NTNU

Norwegian University of
Science and Technology

Implementation of Trust-region Algorithm for Well Placement Optimization in FieldOpt Framework

Lingya Wang

Petroleum Geoscience and Engineering
Submission date: June 2017
Supervisor: Jon Kleppe, IGP
Co-supervisor: Mathias Bellout, IGP

Norwegian University of Science and Technology
Department of Geoscience and Petroleum

Abstract

Oil field development is a challenging engineering task that involves optimization problems such as the types, numbers, scheduling, location, and controls of wells. Optimization algorithm is the key factor for optimization model. Each algorithm has its own strengths and weaknesses, hence we need to apply efficient and effective methodologies to solve optimization problems. Multiple papers propose derivative-free methods based on stochastic global search techniques. However, very few research papers propose the use of derivative-free local search methods in well placement optimization. A main drawback of stochastic search techniques is that they often require an extensive amount of function evaluations to perform their search. These methods are thus not practical for large scale problems where each function evaluation may require hours. As a result, we consider a local and deterministic trust region algorithm in our work. This thesis concerns the implementation of the trust region for well placement into the FieldOpt optimization framework developed at the Petroleum Cybernetics Group, NTNU.

The C++-written FieldOpt software program serves as a framework and enabler for algorithm development aimed at petroleum problems that require the efficient computation of reservoir simulations for cost function evaluation. The reservoir simulation works as a "Black Box", but the FieldOpt software allows for a straightforward one-to-one communication between the algorithm logic and the multiple driver files and settings needed by the simulator.

We start with general introduction of well placement optimization and widely used algorithms for optimization problems. Gradients with respect to well placement variables are commonly not available and therefore derivative-free methods are often proposed for well placement optimization. The trust region method implemented in this work does not require cost function derivatives, but instead constructs surrogate models of the objective function obtained during the optimization process. Our approach is based on building a quadratic interpolation model, which reasonably reflects the local behavior of the original objective function in a subregion.

After implementation of the algorithm in FieldOpt, we apply test functions to validate the performance of the algorithm. One of them is the plated-shaped Matyas function and the other one is the valley-shaped Rosenbrock function. We used Cauchy point calculation and Dogleg method to solve the optimization problem and analyze their convergence properties. The results show that the convergence of the Cauchy point algorithm by taking the steepest descent direction is inefficient in some cases. A future improvement could be achieved by using the Dogleg method. We also notice that values of predetermined parameters can affect algorithm performance. Therefore, it is also important to choose a proper parameter value to improve the convergence of method. Then, we apply the model-based trust-region method to solve some well placement optimization cases, for example single producer or five-spot model.

Preface

This thesis was conducted as a part of the Master's degree in Petroleum Engieering at the Department of Geoscience and Petroleum at the Norwegian University of Science and Technology. It was performed throught my 10^{th} semester, spring 2017 in collaboration with the Petroleum Cybernetics Group at NTNU. This work was done under the supervision of Prof. Jon Kleppe and co-supervised by Postdoc Mathias C. Bellout.

Acknowledgments

I wish to thank my supervisor professor Jon Kleppe for his guidance towards the Petroleum Cybernetics group and his support during the project.

I would like to give a big thank you to my co-supervisor Mathias Bellout for his advice and assistant on almost every aspect for this thesis. Without his help I would probably not have made it.

I would also like to thank friends and classmates for support and discussion: Hilmar Magnusson, Chingz Panahli and Bo Niu. Thank you for helping me with formulating and solving the optimization problem with FieldOpt framework.

In addition, I would like to thank PhD stud. Einar Baumann: without his original FieldOpt code and technical help I would probably not have implemented algorithm in FieldOpt software framework successfully.

Lastly, I will thank my family, my mother and brother for supporting me through all these years. Thanks always being there when I need you. You know who you are.

Table of contents

Chapter 1

Introduction

Oil field development is a very expensive and challenging process, which involves optimization problems such as the types, numbers, scheduling, location, and controls of wells. The well placement optimization problem is one of the most important parts in field development. It is crucial to apply an efficient algorithm to reduce simulation runs and accelerate the process. In our work, we use a model-based trust region algorithm to solve well placement optimization problem and implement this local search derivative free method into FieldOpt optimization framework. The C++-written FieldOpt software program serves as a framework and enabler for algorithm development, which is aimed at petroleum problems that require the efficient computations of reservoir simulations for cost function evaluation. The reservoir simulation works as a "Black Box", but the FieldOpt software allows for a straightforward one-to-one communication between the algorithm logic and the multiple driver files and settings needed by the simulator. This chapter reviews the common methods in well placement optimization and the trust region method.

1.1 Well placement problem

The general well placement optimization problem involves the determination of the well types (e.g., injector or producer, vertical or horizontal) and well locations. The optimal well location that maximizes asset value (such as Net Present Value and cumulative oil production) is a key reservoir management decision. Decision-making is difficult since it depends on many parameters (e.g., heterogeneity of rock and liquids, well and surface equipment specification, economics factors, potential of various well types, spacing and scheduling of wells). In addition, we need to consider the uncertainties in certain variables. For example, permeability, perforation interval, communication between layers and some dynamic uncertainty factors. When we apply efficient algorithms to solve optimization problems, some realistic and practical constraints should also be considered and embedded into a

mathematical optimization formulation. There are three main constraints for well placement optimization problem: well length, inter-well distance and reservoir bound constraints.

1.2 Literature review

1.2.1 Widely used methods for optimal well placement

The derivative-free optimization algorithms are widely used in oil industry. From a different perspective, optimization algorithms can also be classified into deterministic or stochastic. Deterministic optimization methods include general pattern search methods and direct search methods. They are widely used in well control problems. For well placement problem, most research has focused on stochastic search algorithms. In the following section, we will present several widely used derivative-free algorithms in well placement optimization problem.

1.2.1.1 Genetic algorithm

Genetic algorithm (GA) belongs to the evolutionary algorithm and is based on natural selection. This algorithm is the most widely used approach for well placement optimization problem. The main advantage of GA is its ability to find the global optimum solution. However, genetic algorithm requires numerous simulation runs because of its stochastic nature. Johson and Roger [1] used artificial neural networks and a heuristic search method to optimize the design of well field. The heuristic search method includes genetic algorithm and simulated annealing. Artificial neural network (ANN) is trained to predict the model result. Yeten and Durlofksy [2] used genetic algorithm method to find the type, location and trajectory of nonconventional wells. Due to computationally expensive simulations runs, they used three "helper"algorithms (feed-forward artificial neutral network, hill climber and upscaling methodology) to reduce number of simulations and accelerate the optimization procedure. The artificial neutral network is used as an efficient proxy to the objective function. Hill climbing is another acceleration tool to determine the search direction. A near-well upscaling methodology is used to speed up the finite difference simulation runs. It is shown that the optimal type of well depends on the reservoir model, objective function (cumulative oil production or net present value) and the degree of reservoir uncertainty.

There are significant uncertainties in reservoir models and can also affect the well placement optimization. As a result, some new modified genetic algorithms under uncertainties are proposed in recent years. Morales [3] used this algorithm under geological uncertainties which makes genetic algorithm a robust tool. The general genetic algorithm is modified so that engineer can identify the acceptable level of risk. Litvak and Onwunalu [4] also proposed a new modified genetic algorithm in 2011. They considered the surface uncertainties which are represented in multiple reservoir models and proposed two effective approaches to reduce CPU requirements.

1.2.1.2 Particle swarm optimization

Kennedy and Eberhart [5] first introduced particle swarm optimization (PSO) algorithm in 1995. PSO algorithm evaluates the objective function at each particle location and is inspired by social behaviors of bird flocking or fish schooling. In PSO, particles represent potential solutions and move in the search space. Like genetic algorithm, PSO is a global optimization algorithm and based on natural processes, but it has no evolution operators. Compared to GA, PSO is easier to implement and there are fewer parameters to adjust. The main drawback of PSO algorithm is that it is not guaranteed to converge to a stationary point.

Onwunalu and Durlofsky [6] applied PSO algorithm to determine the optimal type and location. Vertical, deviated and dual-lateral wells were considered in their research. In addition to PSO algorithm, GA is also used to maximize objective value (NPV). By a comparion of results from these two global search methods, it indicated that PSO algorithm outperforms the general genetic algorithm for well placement optimization problem. Afterwards, they developed PSO algorithm for large scale field development problem. Well pattern optimization [7] was proposed to handle large numbers of wells as pattern. As a result, number of simulation runs is reduced and optimization procedure is improved. Wang [8] proposed a novel approach for well placement problem under uncertainty using retrospective optimization (RO). RO is suited for optimization problem under geological uncertainty. A sequence of realizations are used in retrospective optimization. Results from several examples in this work demonstrate that RO procedure requires fewer simulation runs and can reduce computational effort obviously.

1.2.1.3 Simulated annealing

Simulated annealing (SA) algorithm is a probabilistic method to approximate the global optimization in a large search space. It was developed in 1983 to deal with nonlinear problems. Both simulated annealing and genetic algorithm start with an initial random population. The main advantage of simulated annealing methods is to avoid getting stuck in local minima, but they are computationally intensive and need carefully chosen parameters. Beckner and Song [9] first used simulated annealing algorithm to optimize well placement and economics. They applied reservoir simulation model and an economic analysis module to maximize NPV. Well placement and scheduling problems are defined as a classical "traveling salesman problem" in this work. Norrena and Deutsch [10] used SA methods to determine the optimal well locations. In this work, they assumed that wells are initially static and dynamics of fluid flow are not considered. Then a flow simulator adjusts the static wells to dynamic. The optimal locations are only determined in expected value because the true reservoir is unknown.

1.2.1.4 Other derivative-free optimization algorithms

In addition to the above methods, there are several other derivative-free optimization methods used in well placement optimization. For example, Branch and

Bounded algorithms (B&B), Covariance Matrix Adaptation-Evolution Strategy (CMA-ES), Nelder-Mead downhill simplex (N-M), Spontaneous Perturbation Stochastic Approximation (SPSA) and Generalized Reduced Gradient (GRG). B&B can be used to solve both linear and nonlinear problems for large scale optimization. Rosenwald and Green [11] first used B&B with mixed integer programming to determine optimal well locations in 1974. The method in this work is in conjunction with a mathematical reservoir model to minimize the difference between the production-demand curve and the flow curve. Like PSO and GA algorithms, CMA-ES also belongs to the evolutionary algorithms and are inspired by biological mechanism. Bouzarkouna [12] proposed two new techniques into CMA-ES (adaptive penalization with rejection and incorporation of a meta-model). Results of his work showed that CMA-ES outperforms the genetic algorithm. N-M algorithm is a direct search method which only uses function evaluations. Tilke [13] presented an automated work flow to well placement optimization problem under uncertainty. The downhill simplex method of Nelder and Mead was coupled with reservoir simulator and adapted to treat bounded constraints to determine the optimal locations of well. SPSA algorithm is based on gradient and used for continuous stochastic optimization problems. Bangerth [14] compared SPSA with Nelder-Mead (N-M) simplex algorithm, very fast Simulated Annealing (VFSA) and Genetic Algorithm (GA). It is shown that SPSA and VFSA are more efficient to find optimal locations of wells with a high probability. For SPSA, there are fewer function evaluations than VFSA. GRG is generalized by allowing nonlinear constraints and arbitrary bounds on variables. John applied GRG algorithm for well spacing optimization in 2010. He proposed an economic optimization model to maximize returns on investment. It is shown that well spacing is more sensitive to oil price.

1.2.2 Trust region method in derivative-free optimization

Function evaluations from reservoir simulation are computationally expensive, hence there is a growing demand for better and more effective algorithms to find the optimal well locations with fewer function evaluations in recent years. Multiple papers propose derivative-free methods based on stochastic global search techniques. However, very few research papers propose the use of derivative-free local search methods in well placement optimization. A main drawback of stochastic search techniques is that they often require an extensive amount of function evaluations to perform their search. These methods are thus not practical for large scale problems where each function evaluation may require hours. As a result, we consider a local and deterministic trust region algorithm in our work.

Trust region is a very important and efficient numerical optimization method for nonlinear programming problems. It has a subregion around the current search point. In general, quadratic model based on interpolation is constructed within this subregion. Winfield [15] was the first to use interpolation models in 1969. Powell [16] proposed COBYLA algorithms which is a sequential trust-region algorithm through interpolation of objective functions and constrained functions. This method can be applied to solve constrained derivative-free optimization

problem. When the number of variables is n, the degree of freedom is $\frac{(n+1)(n+2)}{2}$ for a quadratic function(R^n). Hence, at least $\frac{(n+1)(n+2)}{2}$ interpolation points are needed to construct a unique model. If $n \geqslant 20$, UOBYQA [17] is used. However, UOBAYQA algorithm is not suitable for large scale problems. At each iteration, the amount of computations increases the fourth power with dimension. To overcome this difficulty, Powell also proposed NEWUOA algorithm [18] which is the best one of derivative-free optimization algorithms. It can be used when interpolation point is less than $\frac{(n+1)(n+2)}{2}$. Actually Powell's methods were developed from the modification of minimum change in quasi-newton method [19], which means Broyden modification. From numerical experiments results, it is shown that this modified method can acquire useful quadratic information and accelerate convergence. In the practical calculations, the amount of function computation increase linearly with dimension. Thus, NEWUOA can solve large scale optimization problem. For example, function needs to be evaluated 8504 times to find high precision solution when Powell applied NEWUOA to solve the 160-dimensional Arwhead test function [18].

For both UOBYQA and NEWUOA algorithms, Lagrange interpolation model plays a critical part. Lagrange polynomials can be applied to construct quadratic model, complete poisedness of points and update interpolation points. For constrained optimization problem, Powell improved NEWUOA algorithm and proposed BOBYQA algorithm. Nocedal and Marazzi [20]constructed another trust region derivative-free algorithm based on quadratic interpolation model. In addition to the traditional constraints in subregion, there is a wedge constraint. The purpose of such a wedge constraint is to improve interpolation point when objective values are minimized. CONDOR [21] algorithm is similar to UOBYYQA and can be applied to solve general constrained optimization problem. MNH [22] used Frobenius norm interpolation to construct model. DFLS algorithm [23] is based on the least square method and proposed by Conn and Scheinberg.

As we presented above, the trust region algorithm has been developed a lot. Algorithms based on trust region idea are robust and have been successfully applied in many problems. Bockmann [24] applied a modified trust region Gauss-Newton method to identify physical spectra. Alexandrov [25] used trust region method to solve nonlinear bilevel optimization problems. Bilevel problem is converted into a single-level problem and there is no assumptions on structure. Oeuvray and Bierlaire [26] constructed BOOSTERS algorithm and it has been successfully applied in biomedicine.

The trust region idea has been applied in many fields (e.g. applied mathematics, chemistry, physics, computer science, biology, medicine, economics and sociology), it is rarely used in well placement optimization. As we mentioned before, most of the research focuses on genetic algorithm, particle swarm optimization and simulated annealing. All these algorithms are global search methods. There is less published work that indicates utilization of local search methods for optimal well placement. Thus, trust region derivative-free algorithm arouses our interest. Unlike the line search method, the trust region requires to solve optimization problem in subregion at each iteration. In addiction to that, the convergence

properties of trust region method is strong. Hence, we want to incorporate it into the FieldOpt optimization software toolkit.

1.3 Thesis outline

The next chapter presents the framework for trust region algorithm. We will introduce how we construct a quadratic surrogate model by a set of well-poised interpolation points and solutions of trust region subproblems. In Chapter 3, we describe the implementation of the trust region algorithm in FieldOpt. It includes three C++ classes: **Polynomial** class, **PolyModel** class and **TrustRegionSearch** class. In Chapter 4, we apply two test functions to validate the performance of the algorithm. One of them is the plated-shaped Matyas function and the other one is the valley-shaped Rosenbrock function. Then, we will use the model-based trust region method to solve three well placement optimization cases in Chapter 5. Chapter 6 summarizes the work and the results we have obtained. Although we have implemented trust region algorithm successfully in FieldOpt, there are still some unsolved problems. In Chapter 7, we will discuss and list some important suggestion to improve the trust region algorithm.

Chapter 2

Trust region algorithm

We know that the common methods that are used to solve optimization problem are based on derivatives, for example, Newton methods. However, for well placement optimization problem, we can only get the value of the objective function from a black box. We have the input data for reservoir simulator and get the output data. But there is no accurate expression or derivative information. It may be costly to get derivative by function evaluation. So we consider derivative-free method here.

As we presented before, Derivative-free approaches can be divided into two categories, local and global search method. Compared with other DFO methods, trust region is based on interpolation model. Model-based trust region method has relatively better numerical results and convergence because model is constructed with local information of the true objective function.

The trust region algorithm is a local search model-based method. This method is designed to construct surrogate models of the objective during the optimization process. This surrogate model is usually easy to evaluate and presumed to be accurate in a neighborhood about the current iterate. In our study, we use quadratic interpolation model [27] We start by providing the background of trust region algorithm in Section 2.1 and outlining the interpolation models based on trust region method in Section 2.2. In Section 2.3, we discuss the main solutions of trust the region subproblem.

2.1 Framework of trust region algorithm

The trust region idea is based on a model which is obtained by a set of well-poised interpolation points. This model approximates the true objective function $f(x)$ in a local neighborhood centered at the current point x_k(2.1).

$$B(x_k; \Delta_k) = \{x \in R^n : ||x - x_k|| \leqslant \Delta_k\} \tag{2.1}$$

where x_k is the central point, Δ_k is the trust region radius, $||x - x_k||$ is the trial step at k-th iteration.

We focus on unconstrained optimization problem in our work. Hence, the trust region optimization problem can be defined as

$$\begin{cases} \texttt{min } m(x + s_k) \\ \texttt{s.t.} ||s_k|| \leqslant \Delta_k \end{cases} \tag{2.2}$$

The surrogate model can be accurate or poor. If the trust region step gives a good decrease in the true objective function relatively to the decrease in the model, the step is taken and the trust region size is increased or not altered. Otherwise the step is rejected and the trust region radius is decreased. Therefore, trust region algorithm always drive the radius to zero. When the trust region size is small enough, the optimization process should terminate. The following ratio between the actual reduction in the objective function $\mathbf{f}(\mathbf{x})$ and the predicted reduction using the model $\mathbf{m}(\mathbf{x})$ reflects how much the surrogate model agrees with the objective function within the trust region. This ratio decides whether the solution of (2.2) is accepted as a new iteration x_{k+1}.

$$\rho_k = \frac{\texttt{actual reduction}}{\texttt{predicted reduction}} = \frac{f(x_k) - f(x_k + s_k)}{m(x_k) - m(x_k + s_k)} \tag{2.3}$$

The trust region radius and the center point for next iteration will be updated according to ρ_k. When ρ_k is close to 1, it means the model has a good approximate performance. Trust region radius Δ_k should be increased and $x_k + s_k$ is used as the new center point for the next iteration. If ρ_k is close to 0, the surrogate model gives a poor prediction. The radius have to has be reduced and the center point is still x_k. The updating formulae used for updating x_k and Δ_k can be expressed as follows:

$$x_{k+1} = \begin{cases} x_k + s_k, & \rho_k \geqslant \eta_1 \\ x_k, & \rho_k < \eta_1 \end{cases} \tag{2.4}$$

$$\Delta_{k+1} = \begin{cases} \gamma_2 \Delta_k, & \rho_k \geqslant \eta_2 \\ \Delta_k, & \eta_1 \leqslant \rho_k < \eta_2 \\ \gamma_1 \Delta_k, & \rho_k < \eta_1 \end{cases} \tag{2.5}$$

In our work, the values of these parameters are :

$$\begin{cases} \gamma_1 = 0.5 \\ \gamma_1 = 1.5 \\ \eta_1 = 0.25 \\ \eta_2 = 0.75 \end{cases} \tag{2.6}$$

We construct a surrogate model function of the true objective function at first. Optimization step s_k will be found by solving (2.2). The ratio ρ_k is used to update the center point and trust region size for next iteration. The following algorithm shows the framework of trust region method(1).

Algorithm 1 Trust region derivative free optimization [27]

Step 0: **Initialization**.
Choose an initial point x_0, an initial radius $\Delta_0 \in (0, \Delta_{max})$, an initial interpolation set I_0, given constants parameters: $0 \leqslant \epsilon$, $0 < \eta_1 \leqslant \eta_2$, $0 < \gamma_1 \leqslant 1 < \gamma_2$, set $k = 0$

Step 1: **Create model**.
Complete well-poisedness and create model using Algorithms (3). If $||g_k|| \leqslant \epsilon$, Stop.

Step 2: **Solve the trust region subproblem**.

$$\begin{cases} \texttt{min}\ \ q_k(s) = g_k^T s + \dfrac{1}{2} s^T B_k s \\ \texttt{s.t.}\, ||s||_2 \leqslant \Delta_k \end{cases}$$

Find s_k using Algorithm

Step 3: **Define the next iteration**
Compute

$$\rho_k = \frac{ared}{pred} = \frac{f(x_k) - f(x_k + s_k)}{q(0) - q(s_k)}$$

Then

$$x_{k+1} = \begin{cases} x_k + s_k, & \texttt{if}\ \ \rho_k \geqslant \eta_1; \\ x_k, & \texttt{else}. \end{cases}$$

Step 4: **Update the trust region radius**

$$\Delta_{k+1} = \begin{cases} min(\gamma_2 \Delta_k, \Delta_{max}), & \rho_k \geqslant \eta_2 \\ \Delta_k, & \eta_1 \leqslant \rho_k < \eta_2 \\ \gamma_1 \Delta_k, & \rho_k < \eta_1 \end{cases}$$

Step 5: **Update the interpolation set**
Set $k = k + 1$, select a new set of interpolation points I_k, go to step 1 to improve the current model.

2.2 Problem definition

For the well placement optimization, we need to define variables, objective function values and constraints. In our study, the trajectory of a vertical, horizontal or directional well inside the reservoir is described in the Cartesian coordinate system. The coordinates of hell and toe are used as variables. $(x_{h,i}, y_{h,i}, z_{h,i})$ are the coordinates of heel position, $(x_{t,i}, y_{t,i}, z_{t,i})$ represents the coordinates for the toe of the well. Thus, there are six variables for one well (see Figure 2.1).

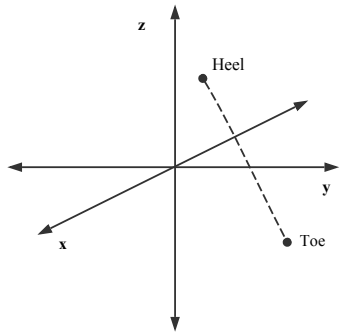

Figure 2.1: The schematic of directional well

The objective function for the well placement optimization is based on the production reservoir. It's difficult to find an accurate formulation $\mathbf{f}(\mathbf{x})$ of production by using any reservoir simulator directly, but we can build approximate surrogate model $\mathbf{m}(\mathbf{x})$ to fit the problem. We define the well placement optimization problem in the following form

$$\begin{cases} \min\ m(x) = -Q \\ \text{s.t. } x_{low} \leqslant x \leqslant x_{up}. \end{cases} \tag{2.7}$$

x is the n-dimensional vector for the well trajectories

$$x = [x_{s,1}, y_{s,1}, z_{s,1}, x_{e,1}, y_{e,1}, z_{e,1}, ..., x_{s,N}, y_{s,N}, z_{s,N}, x_{e,N}, y_{e,N}, z_{e,N},] \tag{2.8}$$

These well trajectory parameters have been defined in Figure 2.1. N is the number of wells, so the dimension of the vector should be

$$n = N \times \text{Number of wells} \tag{2.9}$$

For simplicity, we only consider boundary constraints in our work. x_{low} and x_{up} are the bound constraints from the reservoir boundaries for location of the well.

Q is the objective function value. We can find the optimum placement of wells by maximizing or minimizing objective function value. For example, maximum cumulative oil production, maximum NPV or minimum cost. In FieldOpt , the objective function value is defined as:

$$Q = N_p - 0.2 \times W_p \tag{2.10}$$

Where N_p is the cumulative oil production and W_p is the cumulative water production.

2.3 Construction of quadratic model

Reservoir simulator works as a "Black Box" and only the function values are available. we can't get the accurate calculation formula for objective function. Hence we need to construct an approximate model $m(x)$ at each optimization iteration.

$$m(x) = f(x), x \in I_k \tag{2.11}$$

$m(x)$ is the polynomial model interpolating the function $f(x)$ at a given point if $m(y) = f(y)$. I_k is the set of interpolation points which should be located inside the trust region at the k-th iteration.

When we construct model, we always first consider the space of interpolation function. There are three main different functions: linear polynomial functions, quadratic polynomial functions [28] and radial basis functions [29]. Our approach is based on quadratic interpolation model.

A natural basis $\phi = \{\phi_1, \phi_2, \phi_3, ..., \phi_L\}$ is defined by

$$\phi = \{1, x_{s,1}, y_{s,1}, z_{s,1}, ..., z_{e,N}, \frac{x_{s,1}}{2}, x_{s,1}y_{s,1}, \frac{y_{s,1}}{2}, x_{s,1}z_{s,1}, y_{s,1}z_{s,1}, \frac{z_{s,1}}{2} ..., \frac{z_{e,N}^2}{2}\} \tag{2.12}$$

where $L = \frac{(n+1)(n+2)}{2}$ and $n = N \times$ `Number of wells`.

Given a set of interpolation points $x = \{x^1, x^2, x^3, ..., x^L\} \in I_k$, values of $f(x)$ are obtained from reservoir simulation model and a set of coefficients $\alpha = \{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_L\}$ which can be determined by solving the linear system

$$\begin{cases} M(\phi, x)\alpha_\phi = f(x), x \in I_k \\ \alpha_\phi = M(\phi, x)^{-1} f(x) \end{cases} \tag{2.13}$$

where

$$M(\phi, x) = \begin{bmatrix} \phi_1(x^1) & \phi_2(x^1) & \phi_3(x^1) & \dots & \phi_L(x^1) \\ \phi_1(x^2) & \phi_2(x^2) & \phi_3(x^2) & \dots & \phi_L(x^2) \\ & & \dots\dots\dots\dots\dots & & \\ \phi_1(x^L) & \phi_2(x^L) & \phi_3(x^L) & \dots & \phi_L(x^L) \end{bmatrix} \tag{2.14}$$

$$\alpha_\phi = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_L \end{bmatrix} \tag{2.15}$$

and

$$f(x) = \begin{bmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^L) \end{bmatrix} \tag{2.16}$$

Therefore, for such a basis ϕ, the quadratic model $m(x)$ can be written as

$$m(x) = \sum_{i=1}^{L} \alpha_i \phi_i \tag{2.17}$$

Algorithm 2 Construct a quadratic model

Step 0: Given an Λ- poisedness set of interpolation points by Algorithm (3) and the natural basis (2.12), $u_i(x) = \Phi_i(x), i = 0, 1, ..., L$.

Step 1: Input data to reservoir simulator, get the values of objective function $f(x)$.

Step 2: Construct Matrix $M(x)$ by (2.14)

Step 3: Compute the set of coefficients α_ϕ

$$\alpha_\phi = M(\phi, x)^{-1} f(x)$$

Step 4: Build the quadratic model

$$m(x) = \sum_{i=1}^{L} \alpha_i \phi_i$$

2.4 The updating of interpolation set

Obviously, the above system (2.2) may not have a solution. The interpolation points are very important. If I_k is not well-poised set, the model $m(x)$ acquired may not have a good approximate performance. Hence, we need to update the interpolation set and create a well-poised set to ensure that the matrix $M(\phi, x)$ is non-singular.

There are three main methods to update points: the wedge trust region method [20] the self correcting geometry process [30] and geometry-improvement step [27] [18]. Here we use geometry step to complete a well-poised interpolation set and improve the model.

In well placement optimization, we select total $\frac{(n+1)(n+2)}{2}$ interpolation points inside trust region (2.1). Then complete the non-poised set I_k by using Algorithm (3). Some of these points can work as new interpolation points at next iteration if

they are still located inside the new trust region. Their corresponding values of objective function have been already obtained from reservoir simulator at previous iteration.

With the following approach, we can get a Λ-poised interpolation set after a finite number of iterations. More details and theories can be found in Chapter 6 in [27].

Algorithm 3 Improving well poisedness via LU factorization

Step 0: **Initialization**.
Choose a constant $\Lambda = 0.25$, given a ball $B(x, \Delta)$ (2.1), an initial set $I_k \in B$ and the corresponding monomial basis, $u_i(x) = \Phi_i(x), i = 0, 1, ..., L$, set $i = 1$.

Step 1: **Criticality test**.
If $\Lambda_k = \texttt{max}_{0 \leqslant i \leqslant L} \texttt{max}_{x \in I_k} \mid u_i(x) \mid < \Lambda$, the threshold Λ is too large, then Stop.

Step 2: **Replace**.
Find $j_i = \text{argmax}_{i \leqslant j \leqslant L} \mid u_i(x^j) \mid$. If $\mid u_i(x^j) \mid \geqslant \Lambda$, Then update I_k by performing the point exchange

$$I_k = I_k \cup \{x^{j_i}\} \setminus \{x^i\}$$

Otherwise, compute x^i as

$$x^i \in argmax_{x \in B} \mid u_i(x) \mid$$

Step 3: **Update the polynomials**
For $j = i + 1, ..., L$

$$u_j(x) = u_j(x) - \frac{u_j(x^i)}{u_i(x^i)} u_i(x)$$

If $i < L$, then set $i = i + 1$, go to Step 1.

2.5 Trust-region subproblem

The trust-region subproblem is a constrained optimization problem. As we already mentioned, we concentrate on the quadratic model. The subproblem is n-dimensional ($n = 6 \times$ Number of wells) and based on a simpler objective function. At the current iteration x_k, the trust region subproblem is

$$\begin{cases} \texttt{min } q_k(s) = g_k^T s + \frac{1}{2} s^T B_k s \\ \texttt{s.t.} ||s||_2 \leqslant \Delta_k \end{cases} \tag{2.18}$$

where $q_k(s)$ is the quadratic model in the trust region, s is trial step, $g_k = \nabla m_k(x_k)$ is the gradient at the current iteration x_k, $B_k = \nabla^2 m_k(x_k)$ is approximate Hessian matrix which is symmetric, Δ_k is the trust region radius.

2.5.1 The exact solution

Trust region subproblem can be solved exactly or approximately. Newton-like method is an exact solution. This method is meant to be applied to very small dimensional problems due to its computational complexity. The exact solution to the trust region subproblem 2.18 is found by solving

$$\begin{cases} (B_k + \lambda * I) * s^* = -g, \\ B_k + \lambda * I \geqslant 0 \qquad (\texttt{positive semi-definite}), \\ \lambda \geqslant 0, \\ \lambda(||s||_2^* - \Delta_k) = 0. \end{cases} \tag{2.19}$$

We use the Newton's method to solve the nonlinear system. Then s^* is unique if $B_k + \lambda I > 0$ [27].

2.5.2 The approximate solution

When the dimension is large, factorization to solve 2.19 can be difficult. Exact solution works when $B_k + \lambda * I$ is positive definite and the process is complicated to calculate all eigenvectors and eigenvalues of B_k to find the optimal trajectory. Therefore we consider approximate solutions for a hard trust-region problem. There are three main approaches: Cauchy point, Dogleg method and Steihaug's method [31] [32].

2.5.2.1 Cauchy point algorithm

The Cauchy point is located on the gradient which minimizes the quadratic model subject to the step being within the trust region. Hence, the Cauchy point algorithm is similar to the steepest descent line search algorithm.

$$s_k^C = -\tau_k \frac{\Delta g_k}{||g_k||}$$

$$\tau_k = \begin{cases} 1, & \texttt{if } g_k^T B_k g_k \leqslant 0; \\ \texttt{min } (\dfrac{||g_k||^3}{\Delta_k g_k^T B_k g_k}, 1), & \texttt{else}. \end{cases} \tag{2.20}$$

Although this method is cheap to implement, the convergence of the technique is inefficient and it perform poorly in some cases. Therefore, a further improvement may be achieved if Dogleg method or conjugate gradient method is used.

2.5.2.2 Dogleg algorithm

Dogleg method [16] is simple and cheap to compute and works with a polygon consisting of two line segments. It is a combination of Cauchy point and quasi-Newton point. It attempts to find the minimum along the gradient first and then find the minimum along the current point to the bottom of the quadratic model. This method is based on a positive matrix B_k.

Quasi-Newton point is the minimization point along the Newton direction equation

$$s_k^N = -B_k^{-1} g_k \tag{2.21}$$

The dogleg path is described by

$$s_k = \begin{cases} \tau s_k^C, & 0 \leqslant \tau \leqslant 1; \\ \tau s_k^C + (\tau - 1)(s_k^N - s_k^C), & 1 \leqslant \tau \leqslant 2; \end{cases} \tag{2.22}$$

The single dog leg curve is from the central point x_k along the steepest descent to the Cauchy Point s_k^C and continues along a straight line segment to the quasi-Newton point s_k^N (see Figure 2.2). This curve looks like a dog-leg and approximates the curved optimal trajectory (shown dashed).

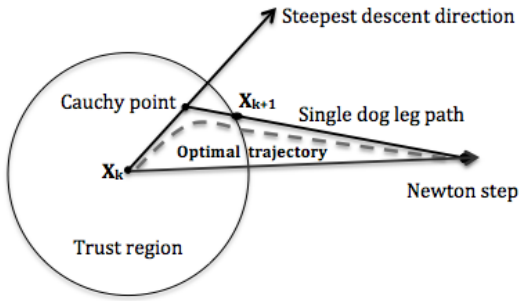

Figure 2.2: Single dogleg path

The value of the objective function is monotonically decreasing along the dogleg path, norm is strictly increasing along the path. If Cauchy point s_k^C is outside the trust region or at the boundary, the approximate solution is located at the truncated Cauchy point s_k^C where τ is 1. If the Cauchy point is interior, then we take a Newton step. Quasi-Newton point s_k^N is taken as approximate solution if it is inside the trust region. Otherwise, the approximate solution is located at the intersection of the dogleg curve and trust region boundary.

Double dogleg method is proposed by Dennis and Mei [33] and is a modified single dogleg algorithm. A new step S_k^{New} is selected in the Newton direction and

Algorithm 4 Dogleg Method

Step 0: Given x_k, g_k, B_k symmetric, Δ_k,

Step 1: Compute $||s_k^C||_2$ by 2.20; if $||s_k^C||_2 \geqslant \Delta_k$, then go to Step 2.

Step 2: $s_k = -\frac{\Delta_k g_k}{||g_k||^2}$; set $x_{k+} = x_k + s_k = x_k - \frac{\Delta_k g_k}{||g_k||^2}$, and Stop.

Step 3: Compute $||s_k^N||_2$" by 2.21; if$||s_k^N||_2$;if $||s_k^N||_2 \leqslant \Delta_k$, then go to Step 4;

Step 4: $s_k = s_k^N$; set $x_{k+1} = x_k + s_k = s_k - B_k^{-1} g_k$, and Stop.

Step 5: Compute λ satisfying $||s_k^C + \lambda(s_k^N - s_k^C)|| = \Delta_k$; set $s_k = s_k^C + \lambda(s_k^N - s_k^C)$; $x_{k+1} = x_k + s_k^C + \lambda(s_k^N - s_k^C)$; and Stop.

replaces the Newton step S_k^{New}. This new point can be found by solving

$$\begin{cases} s_k^{New} = \eta s_k^N, \\ \eta = 0.8\gamma + 0.2, \\ \gamma = \dfrac{||g_k||_2^4}{(g_k^T B_k g_k)(g_k^T B_k^{-1} g_k)}. \end{cases} \tag{2.23}$$

The new dogleg curve passes central point, Cauchy point and the new point towards the Newton direction (see Figure 2.3). Thus double dog-leg method could improve the convergence characteristics of the dog-leg method.

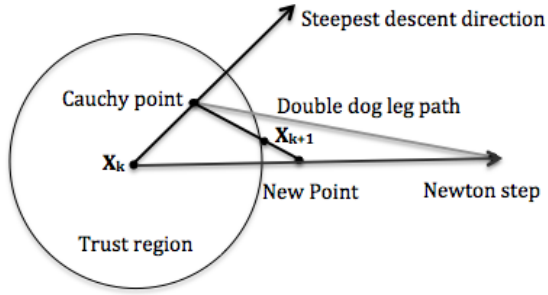

Figure 2.3: Double dogleg path

2.5.2.3 Steihaug's algorithm

Steihaug's method is a modified conjugate gradient approach, which is most widely used to solve the trust region subproblem approximately. It has all the good properties of the dogleg method. Comparing with traditional conjugate gradient method, this truncated conjugate gradient method has two extra exits. The first one is when search direction is zero or along the negative curvature of B_k, the second one is when s_k destroys the trust region constraint. Solution is located at the boundary in both cases.

Algorithm 5 Truncated Conjugate Gradient Method

Step 0: Given $s_0 = 0$, g_0, $d_0 = -g_0$, $k = 0$, B, $\Delta\epsilon$.

Step 1: If $||g_k|| \leqslant \epsilon$, then set $s^* = s_k$ and Stop.

Step 2: If B^{-1} exits, $||B_k^{-1} g_k|| \leqslant \Delta$ and $d_k^T B d_k > 0$, set $s* = -B_k^{-1} g_k$, Stop.

Step 3: If $d_k^T B d_k \leqslant 0$, d_k is a negative curvature, compute τ satisfying $||s_k + \tau d_k|| = \Delta$; set $s^* = s_k + \tau d_k$, and Stop.

Step 4: Compute $\alpha_k = \frac{g_k^T s_k}{d_k^T B d_k}$, $g_{k+1} = g_k + \alpha_k B d_k$, $\beta_k = \frac{g_{k+1}^T g_{k+1}}{d_k^T d_k}$, $d_{k+1} = -g_{k+1} + \beta_k d_k$, set$s_{k+1} = s_k + \alpha_k d_k$.

Step 5: If $||s_{k+1}|| \geqslant \Delta$, compute τ satisfying $||s_k + \tau d_k|| = \Delta$, set $s^* = s_k + \tau d_k$ and Stop.

Step 6: Set $k = k + 1$, then go to Step 1

After we find the approximate solution of subproblem obtained by the above methods. We must check the surrogate model's validity before next iteration. Trust region radius 2.5 and center point 2.4will be updated according to ρ_k.

The Figure 2.4 presents the methods and necessary steps for trust region algorithm. It describes the procedure for cooperation between trust region method and reservoir simulator.The trust region derivative-free optimization strategy decomposes well optimization process into a sequence of optimization subproblems with small regions. At each iteration, a new set of interpolation points are selected and the corresponding polynomial model is constructed instead of computationally expensive objective functions [34]. This optimization algorithm can be described as follows:

- Step 1 Select the initial center case which has been evaluated in simulator, define the values of parameters and the termination conditions
- Step 2 Create a set of interpolation points and construct the surrogate model of the true objective function in the subregion
- Step 3 Find the maximum value of the model in the subregion and check model's validity
- Step 4 Update the center point and trust region size
- Step 5 Check the termination conditions, Stop or return to Step 1

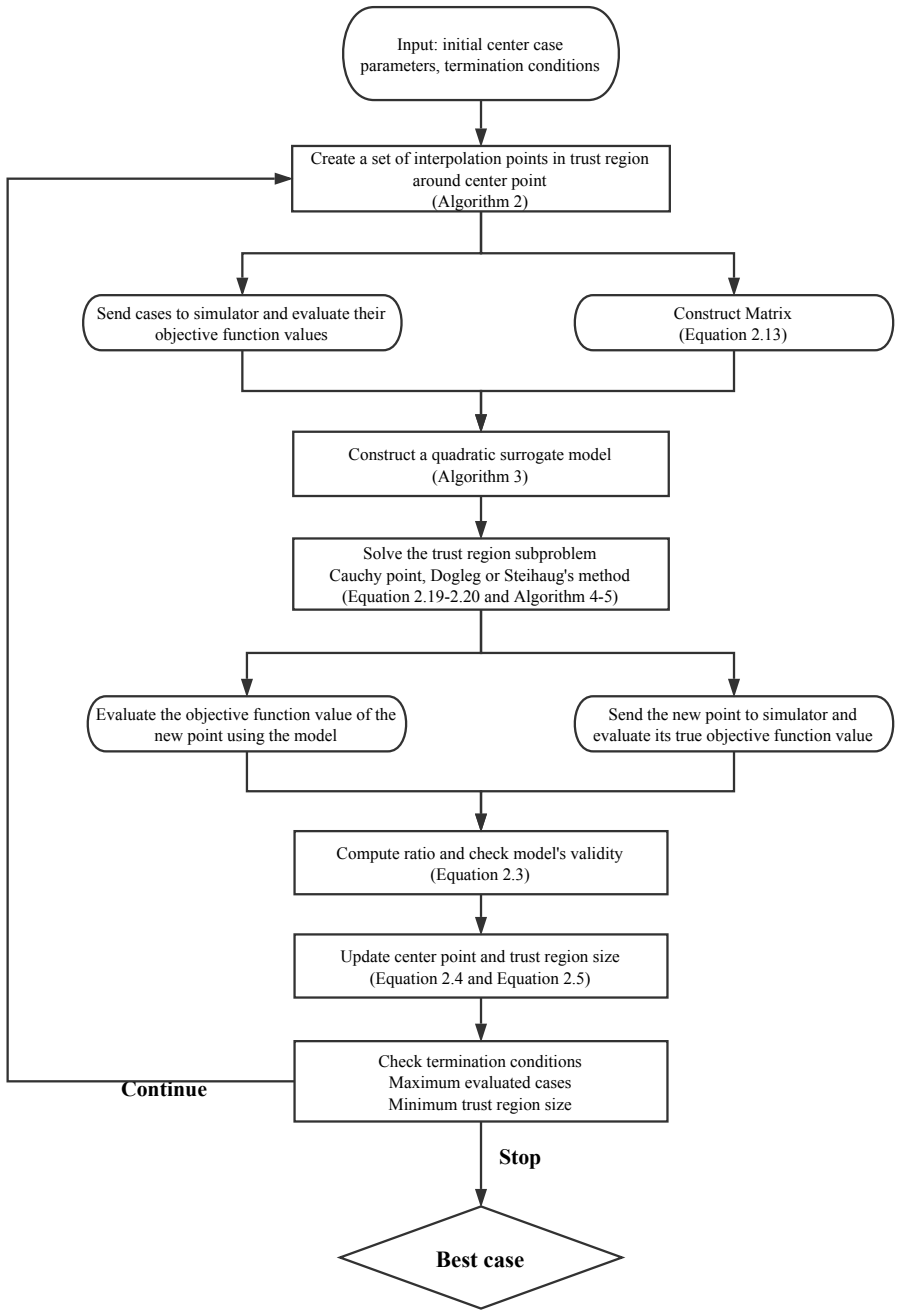

Figure 2.4: Trust region algorithm flowchart in well placement optimization

Chapter 3

Implementation of trust-region algorithm in FieldOpt

In this chapter, we describe how we implemented the derivative free local search trust region algorithm in FieldOpt. The main code is given in Appendix. Here, we present the three classes: `Polynomial`,`PolyModel` and `TrustRegionSearch`.

The full code for trust-region algorithm in FieldOpt is available on its entirety on GitHub [35]. We use class diagram in the UML (Unified Modeling Language) to present the properties of class [36]. Class diagram is a diagram showing different classes in a system their attribute, operation and the relationship among different objects. The name of the class is in bold on the first line. Methods have parentheses after the name. If the attributes and methods are private, they are prefixed with " - ". If they are public, the symbol is " + ".

3.1 Polynomial

Polynomial
- dimension: int - no_elements : int - coefficients : VectorXd - Hessian_Matrix: MatrixXd
+ Polynomial(int dimension, VectorXd coefficients): VectorXd + Evaluate(VectorXd point): double + EvaluateGradient(VectorXd point): VectorXd + Hessian(): MatrixXd + Cauchy_Point(VectorXd points, double radius, VectorXd grad): VectorXd + Newton_Point(VectorXd points, double radius,VectorXd grad): VectorXd + Dogleg_step(VectorXd points, double radius,VectorXd grad: VectorXd

`Polynomial` class has constructor `Polynomial`, which defines a second-order polynomial with dimension of variables and coefficients of each element as parameters. The function `Evaluate` returns the objective function value of the polynomial when values of variables are given. For a given point, the gradient is obtained by function `EvaluateGradient`. Cauchy point, Newton point and Dogleg path are calculated by functions `Cauchy_point`, `Newton_point` and `Dogleg_path`. All of them take interpolation point, gradient at this given point and trust region size as input.

Hessian matrix is a square matrix of second ordered partial derivatives of a scalar function. The general Hessian matrix of n variables is defined as following:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial^2 x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \frac{\partial^2 f}{\partial x_n \partial x_3} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{bmatrix} \tag{3.1}$$

In our work, we apply a second-order polynomial. Hence, Hessian matrix only depends on coefficients. For example, when is polynomial is defined with two variables:

$$f(x_1, x_2) = a_0 + a_1 \times x_1 + a_2 \times x_2 + \tfrac{1}{2} \times a_3 \times x_1^2 + \tfrac{1}{2} \times a_4 \times x_2^2 - a_5 \times x_1 \times x_2 \tag{3.2}$$

The Hessian matrix of this function has in the following form:

$$\mathbf{H} = \begin{bmatrix} a_3 & a_5 \\ a_4 & a_5 \end{bmatrix} \tag{3.3}$$

With the following approach, we can get the Hessian matrix of a second-order polynomial.

Algorithm 6 Hessian matrix of second-order polynomial

Require: Coefficients (vector) and dimension L
  for i=0, ..., L-1 **do**
    Hessian(i,i)=Coefficients(L+i+1);
  end for
  for i=0, ..., L-1 **do**
    k=0
    for j=i+1,...,L-1 **do**
      $a = \frac{((L + 1) + (L - i)) \times (i + 2)}{2} + k$
      Hessian(i,j) = Coefficients(a)
      Hessian(j,i) = Hessian(i,j)
    end for
    k=k+1
  end for

3.2 PolyModel

PolyModel
- points_abs: VectorXd - point: VectorXd - center_: VectorXd - model_coefficients: VectorXd - optimization_step_CP: VectorXd - optimization_step_SDL: VectorXd - Cases_: Case - Cases_not_eval_: Case - Basis_: Polynomial
+ PolyModel(Case base_case, double radius) + CaseFromPoint(VectorXd point, Case prototype): Case + find_new_point(VectorXd point): VectorXd + complete_points_abs(): VectorXd + complete_points: VectorXd + calculate_model_coeffs(): void + optimizationStep_CP (int factor): VectorXd + optimizationStep_SDL (int factor): VectorXd + addBaseCase(Case BaseCase): void + objective_function_value_model(VectorXd point): double

`PolyModel` class includes `Polynomial` class and `Case` class. The `Case` class contains a complete set of variables from a perturbed model and is used to describe perturbations by most of FieldOpt [37]. In this class, we construct surrogate quadratic model from a set of interpolation points. These points need to be well-poised to ensure the matrix $M(\Phi, x)$ is non-singular. We find interpolation points in a ball with radius 1 around origin and create a well-poised set of interpolation

points by Algorithm 3, then points are completed by scaling the ball back to trust region radius and center point.

- `complete_points_abs()`: Use Algorithm 3 to complete well-poised interpolations points `points_abs`, which are located in a ball of radius 1 around the origin.
- `center_`: Values of variables from the initial base case.
- `complete_points()`: Complete the set of interpolation points `points_` in trust region radius around the center point (`center_`).
- `calculate_model_coefficients()`: Find the coefficients of surrogate quadratic model (`model_coefficients`) from the interpolation points and their corresponding objective function value (Algorithm 2.13).
- `optimizationStep_CP (int factor)`: Create a second-order polynomial from `model_coefficients` and find the Cauchy point at current iteration (`center_`). Cauchy point and Dogleg step are based on steepest descent. In maximization problem, we need to transform to maximization by multiplying the objective or `model_coefficients` by -1. Hence, `factor` is used as input in both function `optimizationStep_CP` and function `optimizationStep_SDL`. `factor` $= 1$ in minimization problem, `factor` $= -1$ in maximization problem.
- `CaseFromPoint(VectorXd point, Case prototype)`: Create new case from point by displacing points and objective function value of `Case` prototype.
- `addBaseCase(Case BaseCase)`: After each successful iteration, tentative best case is used as new base case (center point) for next iteration. If the surrogate model is very poor and the candidate solution is not accepted, previous base case is still the center point for next iteration and trust region radius is reduced.

3.3 Trust region search

TrustRegionSearch
- radius_: double - minimum_radius_: double - contraction_factor: double - expansion_factor: double - objective_value: double - polymodel_: PolyModel - Current_CenterPoint: VectordXd - New_CenterPoint: VectordXd - currentBaseCase: Case - newBaseCase: Case
- scaleRaidus(double k): void - InitialModel(): void - completeModel(): void - iteration(): void + TrustRegionSearch(Optimizer setting, Case base_case, VariableProperty-Container variables, Grid grid) + SumbitEvaluatedCases(Case c): void + optimizationStep: void + UpdateModel: void

`TrustRegionSearch` class includes `PolyModel` class and is inherited through public inheritance `Optimizer` class [37]. The abstract `Optimizer` class is designed for implementation of a set of algorithms. The Figure 3.1shows the implementation of trust region algorithm in FieldOpt. We start with an initial evaluated base case and initialize a `PolyModel` with base case. Then we input the center base case and create a set of well-poised interpolation points in function `completeModel` which is inculded in `iterate`. Before `optimizationStep`, all cases must be evaluated. Hence, these unevaluated cases will be added into `CaseHandler` class. After all cases have been evaluated, then we can make the optimization step and find the tentative best case at the current iteration. This case maybe not accepted if the surrogate model is poor. In the function `UpdateModel`, the trust region radius and the base case will be updated such that. If $\rho_k \geq 0.75$, there is a good agreement between the model and the objective function. Trust region size is increased and `New_CenterPoint` is accepted as the new base case for next iteration. If $0.1 \leq \rho_k < 0.75$, trust region size is not altered although it is a successful iteration. In this case, if $\rho_k < 0.1$, the iteration give a poor prediction. The `Current_CenterPoint` is still used at next iteration and the trust region radius is reduced.

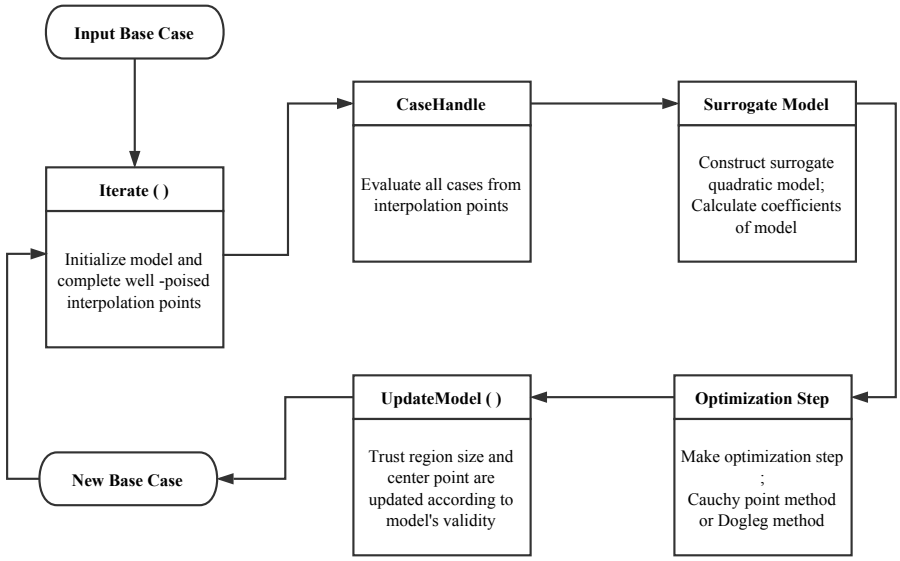

Figure 3.1: Implementation of Trust-region algorithm in FieldOpt

Chapter 4

Computational experiments

Before we apply the trust region optimization algorithm in some specific fields, test functions are used to validate the performance of the algorithm. Test functions can be grouped according to similarities in their significant physical properties and shapes. For example, plate-shaped, valley-shaped and bowl-shaped. In this chapter, we use Matyas function and Rosenbrock function as test functions for the model-based trust region optimization algorithm. The Matyas function is used the test the validity of the surrogate model which is constructed by Algorithm 2 and 3. In addition, we also discuss the convergences of the Dogleg method and the Cauchy point method when the true function is plate-shaped Matyas function. The Rosenbrock function is n-dimensional and valley-shaped, We evaluate the performance and effects of different parameters' values of trust region algorithm when the true function is not quadratic function.

4.1 Matyas test function

4.1.1 Background

Matyas function is a two-dimensional and unimodal test function. This function can be defined on any input domain and it is usually evaluated on the square $x \in [-10, 10]$ and $y \in [-10, 10]$. It has only one global minimum $f(x_1, x_2) = 0$ at $(x_1, x_2) = (0, 0)$. The function is defined by

$$f(x_1, x_2) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{4.1}$$

Matyas function is a quadratic function and plotted in the Figure 4.1.

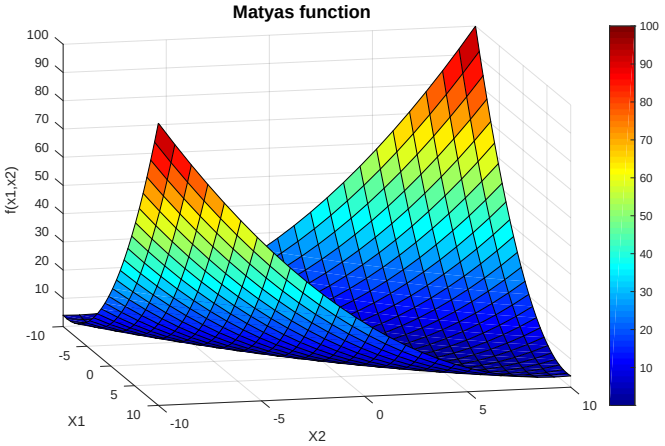

Figure 4.1: Plot of the Matyas function of versus $x_1$and x_2. The global minimum is at the point (0,0).

As we mentioned before, the approximate model is a quadratic function. Here, we use Matyas function as the true objective function, which is also a quadratic function. If the Algorithm 2 and 3 proposed to construct the surrogate model are effective. The surrogate quadratic model should be very close and even identical to true quadratic function. Therefore, we choose Matyas function to test the validity of a surrogate model. In addition, we apply both the Cauchy point method and the Dogleg method for optimization step to analyze their convergences.

4.1.2 Approximate quadratic model

As we know, both the approximate model and the true objective function are quadratic function. Hence, we can check the model's validity by comparing their corresponding coefficients directly. The comparison results are more obvious than ratio between the actual reduction gained by the true objective function (Matyas function) and predicted reduction expected in the model function.

We used two starting points $(x_1, x_2) = (2, 3)$ and $(x_1, x_2) = (4, 5)$ as center points. For each center point, there are three different subregion sizes (0.5, 1 and 1.5). Then we created well-poised interpolation points inside trust region and constructed quadratic models. The Table 4.1 shows the quadratic surrogate models based on Matyas function. We notice that the each coefficient of the quadratic term from the approximate models are very close or even identical to the Matyas function. For example, when starting point is $(x_1, x_2) = (2, 3)$ and trust region size is 0.5, the surrogate quadratic model is

$$f(x_1, x_2) = -1.07 \times 10^{-14} + 0 \times x_1 + 3.55 \times 10^{-15} \times x_2 + \frac{0.56}{2} \times x_1^2 + \frac{0.56}{2} \times x_2^2 - 0.48 x_1 x_2 \quad (4.2)$$

Matyas function can be written as

$$f(x_1, x_2) = 0 \times 1 + 0 \times x_1 + 0 \times x_2 + \frac{0.56}{2} \times x_1^2 + \frac{0.56}{2} \times x_2^2 - 0.48 x_1 x_2 \quad (4.3)$$

Table 4.2 shows that ratio between actual reduction and predicted reduction is 1. This ratio reflects how much the surrogate model agrees with the objective function with the trust region. There is a good agreement between the model and objective function at this iteration when $r \geqslant 0.75$. Figure 4.1 shows the surfaces of approximate quadratic models and Matyas function. Obviously, surfaces are almost same. Based on the coefficients, ratio and surface of surrogate model, we conclude that the constructed model is perfectly accurate when the original true objective function is also quadratic function.

(x_1, x_2)	Δ	a_0	a_1	a_2	a_3	a_4	a_5
(2, 3)	0.5	-1.07E-14	0	3.55E-15	0.52	0.52	-0.48
	1	8.88E-16	4.44E-16	-2.22E-15	0.52	0.52	-0.48
	1.5	-1.33E-15	5.55E-16	1.11E-15	0.52	0.52	-0.48
(4, 5)	0.5	2.84-13	-4.26E-14	-9.95E-14	0.52	0.52	-0.48
	1	-3.55E-14	5.33E-15	6.22E-15	0.52	0.52	-0.48
	1.5	1.24E-14	-2.22E-15	-1.78E-15	0.52	0.52	-0.48
Matyas function		0	0	0	0.52	0.52	-0.48

Table 4.1: Coefficients of approximate models based on Matyas function

(x_1, x_2)	Δ	$(x_1, x_2)_{new}$	Actual reduction	Predicted reduction	ρ
(2, 3)	0.5	(2.277, 2.584)	0.240	0.240	1
	1	(2.000, 2.033)	0.337	0.337	1
	1.5	(1.584, 1.559)	0.401	0.401	1
(4, 5)	0.5	(4.213, 4.548)	0.265	0.265	1
	1	(4.125, 4.125)	0.395	0.395	1
	1.5	(3.646, 3.656)	0.541	0.541	1

Table 4.2: Ratio between actual reduction and predicted reduction

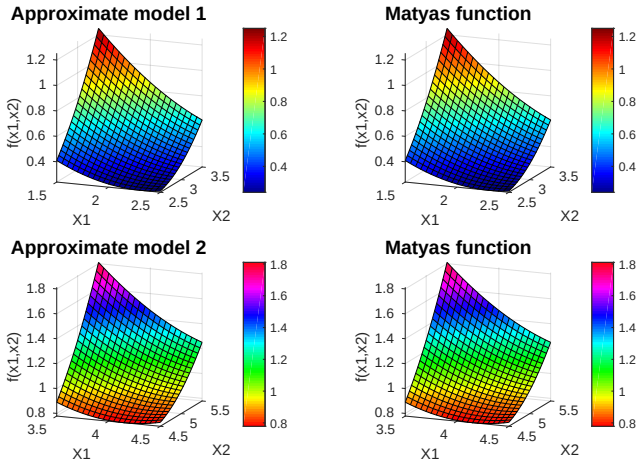

Figure 4.2: Surfaces of approximate models and original function

4.1.3 Optimization results

As we presented in Chapter 2, approximate solutions are widely used for a hard trust region problem. Cauchy point algorithm and Dogleg method are simple and cheap to compute. Here, we applied both methods for optimization step and compared their corresponding results.

4.1.3.1 Cauchy step

Figure 4.3 presents the graphical illustration of optimization results from Cauchy step in contour of Matyas function. The magenta dot represents starting point $(x_1, x_2) = (2, 3)$ and the red square is the global minimum of Matyas function at $(x_1, x_2) = (0, 0)$. The green asterisk marked points are optimization results of each iteration. The magenta circle is the initial trust region radius 1. Expansion factor 1.5 and contraction factor 0.5 are used in this case. As we know, the surrogate quadratic model is perfectly accurate. Hence, the trust region radius is increased at each iteration. The trust region radius is 1.5 for the second iteration and marked as green circle.

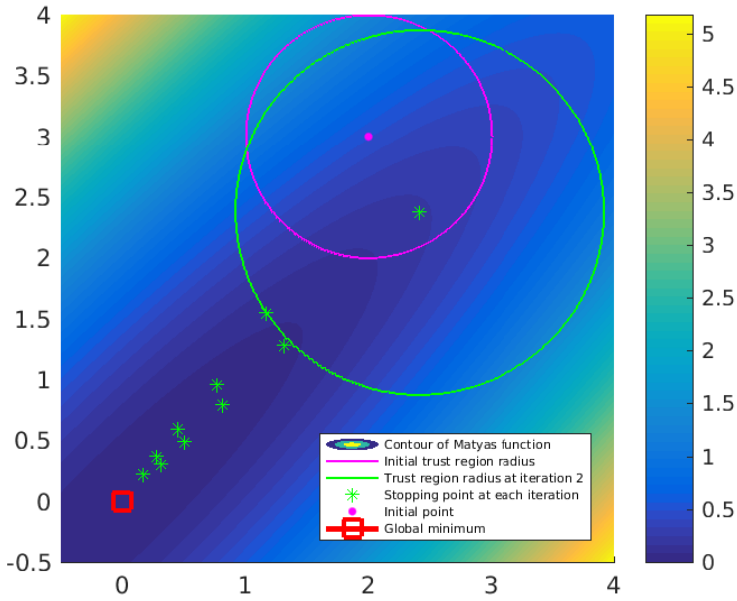

Figure 4.3: Graphical illustration of each iteration based on Cauchy point method. Starting point (2, 3).

Table 4.3 is a summary for the optimization results. Full step is taken at each iteration since the model gives a perfect prediction. At the end stage of the computation, the minimum value is $f(x_{1*}, x_{2*}) = 0.002$ at $(x_{1*}, x_{2*}) =$

$(0.171, 0.226)$. Actually, the Cauchy point calculation doesn't give an efficient convergence rate. We will discuss more about convergence of Cauchy point method later. The following Figure 4.14 and Table 4.4 are results from starting point $(x_1, x_2) = (4, 5)$. It needs 15 iterations to obtain a objective function value $f(x_{1*}, x_{2*}) = 0.015$ at $(x_{1*}, x_{2*}) = (0.616, 0.595)$.

No. of iteration n	Center point (x_1, x_2)	Stopping point (x_{1*}, x_{2*})	$f(x_{1*}, x_{2*})$
1	(2.000, 3.000)	(2.415, 2.377)	0.230
2	(2.415, 2.377)	(1.167, 1.545)	0.109
3	(1.167, 1.545)	(1.312, 1.282)	0.068
4	(1.312, 1.282)	(0.722, 0.956)	0.042
5	(0.722, 0.956)	(0.812, 0.793)	0.026
6	(0.812, 0.793)	(0.447, 0.591)	0.016
7	(0.447, 0.591)	(0.502, 0.490)	0.010
8	(0.502, 0.490)	(0.276, 0.366)	0.006
9	(0.276, 0.366)	(0.310, 0.304)	0.004
10	(0.310, 0.304)	(0.171, 0.226)	0.002

Table 4.3: Optimization results of each iteration based on Cauchy point method. Starting point (2, 3)

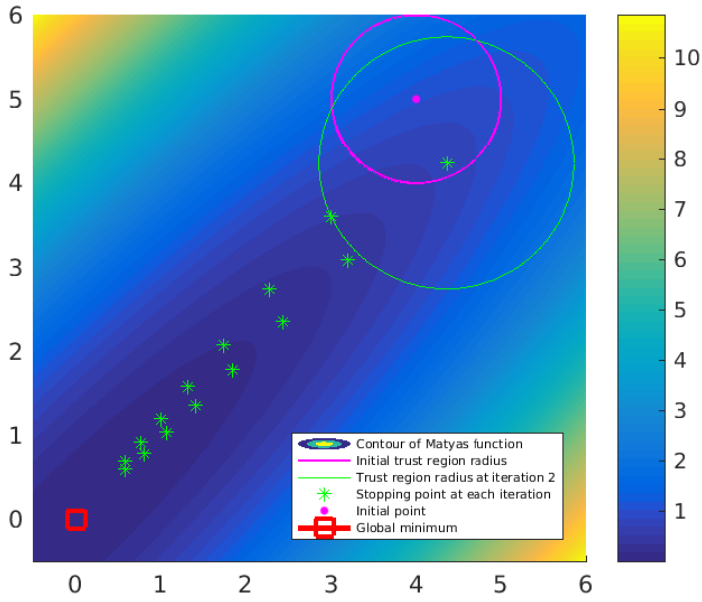

Figure 4.4: Graphic illustration of each iteration based on Cauchy point method. Starting point (4, 5)

No. of iteration n	Center point (x_1, x_2)	Stopping point (x_{1*}, x_{2*})	$f(x_{1*}, x_{2*})$
1	(4.000, 5.000)	(4.360, 4.236)	0.742
2	(4.360, 4.236)	(3.002, 3.597)	0.523
3	(3.002, 3.597)	(3.199, 3.087)	0.398
4	(3.199, 3.087)	(2.283, 2.734)	0.302
5	(2.283, 2.734)	(2.431, 2.346)	0.230
6	(2.431, 2.346)	(1.734, 2.078)	0.175
7	(1.734, 2.078)	(1.848, 1.783)	0.133
8	(1.848, 1.783)	(1.318, 1.579)	0.101
9	(1.318, 1.579)	(1.404, 1.356)	0.077
10	(1.404, 1.356)	(1.002, 1.200)	0.058
11	(1.002, 1.200)	(1.067, 1.030)	0.044
12	(1.067, 1.030)	(0.762, 0.912)	0.034
13	(0.762, 0.912)	(0.811, 0.783)	0.026
14	(0.811, 0.783)	(0.579, 0.694)	0.019
15	(0.579, 0.694)	(0.616, 0.595)	0.015

Table 4.4: Optimization results of each iteration based on Cauchy point method. Starting point (4, 5)

4.1.3.2 Dogleg method

Now we look at the optimization results (see Table 4.5) and the graphical illustration(see Figure 4.5) when Dogleg algorithm is used. After only tree iterations, we get the minimum result $f(x_{1*}, x_{2*}) = 1.9 \times 10^{-31}$ at $(x_{1*}, x_{2*}) = (2.2 \times 10^{-15}, -2.0 \times 10^{-15})$, which is very close to the global minimum point at $(x_{1*}, x_{2*}) = (0, 0)$. Comparing results from Cauchy point and Dogleg method (see Figure 4.6 and Figure 4.7), it is obvious that Dogleg method has much faster convergence. At the same starting points and conditions, it needs 10 iterations to get a stopping point where $f(x_{1*}, x_{2*}) = 0.002$ by using Cauchy step (see Figure 4.3 and Table 4.3).

No. of iteration n	Center point (x_1, x_2)	Stopping point (x_{1*}, x_{2*})	$f(x_{1*}, x_{2*})$
1	(2.000, 3.000)	(2.033, 2.001)	0.163
2	(2.033, 2.001)	(0.785, 1.168)	0.109
3	(0.785, 1.168)	$(2.2 \times 10^{-15}, -2.0 \times 10^{-15})$	1.9×10^{-31}

Table 4.5: Optimization results of each iteration based on Dogleg method

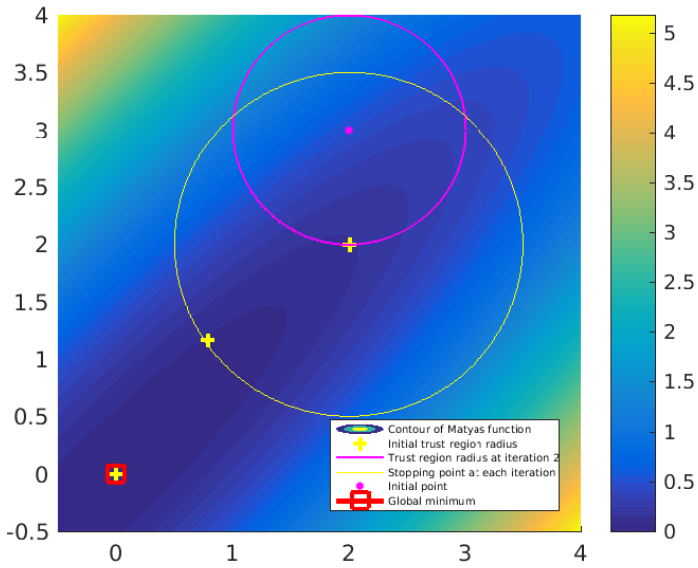

Figure 4.5: Graphical illustration for each iteration based on Dogleg method. Starting point (2.3).

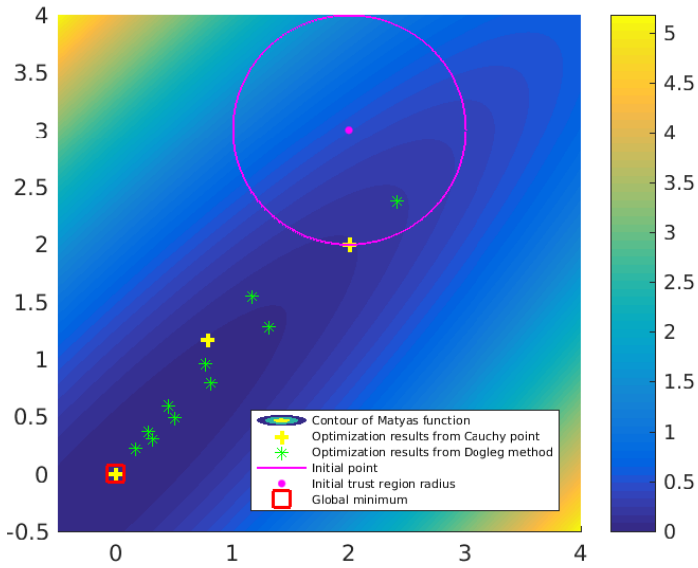

Figure 4.6: Results comparison between Cauchy point method and Dogleg method. Starting point (2.3)

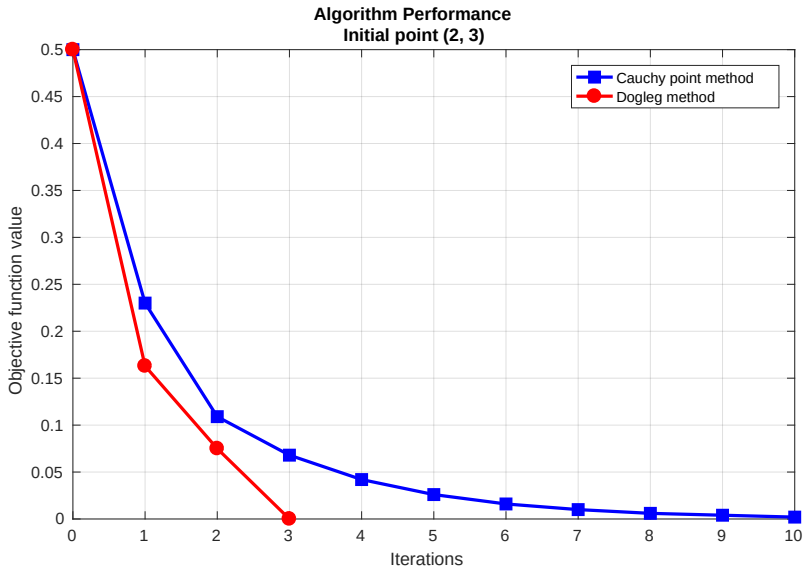

Figure 4.7: Algorithm performance at starting point (2, 3)

For another starting point (4,5), it needs also less iterations for Dogleg method. At the end of stage, the objective function value is 1.9×10^{-30} after four iterations (see Table 4.6), which is much smaller than 0.002 that is obtained from the Cauchy point method (see Figure 4.4 and Table 4.4).

Figure 4.10 and Figure 4.9 also show that Dogleg method has better convergence when the starting point is $(x_1, x_2) = (4, 5)$. The yellow '+' marked points are optimization results from Dogleg algorithm. After four iterations, a yellow '+' is located in the red square which means the global minimum of the original objective function. Hence, we can get a stopping point that is very close to global minimum. The green '$*$' marked points represent results of each iteration based on Cauchy step. After 15 iterations, there is still a gap between the last stopping point $(x_{1*}, x_{2*}) = (0.616, 0.595)$ and the global minimum point $(x_1, x_2) = (0, 0)$.

No. of iteration n	Center point (x_1, x_2)	Stopping point (x_{1*}, x_{2*})	$f(x_{1*}, x_{2*})$
1	(4.000, 5.000)	(4.125, 4.008)	0.665
2	(4.125, 4.008)	(2.768, 3.369)	0.467
3	(2.768, 3.369)	(1.536, 1.486)	0.092
4	(1.536, 1.486)	$(-6.9 \times 10^{-15}, -6.9 \times 10^{-15})$	1.9×10^{-30}

Table 4.6: Optimization results of each iteration based on Dogleg method

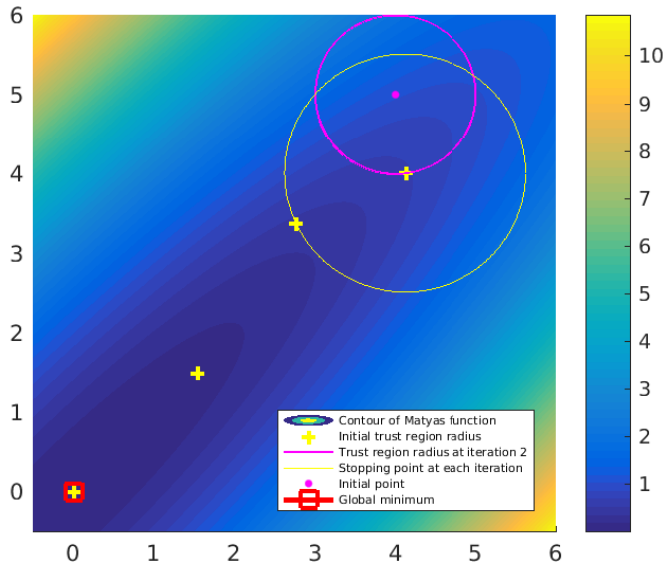

Figure 4.8: Optimization results based on Dogleg method. Starting point (4,5)

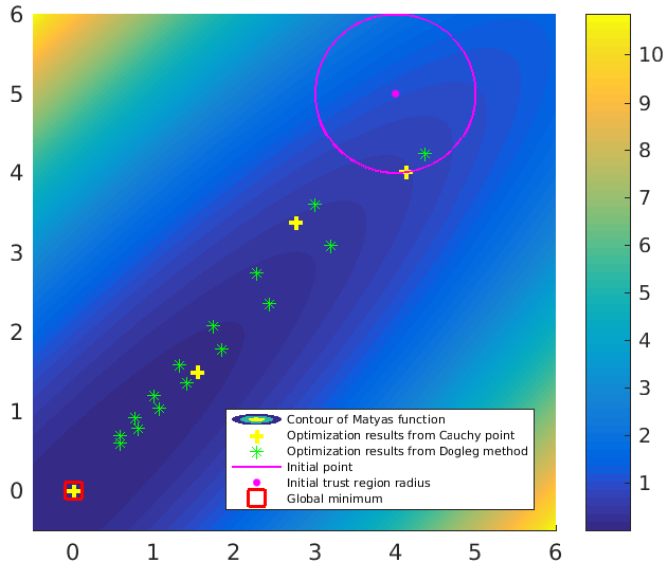

Figure 4.9: Results comparison between Cauchy point method and Dogleg method. Starting point (4,5)

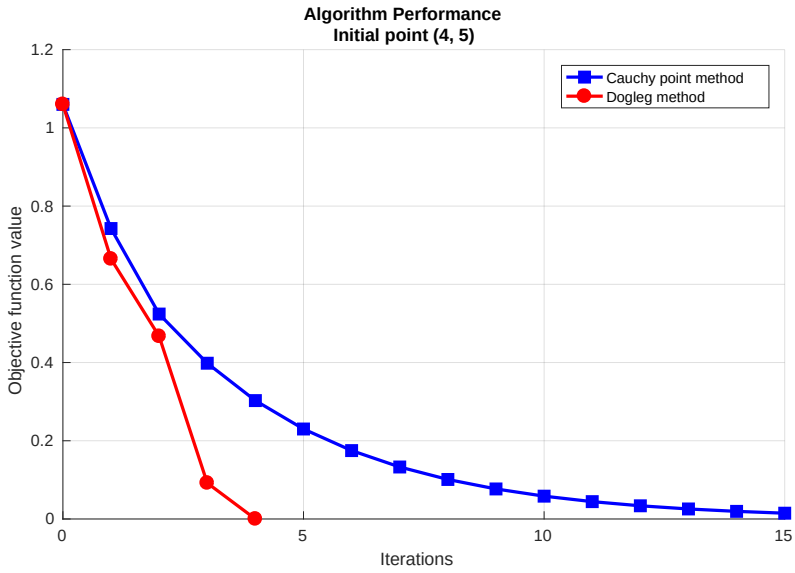

Figure 4.10: Algorithm performance at starting point (4, 5)

We used several different starting points and solved the optimization problem by using both Cauchy step (see Table 4.7) and Dogleg method (see Table 4.8). Obviously, Dogleg algorithm has much better convergence in this case. Hence, we will use Dogleg method to solve the trust region subproblem in our work.

Starting points	No. of iteration	No. of $f(x)$ Evals	Stopping point	$f(x_{1*}, x_{2*})$
(2, 3)	15	91	(0.065, 0.087)	0.00034
(4, 5)	29	175	(0.090, 0.087)	0.00032
(6, 7)	42	253	(0.079, 0.090)	0.00032
(8, 9)	48	289	(0.089, 0.094)	0.00034
(10,11)	54	325	(0.087, 0.096)	0.00036

Table 4.7: Optimization results of Cauchy step from various starting points

Starting points	No. of iteration	No. of $f(x)$ Evals	Stopping point	$f(x_{1*}, x_{2*})$
(2, 3)	3	19	$(2.2 \times 10^{-15}, -2.0 \times 10^{-15})$	1.9×10^{-31}
(4, 5)	4	25	$(-6.9 \times 10^{-15}, -6.9 \times 10^{-15})$	1.9×10^{-30}
(6, 7)	4	31	$(2.2 \times 10^{-15}, 4.4 \times 10^{-15})$	1.7×10^{-32}
(8, 9)	5	31	$(1.7 \times 10^{-15}, 1.6 \times 10^{-15})$	1.2×10^{-29}
(10, 11)	6	37	$(-8.0 \times 10^{-15}, -8.0 \times 10^{-15})$	2.6×10^{-30}

Table 4.8: Optimization results of Dogleg method from various starting points

4.2 Rosenbrock test function

4.2.1 Background

The well known Rosenbrock function is a popular test problem for gradient-based optimization algorithms. It is also referred as Rosenbrock's valley or Rosenbrock's banana function because of its curved contours. The global minimum lies in a narrow and parabolic valley. The n-dimensional function can be formalized as follows

$$f(x_1 \cdots x_2) = \sum_{i=1}^{n-1} (100(x_i - x_{i+1}^2)^2 + (1 - x_i)^2) \qquad \texttt{minimum at } f(1, 1 \cdots, 1) = 0 \tag{4.4}$$

The two-dimensional Rosenbrock function is widely used for numerical optimization problems. It is shown in the following Figure 4.11. It has a global minimum at (1,1), where $f(x_1, x_2)$ is 0. The definition of the function is

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{4.5}$$

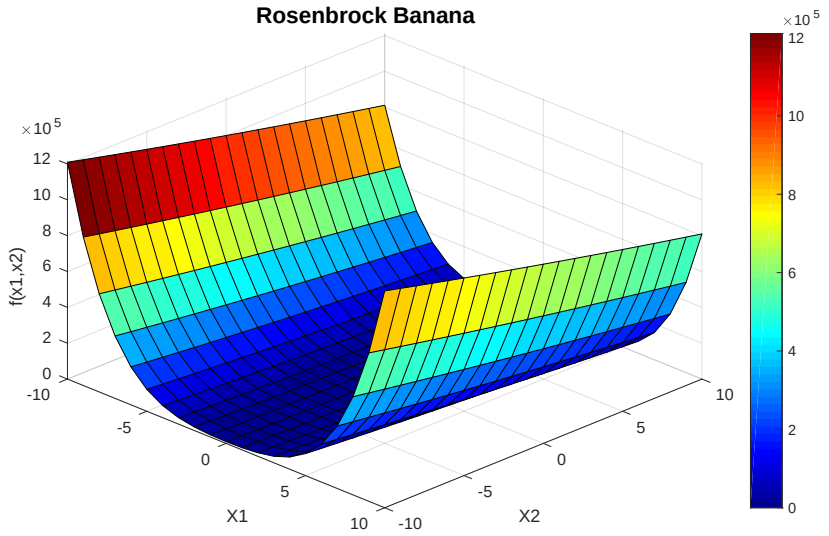

Figure 4.11: Plot of the Rosenbrock function of two variables. The global minimum is at the point (1,1)

As we mentioned before, trust region algorithm is model based. To solve the minimization problem, we first construct a quadratic model in a small subregion by using some interpolation points. Then the minimization problems are solved by Dogleg method. This approximate model's validity needs to be checked at each stage. The candidate solution is not accepted if the iteration is unsuccessful. From the Matyas function, We find that the Dogleg method has much better convergence than The Cauchy point method. Hence, we still use the Dogleg method to solve the trust region subproblem. We use expansion factor 1.5 and contraction factor 0.5 for Matyas function. Here, we also evaluate the performance of trust region algorithm when the parameters' values changes and find the the appropriate values of parameters.

4.2.2 Approximate quadratic model

Starting point	Stopping point	Δ	a_0	a_1	a_2	a_3	a_4	a_5	ρ
(3, 2)	(2.1, 2.1)	1	23641	-20482	-1020	10202	200	-260	0.958
(2.05, 2.14)	(1.37, 2.23)	1.25	5104	-6223	-845	4483	200	3.6	0.403
(1.37, 2.23)	(0.98, 2.19)	1.25	1136	-1727	-427	1688	200	34.9	-1.049

Table 4.9: Approximate models based on Rosenbrock function

We used Rosenbrock function as the original objective function, which is not quadratic. Hence, the surrogate model based on such a function will not be perfectly accurate as Matyas function. The trust region radius and the current iterate will be updated. Table 4.9 shows the surrogate quadratic models for the first tree iterations when starting point is $(x_1, x_2) = (3, 2)$. The initial trust region size is 1 and expansion factor is 1.25. For iteration 1, the center point is $(x_1, x_2) = (3, 2)$. The ratio is 0.958, which is sufficiently high. There is a good agreement between the approximate model and objective function (see Figure 4.12). The stopping point $(x_1, x_2) = (2.046, 2.139)$ is used as the new center point for the next iteration. At this stage, the ratio is 0.403, which means the iteration is successful but not perfect. The candidate solution is accepted and the trust region size is not altered. Hence, the trust region size is 1.25 for both iteration 2 and 3. We notice model difference (see Figure 4.13) between approximate model and the original objective function. For iteration 3, the models differs (see Figure 4.14) and the ratio is -1.049. This iteration is unsuccessful. The trust region size needs to be reduced and surrogate model has to been reconstructed.

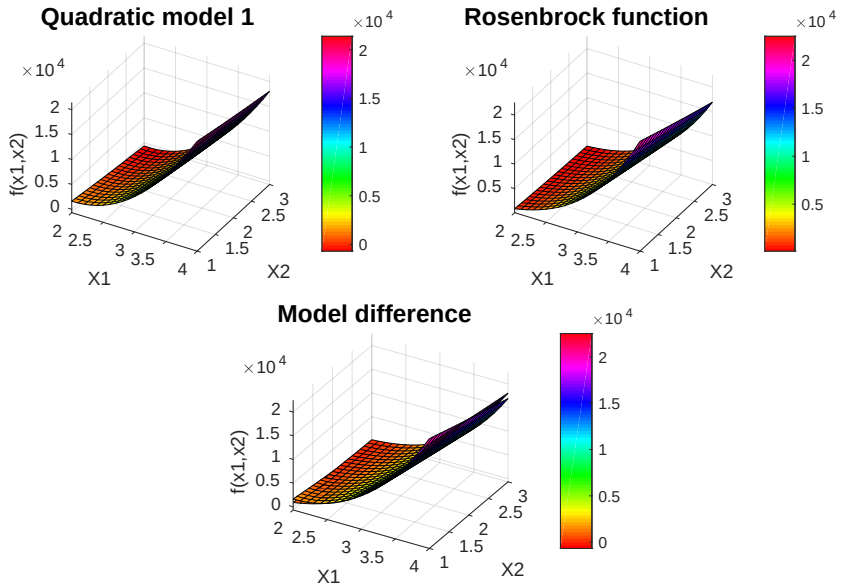

Figure 4.12: Very successful surrogate quadratic model of iteration 1. The ratio is 0.958.

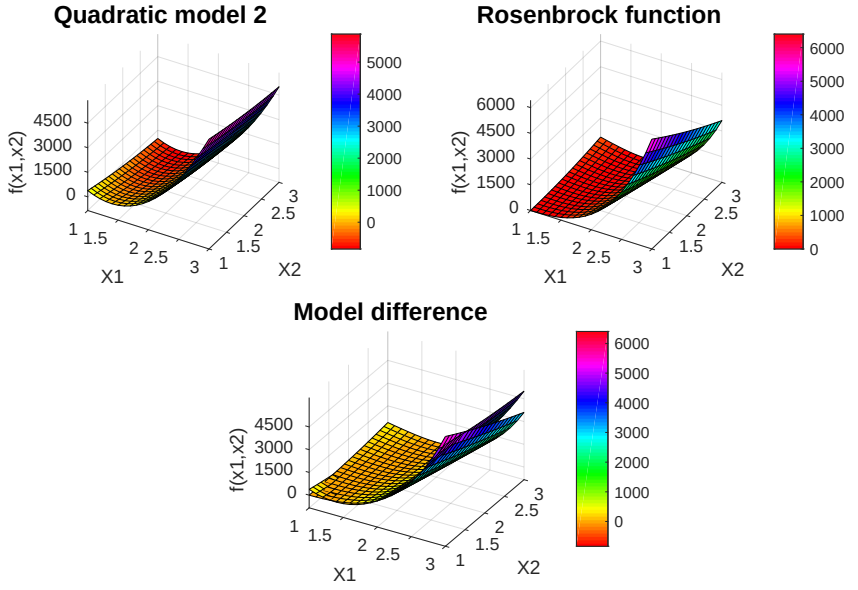

Figure 4.13: Successful surrogate quadratic model of iteration 2. The ratio is 0.403.

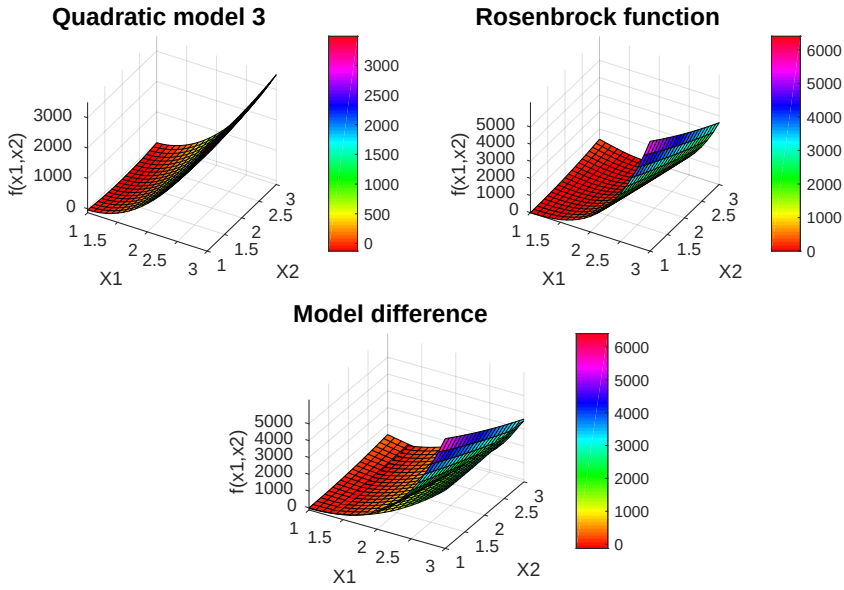

Figure 4.14: Unsuccessful surrogate quadratic model of iteration 3. The ratio is -1.049.

4.2.3 Optimization results

The following figures show the graphical illustration of the first three iterations in the contour of the original objective function. The quadratic model's contours are marked as rose red lines. The black circle is the trust region size. The initial point is marked as a red dot. The stopping point for iteration 1 and the new center point for iteration 2 is marked as a yellow dot (see Figure 4.15). The blue dot represents the stopping point for iteration 2 and the new center point for iteration 3 (see Figure 4.15). The red square is the global minimum at (1,1), where $f(x_1, x_2)$ is 0.

Iteration 1 : The starting point is $(x_1, x_2) = (3, 2)$. The objective function value at this point is 4904. The trust region is defined as the area inside the black circle, which is centered at the starting point and has radius 1. The quadratic model gives a good prediction since the ratio is close to 1. Hence, a full step is taken to the next point $(x_1, x_2) = (2, 0461, 2.139)$. The trust region radius increased from 1 to 1.25 for the next iteration.

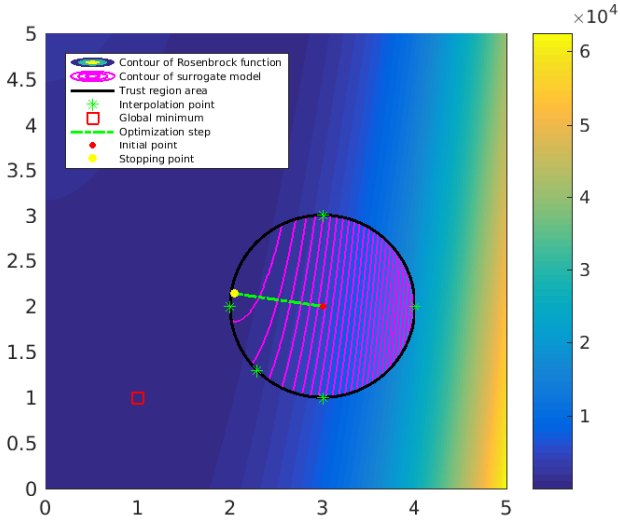

Figure 4.15: Graphical illustration of iteration 1

Iteration 2 : Start with $(x_1, x_2) = (2.046, 2.139)$ and an enlarged trust region radius is 1.25. This iteration is successful but not good enough to the new point $(x_1, x_2) = (1.374, 2.232)$. Ratio is 0.403, which is not sufficient to expand trust region size. Hence, the radius is the same for the next iteration.

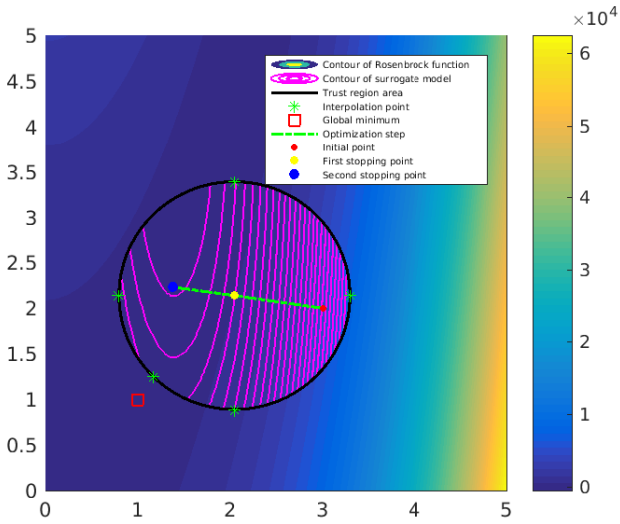

Figure 4.16: Graphical illustration of iteration 2

Iteration 3 : Start with $(x_1, x_2) = (1.374, 2.232)$ and the trust region radius is still 1.25. The ratio is -1.049, which means that the surrogate model gives a poor predictions and the new stopping point is not accepted. We need to reduce the trust region size and reconstruct a new surrogate model. Therefore, the starting point is still at $(x_1, x_2) = (1.374, 2.232)$ and trust region size is reduced to 0.625 for iteration 4.

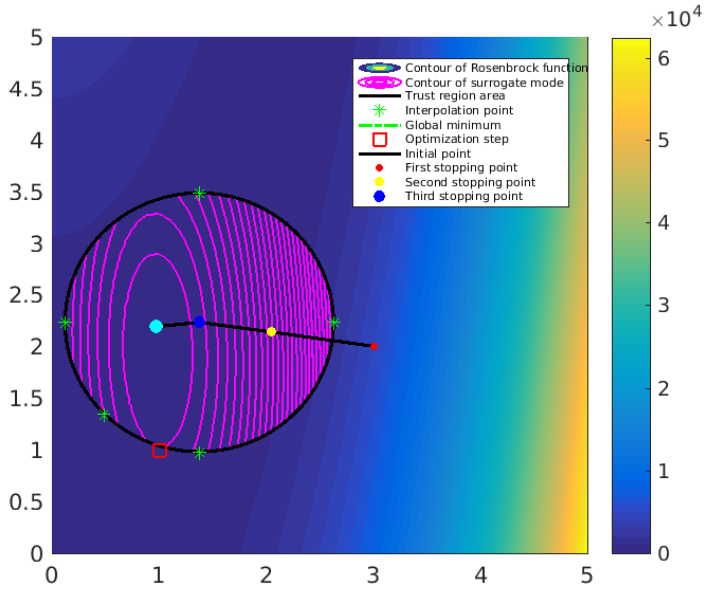

Figure 4.17: Graphical illustration of iteration 3

Table 4.10 is a summary for the optimization process. During the first 10 iterations, there are four iterations giving poor prediction. The surrogate models have to been reconstructed using a in reduced trust region size with same center point.

Iteration	(x_1, x_2)	(x_{1*}, x_{2*})	Δ	$f(x_{1*}, x_{2*})$	ρ	Validity
1	(3.000, 2.000)	(2.046, 2.139)	1	226.027	0.958	Very successful
2	(2.046, 2.139)	(1.374, 2.232)	1.25	11.931	0.403	Successful
3	(1.374, 2.232)	(0.975, 2.191)	1.25	153.962	-1.049	Unsuccessful
4	(1.374, 2.232)	(1.243, 1.896)	0.625	12.083	-0.026	Unsuccessful
5	(1.374, 2.232)	(1.466, 2.185)	0.313	0.344	1.489	Very successful
6	(1.466, 2.185)	(1.424, 2.181)	0.390	2.533	-1.474	Unsuccessful
7	(1.466, 2.185)	(1.429, 2.075)	0.195	0.303	0.095	Unsuccessful
8	(1.466, 2.185)	(1.474, 2.181)	0.098	0.232	1.599	Very successful
9	(1.474, 2.181)	(1.470, 2.180)	0.122	0.255	-2.303	Unsuccessful
10	(1.474, 2.181)	(1.475, 2.180)	0.061	0.227	2.225	Very successful

Table 4.10: Optimization results of each iteration based on Dogleg method

As we known, there are several parameters of trust region algorithm. For example, the initial trust region size, the starting point, expansion factor and contraction factor. Here, we discuss the effects of various initial step size on the performance of trust region algorithm and find the appropriate value of initial trust region radius when the original objective function is Rosenbrock function. Table 4.11 and Table 4.12 show the optimization results from different initial step sizes with the same starting point is (3, 2). The expansion factor is 1.25 and contraction factor is 0.5 for all tests.

The termination condition in Table 4.11 is the maximum number of iterations which is 40. In Test A6, the initial step size is 1.5. We notice that the objective function value of the final stopping point in is 0.154, which is less than the other tests. In Table 4.12, the optimization process will terminate when the objective function value of the stopping point is less than 0.2. In Test B6, The initial trust region radius is 1.5. It needs only 5 iterations to get a new point with objective function value less than 0.2. However, it takes 216 iterations when the initial step size is 0.25 (see Test B1). In addition, Test B6 has the minimum objective function value compared with the other tests in Table 4.12. Hence, we conclude that 1.5 is the appropriate value in this situation. When the initial starting point or other parameters change, we will get another appropriate value. Hence, it is difficult to find all best parameters values since they have interaction effects and it highlights the need for more research.

Test	Initial step	Stopping point	Iterations	Very successful	Successful	Unsuccessful	$f(x_{1*}, x_{2*})$
A1	0.25	(1.501, 2.257)	40	19	10	11	0.251
A2	0.50	(1.501, 2.255)	40	18	11	11	0.247
A3	0.75	(1.478, 2.183)	40	19	10	11	0.228
A4	1.00	(1.454, 2.113)	40	15	15	10	0.207
A5	1.25	(1.456, 2.120)	40	12	18	10	0.208
A6	1.50	(1.393, 1.940)	40	14	15	11	0.154
A7	1.75	(1.446, 2.090)	40	17	12	11	0.199
A8	2.00	(1.476, 2.180)	40	15	13	12	0.227

Table 4.11: Optimization results with various initial trust region sizes. The starting point is (3, 2) and the number of iteration is 40.

Test	Initial step	Stopping point	Iterations	Very successful	Successful	Unsuccessful	$f(x_{1*}, x_{2*})$
B1	0.25	(1.4468, 2.0932)	216	95	87	34	0.1997
B2	0.50	(1.4465, 2.0948)	220	96	89	35	0.2000
B3	0.75	(1.4470, 2.0935)	121	54	45	22	0.1998
B4	1.00	(1.4470, 2.0934)	57	23	22	12	0.1998
B5	1.25	(1.4468, 2.0931)	63	22	28	13	0.1997
B6	1.50	(1.3929, 1.9399)	5	2	1	2	0.1672
B7	1.75	(1.4469, 2.0833)	38	16	11	11	0.1988
B8	2.00	(1.4467, 2.0949)	180	76	73	31	0.1999

Table 4.12: Optimization results with various initial trust region sizes. Starting point is (3, 2) and $f(x_{1*}, x_{2*}) < 0.2$.

Chapter 5

Example cases

In this chapter,we will provide example results generated from trust-region algorithm for well placement problem. We consider three cases in our study. In the first case, there is only one production well in a two-dimensional heterogeneous reservoir. As we presented previously, the values of predetermined parameters can influence the performance of trust region algorithm. Our objective is simply to find the appropriate values of some parameters and method for optimization step in this reservoir. In the second case, we consider a five-spot pattern, where the production well is located in the center surrounded by four injection wells at the corners. In addiction, we also analyze the effects of various expansion factors and contraction factors on optimization process. In the third case, we test the performance of trust region method in a simple three-dimensional reservoir and the appropriate location of producer is already known before trust region search. In our work, the objective function value is defined by $N_p - 0.2 \times W_p$. N_p is cumulative oil production, W_p is cumulative water production. The drivers files for FieldOpt are in a machine-readable format (JSON). The driver files we used for example cases can be found in Appendix B.

5.1 Case 1

5.1.1 Case description

We consider a relatively simple two-dimensional heterogeneous reservoir here. The reservoir is a $1400 \times 1400 \times 24\ m^3$ simulation model. It is discretized by $60 \times 60 \times 1$ grid blocks. The length along each horizontal direction and the depth are 24 m. The porosity and permeability fields of the reservoir are shown in Figure 5.1 and Figure 5.2. First, we find the best location of producer under certain initial conditions. Then, we discuss the optimization results from various parameters values, the Cauchy point method and the Dogleg method.

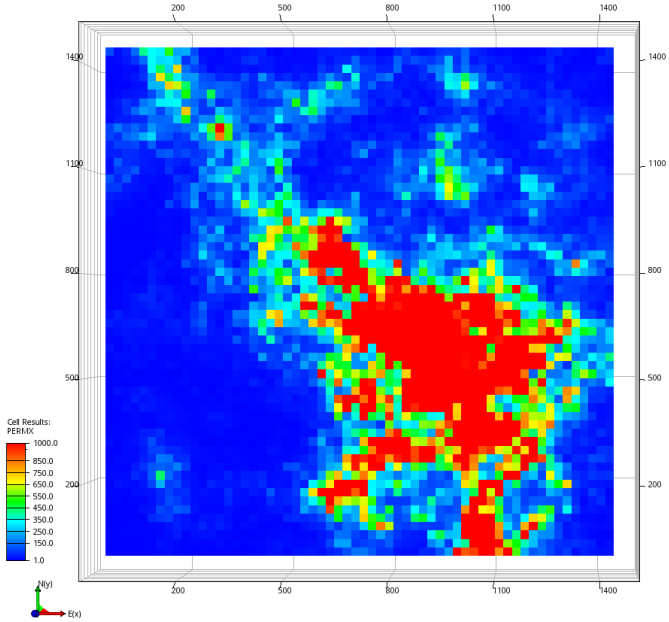

Figure 5.1: Permeability field of reservoir

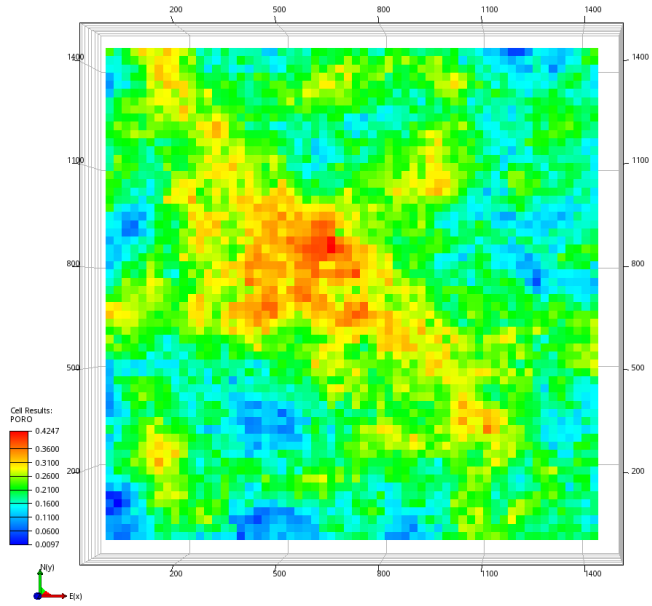

Figure 5.2: Porosity field of reservoir

5.1.2 Optimization solutions

5.1.2.1 Optimization results

The reservoir has only one producer operating at a constant production rate of 5000 STB/D. There are no injection wells. The simulation time is 100 days. The initial step size is 100 m. We use 1.5 as expansion factor and 0.5 as contraction factor. As we explained before, the Dogleg method has better convergence in some cases. Therefore, we use the Dogleg method to decide the optimization step. Table 5.1 shows the optimization results after the trust region search. There are 9 successful models during 18 iterations. Although we used Dogleg method for each iteration, there are only two successful iterations with Dogleg path. The main reason is that most quasi-Newton points are located outside the trust region area. Hence, Cauchy point is used as optimization step. Here, the objective function value is defined by $N_p - 0.2 \times W_p$ (2.10). N_p is the cumulative oil production and the W_p is the cumulative water production. The objective function value is increase by 3 % from 205572 SM^3 to 211570 SM^3. The growth is not significant because the initial location of producer is close to the best location of producer (see Figure 5.3). Figure 5.4 shows the oil saturation at the end of simulation time. We notice that the oil saturation in the southwest corner of reservoir from the best well placement is less than the initial location of producer. It means that more oil is produced when the production well is drilled at the best location after trust region search. Figure 5.5 shows the cumulative oil production (FOPT) and cumulative water production (FWPT) from the initial well placement and the best well placement. The cumulative oil production increases from 205904 SM^3 to 211576 SM^3 at the end of production time. The cumulative water production increases from 28.63 SM^3 to 30 SM^3, which is very small compared with FOPT.

Iteration	Δ	Optimization step	Model's validity	Maximum value
1	100	Cauchy point	Good	208605
2	150	Cauchy point	Satisfactory	209682
3	150	Cauchy point	Good	210376
4	225	Dogleg point	Poor	210376
5	112.5	Cauchy point	Good	211250
6	168.75	Newton point	Poor	211250
7	84.375	Dogleg point	Poor	211250
8	42.1875	Cauchy point	Good	211386
9	63.2812	Newton point	Poor	211386
10	31.6406	Cauchy point	Good	211485
11	47.4609	Dogleg point	Poor	211485
12	23.7305	Dogleg point	Poor	211485
13	11.8652	Cauchy point	Good	211543
14	17.7979	Newton point	Poor	211543
15	8.8989	Cauchy point	Satisfactory	211557
16	8.8989	Dogleg point	Good	211570
17	13.3484	Cauchy point	Poor	211570
18	6.67419	Dogleg point	Poor	211570

Table 5.1: Optimization results at each iteration after trust region search in Case 1

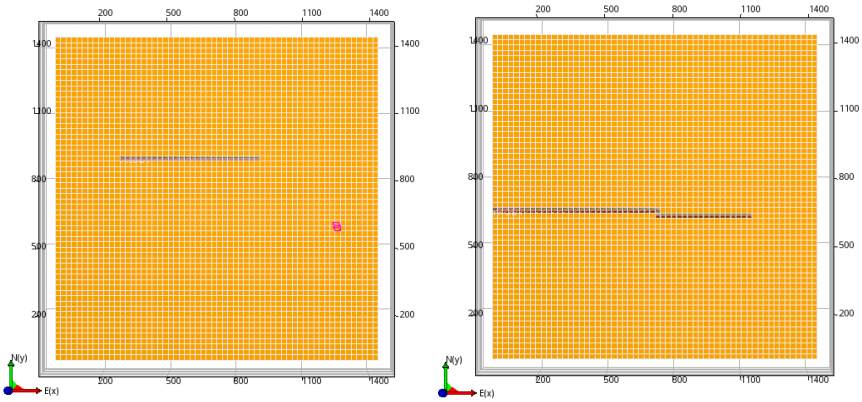

Figure 5.3: The initial location and the best location of producer in Case 1

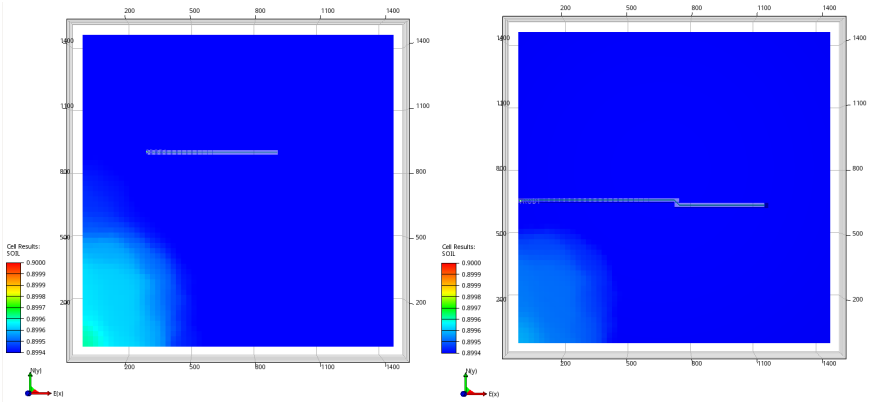

Figure 5.4: Oil saturation at the end of production time from the initial location and the best location of producer in Case 1

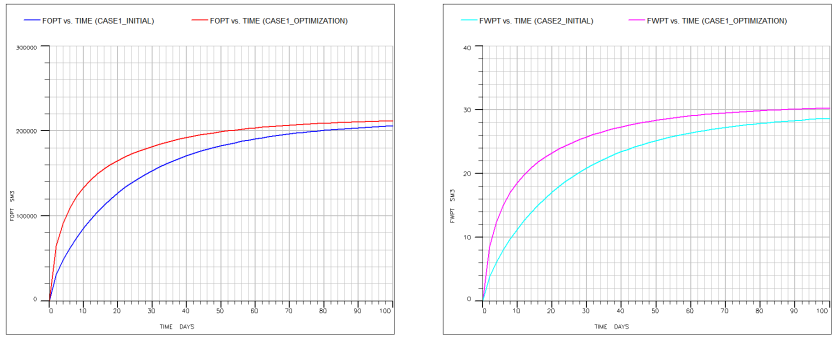

Figure 5.5: Comparison of FOPT and FWPT from the initial location and the best location of producer in Case 1

5.1.2.2 Different values of parameters

As we mentioned before, the values of parameters can influence the optimization results. In order to find the appropriate values of the initial step size and the expansion factor in this reservoir, we use the trust region algorithm with various values of parameters to solve the well placement optimization problem. We use three different initial trust region sizes (60 m ,100 m and 150 m) and three different expansion factors (1.1, 1.25 and 1.5) to analyze the performance of trust region algorithm. All of these tests have the same initial reservoir conditions and location of production well. For simplicity, the simulation time is 100 days here. The iteration will terminate when the trust region size is smaller than 5 m. The contraction factor is 0.5. The production rate is 5000 STB/D.

Table 5.2 shows the optimization results from various initial step sizes and expansion factors. Test 8 evaluated 448 cases when the initial step size is 80 m and the expansion factor is 1.5. The optimization process terminated earlier than the other tests. Hence, the maximum value from this case is 210850 SM^3 and less than the most of tests. In Test 9, the initial trust region size is 100 m and the expansion factor is 1.5. The number of total evaluated cases is 505. The objective function value of the final stopping point is 211570 SM^3 which is largest here.

From Figure 5.6, we notice that the initial trust region size is 60 m or 100 m, we can get larger objective function value with higher expansion factor value. But the effect of various expansion factors is very small when the initial step size is 80 m. In Figure 5.7, when the expansion factor is 1.1, there is no big difference between the objective function values even thought we use different initial step sizes. But it is obvious that higher initial trust region radius have better performance when the expansion factor is 1.25 or 1.5. We also notice that it takes fewer iterations if the trust region size is larger. In order to reduce simulation runs, we prefer 100 m as initial step size and 1.5 as expansion factor in this reservoir.

Test	Initial step	Expansion factor	Iterations	Evaluated cases	Maximum value
1	60	1.1	21	589	210886
2	80	1.1	20	561	211323
3	100	1.1	21	589	210910
4	60	1.25	19	533	210841
5	80	1.25	22	617	210969
6	100	1.25	17	477	211397
7	60	1.5	20	561	211423
8	80	1.5	16	448	210850
9	100	1.5	18	505	211570

Table 5.2: Optimization results of trust region method with various parameters

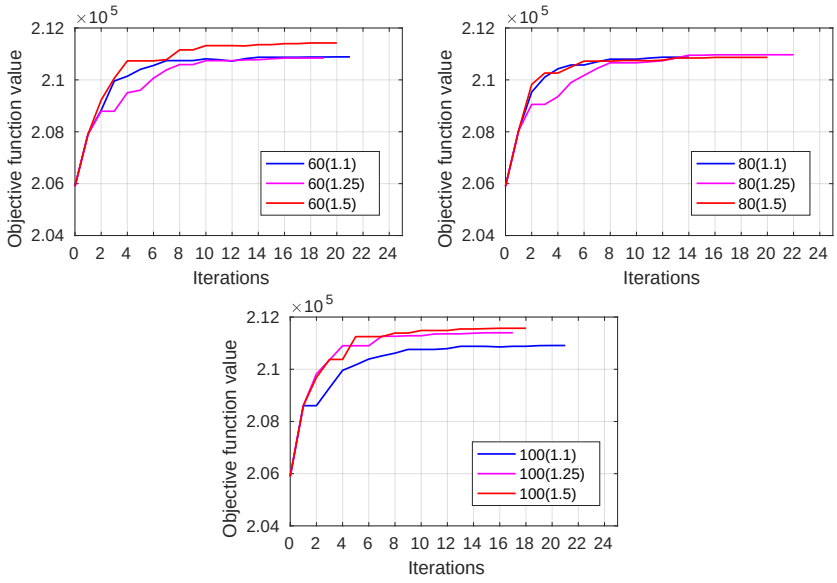

Figure 5.6: Comparison of results with same initial step and different expansion factors.

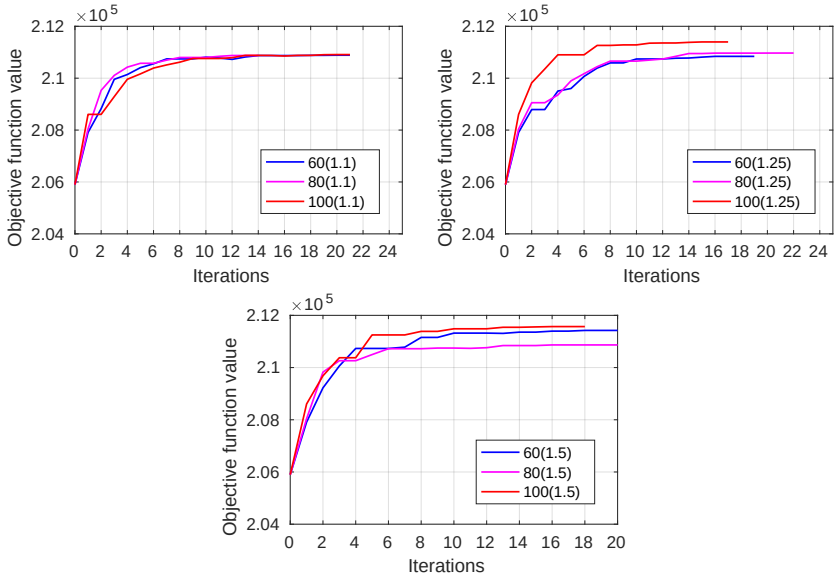

Figure 5.7: Comparison of results with same expansion factor and different initial step sizes.

5.1.2.3 Different methods for optimization step

From the optimization results at each iteration (see Table 5.1), we notice that the most optimization step is from the Cauchy point even thought we have applied the Dogleg method. Therefore, maybe the Dogleg method does not have better convergence than the Cauchy point in this reservoir. To verify this guess, we applied both Cauchy point method and Dogleg method with various initial step sizes to solve the optimization problem in this case. Table 5.3 and Figure 5.8 show the optimization results from both methods. As we expected, the Dogleg method actually does not have much better performance of algorithm here. When we use the Dogleg method, the optimization process terminate earlier. For the Cauchy point method, there will be more successful iterations, which means the optimization process is extended. The final stopping point has a larger objective function value. Therefore, we recommend the Cauchy point as the optimization step at each iteration in this two-dimensional heterogeneous reservoir.

Test	Initial step size	Optimization step	Iterations	Evaluated cases	Maximum value
1	60	Dogleg method	20	561	211423
2	60	Cauchy point	23	645	211443
3	80	Dogleg method	20	561	210867
4	80	Cauchy point	24	673	210914
5	100	Dogleg method	18	505	211570
6	100	Cauchy point	22	617	211599

Table 5.3: Optimization results from Cauchy point method and Dogleg method

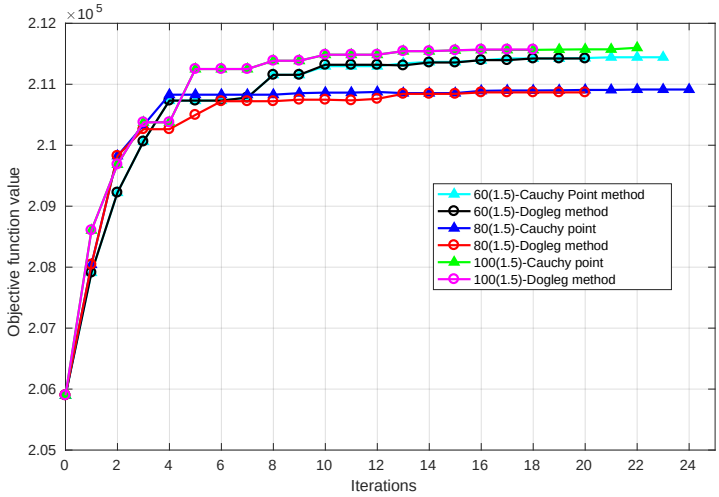

Figure 5.8: Optimization results from Cauchy point method and Dogleg method

5.2 Case 2

5.2.1 Case description

In Case 2, we still use the same reservoir model from the previous case. But it is five-spot model, where four injection wells are located at the corners of reservoir and the production well sits in the center. Table 5.4 summarizes the input parameters for reservoir model and trust region algorithm. In this case, we use trust region algorithm to solve the optimization problem and the Cauchy point is taken as optimization step at each iteration. In addition, we discuss the effects of various parameters values on the optimization results. As we know, the initial well placement also affects the optimization process. Hence, we also solve the optimization problems with different initial well locations.

Number of Grids	$60 \times 60 \times 1$
Each Grid Block Dimension (m)	$24 \times 24 \times 24$
Initial pressure (psi)	2473
Heel of production well (m)	(324,900,1712)
Toe of production well (m)	(612,900,1712)
Production rate (STB/D)	5000
Reservoir life (years)	8
Location of injection well 1 (m)	(84,84,1712)
Location of injection well 2 (m)	(1356,84,1712)
Location of injection well 3 (m)	(84,1356,1712)
Location of injection well 4 (m)	(1356,1356,1712)
Injection rate (STB/D)	1000
Initial trust region radius (m)	100
Minimum trust region radius (m)	5
Expansion factor	1.5
Contraction factor	0.5

Table 5.4: Reservoir parameters and trust region algorithm parameters

5.2.2 Optimization solutions

Table 5.5 shows the optimization results at each iteration in Case 2. The objective function value from the initial well placement is 2097440 SM^3, which increases to 2166840 SM^3 after 309 simulations. In this case, the minimum trust region size is 5 m and there are total 11 iterations when the optimization process terminated. However, there are only four successful iterations (e.g., iteration 1, 2, 4, 9). For the first three iterations, they are very successful models and the trust region sizes are appropriate (100 m, 150 m and 112.5 m). Therefore, the objective function values increase significantly.

From the iteration 5 to iteration 8, there is no successful model. The trust region radius are reduced by halve at each unsuccessful model. Hence, the radius are reduced significantly from 168.75 m to 10.5460 m. The tentative best location of producer during these iterations is found at iteration 4, which has objective function value 2165580 SM^3.

For iteration 9, the increase is not apparently although the quadratic model gives a good prediction. The main reason is relatively small trust region radius (10.5469 m) at this stage limits the increase. However, we can't adjust this subregion directly. Because the trust region radius are depend on previous iterations and some parameters. At this iteration, we have already found the best location of producer in Case 2. The trust region radius is 15.8203 m at this iteration, which is larger than minimum step size 5 m. Hence, the optimization process continues. Unfortunately, the last two iterations are unsuccessful. In such a situation, we could stop optimization process earlier to reduce the simulation runs. For example, the termination condition can be minimum trust region size 10 m.

No. of Iteration	Trust region size	Model's validity	Objective function value
1	100	Good	2129760
2	150	Good	2149150
3	225	Poor	2149150
4	112.5	Good	2165580
5	168.75	Poor	2165580
6	84.375	Poor	2165580
7	42.1875	Poor	2165580
8	21.0938	Poor	2165580
9	10.5469	Good	2166840
10	15.8203	Poor	2166840
11	7.91016	Poor	2166840

Table 5.5: Optimization results at each iteration from Case 2

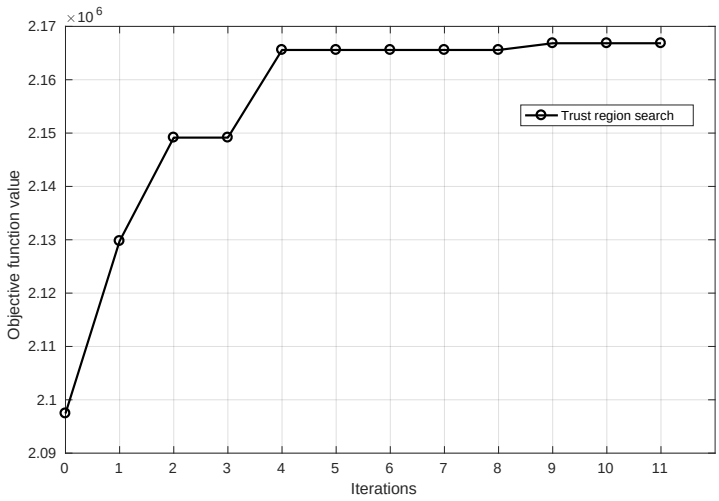

Figure 5.9: Algorithm performance of trust region method in Case 2

Figure 5.10 shows the initial locations and best locations of wells in Case

2. The initial location of producer is along the east-west direction. After trust region search, the best location of producer is along the north-east direction and it is located at the north-west of reservoir. Figure 5.11 displays the oil saturation distributions at the end of the production time. Compared with the initial locations for wells, more oil is produced in the west area and the north area of reservoir from the best locations of wells. Figure 5.12 shows the cumulative oil production and cumulative water production from the initial well locations and the best well locations. The cumulative oil production at the end of production time increases from 3316884.5 SM^3 to 3511177 SM^3. But the cumulative water production from the best locations of wells is less than the initial locations of wells. It is reduced from 6838727 SM^3 to 6721689 SM^3.

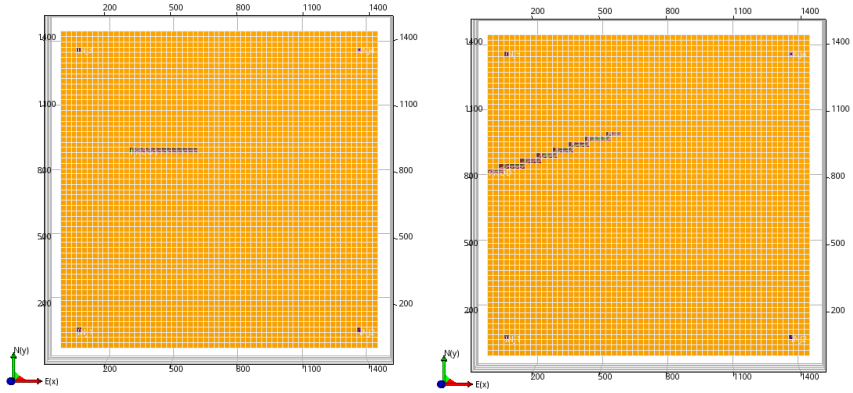

Figure 5.10: The initial location and the best location of producer in Case 2

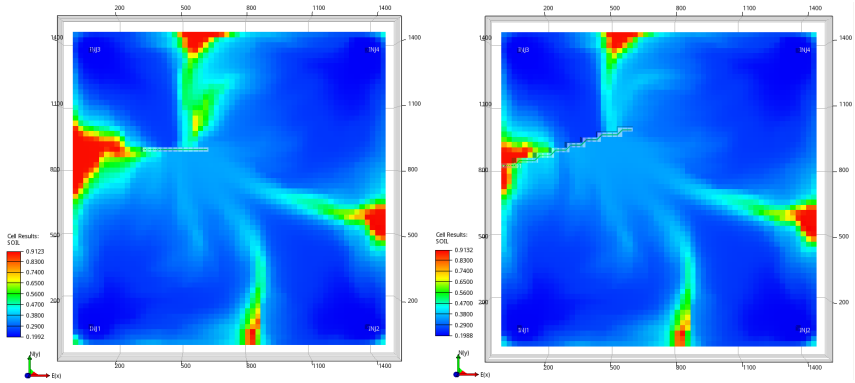

Figure 5.11: Oil saturation distribution at the end of production life from the initial location and the best location of producer in Case 1

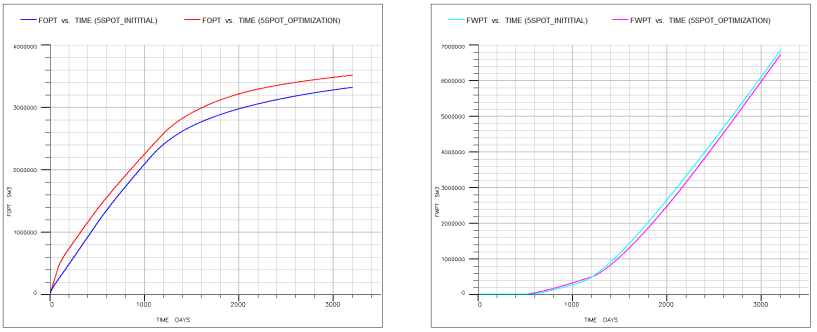

Figure 5.12: Comparison of FOPT and FWPT from the initial location and the best location of producer in Case 2

5.2.2.1 Different values of parameters

In order to extend the optimization process, we can reduce the minimum step size for the termination condition. But there is no significant difference between maximum objective function values, which means that more iterations with small trust region sizes can't improve the optimization process apparently. Therefore we want to adjust values of parameters to get more successful models. In this case, we focus on the expansion factor and the contraction factor. Table 5.6 shows the optimization results in Case 2 from various parameters values.

As we explained previously, the trust region sizes are reduced by halve at each unsuccessful iteration. So the trust region radius are decreased significantly and the optimization process terminated after several unsuccessful iterations. Therefore, we want to use larger contraction factors (0.7 and 0.8) to decrease the reduction in trust region sizes. However, the results show that higher contraction factor didn't improve the performance of algorithm or find better results (see Figure 5.13). When the extraction factor is 0.5, iteration 3 is unsuccessful. When the extraction factor is 0.7, both iteration 3 and iteration 4 are unsuccessful. When the extraction factor is 0.8, iteration 3, iteration 4 and iteration 5 are unsuccessful. Actually, the final best cases from these three situations are similar. But it takes 23 iterations to reach the best locations of wells with the contraction factor 0.8 and there are 19 unsuccessful iterations during the optimization process. Hence, we should not use high contraction factor, which can only result in more unsuccessful iterations.

Figure 5.14 shows the optimization results from three different values of expansion factor (1.5, 2.00 and 2.50). Because the first iteration is successful, the trust region size is increased at iteration 2. The surrogate model is still very good at this iteration. Compared with the expansion factor 1.5, the tentative best case at iteration 2 has larger objective function value with expansion factor 2.0 and 2.5. However the improvement is not significant. In addition, there are more unsuccessful iterations with larger expansion factor. Hence, we suggest expansion factor 1.5 for this five-spot pattern.

Expansion factor	Contraction factor	Successful iterations	Unsuccessful iterations	Maximum value
1.5	0.5	4	7	2166840
1.5	0.7	4	13	2166520
1.5	0.8	4	19	2166920
1.50	0.5	4	7	2166840
2.00	0.5	5	11	2168200
2.50	0.5	5	10	2166420

Table 5.6: Results of iterations from different contraction factors in Case 2

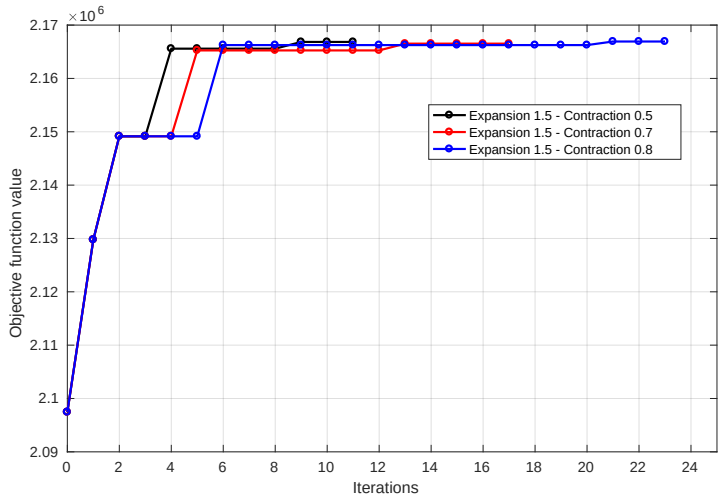

Figure 5.13: Optimization results from different contraction factors in Case 2

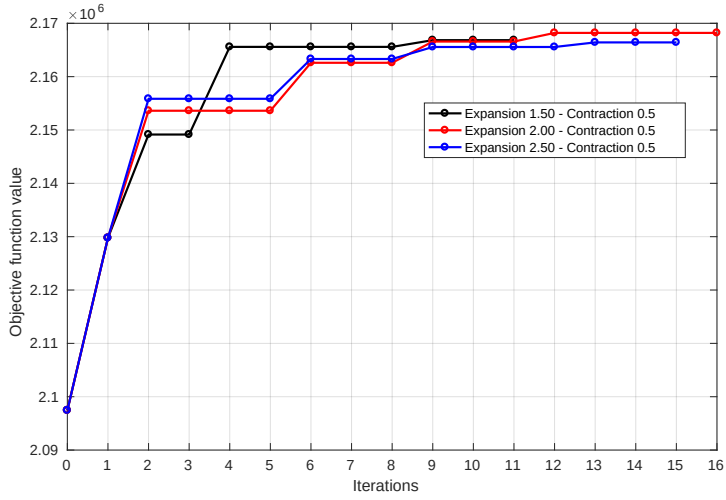

Figure 5.14: Optimization results from different contraction factors in Case 2

5.2.2.2 Different initial locations of producer

From the optimization results (see Table 5.5), the objective function value of the best locations of wells is increased only by 3 % from 2097440 SM^3 to 2166840 SM^3. There are maybe two reasons for such a small growth. First, the initial well locations is good and close to best well locations (see Figure 5.10). In addition, the simulation time could be another reason. After long time of production, a lot of oil in the reservoir will be produced regardless of the locations of wells. Therefore, there is no significant difference in the oil saturation distribution at the end of production time. In order to get more significant improvement, we reduce the simulation time from 8 years to 5 years and solve the well placement optimization problem from another four different initial locations of wells (north-east, north-west,north-south and east-west). The other initial conditions are unchanged.

Table 5.7 summaries the initial locations of wells and theirs corresponding optimization results after trust region search. Although we have used different initial locations of producer, the best location is always along the north-east direction (see Figure 5.10, 5.15 5.18 and 5.21). In Test 2 and Test 4, the initial locations of wells are far way from the best locations of well. Hence, the increasing in objective function value should be significant and much more oil will be produced from the best locations of well. As we expected, the oil saturation at the end of production time from the best locations of well is much less than the initial locations of wells (see Figure 5.19 and 5.22). The objective function value is increased by 33% from 2083140 $SM^3$to 2771040 SM^3 in Test 3, which has the most significant improvement in all the tests. Cumulative oil production at the end of production time is increased from 2590176 SM^3 to 3134591 SM^3 and the cumulative water production at the end of production time is reduced from 2535182 SM^3 to 1817738 SM^3 (see Figure 5.23).

	Test 1	Test 2	Test 3	Test 4
Production rate (STB/D)	5000	5000	5000	5000
Injection rate (STB/D)	1000	1000	1000	1000
Simulation time (years)	5	5	5	5
Initial heel (i,j,k)	(14,14,1)	(46,14,1)	(13,13,1)	(23,32,1)
Initial toe (i,j,k)	(41,41,1)	(46,33,1)	(30,13,1)	(41,16,1)
Initial direction	north-east	north-south	north-west	east-west
Best heel (i,j,k)	(7,31,1)	(7,31,1)	(5,32,1)	(2,32,1)
Best toe (i,j,k)	(28,52,1)	(29,52,1)	(26,48,1)	(26,36,1)
Best direction	north-east	north-east	north-east	north-east
Initial value (SM^3)	2513320	2357380	2083140	2527240
Maximum value (SM^3)	**2767310**	**2770040**	**2771040**	**2710850**
Percent increase (%)	**10.1%**	**17.5%**	**33%**	**7.3%**
FOPT - Initial location (SM^3)	2955580	2842019	2590176	2966097
FOPT - Best location (SM^3)	3132627	3134558	3134591	3096942
Percent increase -FOPT (%)	6%	10.3%	21.2%	4.5%
FWPT - Initial location (SM^3)	2211300	2423213	2535182	2194306
FWPT - Best location (SM^3)	1826596	1822586	1817738	1930438
Percent decrease -FWPT (%)	17.4%	24.8%	28.3%	12%

Table 5.7: Optimization results of different initial conditions

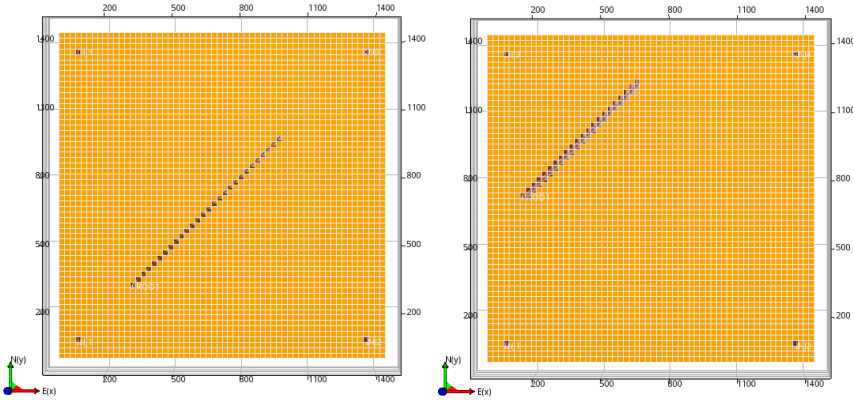

Figure 5.15: The initial locations and the best locations of wells in Test 1

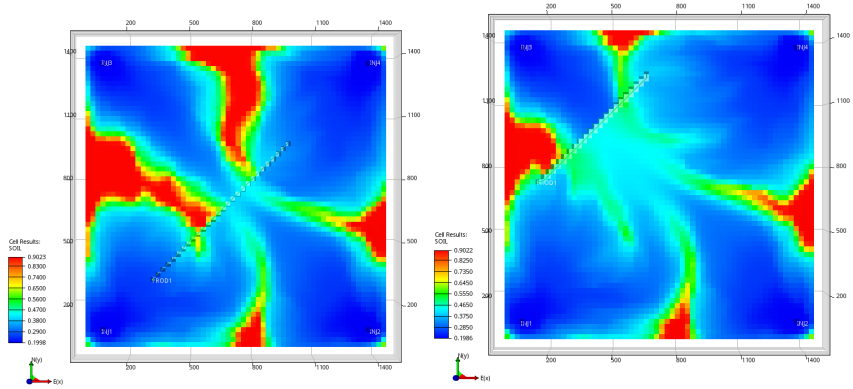

Figure 5.16: Oil saturation at the end of production time from the initial locations and the best locations of wells in Test 1

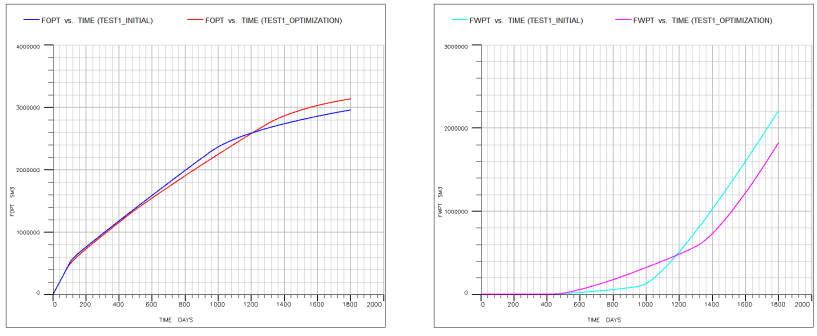

Figure 5.17: Comparison of FOPT and FWPT from the initial locations and the best locations of wells in Test 1

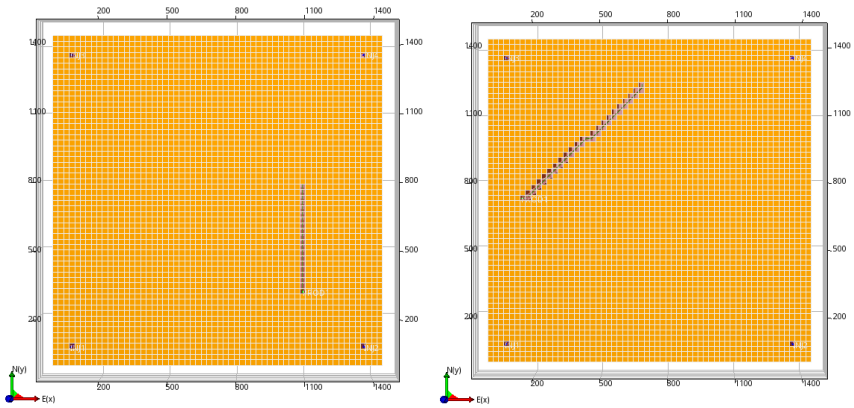

Figure 5.18: The initial locations and the best locations of wells in Test 2

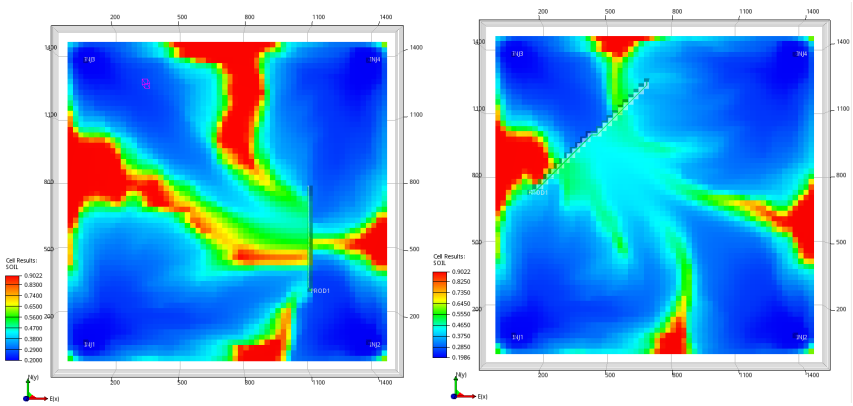

Figure 5.19: Oil saturation at the end of production time from the initial locations and the best locations of wells in Test 2

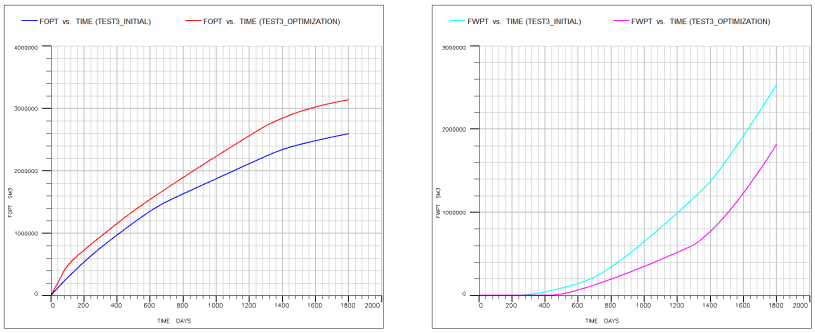

Figure 5.20: Comparison of FOPT and FWPT from the initial locations and the best locations of wells in Test 2

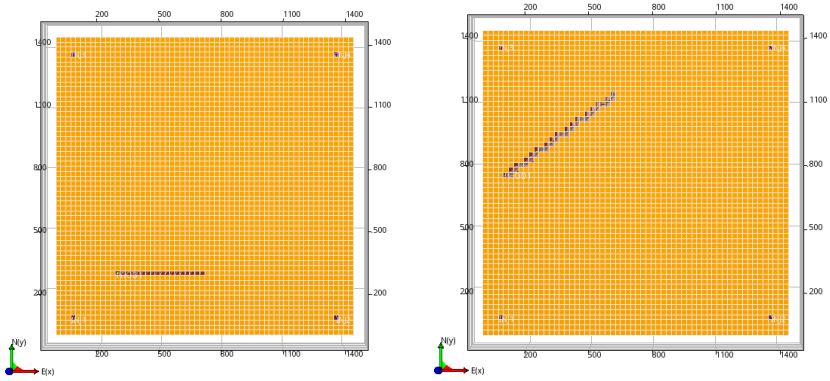

Figure 5.21: The initial locations and the best locations of wells in Test 3

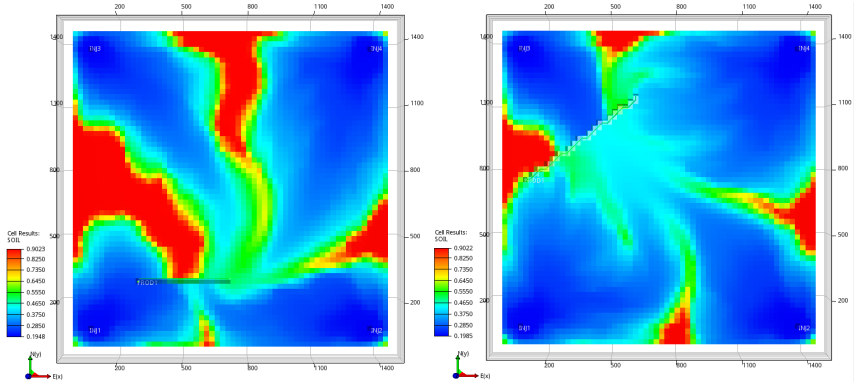

Figure 5.22: Oil saturation at the end of production time from the initial locations and the best locations of wells in Test 3

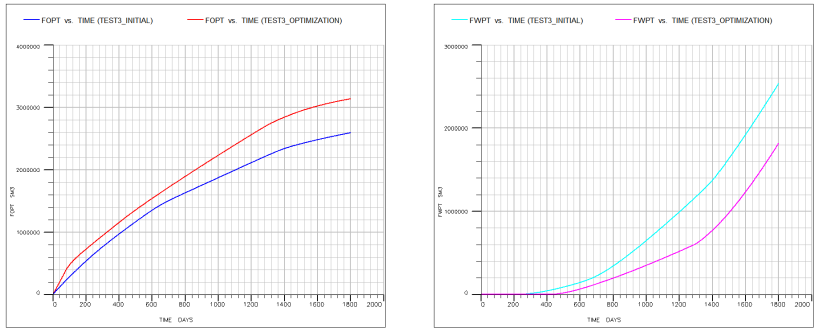

Figure 5.23: Comparison of FOPT and FWPT from the initial locations and the best locations of wells in Test 3

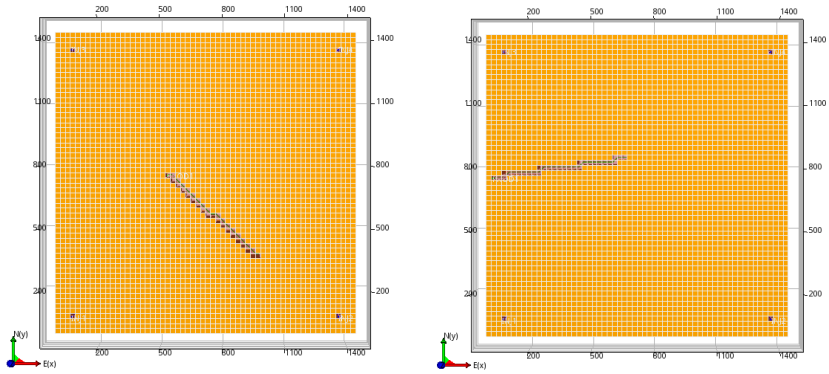

Figure 5.24: The initial locations and the best locations of wells in Test 4

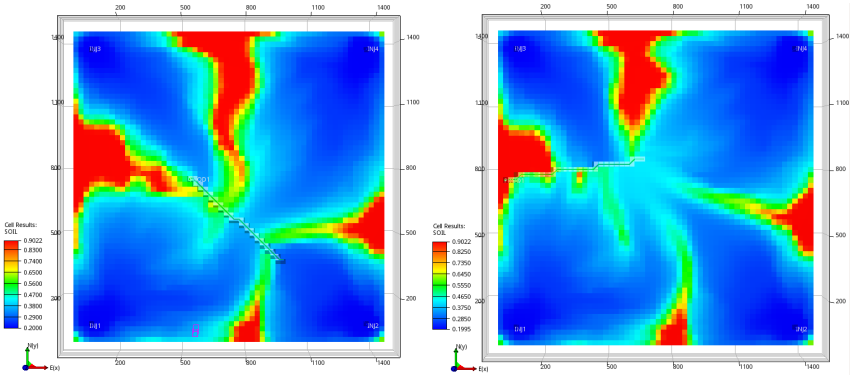

Figure 5.25: Oil saturation at the end of production time from the initial locations and the best locations of wells in Test 4

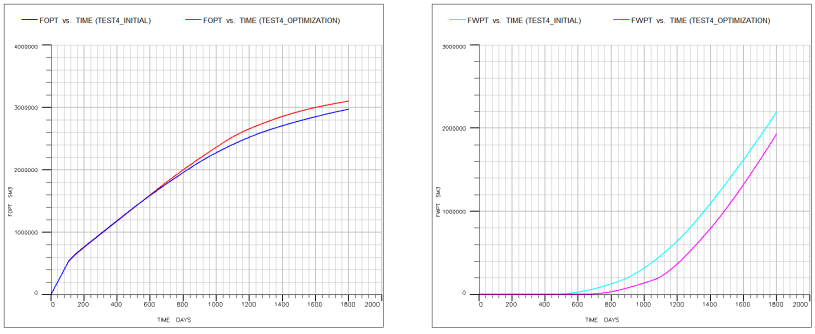

Figure 5.26: Comparison of FOPT and FWPT from the initial locations and the best locations of wells in Test 4

5.3 Case 3

5.3.1 Case description

In this case, we consider a simple three dimensional reservoir with one production well. The dimensions of reservoir are $2000 \times 990 \times 300$ ft^3 and represented by $20 \times 9 \times 9$ grid system. The reservoir and initial data are shown in Table 5.8 and 5.9. The reservoir has nine layers and porosity is constant 25% (see Figure 5.27). The horizontal and vertical permeability is 100 md and 5 md, respectively. The initial oil saturation distribution is shown in the Figure 5.28. In the top two layers, the gas saturation is very high. The bottom two layers contain mostly water. The oil saturation is stratified horizontally and the third layer has largest oil saturation (83.9%). Therefore, the appropriate locations of producer should be horizontal between layer 2 and layers 7. As we already know the good locations of production well, this reservoir model can be used to evaluate the validity of trust region algorithm in well placement optimization problem.

Layer	Thickness ft	Depth of the center of layer ft	k_h md	k_v md
1	50	7025	100	5
2	50	7075	100	5
3	30	7115	100	5
4	15	7137.5	100	5
5	10	7150	100	5
6	15	7162.5	100	5
7	30	7185	100	5
8	50	7225	100	5
9	50	7275	100	5

Table 5.8: Reservoir data

Layer	Thickness ft	p_o psia	S_o	S_w	S_g
1	50	3845	0	0.1	0.9
2	50	3854	0.151	0.115	0.734
3	30	3866	0.839	0.147	0.014
4	15	3872	0.831	0.169	0
5	10	3876	0.806	0.194	0
6	15	3879	0.769	0.231	0
7	30	3885	0.506	0.494	0
8	50	3895	0.005	0.995	0
9	50	3918	0	1	0

Table 5.9: Reservoir initial data

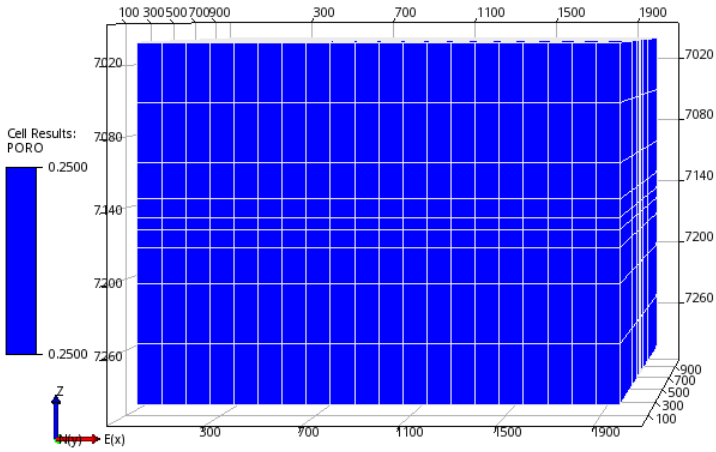

Figure 5.27: Porosity field of reservoir

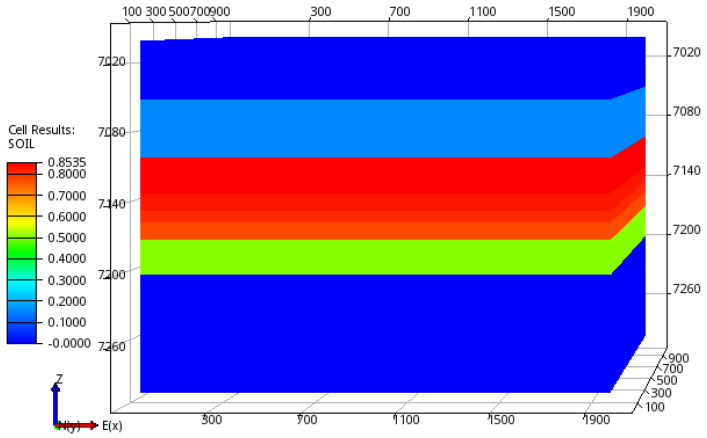

Figure 5.28: Initial oil saturation distribution

5.3.2 Optimization solutions

To test the performance of trust region algorithm in this case, we consider three different initial locations of production well: vertical well, inclined well and horizontal well. The reservoir has only one production well operating at a constant production rate of 2000 STB/D. The simulation time is 100 days. Table 5.10 summaries the initial well placement and conditions for each test.

	Test 1	Test 2	Test 3
Liquid production rate (STB/D)	2000	2000	2000
Direction of well	Vertical	Inclined	Horizontal
Producer length (ft)	110	907	900
Layer	Layer 2-7	Layer 2-7	Layer 7
Initial heel (ft)	(1050,495,7075)	(550,495,7075)	(550,495,7185)
Initial toe (ft)	((1050,495,7185)	(1450,495,7185)	(1450,495,7075)

Table 5.10: Initial well placement for each test in Case 3

As we explained before, the oil saturation is stratified horizontally (see Figure 5.28. Hence, we suggest that production well should be drilled horizontally on the layers with large oil saturation, for example, the layer 3. Table 5.11 shows the optimization results of all the tests. The best locations of producer is always on the layer 3 (see Figure 5.29,5.31 and5.33). Figure 5.30, 5.32 and 5.34 show the cumulative oil production and the cumulative water production from the initial location of producer and the best location of producer. We notice that much more oil is produced when the producer is drilled on the layer 3. The cumulative water production in each test is reduced significantly, which are are negligibly small. The optimization results after trust region search are as we expected. Hence, the trust region algorithm is efficient in this case.

	Test 1	Test 2	Test 3
Initial direction	Vertical	Inclined	Horizontal
Best direction	Horizontal	Horizontal	Horizontal
Best layer	Layer 3	Layer 3	Layer 3
Heel (i,j,k)	(4,9,3)	(6,6,3)	(6,5,3)
Toe (i,j,k)	(14,2,3)	(19,5,3)	(15,5,3)
Maximum value	199701	199712	199683

Table 5.11: Best locations of producer after trust region search

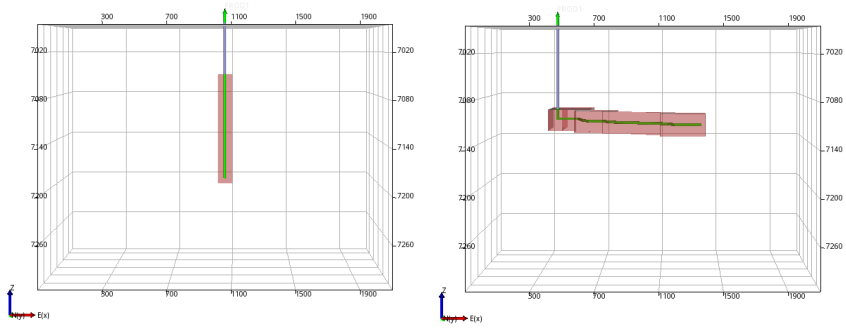

Figure 5.29: The initial location and the best location of producer in Test 1

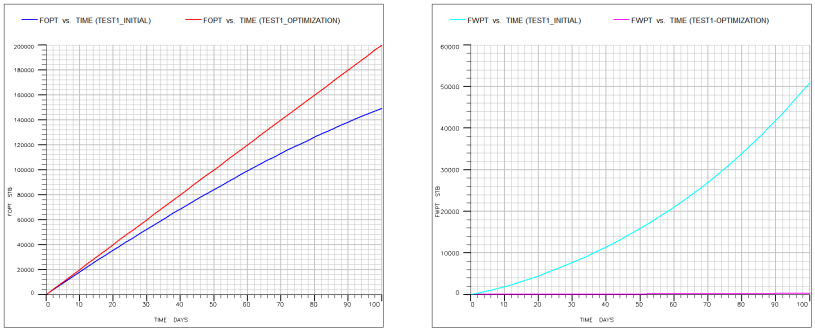

Figure 5.30: Comparison of FOPT and FWPT from the initial location and the best location of producer in Test 1

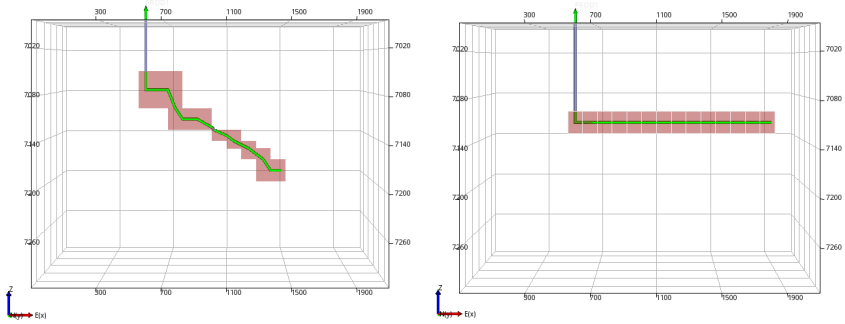

Figure 5.31: The initial location and the best location of producer in Test 2

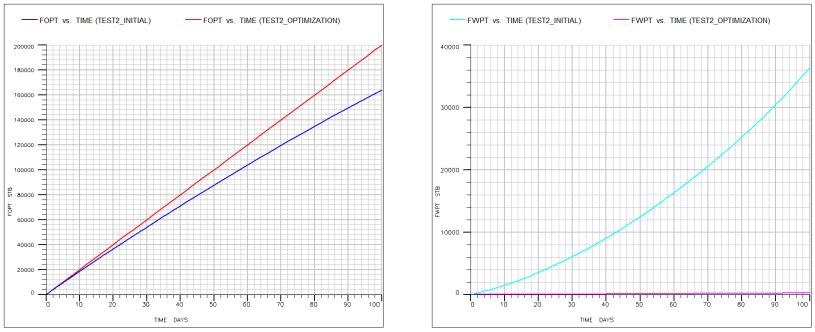

Figure 5.32: Comparison of FOPT and FWPT from the initial location and the best location of producer in Test 2

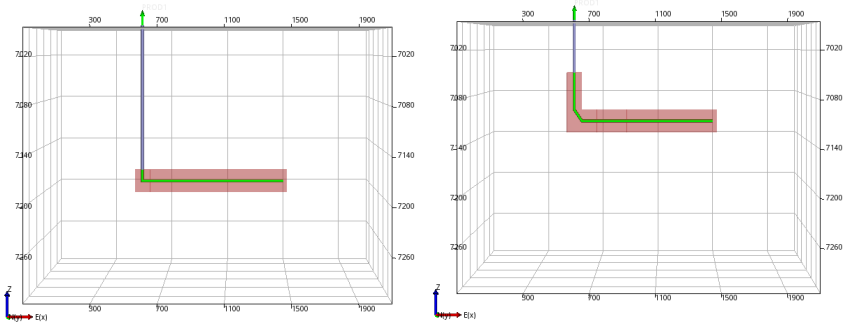

Figure 5.33: The initial location and the best location of producer in Test 3

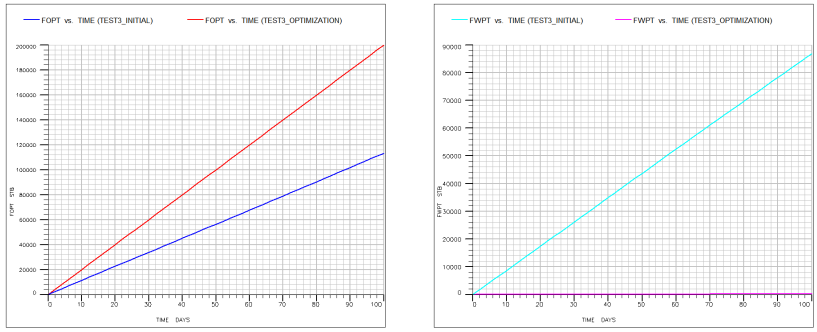

Figure 5.34: Comparison of FOPT and FWPT from the initial location and the best location of producer in Test 3

Chapter 6

Summary

Well placement optimization is one of the most important part in field development. It is a challenging process because evaluating this type of objectives typically requires performing a time-consuming reservoir simulation. An efficient algorithm is essential to reduce simulation runs and accelerate optimization process. In our work, we implement model-based trust region algorithm in FieldOpt for well placement optimization problems. The values of parameters can influence the performance of this algorithm. We should choose proper parameters to improve the convergence of method. In our work, we recommend expansion factor 1.5, contraction factor 0.5 and intial step size 100 ft as input parameters for example cases.

We have applied both the Cauchy point method and Dogleg method to solve the trust region subproblem. Cauchy point performs poorly in some cases, for example, Matyas function. A significant improvement in convergence could be achieved by using Dogleg method. However, when we used Dogleg method for well placement in FieldOpt. The results show that the Dogleg method does not accelerate the optimization process significantly. In well placement problem, quasi-Newton points are always located outside the trust region area. Althought we used Dogleg method to solve the trust region subproblem, the Dogleg path is not available when quasi-Newton point is outerior. Then Cauchy point is taken as optimization step in such a situation. When the Cauchy point method is applied, there are more successful iteartions before the optimization process is terminated. Therefore the final case based on Cauchy point is better than that from Dogleg method. In addition to that, we also notice that appropriate termation condition can reduce unnecessary iterations and inital well placement is aslo an imporant factor during the optimization process in Case 2. In Case 3, the reservoir has horizontal oil saturation distrution and the approriate location of producer is known. The optimization results validated the perfomance of trust region algorithm.

Chapter 7

Further work

The current trust region algorithm implemented with FieldOpt has some drawbacks. Due to lack of time, we were not able to solve some problems. In the future, we can improve the performance of the trust region algorithm from the following aspects.

7.1 Constrained optimization

In our work, we only optimize a single well placement. There are six variables for one well and only reservoir boundary conditions are considered as constraints. In the future, we will solve multiple-well placement optimization problem. Therefore, there are more variables in optimization process and we have to take into account more constraints. For example, well length constraint and inter-well distance constraint.

7.2 Surrogate model

Compared to the true function, surrogate model is computationally simple to evaluate and processes well-behaved derivatives. But there is a significant drawback of quadratic model. The number of interpolation points $\frac{(\mathbf{n}+1)(\mathbf{n}+2)}{2}$ is the quadratic in the dimension $\mathbf{n}$ of the true function. For example, $\mathbf{n} = 24$ for four wells, there are nearly 325 cases needed to be evaluated at each iteration. Hence, the procedure to evaluate objective function from simulator can be very time consuming. Powell [18] proposed an algorithm that uses a quadratic model relying on fewer than $\frac{(\mathbf{n}+1)(\mathbf{n}+2)}{2}$ interpolation points. Beyond quadratic model, we can also consider radial basis function model, which uses fewer function evaluations than the polynomial models.

7.3 Optimization step

Actually, both exact solution and Dogleg method have some drawbacks. For exact solution, $B_k + \lambda I$ need to be positive definite and the process is complicated to calculate all eigenvectors and eigenvalues of B_k to find the optimal trajectory. So the exact solution should be applied to small dimension. Dogleg method works when B_k is positive and it is not suited for large optimization scale. However, truncated conjugate method works for arbitrary B_k and has more and all the good properties of Dogleg method. In addition to that, the solution calculated by this method has a good sufficient reduction property [38]. Therefore, we suggest the truncated conjugate gradient method to solve the trust region subproblem.

Appendix A

Code

A.1 Code for interpolation points

`find_new_point()`

```
QList<Eigen::VectorXd> PolyModel::find_new_point(
    Polynomial basis_function)
{
    int dimension = basis_function.return_dimension();
    Eigen::VectorXd coeffs = basis_function.return_coeffs();
    Eigen::VectorXd x0, x1, x2, x3, x4;
    x0 = x1 = x2 = x3 = x4 = Eigen::VectorXd::Zero(dimension);

    // Find largest monomial coefficient (excluding constant term
        ↪ which has already been assigned to first point)
    double max = 0.0;
    int max_coeff = -1;
    for (int i = 1; i < coeffs.size(); ++i) {
        if(fabs(coeffs(i)) > max) {
            max = fabs(coeffs(i));
            max_coeff = i;
        }
    }

    if(max_coeff == -1){
        std::cout << "Good point alg, Problem: all coefficients are
            ↪ zero, should never happen" << std::endl;
        for(int i = 0; i < 3; i++){
            std::cout << coeffs[i] << std::endl;
        }
    }

    if(max_coeff <= dimension){
        // The largest coefficient is from a linear term
        x1(max_coeff-1) = 1;
        x2 = -x1;
    }
```

```
    else if(max_coeff <= 2*dimension){
        // Largest coefficient is quadratic monomial
        x1(max_coeff-dimension-1) = 1;
        x2 = -x1;
    }
    else{
        int l,m = -1;
        int coeff_dummy = 2*dimension+1;

        for(int i = 0; i < dimension-1; i++){
            for (int j = i+1; j < dimension; j++) {
                if (max_coeff == coeff_dummy) {
                    l = i;
                    m = j;
                    goto endloop;
                }
                coeff_dummy = coeff_dummy+1;
            }
        }
        endloop:

        x1(l) =  1.0/sqrt(2);
        x1(m) = -1.0/sqrt(2);
        x2 = -x1;
        for(int i = 0; i < dimension; i++){
            x3(i) = fabs(x1(i));
            x4(i) = -fabs(x1(i));
        }
    }

    Eigen::VectorXd best_point;
    double best_value = 0.0;
    QList<Eigen::VectorXd> points;
    points.append(x0);
    points.append(x1);
    points.append(x2);
    points.append(x3);
    points.append(x4);

    // Determine which of the 5 points is the best one
    for(int i = 0; i < 5; i++){
    if(fabs(basis_function.evaluate(points.at(i))) >= best_value){
        best_point = points.at(i);
        best_value = fabs(basis_function.evaluate(points.at(i)));
      }

    }

 return best_point;
}
```

`complete_points_abs()`

```
void PolyModel:: complete_points_abs()
{
    Eigen::VectorXd centre_point = points_.at(0);
     // points_abs=scaled interpolation point.
     // points_abs.at(0) is always zero
     points_abs.append(points_.at(0) - centre_point);
     int n_Polynomials = basis_.length();
     int n_points = points_.length();

     QList<Polynomial> temp_basis = get_basis();
     for (int i = 0; i < n_Polynomials; i++) {
         Polynomial cur_pol = temp_basis.at(i);
         if(i == 0){
             std::cout << "we don't need to find new point, basis
                 ↪ polynomial, i = " << i << std::endl;
             std::cout << " This point is " << points_abs.at(i) <<
                 ↪ std::endl;
         }
         else{
             std::cout << "we need to find new point, basis
                 ↪ polynomial, i = " << i << std::endl;
             Polynomial temp_poly_here = temp_basis.at(i);
             // Append new point and swap it to current position
             points_abs.append(find_new_point(temp_poly_here));

             //points_abs.swap(i, points_abs.length() - 1);
             //std::cout<<"this point is " << points_abs.at(i) << std
                 ↪ ::endl;
             Polynomial temp_i = temp_basis.at(i);
             auto temp_point = points_abs.at(i);

             for (int j = i + 1; j < n_Polynomials; j++) {
                Polynomial uj = temp_basis.at(j);
                Polynomial ui = temp_basis.at(i);
                double ratio;
                //Testing division by zero
                if(ui.evaluate(temp_point) == 0 && j == i+1){
                 std::cout << "Division by zero because U_i(y_i) = 0"
                     ↪ << std::endl;
                 }
                if(uj.evaluate(temp_point) == ui.evaluate(temp_point)
                    ↪ ){
                 ratio = 1.0;
                 }
                else{
                     ratio = uj.evaluate(temp_point) / ui.evaluate(
                         ↪ temp_point);
                 }
                 ui.multiply(-1.0 * ratio);
                 uj.add(ui);
                 temp_basis[j] = uj;
             }
         }
     }

}
```

`complete_points()`

```
void PolyModel:: complete_points()
{
    int n_Polynomials = basis_.length();
    Eigen::VectorXd centre_point = points_.at(0);
    std::cout << "for i=0" << std::endl;
    std::cout << "This interpolation point is:" << points_.at(0) <<
        ↪ std::endl;
    //Scale points back
    for (int i = 1; i < n_Polynomials; ++i) {
        points_.append(centre_point + radius_*points_abs.at(i));
        std::cout << "for i=" << i << std::endl;
        std::cout << "This interpolation point is:" << points_.at(
            ↪ points_.size()-1) << std::endl;

        //Create case from new interpolation point
        cases_.append(CaseFromPoint(centre_point + radius_*
            ↪ points_abs.at(i), cases_.at(0)));

        // Append case to list of unevaluated cases
         cases_not_eval_.append(cases_.at(i));

    }
    needs_evals_ = true;
    needs_set_of_points_ = false;
    std::cout << "Interpolation points are completed:" << std::endl
        ↪ ;
}
```

A.2 Code for optimization step

`Hessain()`

```
QList<Eigen::VectorXd> Polynomial::Hessian()
{
  /* Hessian matrix is a square matrix of second-order
   * partial derivative of a scalar-valued function.
   * for quadratic poly model, Hessian matrix only
   * depend on cofficients
   */
  Eigen::MatrixXd B = Eigen::MatrixXd::Zero(dimension_, dimension_);

    for(int i = 0; i < dimension_; ++i){
            B(i,i) = coeffs_(dimension_+i+1);
    }

    for (int i = 0;i < dimension_; ++i){
        int k = 0;
        for (int j = i+1; j< dimension_; ++j) {
            int a = ((dimension_ + 1) + (dimension_ - i)) * (i + 2)
                ↪ / 2 + k;;
            B(i,j) = coeffs_(a);
            B(j,i) = B(i,j);
            k = k+1;
        }
```

```
    }

    Hessian_Matrix = B;
    //std::cout << "Hessian matrix: " << B << std::endl;

 return Hessian_Matrix;
}
```

`Cauchy_point()`

```
QList<Eigen::VectorXd> Polynomial::Cauchy_Point(
    Eigen::VectorXd points, double radius_, Eigen::VectorXd grad)
{
    double gBg = grad.transpose()*Hessian_Matrix*grad;
    double tau;
    if(gBg <= 0) {
       tau = 1;
    }
    else {
       double length = grad.norm()*grad.norm()*grad.norm()/(radius_*
           ↪ gBg);
       if (length >= 1){
           std::cout << "The Cauchy Point lies on the boundary, then
               ↪  take a step to the boundary along steepest
               ↪ descent direction:" << std::endl;
           tau = 1;
       }
       else{
           std::cout << "The Cauchy Point lies inside the trust
               ↪ region" << std::endl;
           tau = length;
       }
   }
    Eigen::VectorXd Cauchy_Point = -tau*radius_*grad/grad.norm();
    // std::cout << "tau is: " << tau << std::endl;
    std::cout << "The Cauchy Point is:\n" << Cauchy_Point << std::
        ↪ endl;
    std::cout << "Length of Cauchy Point is:\n" << Cauchy_Point.norm
        ↪ () << std::endl;
    //std::cout << "objective function value of Current Point is:"
        ↪ << evaluate(points) << std::endl;
    //std::cout << "objective function value of Cauchy Point is: "
        ↪ << evaluate(Cauchy_Point) << std::endl;
    return Cauchy_Point;
}
```

`Newton_point()`

```
QList<Eigen::VectorXd> Polynomial:: Newton_Point(
    Eigen::VectorXd points, double radius_, Eigen::VectorXd grad)
{
    std::cout << "Find Quastion-Newton point:" << std::endl;
    Eigen::MatrixXd Inverse_Matrix = Hessian_Matrix.inverse();
    Eigen::VectorXd Newton_Point = -Inverse_Matrix*grad;
    if (Newton_Point.norm() > radius_){
        std::cout << "The Newton Point lies outside the trust region
            ↪ " << std::endl;
    }
    else{
        std::cout << "The Newton Point lies inside the trust region"
            ↪  << std::endl;
    }

    std::cout << "The Quasi-Newton Point is:\n" << Newton_Point <<
        ↪ std::endl;
    std::cout << "length of Newton Point is:\n" << Newton_Point.norm
        ↪ () << std::endl;
    //std::cout << "objective function value of Current Point is:\n"
        ↪  << evaluate(points) << std::endl;
    //std::cout << "objective function value of Newton Point is:\n"
        ↪ << evaluate(Newton_Point) << std::endl;

    return Newton_Point;

}
```

`Dogleg_step()`

```
QList<Eigen::VectorXd> Polynomial::Dogleg_step(
    Eigen::VectorXd points, double radius_, Eigen::VectorXd grad)
{
    Eigen::VectorXd Dogleg_step;
    Eigen::VectorXd d_cp = Cauchy_Point(points, radius_,grad);
    Eigen::VectorXd d_nw = Newton_Point(points, radius_,grad);
    std::cout << "length of raidus" << radius_ << std::endl;
    std::cout << "length of Cauchy point is" << d_cp.norm() << std::
        ↪ endl;
    std::cout << "length of Newton point is" << d_nw.norm() << std::
        ↪ endl;

    if(d_cp.norm() >= radius_){
        Dogleg_step = d_cp;
        std::cout << "Cauchy point lies on the boundary, then take
            ↪ Cauchy point as optimization step: " << Dogleg_step
            ↪ << std::endl;
    }
    else if (d_nw.norm() <= radius_)
    {
        Dogleg_step = d_nw;
        std::cout << "Both Cauchy point and Quastion-Newton point
            ↪ lie inside trust region, then take Newton point as
            ↪ optimization step: " << Dogleg_step << std::endl;

    }
```

```
    else{
        std::cout << "Cauchy point lies inside trust region, 
            ↪ Quastion-Newton point lies outside trust region, then
            ↪ take Dogleg optimization step:" << Dogleg_step <<
            ↪ std::endl;
        std::cout << "Find tau such that || d_cp + tau*(d_nw - d_cp)
            ↪ || _2 = radius :" << std::endl;
        // solve a single quadratic equation
        double tau;
        Eigen::VectorXd diff = d_nw-d_cp;
        double a = diff.dot(diff);
        double b = 2*d_cp.dot(diff);
        double c = d_cp.dot(d_cp)-radius_*radius_;
        tau = (-b+sqrt(b*b-4*a*c))/(2*a);
        std::cout << "tau is:" << tau << std::endl;
        Dogleg_step = d_cp+tau*(d_nw-d_cp);
        std::cout << "Dogleg optimization step is:" << Dogleg_step
            ↪ << std::endl;
        std::cout << "length of Dogleg point is:" << Dogleg_step.
            ↪ norm() << std::endl;
    }
    return Dogleg_step;

}
```

Appendix B

Driver Files

B.1 Case 1

```
{
    "Global": {
        "Name": "ONE_PRODUCER",
        "BookkeeperTolerance": 32.0
    },
    "Optimizer": {
        "Type": "Trustregion",
        "Mode": "Maximize",
        "Parameters": {
            "MaxEvaluations": 800,
            "InitialStepLength": 100.0,
            "MinimumStepLength": 5,
            "ContractionFactor": 0.5,
            "ExpansionFactor": 1.5,
        },
        "Objective": {
            "Type": "WeightedSum",
            "WeightedSumComponents": [
                {
                    "Coefficient": 1.0, "Property": "
                        ↪ CumulativeOilProduction", "TimeStep": -1,
                    "IsWellProp": false
                },
                {
                    "Coefficient": -0.2, "Property": "
                        ↪ CumulativeWaterProduction", "TimeStep":
                        ↪ -1,
                    "IsWellProp": false
                }
            ]
        },
        "Constraints": [
            {
                "Wells": ["PROD1"],
```

```
                "Type": "ReservoirBoundary",
                "BoxImin": 0,
                "BoxImax": 59,
                "BoxJmin": 0,
                "BoxJmax": 59,
                "BoxKmin": 0,
                "BoxKmax": 0
            }
        ]
    },
    "Simulator": {
        "Type": "ECLIPSE",
        "ExecutionScript": "csh_eclrun"
    },
    "Model": {
        "ControlTimes": [0, 50, 100],
        "Reservoir": {
            "Type": "ECLIPSE"
        },
        "Wells": [
            {
                "Name": "PROD1",
                "Group": "G1",
                "Type": "Producer",
                "DefinitionType": "WellSpline",
                "PreferredPhase": "Oil",
                "WellboreRadius": 0.1905,
                "SplinePoints": {
                    "Heel": {
                        "x": 300.0,
                        "y": 900.0,
                        "z": 1712.0,
                        "IsVariable": true
                    },
                    "Toe": {
                        "x": 900.0,
                        "y": 900.0,
                        "z": 1712.0,
                        "IsVariable": true
                    }
                },
                "Controls": [
                    {
                        "TimeStep": 0,
                        "State": "Open",
                        "Mode": "BHP",
                        "BHP": 100.0
                    }
                ]
            }
        ]
    }
}
```

B.2 Case 2

```
{
    "Global": {
        "Name": "5SPOT_INPUT",
        "BookkeeperTolerance": 32.0
    },
    "Optimizer": {
        "Type": "Trustregion",
        "Mode": "Maximize",
        "Parameters": {
            "MaxEvaluations": 1000,
            "InitialStepLength": 100.0,
            "MinimumStepLength": 5,
            "ContractionFactor": 0.5,
            "ExpansionFactor": 1.5
        },
        "Objective": {
            "Type": "WeightedSum",
            "WeightedSumComponents": [
                {
                    "Coefficient": 1.0, "Property": "
                        ↪ CumulativeOilProduction", "TimeStep": -1,
                    "IsWellProp": false
                },
                {
                    "Coefficient": -0.2, "Property": "
                        ↪ CumulativeWaterProduction", "TimeStep":
                        ↪ -1,
                    "IsWellProp": false
                }
            ]
        },
   "Constraints": [
            {
                "Wells": ["PROD1"],
                "Type": "ReservoirBoundary",
                "BoxImin": 0,
                "BoxImax": 59,
                "BoxJmin": 0,
                "BoxJmax": 59,
                "BoxKmin": 0,
                "BoxKmax": 0
            }
        ]
    },
   "Simulator": {
        "Type": "ECLIPSE",
        "ExecutionScript": "csh_eclrun"
    },
    "Model": {
        "ControlTimes": [0, 400, 800, 1200, 1600, 2000, 2400, 2800,
            ↪ 3200],
        "Reservoir": {
            "Type": "ECLIPSE"
        },
        "Wells": [
```

```
        {
            "Name": "PROD1",
            "Group": "G1",
            "Type": "Producer",
            "DefinitionType": "WellSpline",
            "PreferredPhase": "Oil",
            "WellboreRadius": 0.1905,
            "SplinePoints": {
                "Heel": {
                    "x": 12.0,
                    "y": 828.0,
                    "z": 1712.0,
                    "IsVariable": true
                },
                "Toe": {
                    "x": 588.0,
                    "y": 996.0,
                    "z": 1712.0,
                    "IsVariable": true
                }
            },
            "Controls": [
                {
                    "TimeStep": 0,
                    "State": "Open",
                    "Mode": "Rate",
                    "Rate": 5000.0
                }
            ]
        },
         {
            "Name": "INJ1",
            "Group": "G2",
            "Type": "Injector",
            "DefinitionType": "WellSpline",
            "PreferredPhase": "Water",
            "WellboreRadius": 0.1905,
            "SplinePoints": {
                "Heel": {
                    "x": 84.0,
                    "y": 84.0,
                    "z": 1712.0,
                    "IsVariable":true
                },
                "Toe": {
                    "x": 84.0,
                    "y": 84.0,
                    "z": 1712.0,
                    "IsVariable": true
                }
            },
            "Controls": [
                {
                    "TimeStep": 0,
                    "Type": "Water",
                    "State": "Open",
                    "Mode": "Rate",
```

```
                    "Rate": 1000.0
                }
            ]
        },
        {
            "Name": "INJ2",
            "Group": "G2",
            "Type": "Injector",
            "DefinitionType": "WellSpline",
            "PreferredPhase": "Water",
            "WellboreRadius": 0.1905,
            "SplinePoints": {
                "Heel": {
                    "x": 1356.0,
                    "y": 84.0,
                    "z": 1712.0,
                    "IsVariable": true
                },
                "Toe": {
                    "x": 1356.0,
                    "y": 84.0,
                    "z": 1712.0,
                    "IsVariable": true
                }
            },
            "Controls": [
                {
                    "TimeStep": 0,
                    "Type": "Water",
                    "State": "Open",
                    "Mode": "Rate",
                    "Rate": 1000.0
                }
            ]
        },
        {
            "Name": "INJ3",
            "Group": "G2",
            "Type": "Injector",
            "DefinitionType": "WellSpline",
            "PreferredPhase": "Water",
            "WellboreRadius": 0.1905,
            "SplinePoints": {
                "Heel": {
                    "x": 84.0,
                    "y": 1356.0,
                    "z": 1712.0,
                    "IsVariable": true
                },
                "Toe": {
                    "x": 84.0,
                    "y": 1356.0,
                    "z": 1712.0,
                    "IsVariable": true
                }
            },
            "Controls": [
```

```
                    {
                        "TimeStep": 0,
                        "Type": "Water",
                        "State": "Open",
                        "Mode": "Rate",
                        "Rate": 1000.0
                    }
                ]
            },
             {
                "Name": "INJ4",
                "Group": "G2",
                "Type": "Injector",
                "DefinitionType": "WellSpline",
                "PreferredPhase": "Water",
                "WellboreRadius": 0.1905,
                "SplinePoints": {
                    "Heel": {
                        "x": 1356.0,
                        "y": 1356.0,
                        "z": 1712.0,
                        "IsVariable": true
                    },
                    "Toe": {
                        "x": 1356.0,
                        "y": 1356.0,
                        "z": 1712.0,
                        "IsVariable": true
                    }
                },
                "Controls": [
                    {
                        "TimeStep": 0,
                        "Type": "Water",
                        "State": "Open",
                        "Mode": "Rate",
                        "Rate": 1000.0
                    }
                ]
            }

        ]
    }
}
```

B.3 Case 3

```
{
    "Global": {
        "Name": "SPE7_TEST2",
        "BookkeeperTolerance": 32.0
    },
    "Optimizer": {
        "Type": "Trustregion",
        "Mode": "Maximize",
        "Parameters": {
            "MaxEvaluations": 800,
            "InitialStepLength": 100.0,
            "MinimumStepLength": 5,
            "ContractionFactor": 0.5,
            "ExpansionFactor": 1.5
        },
        "Objective": {
            "Type": "WeightedSum",
            "WeightedSumComponents": [
                {
                    "Coefficient": 1.0, "Property": "
                        ↪ CumulativeOilProduction", "TimeStep": -1,
                    "IsWellProp": false
                },
                {
                    "Coefficient": -0.2, "Property": "
                        ↪ CumulativeWaterProduction", "TimeStep":
                        ↪ -1,
                    "IsWellProp": false
                }
            ]
        },
        "Constraints": [
            {
                "Wells": ["PROD1"],
                "Type": "ReservoirBoundary",
                "BoxImin": 0,
                "BoxImax": 19,
                "BoxJmin": 0,
                "BoxJmax": 8,
                "BoxKmin": 0,
                "BoxKmax": 8
            }
        ]
    },
    "Simulator": {
        "Type": "ECLIPSE",
        "ExecutionScript": "csh_eclrun"
    },
    "Model": {
        "ControlTimes": [0, 50, 100],
        "Reservoir": {
            "Type": "ECLIPSE"
        },
        "Wells": [
            {
```

```
                "Name": "PROD1",
                "Group": "G1",
                "Type": "Producer",
                "DefinitionType": "WellSpline",
                "PreferredPhase": "Oil",
                "WellboreRadius": 0.1905,
                "SplinePoints": {
                    "Heel": {
                        "x": 550.00,
                        "y": 495.00,
                        "z": 7075.00,
                        "IsVariable": true
                    },
                    "Toe": {
                        "x": 1450.00,
                        "y": 495.00,
                        "z": 7185.00,
                        "IsVariable": true
                    }
                },
                "Controls": [
                    {
                        "TimeStep": 0,
                        "State": "Open",
                        "Mode": "Rate",
                        "Rate": 2000.0
                    }
                ]
            }
        ]
    }
}
```

Bibliography

[1] L. L. Rogers, F. U. Dowla, and V. M. Johnson, "Optimal field-scale groundwater remediation using neural networks and the genetic algorithm," *Environmental Science & Technology*, vol. 29, no. 5, pp. 1145–1155, 1995.

[2] B. Yeten, L. J. Durlofsky, K. Aziz *et al.*, "Optimization of nonconventional well type, location and trajectory," in *SPE annual technical conference and exhibition*. Society of Petroleum Engineers, 2002.

[3] A. N. Morales, H. Nasrabadi, D. Zhu *et al.*, "A new modified genetic algorithm for well placement optimization under geological uncertainties," in *SPE EUROPEC/EAGE Annual Conference and Exhibition*. Society of Petroleum Engineers, 2011.

[4] M. L. Litvak, J. E. Onwunalu, J. Baxter *et al.*, "Field development optimization with subsurface uncertainties," in *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2011.

[5] R. C. Eberhart, J. Kennedy *et al.*, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[6] J. E. Onwunalu and L. J. Durlofsky, "Application of a particle swarm optimization algorithm for determining optimum well location and type," *Computational Geosciences*, vol. 14, no. 1, pp. 183–198, 2010.

[7] J. E. Onwunalu, L. Durlofsky *et al.*, "A new well-pattern-optimization procedure for large-scale field development," *SPE Journal*, vol. 16, no. 03, pp. 594–607, 2011.

[8] H. Wang, D. Echeverría-Ciaurri, L. Durlofsky, A. Cominelli *et al.*, "Optimal well placement under uncertainty using a retrospective optimization framework," *Spe Journal*, vol. 17, no. 01, pp. 112–121, 2012.

[9] B. Beckner, X. Song *et al.*, "Field development planning using simulated annealing-optimal economic well scheduling and placement," in *SPE annual technical conference and exhibition*. Society of Petroleum Engineers, 1995.

[10] K. P. Norrena, C. V. Deutsch *et al.*, "Automatic determination of well placement subject to geostatistical and economic constraints," in *SPE International Thermal Operations and Heavy Oil Symposium and International Horizontal Well Technology Conference*. Society of Petroleum Engineers, 2002.

[11] G. W. Rosenwald, D. W. Green *et al.*, "A method for determining the optimum location of wells in a reservoir using mixed-integer programming," *Society of Petroleum Engineers Journal*, vol. 14, no. 01, pp. 44–54, 1974.

[12] Z. Bouzarkouna, D. Y. Ding, and A. Auger, "Well placement optimization with the covariance matrix adaptation evolution strategy and meta-models," *Computational Geosciences*, vol. 16, no. 1, pp. 75–92, 2012.

[13] P. G. Tilke, R. Banerjee, V. B. Halabe, B. Balci, M. Thambynayagam, J. B. Spath *et al.*, "Automated field development planning in the presence of subsurface uncertainty and operational risk tolerance," in *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2010.

[14] W. Bangerth, H. Klie, M. Wheeler, P. Stoffa, and M. Sen, "On optimization algorithms for the reservoir oil well placement problem," *Computational Geosciences*, vol. 10, no. 3, pp. 303–319, 2006.

[15] D. Winfield, "Function minimization by interpolation in a data table," *IMA Journal of Applied Mathematics*, vol. 12, no. 3, pp. 339–347, 1973.

[16] M. Powell, *A new algorithm for unconstrained optimization*. Academic Press, 1970.

[17] M. J. Powell, "Uobyqa: unconstrained optimization by quadratic approximation," *Mathematical Programming*, vol. 92, no. 3, pp. 555–582, 2002.

[18] M. Powell, "The newuoa software for unconstrained optmization without derivatives," pp. 255–297, 2006.

[19] J. E. Dennis, Jr and R. B. Schnabel, "Least change secant updates for quasi-newton methods," *Siam Review*, vol. 21, no. 4, pp. 443–459, 1979.

[20] M. Marazzi and J. Nocedal, "Wedge trust region methods for derivative free optimization," *Mathematical programming*, vol. 91, no. 2, pp. 289–305, 2002.

[21] F. V. Berghen and H. Bersini, "Condor, a new parallel, constrained extension of powell's uobyqa algorithm: Experimental results and comparison with the dfo algorithm," *Journal of computational and applied mathematics*, vol. 181, no. 1, pp. 157–175, 2005.

[22] S. M. Wild, "Mnh: A derivative-free optimization algorithm using minimal norm hessians," 2008.

[23] H. Zhang and A. R. Conn, "On the local convergence of a derivative-free algorithm for least-squares minimization," *Computational Optimization and Applications*, vol. 51, no. 2, pp. 481–507, 2012.

[24] C. Böckmann, "Curve fitting and identification of physical spectra," *Journal of computational and applied mathematics*, vol. 70, no. 2, pp. 207–224, 1996.

[25] N. Alexandrov and J. E. Dennis, *Algorithms for bilevel optimization.* Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1994.

[26] R. Oeuvray, "Trust-region method based on radial basis functions with application to biomedical imaging," Ph.D. dissertation, Ecole Polytechique Federale de Lausanne, 2005.

[27] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization.* Siam, 2009, vol. 8.

[28] A. R. Conn and P. L. Toint, "An algorithm using quadratic interpolation for unconstrained derivative free optimization," in *Nonlinear optimization and applications.* Springer, 1996, pp. 27–47.

[29] M. D. Buhmann, "Radial basis functions: theory and implementations," *Cambridge Monographs on Applied and Computational Mathematics*, vol. 12, pp. 147–165, 2003.

[30] K. Scheinberg and P. L. Toint, "Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization," *SIAM Journal on Optimization*, vol. 20, no. 6, pp. 3512–3532, 2010.

[31] Y. xian Yuan, "A review of trust region algorithms for optimization."

[32] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM Journal on Numerical Analysis*, vol. 20, no. 3, pp. 626–637, 1983.

[33] J. Dennis Jr and H. Mei, "Two new unconstrained optimization algorithms which use function and gradient values," *Journal of Optimization Theory and Applications*, vol. 28, no. 4, pp. 453–482, 1979.

[34] C. M. Giuliani, "Distributed derivative free optimization," Ph.D. dissertation, Universidade Federal de Santa Catarina, 2016.

[35] https://github.com/lingyaw/FieldOpt/tree/TRO-new_polymodel/.

[36] https://msdn.microsoft.com/en-us/library/dd409416.aspx/.

[37] E. J. M. Baumann, "Fieldopt: Enhance software framework for petroleum field optimization," Master's thesis, Norweigian University of Sicenes and Technology, 2015.

[38] Y. Yuan, "On the truncated conjugate gradient method," *Mathematical Programming*, vol. 87, no. 3, pp. 561–573, 2000.