbol{\xi}}_2 + \hat{\boldsymbol{\xi}}_3). \tag{3.67}$$

A vector $\mathbf{s}$ vector satisfies the necessary KKT conditions if it solves (3.50) with

$$A = \sum_{i=1}^{3}\hat{\boldsymbol{\xi}}_i \otimes \hat{\boldsymbol{\xi}}_i \quad \text{and} \quad b = \frac{2d}{3}\hat{\boldsymbol{\xi}}_1 - \frac{d}{3}(\hat{\boldsymbol{\xi}}_2 - \hat{\boldsymbol{\xi}}_3). \tag{3.68}$$

Solving (3.50) can be done in exactly the same way as described in the four-point case.

3.2.4 Two-point solution

If the two points are too close to each other they need to be moved as little as possible away from each other so that they are a minimum distance d apart. This problem is identical to solving the well length constraint in equation (3.1) with $L_{\min} = d$ and $L_{\max} = \infty$.

3.2.5 Complete inter-well distance constraint solution

We have solved the two-, three-, and four-point problems individually and combining them is the solution to the complete inter-well distance problem. Below follows a pseudo code of the algorithm that solves the complete inter-well distance problem. Note that by point we mean an endpoint of a line segment.

Algorithm 3 Inter-well distance projection

```
1: procedure PROJECT LINE SEGMENTS TO FEASIBLE SPACE
2:     Get four initial points
3:
4:     for all subsets with two points from different line segments do
5:         Optimal projection of two points
6:         if Four point configuration satisfies inter-well constraint then
7:             Calculate movement cost and save configuration
8:         end if
9:     end for
10:
11:    if any two point solution satisfies constraints then
12:        return best two point solution
13:    end if
14:
15:    for all subsets with three points do
16:        Optimal projection of three points
17:        if Four point configuration satisfies inter-well constraint then
18:            Calculate movement cost and save configuration
19:        end if
20:    end for
21:
22:    if any three point solution satisfies constraints then
23:        return best three point solution
24:    end if
25:
26:    Optimal projection of four points
27:    return four point solution
28: end procedure
```

Chapter 4

Well index calculation

The well index is an essential part of well placement optimization. In a reservoir simulation, the flowing bottom hole pressure differs from the measured well block pressure. In order to connect the two D. Peaceman introduced the well index, or well transmisibillity factor, and suggested a way to calculate it. We refer to Peaceman's paper [9] for a more in-depth explanation on the topic. The bottom hole pressure determines the rate of flow through the well, which again determines the production rate of the well. So because the well index determines the production rate, the objective function J greatly depends on it. This means that, if we wish to develop a good algorithm for the well placement problem optimization, it is essential to be able to calculate the well index for all well blocks. Peaceman's original paper only considered horizontal wells, but new algorithms for slanted (i.e., not fully horizontal) wells have been developed, three of which are described by Shu in his report [5].

4.1 Projection well index

In this thesis we will use the projection well index, originally developed by Jonathan Holmes [10], which is described by Shu in Chapter 2 of his report. The main assumptions of the model is a uniform Cartesian grid with single-phase radial flow without interaction with boundaries or other wells. In the projection well index method, the well trajectory is projected onto three orthogonal axes as shown in Figure 4.1.

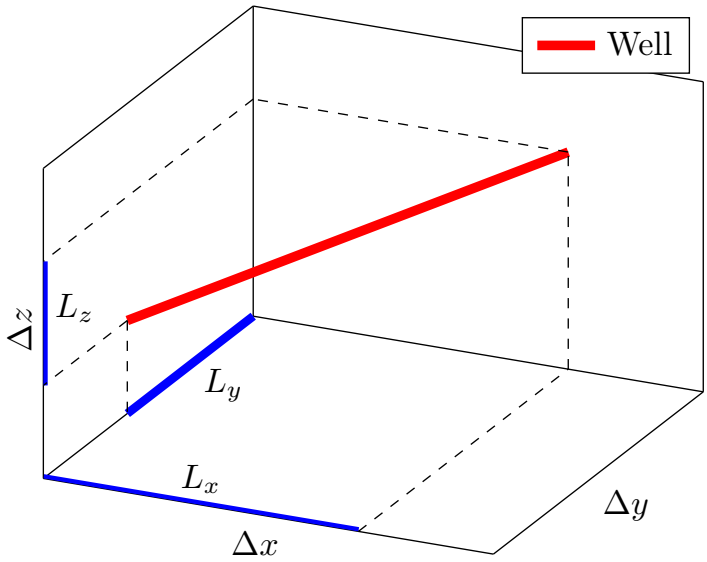


Figure 4.1: Well trajectory in a single well block projected onto the three axes. The projections have lengths L_x, L_y and L_z, and the well block has dimensions Δ_x, Δ_y and Δ_z. This figure is a copy of the one used by Shu in his report [5]

The well index can now be calculated for slanted wells and non-square blocks by first calculating the well index for each direction with

$$WI_x = \frac{2\pi\sqrt{k_y k_z}L_x}{\ln\frac{r_{0,x}}{r_w} + s}, \quad WI_y = \frac{2\pi\sqrt{k_x k_z}L_y}{\ln\frac{r_{0,y}}{r_w} + s} \quad \text{and} \quad WI_z = \frac{2\pi\sqrt{k_x k_y}L_z}{\ln\frac{r_{0,z}}{r_w} + s}, \tag{4.1}$$

where k_i is the permeability in direction i, r_w is the wellbore radius, s is the skin factor and

$$r_{0,x} = 0.28\frac{\left(\Delta z^2\left(\frac{k_y}{k_z}\right)^{\frac{1}{2}} + \Delta y^2\left(\frac{k_z}{k_y}\right)^{\frac{1}{2}}\right)^{\frac{1}{2}}}{\left(\frac{k_y}{k_z}\right)^{\frac{1}{4}} + \left(\frac{k_z}{k_y}\right)^{\frac{1}{4}}}. \tag{4.2}$$

The functions $r_{0,y}$ and $r_{0,y}$ are defined in the same manner but with the indices x, y and z shifted. At last we take the square root of the sum of the squares of the directional well indices to get the well index for the block

$$WI = \sqrt{WI_x^2 + WI_y^2 + WI_z^2}. \tag{4.3}$$

For the rest of this thesis we will simplify the equation by setting $s = 0$, i.e., neglecting the skin factor.

4.2 Computing well trajectory and projection

In order to compute the well index with (4.3) we need not only determine which blocks are penetrated by the wells, but also determine the point of entry and exit. This is done by first using a function called `GetblockEnvelopingPoint()`. This function simply iterates over all the well blocks of the reservoir and returns the first block that contains a given point. This is done by checking that the point is on the correct side (i.e., towards the center of the well block) of every face of the block. Using this function on the start point will give us the well block that contains the starting point as shown in Figure 4.2.

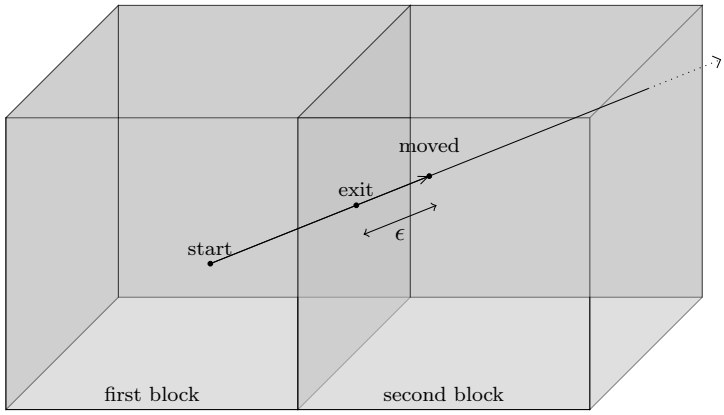


Figure 4.2: Moving a small distance ϵ along the line segment towards the end of the line segment will ensure that we are completely inside the next block. In the new block we can perform the same steps as we did in the previous block.

We then find the intersection point between the well and this first block, which we will call the exit point of the well. Now we have acquired the intersection points for the first well block. After this we simply move a sufficiently small distance ϵ from this exit point towards the end of the well, and call this new point moved point. With sufficiently small we mean a small fraction ($\sim \frac{1}{100}$) of the shortest well side length. Note that this could lead to numerical errors as we might jump over some blocks if the intersection between the well and the block is very short. But if the length of the intersection is very short, then value of the corresponding well index will be very small as well, thus we can neglect it. Now we can use the same method as we did for the starting point to get the intersection points of the second block, and this process can be repeated until we reach the end of the well. Because the corner coordinates of the well block are known, we can compute the lengths Δ_x, Δ_y and Δ_z for each side of the block. Then we can find the unit vector $\mathbf{u}_i$ for the direction of the sides of the well block. Thus given a well block and its two intersection points, $\mathbf{p}_1$ and $\mathbf{p}_2$, the projected well lengths L_x, L_y and

L_z are given by

$$L_i = \big\langle (\mathbf{p}_2 - \mathbf{p}_1), \mathbf{u}_i \big\rangle. \tag{4.4}$$

Supplying the permeabilities k_i and wellbore radius r_w and in the absence of skin ($s = 0$), we can compute the well index of the block with (4.3).

Chapter 5

Implementation

Here we briefly explain the implementations that were made and the most important part of the code can be seen in the Appendix. For the full code we refer to the author's Github repository [11].

5.1 Software

All code was written in Qt 5.5, a cross-platform application and UI framework for C++ developed by The Qt Company [12]. The code makes great use of the Eigen library [13] which is a template library for linear algebra that includes matrix and vector classes and algorithms for matrix decompositions. This is required in the implementation of the inter-well distance projection for finding the eigenvalue decomposition needed in (3.52). The vector class `Vector3d` of the Eigen library is also used frequently throughout most implementations as it contains many useful functions. In order to find the roots of polynomials we make use of the RPOLY library [14], which is an implementation of the Jenkins-Traub algorithm [15]. This is needed for solving the sixth degree equation (3.54) which is the most important part of the inter-well distance projection.

5.2 Well length constraint projection

The function `well_length_projection_eigen()`, takes the initial coordinates of the heel and toe of a well, and by calculating the distance between them it determines which of the solutions (3.5) – (3.8) it should return.

5.3 Inter-well distance constraint projection

In the function `interwell_constraint_projection_eigen()` which was implemented we take the initial coordinates of two wells and a distance d as input. Then

we try to find solutions by moving as few points as possible. First we try to move only two points by using `well_length_projection_eigen()`. If some two-point solution is feasible for the complete problem we return the best one. If no two-point solution is found we try the three-point solutions. This is done by building A and b according to (3.68), and then running `kkt_eq_solutions_eigen(`A, b`)` which returns all candidate solutions of (3.50). If one or more feasible solutions are found we pick the best one. If no solution has been found yet we must have a four-point solution. We build A and b according to (3.68) and pick the best solution returned by `kkt_eq_solutions_eigen(`A, b`)`.

5.4 Alternating projections

Now that both well length projection and inter-well distance projection are available, the method of alternating projections is simply done by running one projection after the other inside a while loop until the well positions are feasible. Note that we can also change the order of the projections, which might impact the solution. In our implementation the inter-well distance projection was done first.

5.5 Well index calculation and intersecting blocks

In order to compute the well indices for the well blocks of a reservoir, we first need to determine which blocks are penetrated by a well and what the entry and exit points are. If these two steps are handled, then computing the well index for each block is done by computing the well block projections as shown in Figure 4.1 and then supplying the block dimensions and permeabilities and using (4.3). Here we describe the algorithm used to find the well blocks which are intersected by a well and in which points the intersections occur. Well blocks will only be referred to as blocks. Assume that `GetblockEnvelopingPoint(`p`)` returns a block that contains the point p. Assume also that `FindIntersectionPoints`(block, line) calculates the two intersection points between a line segment and a block.
A list of intersected blocks and their entry and exit points are created and returned at the end of the algorithm.

Algorithm 4 Input: reservoir(blocks), line(start point, end point)

```
1:  procedure FIND INTERSECTED BLOCKS
2:      first block ← GetblockEnvelopingPoint(start point)
3:      last block ← GetblockEnvelopingPoint(end point)
4:      Set current block = first block
5:
6:      while current block ! = last block do
7:          intersection points ← FindIntersectionPoints(current block, line)
8:          add intersection points to list
9:          add current block to list
10:
11:         new point ← Move small distance out of current block in direction of
    .           end point
12:
13:         current block ← GetblockEnvelopingPoint(new point)
14:
15:     end while
16:     Add last block to list
17:     Return lists
18: end procedure
```

Chapter 6

Results and numerical tests

In this chapter we present the results of the implementations of the well distance projection and the inter-well distance projection and the alternating projection method applied to the joint problem. We also include numerical results from the computations of well indices and compare them to the results found using the reservoir simulator RMS. Figures are also supplied whenever possible.

6.1 Well constraint projections

For all projections in this section we set the well length constraint (i.e., shortest and longest wells allowed) and inter-well distance constraint (i.e., the minimum distance required to be between all pairs of wells) parameters to the following values:

$$L_{\min} = 5, \tag{6.1}$$

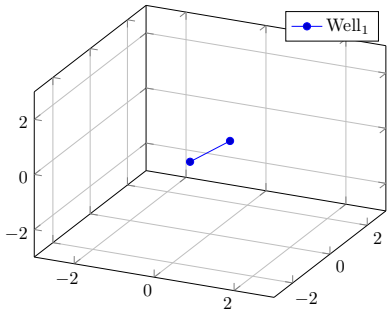
$$L_{\max} = 10, \tag{6.2}$$

$$d = 4. \tag{6.3}$$

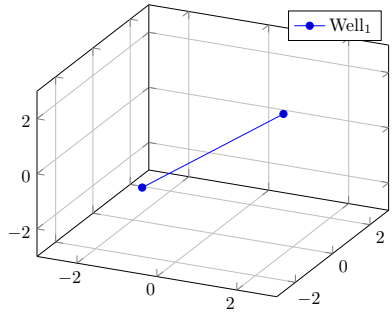
This means that in the final configuration all wells must have a length between 5 and 10, and no two wells are allowed to be closer than a distance 4 to each other. The well length projection was tested on a single well and simultaneously on a set of five wells. the inter-well distance projection was tested on two wells and then as an alternating projection on five wells. Both projections were then applied alternatingly on one set of two wells and one set of five wells until a feasible solution was reached.

6.1.1 Well length projection

Well length constraint on a single well First consider the well with endpoints $(-\frac{1}{2}, 0, \frac{1}{2})$ and $(\frac{1}{2}, 0, \frac{1}{2})$. This well has length 2 and its length is increased as shown in Figure 6.1.



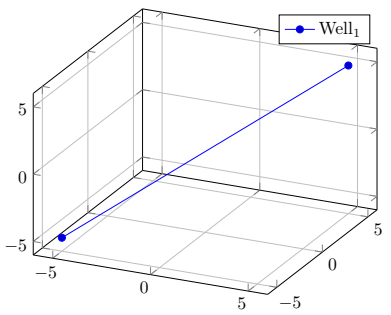
(a) Initial well. The well has length 2 and violates the well length constraint because it is too short.



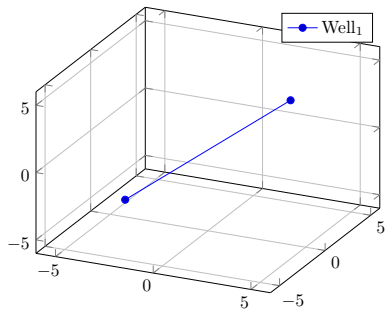
(b) The well has been projected to satisfy the well length length constraint. The projected well has length 5 which is equal to $L_{\min}$.

Figure 6.1: The well on the left that violates the well length constraint has been projected to a feasible solution on the right.

Now consider the well with endpoints $(-5, -5, -5)$, $(5, 5, 5)$. This well has length $10\sqrt{3}$ and its length is decreased as shown in Figure 6.2.



(a) Initial well. The well has length $10\sqrt{3}$ and violates the well length constraint because it is too long.



(b) The well has been projected to satisfy the well length length constraint. The projected well has length 10 which is equal to $L_{\max}$.

Figure 6.2: The well on the left that violates the well length constraint has been projected to a feasible solution on the right.

Lastly five wells were were created as shown in Figure 6.3.

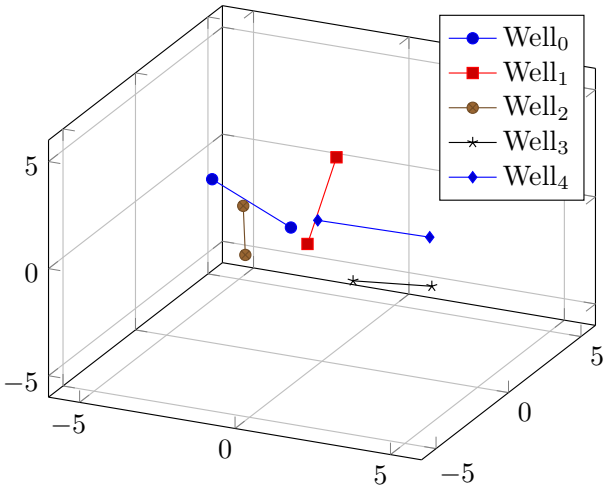


Figure 6.3: Initial positions of five wells. Both the well distance constraint and the inter-well distance constraint are violated by one or more wells.

Note that since the well lenghts are independent from each other the well lenght projection of multiple wells is eqivalent to sequential projection of single wells. In Figure 6.4 we can see the five well length projections.

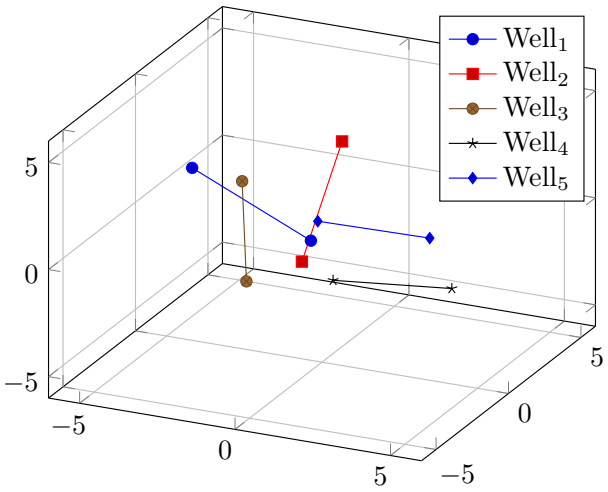


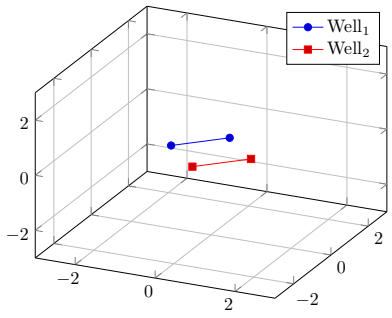
Figure 6.4: Well length projection on five wells. Five wells have been moved so that the well length constraint is satisfied for all wells. The inter-well distance constraint however is not satisfied.

6.1.2 Inter-well distance projection

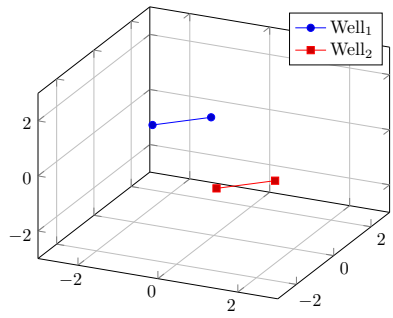
The inter-well distance projection was first tested on two wells, Well_1 and Well_2, with initial coordinates

$$\text{Well}_1 = ((-1,0,0),(0,1,0)) \quad \text{and} \quad \text{Well}_2 = ((0,-1,0),(1,0,0))\,.$$

Note that the wells in this case are parallel and, as mentioned in Section 3.2.1, there is no unique shortest distance line between the two wells. Still the numerical solution is found and the resulting four-point solution can be seen in Figure 6.5.



(a) Initial wells. The shortest distance between the wells is $\sqrt{2}$ which is less than $d = 4$, and thus the inter-well distance constraint is violated.



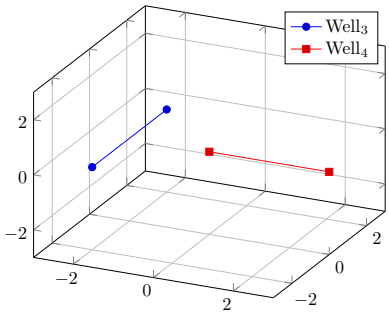
(b) The wells have been moved so the the shortest distance between them is 4.

Figure 6.5: The wells initial positions don't satisfy the inter-well distance constraint. The wells are moved so that the shortest distance between them is exactly equal to $d = 4$.

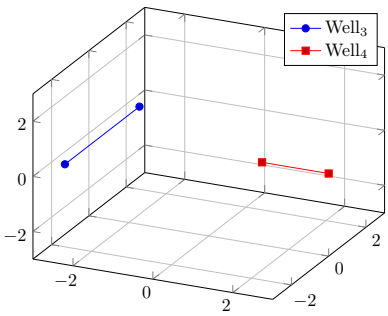
The inter-well distance projection was then tested on two wells Well_3 and Well_4, with initial coordinates

$$\text{Well}_3 = ((-2,-2,0),(-2,2,0)) \quad \text{and} \quad \text{Well}_4 = ((0,0,0),(3,0,0))\,.$$

The point to the right on Well_4 is exactly a distance 4 away from Well_3, but clearly the points of Well_3 will be moved to the left, so we expect the right point on Well_4 to remain static. Indeed the projection, which is shown in Figure 6.6 only moves 3 points, and we have a three-point solution.



(a) Initial wells. The shortest distance between the wells is 1 which is less than $d = 4$, and thus the inter-well distance constraint is violated.



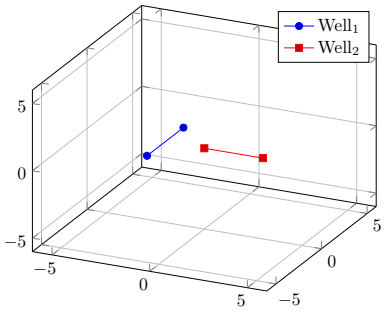
(b) The wells have been moved so the the shortest distance between them is 4.

Figure 6.6: The wells initial positions don't satisfy the inter-well distance constraint. The wells are moved so that the shortest distance between them is exactly equal to $d = 4$. The point to the right on Well_4 is not moved at all, which means this is a three-point solution as discussed in Chapter 3.

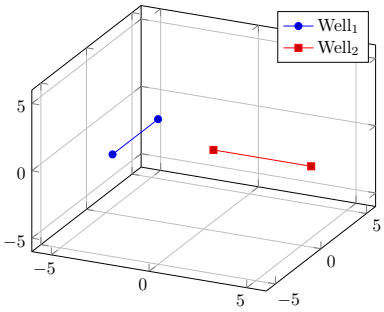
We refer to Chapter 8 for discussion on a single projection that failed.

6.1.3 Alternating projections to joint problem

We start with alternating projections on two wells with initial coordinates $((-2, -2, 0), (-2, 2, 0))$ and $((0, 0, 0), (3, 0, 0))$ as seen in Figure 6.7a. The resulting position in Figure 6.7b took 8 alternating iterations to be reached.



(a) Initial wells. The shortest distance between the wells is 1 which is less than $d = 4$, and thus the inter-well distance constraint is violated.



(b) The wells have been moved so the the shortest distance between them is 4 and the lengths of the wells have been adjusted.

Figure 6.7: The wells initial positions don't satisfy the inter-well distance constraint. The wells are moved so that the shortest distance between them is exactly equal to $d = 4$. The point to the right on Well_4 is not moved at all, which means this is a three-point solution as discussed in Chapter 3.

Actually the constraints are never fully satisfied in this case because the projections work against each other. We have implemented a tolerance for accepting a position as feasible, but in this case we can get arbitrarily close to a feasible solution given enough iterations.

Now the five wells seen in Figure 6.3 are projected using alternating projections. Theoretically there is no guarantee that any of the projections will converge. First we use the inter-well distance projection on pairs of wells until a feasible solution is reached. After iterating four times over all pairs of wells the solution in Figure 6.8 was found.

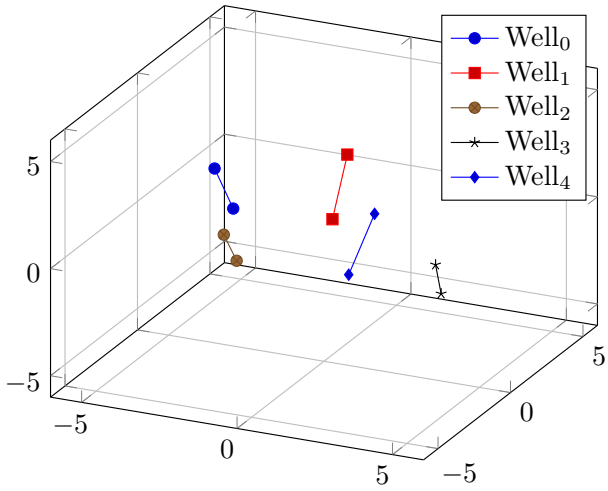


Figure 6.8: Inter-well distance projection on five wells. By running inter-well projection on wells pairwise, the five wells have been moved so that the inter-well distance constraint is satisfied for all pairs of wells. The Solution took four steps of iterating over all pairs. The well length constraint is not satisfied for any of the wells because they are all too short. This is not surprising because the inter-well distance projection can only shorten the length of a well.

Finally both projections were done alternatingly until the five wells satisfied both constraints. The final positions of the wells, which can be seen in Figure 6.9, was found after six iterations of each projection. Again the solution is not feasible but the error is sufficiently small to stop the iterations.

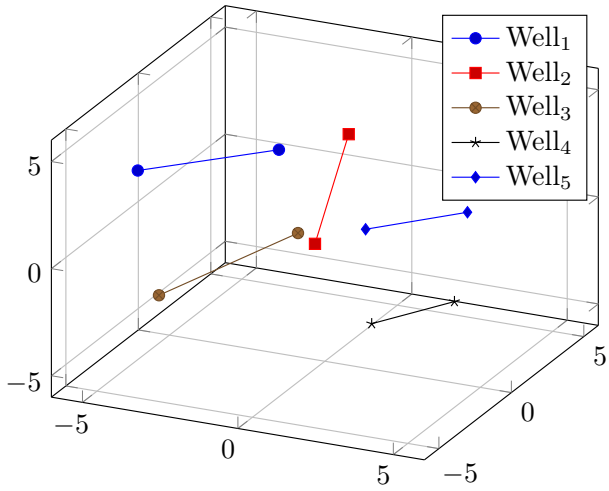


Figure 6.9: Alternating well length- and inter-well distance projection. By alternating between well length projection and inter-well distance projection the wells have been moved so that the well length constraint is satisfied for all wells and the inter-well distance constraint is satisfied for all pairs of wells. The solution took six steps of alternating projections.

6.2 Well index calculation

We use a reservoir containing 60×60 well blocks with dimensions $\Delta_x = \Delta_y = \Delta_z = 24$ and varying permeabilities. We ran the intersecting well blocks and well index calculation algorithms on a well with wellbore radius $r_w = \frac{0.1905}{2}$, which runs from the middle of the block located in the bottom left corner and straight to the block in the bottom right corner as indicated by Figure 6.10.

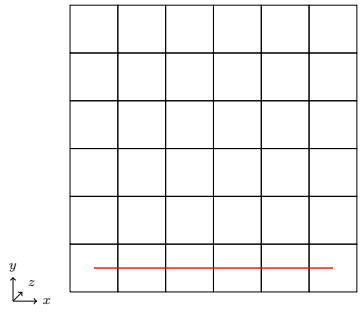


Figure 6.10: Well that goes from block 0 in the bottom left corner and ends up in block 59 in the bottom right corner. For illustration purposes the figure has been edited and every square actually contains 10×10 blocks

The calculated well indices from the first few well blocks of our implementation and the results obtained from RMS can be seen in Table 6.1.

Table 6.1: Computed well indices compared to the well indices calculated by RMS

Block number	Well index (Our algorithm)	Well index (RMS)
0	0.213273	0.21327
1	0.328879	0.32888
2	0.328879	0.32888
3	0.501907	0.50191
4	0.471228	0.47123
5	0.593556	0.59356
6	0.924533	0.92453
7	1.287440	1.28744
8	0.905511	0.90551

Chapter 7

Summary and discussion

We summarize the results of the projection methods and the well index calculator and comment on the main points of them.

The main goal of the thesis was the solution of two sub-problems occurring during the tackling of well-placement optimization, namely handling of well constraints and well index calculations. These are important because they reflect physical properties of the reservoir and are essential for a reliable and practical way of solving the overall well placement problem.

7.1 Projection to feasible space

The first problem we discussed was the handling of well-placement constraints. Here we introduced and implemented a method based on alternating projections, where one of the projections could be solved analytically, while the other projection had to be solved numerically.

The alternating projection method for satisfaction of inter-well distance and well distance constraint was implemented. The well distance constraint was solved and the analytical solutions were derived and implemented.

The main issue was the derivation of an accurate method for the projection on the inter-well distance constraint. The idea of splitting up the problem into k-point solutions leads to different cases. The main sub-case is finding the roots of the sixth degree polynomial in (3.54), which can be done efficiently with arbitrarily high precision. The implemented version preformed well overall and managed to even solve cases where one would expect numerical issues, such as when all wells are along the same line. However, the implementation was not able to solve one case with parallel line segments which is presented in Chapter 8. The reason for this is unknown.

7.2 Well index calculation

The goal for the well index calculator was to implement an alternative method capable of dealing with deviated and slanted wells. The method works by taking weighted means of the well indices one would obtain for centered wells parallel to the block axes, with weights depending on the projections of the well on these axes. To do this we require computations of entry and exit points of each of the blocks penetrated by the well.

The implemented algorithm for finding well indices compared well to industry standards for well index calculations.

Chapter 8

Further work

There were several things that we were not able to do due to either lack of time or simply because we were not able to solve the problem. Here we list the most important things and provide some information for further work.

8.1 FieldOpt integration

All functions need to be slightly adjusted so they can be included into FieldOpt and integrated for the optimization problem. The main reason is that FieldOpt uses its own well and coordinate classes, so we have to adjust the input and output format of our code.

8.2 Well length constraint projection

The well length constraint projection only handles wells that consist of a single straight line segment. There is no obvious extension of the current solution for non-straight wells. In the case of wells consisting of several connected straight lines there is an extension of our current algorithm which is to only move the heel and toe along the trajectory of the initial well. Extending the lines beyond the initial lenght of the well doesn't have a unique solution, since all directions are feasible. This works perfectly well for wells consisting of multiple line segments as long as the angles between the line segments are less than 45°.

8.3 Inter-well distance constraint projection

Note first that we were able to find one case where our algorithm was not able to solve the inter-well distance problem. This is a simple case with two parallel shifted line segments as shown in Figure 8.1.

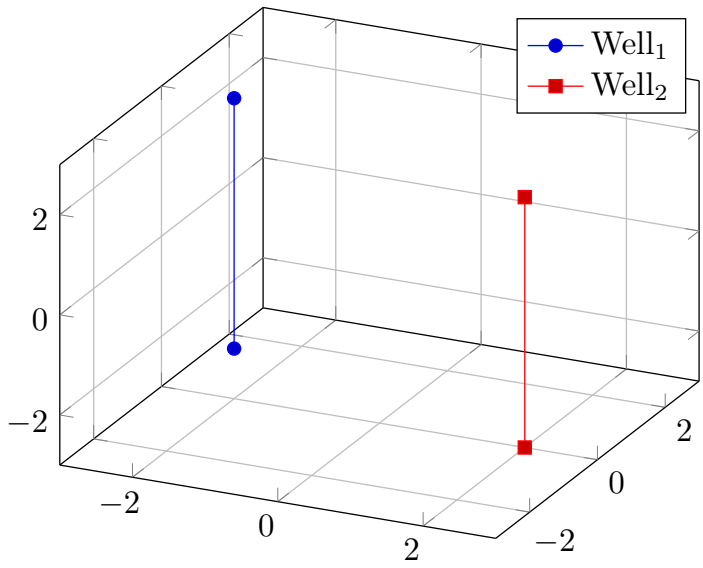


Figure 8.1: Well length projection on five wells. Five wells have been moved so that the well length constraint is satisfied for all wells. The inter-well distance constraint, however, is not satisfied.

Several very similar cases were tried and solutions were found for all of them. We did not have time to resolve the problem for this exact case, but we believe that the problem is due to numerical errors. Every time we solve equation (3.52) there might be numerical issues when computing the eigendecomposition of A. Mathematically speaking there are no problems involved in solving (3.52), but some care should be taken in these computations to guarantee that a solution is found.

8.4 Alternating projections

The convergence of the alternating projection of well length projection and inter-well distance projection was not proven. From the results obtained for several wells it is reasonable to expect that the alternating projection will always converge. The inter-well distance projection always moves pairs of wells away from the average coordinate of their endpoints, whilst the well length projection always leaves the average of the heel and toe of a well unchanged. Because there was no spatial restriction in our work, we hypothesize that the inter-well distance projection will simply move points away from the average of all the points until a solution is found. Possible alternatives to the method of alternating projections, which might be worthwhile looking into, could be averaged projections [16] or Dykstra's projection algorithm.

Appendix A

Code

A.1 Code for well constraint projection

`well_length_projection_eigen()`

```
QList<Eigen::Vector3d> WellConstraintProjections::
    ↪ well_length_projection_eigen(
    Eigen::Vector3d heel, Eigen::Vector3d toe, double max, double
        ↪ min, double epsilon)
{
    QList<Eigen::Vector3d> projected_points;
    Eigen::Vector3d moved_heel;
    Eigen::Vector3d moved_toe;

    // Need the vector going from heel to toe to project points
    Eigen::Vector3d heel_to_toe_vec = toe-heel;
    // Distance between heel and toe.
    double d = heel_to_toe_vec.norm();

    // heel and toe same point.
    // All directions equally good.
    if(d==0){
        Eigen::Vector3d unit_vector;
        unit_vector << 1,0,0;
        moved_heel = heel + (min/2)*unit_vector;
        moved_toe  = heel - (min/2)*unit_vector;
        projected_points.append(moved_heel);
        projected_points.append(moved_toe);
        return projected_points;
    }

    // Normalize vector to get correct distance
    heel_to_toe_vec.normalize();

    // Trivial case
    if(d<=max && d>=min){
        projected_points.append(heel);
```

```
        projected_points.append(toe);
    }

    // Distance too long
    else if(d>max){
        double move_distance = 0.5*(d-max+(epsilon/2));
        moved_heel = heel + move_distance*heel_to_toe_vec;
        moved_toe =  toe  - move_distance*heel_to_toe_vec;
        projected_points.append(moved_heel);
        projected_points.append(moved_toe);
    }

    // Distance too short
    else{
        double move_distance = 0.5*(d-min-(epsilon/2));
        moved_heel = heel + move_distance*heel_to_toe_vec;
        moved_toe =  toe  - move_distance*heel_to_toe_vec;
        projected_points.append(moved_heel);
        projected_points.append(moved_toe);
    }

    return projected_points;
}
```

A.2 Code inter-well distance projection

`interwell_constraint_projection_eigen()`

```
QList<Eigen::Vector3d> WellConstraintProjections::
    ↪ interwell_constraint_projection_eigen(QList<Eigen::Vector3d>
    ↪ coords, double d)
{
    /* If the two line segments already satisfy
     * the interwell distance constraint,
     * simply return the same coordinates
     */
    if( WellConstraintProjections::shortest_distance_eigen(coords)
        ↪ >=d){
        std::cout <<"Initial points satisfy constraints" << std::
            ↪ endl;
        return coords;
    }

    QList<Eigen::Vector3d> solution_coords;
    QList<Eigen::Vector3d> moved_coords;
    QList<Eigen::Vector3d> temp_coords;
    /* Iterate through moving points. First try moving 2 points,
        ↪ then 3 points
     * then 4 points. If problem can be solved moving k points,
         ↪ moving k+1 points
     * will be a worse solution. Return the best k point solution.
     */

    double cost = INFINITY;
```

```
// Try moving 2 points
std::cout <<"Initial points not feasible. Try moving 2 points"
    << std::endl;
int two_point_index[4][2] = { {0, 2} ,
                              {0, 3} ,
                              {1, 2} ,
                              {1, 3}  };

for (int ii=0; ii<4; ii++){
    moved_coords = coords;
    temp_coords = WellConstraintProjections::
        well_length_projection_eigen(coords.at(
        two_point_index[ii][0]),coords.at(two_point_index[ii
        ][1]),INFINITY,d,10e-5);
    moved_coords.replace(two_point_index[ii][0],temp_coords.at
        (0));
    moved_coords.replace(two_point_index[ii][1],temp_coords.at
        (1));
    if(WellConstraintProjections::shortest_distance_eigen(
        moved_coords) >=d && WellConstraintProjections::
        movement_cost_eig(coords,moved_coords) < cost){
        // If several moves of two points work, save the one
            with lovest movement cost
        cost = WellConstraintProjections::movement_cost_eig(
            coords,moved_coords);
        solution_coords = moved_coords;
    }
}
// If there were any succesful configurations, return the best
    one.
if (cost < INFINITY){
    std::cout <<"Found 2-point solution" << std::endl;
    return solution_coords;
}

// ################### 3 POINT PART ##############################
// If no 2 point movements were succesful, try moving 3 points.
std::cout <<"No 2 point solution. Try moving 3 points" << std::
    endl;
int three_point_index[4][3] = { {2, 0, 1} ,
                                {3, 0, 1} ,
                                {0, 2, 3} ,
                                {1, 2, 3}  };
for (int ii=0; ii<4; ii++){
    // Reset moved coords to initial state
    moved_coords = coords;

    // Choose which 3 points to move. (order is important,
        second and third entry should belong to same line
        segment)
    QList<Eigen::Vector3d> input_cords_3p;
    for (int jj=0; jj<3; jj++){
        input_cords_3p.append(coords.at(three_point_index[ii][jj
            ]));
    }
    Eigen::Matrix3d temp_A = WellConstraintProjections::
        build_A_3p_eigen(input_cords_3p);
```

```
        Eigen::Vector3d temp_b = WellConstraintProjections::
            ↪ build_b_3p_eigen(input_cords_3p,d);

        /* The kkt_eg_solutions solver handles some numerical issues
         * like A having some values close to machine epsilon and
         * eigenvalues being close to 0. Just assume that any
            ↪ solution
         * must be among the ones given in solution candidates. we
            ↪ check
         * all of them.
         */
        QList<Eigen::Vector3d> solution_candidates =
            ↪ WellConstraintProjections::kkt_eq_solutions_eigen(
            ↪ temp_A,temp_b);
        //std::cout << "there are " << solution_candidates.length()
            ↪ << " solution candidates" << std::endl;

        for(int sol_num = 0; sol_num < solution_candidates.length();
            ↪  sol_num++){
            // Solution of three point problem
            temp_coords = WellConstraintProjections::
                ↪ move_points_3p_eigen(input_cords_3p,d,
                ↪ solution_candidates.at(sol_num));
            if(temp_coords.length()<1){temp_coords = input_cords_3p
                ↪ ;}

            for (int jj=0; jj<3; jj++){
                moved_coords.replace(three_point_index[ii][jj],
                    ↪ temp_coords.at(jj));
            }

            /*std::cout << "shortest distance unmoved 3p = " <<
                ↪ shortest_distance_3p_eigen(input_cords_3p) << std
                ↪ ::endl;
            std::cout << "shortest distance 3p = " <<
                ↪ shortest_distance_3p_eigen(temp_coords) << std::
                ↪ endl;
            std::cout << "shortest distance 4p = " <<
                ↪ shortest_distance_eigen(moved_coords) << std::
                ↪ endl;
            std::cout << "movement cost = " <<
                ↪ WellConstraintProjections::movement_cost_eig(
                ↪ coords,moved_coords) << std::endl;*/

            if(WellConstraintProjections::shortest_distance_eigen(
                ↪ moved_coords) >=d-0.001 &&
                    WellConstraintProjections::movement_cost_eig(
                        ↪ coords,moved_coords) < cost){
                // If several moves of two points work, save the one
                    ↪  with lovest movement cost
                cost = WellConstraintProjections::movement_cost_eig(
                    ↪ coords,moved_coords);
                solution_coords = moved_coords;
            }
        }

    }
```

```
    // If there were any succesful configurations, return the best
        ↪ one.
    if (cost < INFINITY){
        std::cout <<"Found 3-point solution" << std::endl;
        return solution_coords;}
    // ##################### END 3 POINT PART ##########################



    // ##################### 4 POINT PART ##############################
    std::cout <<"Found no 3-point solution. Try 4 points" << std::
        ↪ endl;

    // Get all candidates for vector s
    /* The kkt_eg_solutions solver handles some numerical issues
     * like A having some values close to machine epsilon and
     * eigenvalues being close to 0. Just assume that any solution
     * must be among the ones given in solution candidates. we check
     * all of them.
     */
    Eigen::Matrix3d temp_A = WellConstraintProjections::
        ↪ build_A_4p_eigen(coords);
    Eigen::Vector3d temp_b = WellConstraintProjections::
        ↪ build_b_4p_eigen(coords,d);
    QList<Eigen::Vector3d> solution_candidates =
        ↪ WellConstraintProjections::kkt_eq_solutions_eigen(temp_A,
        ↪ temp_b);

    // Go through candidates s and pick the best one
    for(int sol_num = 0; sol_num < solution_candidates.length();
        ↪ sol_num++){

        moved_coords = WellConstraintProjections::
            ↪ move_points_4p_eigen(coords,d,solution_candidates.at(
            ↪ sol_num));
        if(WellConstraintProjections::shortest_distance_eigen(
            ↪ moved_coords) >=d-0.001 && WellConstraintProjections
            ↪ ::movement_cost_eig(coords,moved_coords) < cost){
            // If several candidates for s work, save the one with
                ↪ lovest movement cost
            cost = WellConstraintProjections::movement_cost_eig(
                ↪ coords,moved_coords);
            solution_coords = moved_coords;
        }
    }

    if(solution_coords.length()>0){std::cout <<"Found 4-point
        ↪ solution" << std::endl;}
    else std::cout <<"Found no solution to problem" << std::endl;

    return solution_coords;
}
```

`kkt_eq_solutions_eigen()`

```
QList<Eigen::Vector3d> kkt_eq_solutions_eigen(Eigen::Matrix3d A,
    ↪ Eigen::Vector3d b)
{
```

```
QList<Eigen::Vector3d> candidate_solutions;

/* First assume that A-\mu I has an inverse.
 * We can find the inverse of it and solve
 * a sixth degree equation for \mu.
 */

// Remove values that are close to zero. Mostly caused by
    ↪ initialization.
A = WellIndexCalculator::WellConstraintProjections::
    ↪ rm_entries_eps_matrix(A,10e-12);
Eigen::SelfAdjointEigenSolver<Eigen::Matrix3d> A_es(A);

// Need to remove eigenvalues which are close to zero
Eigen::Vector3d eigenvalues = WellConstraintProjections::
    ↪ rm_entries_eps(A_es.eigenvalues(),10e-12);

// Calculate the coefficients of the sixth degree polynomial
Eigen::VectorXd coeffs = WellConstraintProjections::
    ↪ coeff_vector_eigen
                                (eigenvalues, A_es.eigenvectors
                                    ↪ ().inverse(), b);

/* There is an issue where coefficients should be zero but are
    ↪ not
 * but because of numerical issues these need to be handled
    ↪ manually.
 * Simply set all whose fabs(x)<10-e12 to zero.
 */
coeffs = WellConstraintProjections::rm_entries_eps_coeffs(coeffs
    ↪ ,10e-12);


// Compute roots of polynomial with RPOLY library
Eigen::VectorXd realroots(6);
Eigen::VectorXd comproots(6);
rpoly_plus_plus::FindPolynomialRootsJenkinsTraub(coeffs,&
    ↪ realroots,&comproots);

// Loop through all roots of polynomial
for (int ii=0;ii<6;ii++){

    // Root may not be complex or an eigenvalue of A
    if(comproots[ii]==0 && eigenvalues[0]!=realroots[ii] &&
            eigenvalues[1]!=realroots[ii] && eigenvalues[2]!=
                ↪ realroots[ii]){

        // We have found a valid root. Get vector s.
        double cur_root = realroots[ii];
        Eigen::Vector3d cur_root_vec;
        cur_root_vec << cur_root, cur_root, cur_root;
        Eigen::Matrix3d invmatr = (eigenvalues-cur_root_vec).
            ↪ asDiagonal();

        // Get vector candidate vector s
```

```
            Eigen::Vector3d s = A_es.eigenvectors()*invmatr.inverse
                ↪ ()*A_es.eigenvectors().inverse()*b;
            candidate_solutions.append(s);
        }
    }

    /* Now for the second part assume that A-\mu I is not
     * invertible, i.e. \mu is an eigenvalue of A. Then
     * we either have an infinite amount of solutions of
     * (A-\mu I)s = b. Require s have length 1 to find
     * at most two solutions as long as all points are
     * not on the same line.
     */

    // Loop through all 3 eigenvalues of A
    for(int i=0; i<3; i++){

        QList<Eigen::Vector3d> eigenvalue_solutions;

        // Create linear system (A-\my I)s = b
        Eigen::Matrix3d A_eig = A- eigenvalues[i]*Eigen::Matrix3d::
            ↪ Identity();
        Eigen::Vector3d b_eig = b;

        // Check for existence of solutions
        if(WellConstraintProjections::solution_existence(A_eig,b_eig
            ↪ )){

            // Solves non-invertible case and returns, if any, the
                ↪ feasible vectors s
            eigenvalue_solutions = WellConstraintProjections::
                ↪ non_inv_solution(A_eig, b_eig);
        }

        // If any solutions exist, add them to solution_vectors
        for(int jj = 0; jj < eigenvalue_solutions.length(); jj++){
            candidate_solutions.append(eigenvalue_solutions.at(jj));
        }
    }

    return candidate_solutions;
}
```

Bibliography

[1] P. C. Group, "Fieldopt," 2016. [Online]. Available: https://github.com/PetroleumCyberneticsGroup/FieldOpt

[2] M. C. Bellout, D. E. Ciaurri, L. J. Durlofsky, B. Foss, and J. Kleppe, "Joint optimization of oil well placement and controls," *Computational Geosciences*, vol. 16, no. 4.

[3] Schlumberger. (2014). [Online]. Available: https://www.software.slb.com/products/eclipse

[4] C. Wolfsteiner, L. J. Durlofsky, and K. Aziz, "Calculation of well index for nonconventional wells on arbitrary grids," *Computational Geosciences*, vol. 7, pp. 61–82, 2003.

[5] J. Shu, "Comparisons of various techinques for computing well index," August 2005. [Online]. Available: https://earthsci.stanford.edu/ERE/research/suprihw/publications/pub-docs/JonesShu_MSReport.pdf

[6] Mathworks, "Matlab," 2016. [Online]. Available: http://se.mathworks.com/products/matlab/

[7] A. S. Lewis, D. R. Luke, and J. Malick, "Local linear convergence for alternating and averaged nonconvex projections," *Foundations of computational mathematics*, vol. 9, pp. 485–513, 2009.

[8] J. Nocedal and S. J. Wright, *Numerical Optimization*, second edition ed. Springer, 2006.

[9] D. Peaceman, "Interpretation of well-block pressures in numerical reservoir simulation," *Society of Petroleum engineers journal*, vol. 18, no. 3, pp. 183–194, June 1978.

[10] S. ECLIPSE, *Schedule User Guide*, 2004th ed., 2004.

[11] H. on github. Hilmarm's github repositories. [Online]. Available: https://github.com/hilmarm

[12] T. Q. Company, "Qt framework web page," 2016, accessed: 31.03.2016. [Online]. Available: http://www.qt.io/about-us/

[13] "The eigen library for c++," 2016, accessed: 31.03.2016. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page

[14] C. Sweeney, "Rpoly - a polynomial root-finding library," 2016, accessed: 31.03.2016. [Online]. Available: https://github.com/sweeneychris/RpolyPlusPlus

[15] M. A. Jenkins and J. F. Traub, "A three-stage variable-shift iteration for polynomial zeros and its relation to generalized rayleigh iteration," *Numerische Mathematik*, vol. 14, no. 3, pp. 252–263, 1970.

[16] A. S. Lewis, D. R. Luke, and J. Malick. (2009) Local convergence for alternating and averaged nonconvex projections.