49th IEEE Conference on Decision and Control
December 15-17, 2010
Hilton Atlanta Hotel, Atlanta, GA, USA

Second-Order Adjoint-Based Control for Multiphase Flow in Subsurface Oil Reservoirs

Eka Suwartadi†, Stein Krogstad‡, Bjarne Foss†

Abstract— **This paper presents an efficient way to compute second-order gradients by using the adjoint method for PDE-constrained optimization. The gradient thus obtained will then be used in an optimization algorithm. We propose a conjugate gradient combined with the trust-region method, which may have a quadratic convergence rate of Newton's method. Furthermore, we compare the proposed algorithm to a quasi-Newton method (BFGS). We apply the method for production optimization of oil reservoirs. Two numerical cases are presented, showing that our proposed method requires fewer function and gradient evaluations.**

I. INTRODUCTION

The development of the Hessian matrix in the framework of adjoint methods can be traced to [1], where an auxiliary convex quadratic optimization was used to compute the Newton step. In that work, the Hessian matrix was not computed explicitly. In [2], the use of finite difference methods to compute the Hessian was proposed. Similarly, [3] presented a hierarchical optimization approach for oil reservoirs that used the central finite difference method to estimate the Hessian. In the optimization community, an analysis of the Newton method was presented by, among other works, [4]. In [5] an inexact Newton method was applied, that is, Truncated Newton or Newton Conjugate Gradient, to a PDE-constrained optimization.

In this paper, in the spirit of [4] and [5], we implement the adjoint method for the Hessian matrix. A Hessian-times-vector, as in [5], is derived for production optimization of an oil reservoir model. In addition, we use the conjugate gradient algorithm, following [6], to perform the optimization using the gradients (first and second order) from the adjoint method. We discuss the second-order adjoint algorithm in Section II. In Section III we explain our incompressible oil reservoir model along with the optimization goal. Two numerical cases demonstrating our proposed algorithm are presented in Section IV. In Section V we discuss the results of the numerical cases, and in Section VI we present our conclusions.

This work was supported by The Center for Integrated Operations at the Norwegian University of Science and Technology, Norway

E. Suwartadi and B. Foss are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway Eka.Suwartadi@itk.ntnu.no , Bjarne.Foss@itk.ntnu.no

S. Krogstad is with the Department of Applied Mathematics, SINTEF ICT, Blindern, 0314 Oslo, Norway Stein.Krogstad@sintef.no

II. THE ADJOINT GRADIENT FOR THE HESSIAN

In general, we consider an optimization problem,

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^{n_x},\mathbf{u}\in\mathbb{R}^{n_u}} \mathcal{J}(\mathbf{x},\mathbf{u}) \\ &\text{subject to} \quad c(\mathbf{x},\mathbf{u})=0, \quad (\mathbf{x},\mathbf{u})\in W_{\text{ad}}, \end{aligned} \tag{1}$$

where $\mathcal{J}:\mathbb{R}^{n_x}\times\mathbb{R}^{n_u}\to\mathbb{R}$ is the objective function, $c:\mathbb{R}^{n_x}\times\mathbb{R}^{n_u}\to\mathbb{R}^{n_c}$ is the implicit constraint representing a dynamic model, and $W_{ad}\subset W := \mathbb{R}^{n_x}\times\mathbb{R}^{n_u}$ is a nonempty closed set. $\mathcal{J}$ and c are continuously Fréchet-differentiable. We assume that for a given control input $\mathbf{u}$, there is always a unique solution of the state variable $\mathbf{x}$, i.e., $\mathbf{u}\in\mathbb{R}^{n_u}\mapsto\mathbf{x}\in\mathbb{R}^{n_x}$. Moreover, we also assume that $c_{\mathbf{x}}(\mathbf{x}(\mathbf{u}),\mathbf{u})$ is continuously invertible. Hence, the implicit function theorem guarantees that $\mathbf{x}(\mathbf{u})$ is continuously differentiable. Since $\mathbf{x}$ is dependent on $\mathbf{u}$, we obtain the reduced problem

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^{n_u}} \widehat{\mathcal{J}}(\mathbf{u}): \quad &= \quad \mathcal{J}(\mathbf{x}(\mathbf{u}),\mathbf{u}) \quad , \\ \text{subject to} \quad &\mathbf{u}\in\widehat{U}_{ad}:=\{\mathbf{u}\in\mathbb{R}^{n_u}:(\mathbf{x}(\mathbf{u}),\mathbf{u})\in W_{ad}\}. \end{aligned} \tag{2}$$

Furthermore, an equation for $\mathbf{x}_{\mathbf{u}}(\mathbf{u})$ is obtained by taking the derivative of $c(\mathbf{x},\mathbf{u})$:

$$c_{\mathbf{x}}(\mathbf{x},\mathbf{u})\,\mathbf{x}_{\mathbf{u}}(\mathbf{u})+c_{\mathbf{u}}(\mathbf{x},\mathbf{u})=0. \tag{3}$$

We will use this derivative for gradient computation which will be explained shortly.

A. The Adjoint Method

Following [5], we now derive the first-order gradient using the adjoint method. We construct a Lagrangian:

$$\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})=\mathcal{J}(\mathbf{x},\mathbf{u})+\boldsymbol{\lambda}^T c(\mathbf{x},\mathbf{u}). \tag{4}$$

By the optimality condition $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})=0$, we can solve the following linear equation

$$c_{\mathbf{x}}(\mathbf{x}(\mathbf{u}),\mathbf{u})^T\boldsymbol{\lambda}=-\nabla_{\mathbf{x}}\mathcal{J}(\mathbf{x}(\mathbf{u}),\mathbf{u}). \tag{5}$$

Denote the solution as the Lagrangian multiplier $\boldsymbol{\lambda}(\mathbf{u})$ and the first-order gradient as

$$\nabla_{\mathbf{u}}\mathcal{J}(\mathbf{x}(\mathbf{u}),\mathbf{u})=\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})|_{\mathbf{x}=\mathbf{x}(\mathbf{u}),\boldsymbol{\lambda}=\boldsymbol{\lambda}(\mathbf{u})}. \tag{6}$$

If one is interested in the use of a quasi-Newton method, for example, the BFGS algorithm, (6) will be supplied to the algorithm to numerically approximate the inverse of the Hessian matrix. Now, we continue to derive the adjoint for the Hessian matrix.

For brevity, we denote the Hessian $\nabla^2\mathcal{J}(\mathbf{u})$ referring to $\nabla_{\mathbf{uu}}\mathcal{J}(\mathbf{x}(\mathbf{u}),\mathbf{u})$, and let $\mathbf{x}$ and $\boldsymbol{\lambda}$ denote $\mathbf{x}(\mathbf{u})$ and $\boldsymbol{\lambda}(\mathbf{u})$,

respectively. From (6), we take the derivative with respect to $\mathbf{x}$, $\mathbf{u}$, $\boldsymbol{\lambda}$, yielding the Hessian, as follows:

$$\begin{aligned}\nabla^2\mathcal{J}(\mathbf{u}) &= \nabla_{\mathbf{ux}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})\,\mathbf{x_u}(\mathbf{u}) + \nabla_{\mathbf{uu}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) \\ &\quad +\nabla_{\mathbf{u\lambda}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})\,\boldsymbol{\lambda_u}(\mathbf{u}).\end{aligned} \tag{7}$$

Furthermore, if we differentiate $\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) = 0$ we obtain

$$\begin{aligned}&\nabla_{\mathbf{xx}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{x_u}\left(\mathbf{u}\right) \\ &\quad +\nabla_{\mathbf{xu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right) \\ &+\nabla_{\mathbf{x\lambda}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\boldsymbol{\lambda_u}\left(\mathbf{u}\right) = 0.\end{aligned} \tag{8}$$

Using the implicit function theorem, we take the derivative of the Lagrangian (4) and obtain

$$\nabla_{\mathbf{x\lambda}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) = c_{\mathbf{x}}(\mathbf{x},\mathbf{u})^T \tag{9}$$

$$\nabla_{\mathbf{u\lambda}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) = c_{\mathbf{u}}(\mathbf{x},\mathbf{u})^T. \tag{10}$$

Substitute (9) into (8) with the aim of computing the derivative of the Lagrangian multiplier $\lambda_{\mathbf{u}}(\mathbf{u})$ will lead to

$$\begin{aligned}\boldsymbol{\lambda_u}\left(\mathbf{u}\right) &= -c_{\mathbf{x}}\left(\mathbf{x},\mathbf{u}\right)^{-T} \\ &\quad \left(\nabla_{\mathbf{xx}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{x_u}\left(\mathbf{u}\right) + \nabla_{\mathbf{xu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\right).\end{aligned} \tag{11}$$

Equation (3) also implies

$$\mathbf{x_u}(\mathbf{u}) = -c_{\mathbf{x}}(\mathbf{x},\mathbf{u})^{-1}c_{\mathbf{u}}(\mathbf{x},\mathbf{u}). \tag{12}$$

Then, we substitute (10), (11), and (12) to the Hessian matrix in (7). After some rearrangement, we get

$$\begin{aligned}\nabla^2\mathcal{J}(\mathbf{u}) &= \nabla_{\mathbf{ux}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})\,\mathbf{x_u}(\mathbf{u}) + \nabla_{\mathbf{uu}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) \\ &\quad -\mathbf{x_u}(\mathbf{u})^T\,\nabla_{\mathbf{xx}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda})\,\mathbf{x_u}(\mathbf{u}) \\ &\quad -\mathbf{x_u}(\mathbf{u})^T\,\nabla_{\mathbf{xu}}\mathcal{L}(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}).\end{aligned} \tag{13}$$

If we define $W(\mathbf{x},\mathbf{u}) = \begin{pmatrix}\mathbf{x_u}(\mathbf{u}) \\ \mathbf{I}\end{pmatrix}$, then (13) will become

$$\begin{aligned}\nabla^2\mathcal{J}\left(\mathbf{u}\right) &= W\left(\mathbf{x},\mathbf{u}\right)^T \\ &\quad \begin{pmatrix}-\nabla_{\mathbf{xx}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right) & -\nabla_{\mathbf{xu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right) \\ \nabla_{\mathbf{ux}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right) & \nabla_{\mathbf{uu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\end{pmatrix} \\ &\quad W\left(\mathbf{x},\mathbf{u}\right).\end{aligned} \tag{14}$$

The identity (14) can be used to compute the Hessian matrix. However, in practice, the pure Newton's method is prohibitive in terms of computational cost. Therefore, we instead use a Hessian-times-vector product. This Hessian-times-vector product will be used in a conjugate gradient-based method, which we will describe in the next section. Let us summarize the procedure in the following steps.

1) Given the control input $\mathbf{u}$, solve the system model $c\left(\mathbf{x},\mathbf{u}\right) = 0$. Denote the solution by the state variable $\mathbf{x}$.
2) Solve the adjoint equation $c_{\mathbf{x}}(\mathbf{x}(\mathbf{u}),\mathbf{u})^T\boldsymbol{\lambda} = -\nabla_{\mathbf{x}}\mathcal{J}(\mathbf{x}(\mathbf{u}),\mathbf{u})$. The solutions are the Lagrangian multipliers $\boldsymbol{\lambda}$.
3) Compute the derivative of the state variable in the direction of $\mathbf{s}_1$

$$\mathbf{x_u}\left(\mathbf{u}\right)\mathbf{s}_1 = -c_{\mathbf{x}}\left(\mathbf{x},\mathbf{u}\right)^{-1}c_{\mathbf{u}}\left(\mathbf{x},\mathbf{u}\right)\mathbf{s}_1$$

4) Also compute the derivative of the Lagrangian multiplier in the direction of $\mathbf{s}_1$

$$\begin{aligned}\boldsymbol{\lambda_u}\left(\mathbf{u}\right)\mathbf{s}_1 &= -\mathbf{c_x}\left(\mathbf{x},\mathbf{u}\right)^{-T} \\ &\quad \left(\nabla_{\mathbf{xx}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{x_u}\left(\mathbf{u}\right) + \nabla_{\mathbf{xu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{s}_1\right)\end{aligned}$$

5) Compute the Hessian-times-vector product

$$\begin{aligned}\nabla^2\mathcal{J}\left(\mathbf{u}\right)\mathbf{s}_1 &= \nabla_{\mathbf{ux}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{x_u}\left(\mathbf{u}\right)\mathbf{s}_1 \\ &+ \nabla_{\mathbf{uu}}\mathcal{L}\left(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}\right)\mathbf{s}_1 \\ &+ c_{\mathbf{u}}\left(\mathbf{x},\mathbf{u}\right)^T\boldsymbol{\lambda_u}\left(\mathbf{u}\right)\mathbf{s}_1\end{aligned}$$

Note that steps 3 and 4 are linear equations that supplement the adjoint equation, which is also a linear equation.

B. Truncated Newton Method

Newton's method, $\nabla^2\mathcal{J}(\mathbf{u}_k)\,\mathbf{s}_k = -\nabla\mathcal{J}(\mathbf{u}_k)$, is approximately solved using the conjugate gradient (CG) method. The method consists of two layers of iterations, i.e., an inner and an outer iteration. The inner iteration finds a Newton step $\mathbf{s}_k$ at iteration k, while the outer iteration is a trust-region globalization strategy. Algorithm 1 below describes the outer iteration. The algorithm setting here is similar to [7].

Algorithm 1 Trust-region Method (Outer iterations)

1. **Initialization** :
   - Set the initial control input $\mathbf{u}_0$ and initial trust region radius $\triangle_0$.
   - Set constants η_1,η_2, γ_1, γ_2, which satisfy $0 < \eta_1 \leq \eta_2 < 1$ and $0 < \gamma_1 \leq \gamma_2 < 1$.
   - Set the gradient convergence tolerance $\omega_* \ll 1$
2. For $k = 0, 1, \ldots$
   - If $\|\nabla_{\mathbf{u}_k}\mathcal{J}\left(\mathbf{u}_k\right)\| \leq \omega_*$, then **stop**.
   - Compute a step $\mathbf{s}_k$ by solving the trust region sub-problem :
   $$\min_{\mathbf{s}} \quad q_k\left(\mathbf{s}\right) \qquad \text{s.t} : \|\mathbf{s}\| \leq \triangle_k,$$
   where $q_k\left(\mathbf{s}\right) = \nabla\mathcal{J}\left(\mathbf{u}_k\right)^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T\nabla^2\mathcal{J}\left(\mathbf{u}_k\right)\mathbf{s}$, a quadratic approximation of the objective function $\mathcal{J}\left(\mathbf{u}_k\right)$.
   - Compute a ratio ρ_k :
   $$\rho_k = \frac{\mathcal{J}\left(\mathbf{u}_k + \mathbf{s}_k\right) - \mathcal{J}\left(\mathbf{u}_k\right)}{q_k\left(\mathbf{s}_k\right)}.$$
   - Update $\mathbf{u}_k$ based on ρ_k value:
   $$\mathbf{u}_{k+1} = \begin{cases}\mathbf{u}_k + \mathbf{s}_k & \text{if } \rho_k \geq \eta_1 \\ \mathbf{u}_k & \text{if } \rho_k < \eta_1\end{cases}.$$
   - Update the trust region radius:
   $$\triangle_{k+1} \in \begin{cases}[\triangle_k, \infty) & \text{if } \rho_k \geq \eta_2, \\ [\gamma_2\triangle_k, \triangle_k] & \text{if } \rho_k \in [\eta_1, \eta_2), \\ [\gamma_1\triangle_k, \gamma_2\triangle_k] & \text{if } \rho_k < \eta_1.\end{cases}$$

The trust region sub-problem in Algorithm 1 is solved using Steihaug's Conjugate Gradient method [6]. We choose this method because the Hessian matrix may be negative definite. Hence, if we use line search methods, we will not succeed in finding a descent direction (minimization problem). The method is described in Algorithm 2. The algorithm is terminated if one of the following criteria is met: residual tolerance, crossing the boundary of the trust region, or negative curvature. Furthermore, since the

Conjugate Gradient algorithm might have lengthy iterations, the iteration is "truncated" in the early stages of the inner iteration. This algorithm is also known as the Truncated Newton or Inexact Newton method. The truncation is done according to the residual, which is described by the following inequality

$$\mathbf{r}_i = \left\|\nabla\mathcal{J}(\mathbf{u}_k) + \nabla^2\mathcal{J}(\mathbf{u}_k)\,\underline{\mathbf{s}}_i\right\| \leq \eta_k \left\|\nabla\mathcal{J}(\mathbf{u}_k)\right\|. \quad (15)$$

The subscript i represents the iteration within the Conjugate Gradient algorithm. The $\underline{\mathbf{s}}$ is the approximate solution of the Newton step.

Algorithm 2 Steihaug Conjugate Gradient (Inner iterations)

1. Set $\eta_k < 1$, $\triangle_k > 0$, $\underline{\mathbf{s}}_0 = \mathbf{0}$, $\mathbf{r}_0 = -\nabla\mathcal{J}(\mathbf{u}_k)$, and $\mathbf{d}_0 = \mathbf{r}_0$.
2. For $i = 0, 1, \ldots$
   - If $\|\mathbf{r}_i\| \leq \eta_k \|\nabla\mathcal{J}(\mathbf{u}_k)\|$, then $\mathbf{s}_k = \underline{\mathbf{s}}_i$ and **stop**.
   - $\mathbf{H}\mathbf{v}_i = \nabla^2\mathcal{J}(\mathbf{u}_k)\,\mathbf{d}_i$
   - If $\mathbf{d}_i^T\mathbf{H}\mathbf{v}_i \leq 0$, compute τ such that $\|\underline{\mathbf{s}}_i + \tau\mathbf{d}_i\| = \triangle_k$, then $\mathbf{s}_k = \underline{\mathbf{s}}_i + \tau\mathbf{d}_i$ and **stop**.
   - $\vartheta_i = \|\mathbf{r}_i\|^2 / \mathbf{d}_i^T\mathbf{H}\mathbf{v}_i$
   - $\underline{\mathbf{s}}_{i+1} = \underline{\mathbf{s}}_i + \vartheta_i\mathbf{d}_i$
   - If $\left\|\underline{\mathbf{s}}_{i+1}\right\| \geq \triangle_k$, compute τ such that $\|\underline{\mathbf{s}}_i + \tau\mathbf{d}_i\| = \triangle_k$, then $\mathbf{s}_k = \underline{\mathbf{s}}_i + \tau\mathbf{d}_i$ and **stop**.
   - $\mathbf{r}_{i+1} = \mathbf{r}_i - \vartheta_i\mathbf{H}\mathbf{v}_i$
   - $\phi_i = \|\mathbf{r}_{i+1}\|^2 / \|\mathbf{r}_i\|^2$
   - $\mathbf{d}_{i+1} = \mathbf{r}_{i+1} + \phi_i\mathbf{d}_i$.

The Hessian-times-vector ($\mathbf{H}\mathbf{v}_i$) in Algorithm 2 is obtained from the adjoint method explained in the previous section. The theorem below further explains the relationship between the convergence rate and the residual value in (15).

Theorem 1. *Assume that $\nabla\mathcal{J}(\mathbf{u})$ is continuously differentiable in a neighborhood of a local solution $\mathbf{u}^*$ of* (1)*. In addition, assume that $\nabla^2\mathcal{J}(\mathbf{u}^*)$ is nonsingular and that it is Lipschitz continuous at $\mathbf{u}^*$. Assume that iteration k of the truncated-Newton method computes a step $\mathbf{s}_k$ that satisfies*

$$\left\|\nabla\mathcal{J}(\mathbf{u}_k) + \nabla^2\mathcal{J}(\mathbf{u}_k)\,\mathbf{s}_k\right\| \leq \eta_k \left\|\nabla\mathcal{J}(\mathbf{u}_k)\right\|$$

for a specified value of η_k ; the new estimate of the solution is computed using $\mathbf{u}_k + \mathbf{s}_k \rightarrow \mathbf{u}_{k+1}$. If $\mathbf{u}_0$ is sufficiently close to $\mathbf{u}^*$ and $0 \leq \eta_k \leq \eta_{max} < 1$, then $\{\mathbf{u}_k\}$ converges to $\mathbf{u}^*$ q-linearly in the norm $\|\cdot\|_*$, defined by $\|v\|_* \equiv \left\|\nabla^2\mathcal{J}(\mathbf{u}^*)v\right\|$, with asymptotic rate constant no greater than η_{max}. If $lim_{k\rightarrow\infty}\eta_k = 0$, then the convergence is q-superlinear. If $\eta_k = O\left(\left\|\nabla\mathcal{J}(\mathbf{u}_k)\right\|^r\right)$ for $0 < r \leq 1$, then the convergence is of order at least $(1 + r)$.

Proof: see [8]. □

The η_k is called the *forcing sequence*, which, as suggested in [8], has practical value $\eta_k = \min\left\{\frac{1}{2}, \kappa\left\|\nabla\mathcal{J}(u_k)\right\|^a\right\}$, where κ is a positive constant and $0 < a \leq 1$.

The algorithms above have been described without constraints handling. Later, we will present case examples with constraints. To handle the constraints, we use an active set Sequential Linear-Quadratic Programming (SLQP) method, as implemented in the software package KNITRO [9]. This method is based on the projected Conjugate Gradient (PCG). In short explanation, the constraints are represented by equality constraints g_E and inequality constraints g_I. The objective function will be in the form

$$P(\mathbf{u};\nu) = \widehat{\mathcal{J}}(\mathbf{u}) + \nu\sum_{i\in\mathcal{E}}|g_i(\mathbf{u})| + \nu\sum_{i\in\mathcal{I}}\left(\max\left(0, -g_i(\mathbf{u})\right)\right). \quad (16)$$

Here, ν is the penalty parameter and $i \in \mathcal{E}$ and $i \in \mathcal{I}$ represent the vectors g_E and g_I, respectively. The Hessian-times-vector derived in the previous section is supplied to the optimizer. Refer to [9] for further details of constraints handling in the SLQP method.

III. PRODUCTION OPTIMIZATION OF OIL RESERVOIRS

The problem that we are interested in for demonstrating the algorithms is that of oil reservoirs. An oil reservoir model is posed as the implicit constraint $c(\mathbf{x}, \mathbf{u})$ in (1). An economic objective function is selected to represent the objective function $\mathcal{J}(\mathbf{x}, \mathbf{u})$. Furthermore, our problem here is an open-loop optimal control setting, with the goal being to find the best possible solution for production strategy.

A. Oil Reservoir Model

We focus on the optimal production case for two-phase (oil and water) reservoirs. Furthermore, we assume immiscible and incompressible fluids and rocks, no gravity effects or capillary pressure, no-flow boundaries, and finally isothermal conditions. Let Ω be a porous media domain with boundary $\partial\Omega$. The corresponding state equations are referred to as the *pressure equation* and the *saturation equation*. The pressure equation is given by

$$\vec{v} = -\mathbf{K}\lambda_t(s)\nabla p, \quad \nabla\cdot\vec{v} = q \quad \text{in} \quad \Omega, \quad (17)$$

where $\vec{v}$ is the Darcy velocity, $\mathbf{K}$ is the permeability tensor, and q is the volumetric source/sink term. Finally, λ_t is the total mobility, which in this setting is the sum of the water and oil mobility functions,

$$\lambda_t(s) = \lambda_w(s) + \lambda_o(s) = k_{rw}(s)/\mu_w + k_{ro}(s)/\mu_o. \quad (18)$$

Here, k_{rw}, k_{ro} and μ_w, μ_o are relative permeabilities and viscosities for water and oil, respectively. Assuming no-flow boundaries mean that the normal component of the Darcy velocity across boundaries is zero. The saturation equation is given by

$$\phi\frac{\partial s}{\partial t} + \nabla\cdot\left(f_w(s)\vec{v}\right) = q_w \quad \text{in} \quad \Omega, \quad (19)$$

where ϕ is the porosity and q_w is the volumetric water source. Finally, f_w is the water fractional flow function $f_w(s) = \lambda_w(s)/\lambda_t(s)$. The nonlinear behavior of the above equations is mainly dictated by the shape of the relative permeability functions, which in this paper are taken to be quadratic.

The reservoir simulation model is typically given as a (non-matching) hexahedral grid, where each grid block is assigned reservoir properties such as permeability and porosity. For ease of exposition, we assume here that the pressure

equation (17) is discretized using a *two-point flux approximation* (TPFA) (see, e.g., [11]), although the MATLAB implementation employed in this work [12] is more general in the sense that a wider range of discretization schemes can be used. For a given saturation field, the TPFA assumes that the Darcy flux from one grid block i to its neighbor j is proportional to the pressure drop between the two blocks, i.e.,

$$v_{ij} = t_{ij}(s_i, s_j)(p_i - p_j). \tag{20}$$

In the above equation, t_{ij} is referred to as the transmissibility, which in this formulation is taken to be dependent on the saturation in the two blocks (or rather, $\lambda(s_i)$ and $\lambda(s_j)$). Wells are implemented using the Peaceman well model ([13])

$$q_i^w = -WI_i^w(s_i)\left(p^w - p_i\right). \tag{21}$$

Here, q_i^w is the flow rate from well w into grid block i, and p^w is the wellbore pressure (assumed to be constant since we neglect gravity and wellbore flow effects). Finally, $WI_i^w(s_i)$ is the Peaceman well-index as a function of the grid block saturation s_i. The implementation uses a sequential splitting, that is, the pressure field at time-step n is calculated based on the saturation at time step $n-1$, and the saturation at time step n is calculated based on the pressure field at time step n. Let $\mathbf{p}^n$ denote the vector containing the grid block pressures and unknown wellbore pressures at time-step n. Similarly, let $\mathbf{s}^{n-1}$ denote the grid block saturations at time step $n-1$. Enforcing volume balance, i.e., setting the sum of all outfluxes (expressed as (20) and (21)) from each block equal to the source, leads to a positive definite matrix $\mathcal{A}(\mathbf{s}^{n-1})$ in a linear system equation

$$\mathcal{A}\left(\mathbf{s}^{n-1}\right)\mathbf{p}^n = \mathcal{B}\mathbf{u}^n. \tag{22}$$

Here, we have taken the right-hand-side as a function of a control input vector $\mathbf{u}^n$ for time step n, which can either be well rates or well pressures (bottom hole pressure/BHP). We note that the right-hand-side is linear in $\mathbf{u}^n$ and refer to (22) as the discretized pressure equation.

We discretize the saturation equation (19) using a standard upstream weighted implicit finite volume method to form

$$\mathbf{s}^n = \mathbf{s}^{n-1} + \triangle t^n\, \mathbf{D}_{PV}^{-1}\left(\mathbf{A}\left(\mathbf{v}^n\right) f_w\left(\mathbf{s}^n\right) + \mathbf{q}\left(\mathbf{v}^n\right)_+\right). \tag{23}$$

Here, $\triangle t^n$ is the time step and $\mathbf{D}_{PV}$ is the diagonal matrix containing the grid block pore volumes. The matrix $\mathbf{A}\left(\mathbf{v}^n\right)$ is the sparse flux matrix based on the upstream weighted discretization scheme, and $\mathbf{q}(\mathbf{v}^n)_+$ is the vector of positive sources (in this setting, water injection rates). We note that the matrix $\mathbf{A}$ and vector $\mathbf{q}$ are linear functions of $\mathbf{v}^n$, while $\mathbf{v}^n = \mathbf{T}(\mathbf{s}^{n-1})\mathbf{p}^n$, where $\mathbf{T}(\mathbf{s}^{n-1})$ is a matrix containing the transmissibilities and well indices based on $\mathbf{s}^{n-1}$. We refer to (23) as the discretized saturation equation.

The discrete state equations (22) and (23) can be written in an implicit form $F(\mathbf{x}, \mathbf{u}) = 0$ as

$$\begin{aligned}
\mathbf{F}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{u}}) &= \begin{pmatrix} \mathbf{F}^1\left(\mathbf{p}^1, \mathbf{s}^0, \mathbf{s}^1, \mathbf{u}^1\right) \\ \vdots \\ \mathbf{F}^N\left(\mathbf{p}^N, \mathbf{s}^{N-1}, \mathbf{s}^N, \mathbf{u}^N\right), \end{pmatrix} \\
\mathbf{x}^{nT} &= (\mathbf{p}^{nT}, \mathbf{s}^{nT}), \quad n = 1, ..., N, \\
\widetilde{\mathbf{x}}^T &= (\mathbf{x}^{1T}, ..., \mathbf{x}^{NT}), \\
\widetilde{\mathbf{u}}^T &= (\mathbf{u}^{1T}, ..., \mathbf{u}^{NT}).
\end{aligned} \tag{24}$$

The state vectors and control input vectors are stacked for all time instances from $n = 1, \ldots, N$.

In this study, we use Net Present Value (NPV) as the objective function, having the following formula

$$\begin{aligned}
\mathcal{J}(\tilde{\mathbf{u}}) &= \sum_{n=0}^{N-1}\left[\sum_{j=1}^{\mathcal{N}_{prod}}\left(\frac{r_o q_{o,j}^n - r_w q_{w,j}^n}{(1+d)^n}\right) - \sum_{l=1}^{\mathcal{N}_{inj}} r_{inj} q_l^n\right]\triangle t \\
&= \sum_{n=0}^{N-1}\mathcal{J}^n,
\end{aligned} \tag{25}$$

where r_o, r_w, r_{inj} are the represented oil price, water separation cost, and water injection cost, respectively. The well rate at the injector wells is denoted by q_l, the water rate at the producer wells is q_w, the oil rate at the producers is q_o, d is the discount factor, and $\triangle t$ is the time interval. In addition, $\mathcal{N}_{prod}, \mathcal{N}_{inj}$ denote the number of producer and injector wells, respectively.

B. Adjoint Implementation

Due to page limitations, we omit the detailed derivation of the procedure explained in Section IIA. The first order adjoint-gradient derivation in this work is the same as that presented in [14]. The Hessian-times-vector derivations are analogous to the first order gradient where one forward and one backward simulations are performed to compute the linearized state variables and the derivative of Lagrangian multipliers, respectively. Hence, a lot of code from the first order gradient computation can be reused for the Hessian-times-vector computations.

To check the adjoint implementation, we compare the obtained gradients with those of finite difference method. Furthermore, the Hessian must be a symmetric matrix since it has the *self-adjoint* property.

IV. NUMERICAL CASES

In this section, we use a five-spot 2-dimensional oil reservoir consisting of oil and water. The reservoir is discretized into 60×60 grid blocks and it originates from layer 65 of the SPE 10th comparative study [15]. The grid block dimension is $10\,\mathrm{ft} \times 10\,\mathrm{ft} \times 2\,\mathrm{ft}$. There are four producer wells at the corners and one injector well in the middle. The porosity is set homogenously to 0.3 in the reservoir, but the permeability is heterogenous, as depicted in Fig.1. The mobility ratio between oil and water is one. The initial water saturation is 0.2, and pressure in the reservoir is 400 bar. The oil price is set to $126\ \$/\mathrm{m}^3$, the water separation cost to 19

$\$/\mathrm{m}^3$, and the water injection cost to 6 $\$/\mathrm{m}^3$. We set up two examples by distinguishing the control input. Furthermore, we parameterize the control input into five control intervals, where a control interval is equal to 30 days.

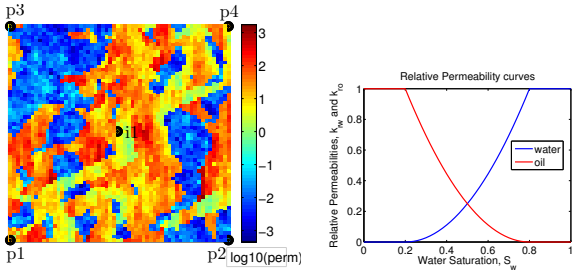

Fig. 1. Five-spot well pattern with a heterogenous permeability field (with units in milidarcy), 2-dimensional 60×60 grid block. $i1$ is the injector well in the middle, and $p1$, $p2$, $p3$, $p4$ are the producer wells at the corners.

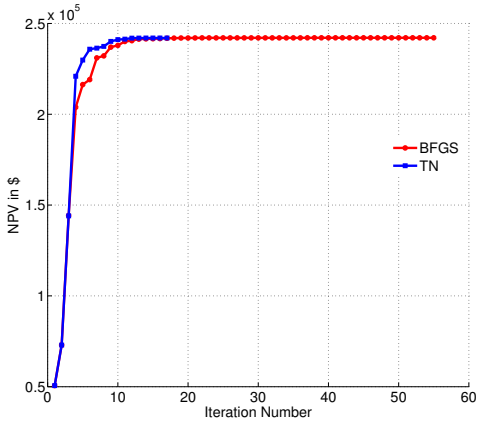

Fig. 2. Comparison of the objective function (NPV) evaluation between BFGS and Truncated Newton (TN) for the first case. The control inputs are well rates at the injector and producer wells.

A. Case 1

In this case, we use well rates as our control input, namely $\mathbf{u}^n = \begin{bmatrix} q^n_{i1} & q^n_{p1} & q^n_{p2} & q^n_{p3} & q^n_{p4} \end{bmatrix}^T$. Hence, we have five control intervals, which means that this case has 25 decision variables. Since we assume incompressible flow, the total injection rate is equal to the total producer rates, that is, $q^n_{i1} = \sum_{j=1}^{4} q^n_{p\,j}$. Moreover, the well rate is bounded to be greater than 0 $\mathrm{m}^3/\mathrm{day}$. The initial injection rate is 10 $\mathrm{m}^3/\mathrm{day}$.

B. Case 2

Now, we use BHP in the producer wells as the control input and set a fixed BHP in the injector well. The control input is $\mathbf{u}^n = \begin{bmatrix} p^n_{w,p1} & p^n_{w,p2} & p^n_{w,p3} & p^n_{w,p4} \end{bmatrix}^T$. In total, the number of control inputs is 20. We fix the BHP at the injector well to 411 bar, and the initial BHP at each producer well is 408 bar. Furthermore, we constrain BHP below 410 bar.

V. RESULTS AND DISCUSSION

We compare our proposed method, the Truncated Newton (TN), to a quasi-Newton method, that is, BFGS. The BFGS method is also used in a conjugate gradient algorithm. Table I summarizes the performance of the methods using the given initial control input. We run each case for 100 different initial control inputs, yielding the statistics presented in Tables II and III. The simulations were performed on a Linux 64-bit machine with an Intel Xeon(R) 3.00 GHz and 16 GB of RAM. The constraints appear as equality and bound constraints in the first case, and only as bound constraints in the second case. Moreover, we set the number of inner conjugate gradient iterations to 5.

A. Case 1

The stopping criteria for the optimizer are relative gradient and step size, which are set to 10^{-3} and 10^{-15}, respectively. Both methods terminate due to the gradient tolerance and lead to the same value of NPV. The TN method excels in the number of iterations and the number of function evaluations.

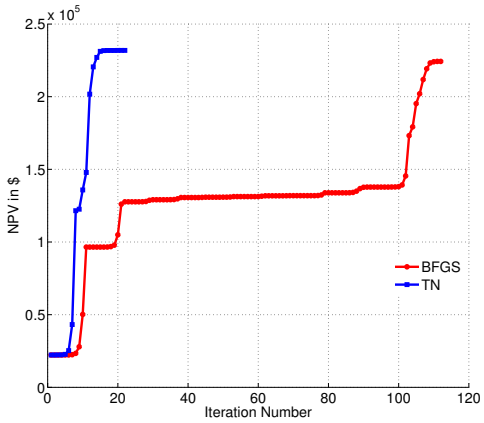

Fig. 3. Comparison of the objective function (NPV) evaluation between BFGS and TN for the second case. The control inputs are BHP at the producer wells.

B. Case 2

We use the same stopping criteria as in Case 1. In this case, both methods stop due to the gradient criterion. The TN method yields higher NPV than the BFGS method. However, the TN method is more CPU-intensive, as can be seen in Table I. This can be explained from the Hessian-times-vector procedure in Section II, particularly steps 3 and 4, which require the solving of linear equations. To solve the linear equations, we use the built-in solver in MATLAB which is a direct sparse method (see [16]).

VI. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

As shown, the TN method requires fewer function and gradient evaluations. Furthermore, in Case 2, the TN method results in better objective function values than the BFGS method. However, the TN method requires more CPU time. This is due to the fact that the computation of the Hessian-times-vector product requires more linear equations to be

TABLE I

PERFORMANCE COMPARISON OF BFGS AND TN IN TERMS OF: NUMBER OF ITERATIONS; NUMBER OF FUNCTION, GRADIENT, AND HESSIAN-VECTOR EVALUATIONS; CPU TIME; AND OBJECTIVE FUNCTION VALUES

Method	Case 1		Case 2	
	BFGS	TN	BFGS	TN
# of iteration	54	16	111	21
# of function evals	112	34	112	25
# of grad evals	55	17	112	22
# of Hessian-vec evals	-	250	-	275
CPU time (in sec)	170	370	280	415
NPV (in 10^5)	2.42	2.42	2.24	2.32

TABLE II

STATISTICAL PERFORMANCE COMPARISON OF BFGS AND TN FOR CASE 1 FROM 100 DIFFERENT INITIAL CONTROL INPUTS.

Method	Case 1					
	BFGS			TN		
	min	max	average	min	max	average
# iteration	35	74	46	14	50	23
# function evals.	74	145	97	30	100	50
# grad. evals.	36	75	47	15	50	24
# Hessian-vec. evals.	-	-	-	218	669	365
CPU Time (in sec)	114	228	150	323	990	546
NPV (in $\$10^5$)	2.42	2.42	2.42	2.42	2.42	2.42

solved. The Hessian-times-vector procedure requires 4 simulations: 2 forward simulations and 2 backward simulations. The forward simulations are needed to compute the state variables and the linearized state variables. The backward simulations are used to obtain the Lagrangian multiplier and its derivative.

B. Future Work

To reduce the CPU time of the Hessian-times-vector algorithm, it would be beneficial to apply model order reduction techniques. Using different kinds of preconditioner matrices for the Hessian might also reduce the CPU time. Furthermore, the Lanczos-CG method may lead to better results than the Steihaug-CG. We will proceed along these directions for our next study.

TABLE III

STATISTICAL PERFORMANCE COMPARISON OF BFGS AND TN FOR CASE 2 FROM 100 DIFFERENT INITIAL CONTROL INPUTS.

Method	Case 2					
	BFGS			TN		
	min	max	average	min	max	average
# iteration	87	133	113	16	43	25
# function evals.	107	144	119	22	81	42
# grad. evals.	88	134	114	17	44	26
# Hessian-vec. evals.	-	-	-	214	894	474
CPU Time (in sec)	214	301	331	320	1412	686
NPV (in $\$10^5$)	1.23	2.28	2.16	2.28	2.32	2.31

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the Integrated Operations Center at NTNU.

REFERENCES

[1] R.L. Raffard and C.J. Tomlin, "Second Order Adjoint-based Optimization of Ordinary and Partial Differential Equations with Application to Air Traffic Flow", *Proceedings of 2005 American Control Conference*, 2005, pp. 798-803.

[2] R.L. Raffard, K. Amonlirdviman, J.D. Axelrod, and C.J. Tomlin, "An Adjoint-Based Parameter Identification Algorithm Applied to Planar Cell Polarity Signaling", *Transaction of Automatic Control, Special Issue on System Biology*, 2008, pp. 109-121.

[3] G.M. van Essen, P.M.J. Van den Hof, and J.D. Jansen, "Hierarchical Long-Term and Short-Term Production Optimization", Paper SPE 124332 presented at the 2009 SPE ATCE , New Orleans, Louisiana, USA, 4-7 October 2009.

[4] K. Ito, and K. Kunisch, "Newton's Method for A Class of Weakly Singular Optimal Control Problems", *SIAM Journals Optimization*, 2000, Vol 10, No.3, pp: 896-916.

[5] M. Heinkenschloss, "Numerical Solution of Implicitly Constrained Optimization Problems", Technical Report Dept. of Computational and App. Math., Rice University, 2008.

[6] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization", *SIAM J.Numer.Anal*, 1983, 20:pp. 626-637.

[7] C.J. Lin, R.C. Weng, and S.S. Keerthi, "Trust Region Newton Method for Large-Scale Logistic Regression", *Journal of Machine Learning Research*, 2008, 9:pp. 627-650.

[8] R.S. Dembo, S.C. Eisenstat, and T. Steihaug, "Inexact Newton Methods", *SIAM J.Numer.Anal*, 1982, 19:pp. 400-408.

[9] R.H. Byrd, J. Nocedal, and R.A. Waltz, "KNITRO: An Integrated Package for Nonlinear Optimization", *Large-Scale Nonlinear Optimization*, Springer US, editor. G. Di Pillo and M. Roma, 2006.

[10] P.M.J. Van den Hof, J.D. Jansen, G. van Essen, and O.H. Bosgra, "Model-Based Control and Optimization of Large Scale Physical Systems - Challenges in Reservoir Engineering", *Proceedings of Chinese Control and Decision Conference*, 2009.

[11] K. Aziz, and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publisher, 1979.

[12] MATLAB Reservoir Simulation Toolbox, url: `http://sintef.org/Projectweb/MRST/`.

[13] D. Peaceman, "Interpretation of Well-Block Pressures in Numerical Reservoir Simulation with Nonsquare Grid Blocks and Anisotropic Permeability", *SPE Journal*, 1983, 23(3):531–543.

[14] S. Krogstad, V. L. Hauge, and A. F. Gulbransen, "Adjoint multiscale mixed finite elements", *Proceedings of SPE Reservoir Simulation Symposium*, 2009, The Woodlands, Texas, USA, 2-4 February. `http://dx.doi.org/10.2118/119112-MS`

[15] M.A. Christie, and M.J. Blunt "Tenth SPE Comparative Solution Project: A Comparative of Upscaling Technique", *SPE Reservoir Eval.Eng.*, 2001, vol 4, pp: 308-317.

[16] T.A. Davis, *Direct Methods for Sparse Linear Systems*, SIAM Publisher, 2006.