
SPE 141677

# Use of Approximate Dynamic Programming for Production Optimization

Zheng Wen, Louis J. Durlofsky, SPE, Benjamin Van Roy, Khalid Aziz, SPE, Stanford University



This paper was prepared for presentation at the 2011 SPE Reservoir Simulation Symposium held in Woodlands, Texas, U.S.A., 21-23 February 2011.



## Abstract

In production optimization, we seek to determine the well settings (bottomhole pressures, flow rates) that maximize an objective function such as net present value. In this paper we introduce and apply a new approximate dynamic programming (ADP) algorithm for this optimization problem. ADP aims to approximate the global optimum using limited computational resources via a systematic set of procedures that approximate exact dynamic programming algorithms. The method is able to satisfy general constraints such as maximum watercut and maximum liquid production rate in addition to bound constraints. ADP has been used in many application areas, but it does not appear to have been implemented previously for production optimization. The ADP algorithm is applied to two-dimensional problems involving primary production and water injection. We demonstrate that the algorithm is able to provide clear improvement in the objective function compared to baseline strategies. It is also observed that, in cases where the global optimum is known (or surmised), ADP provides a result within 1-2% of the global optimum. Thus the ADP procedure may be appropriate for practical production optimization problems.

## Introduction

Many optimization algorithms have been applied to maximize reservoir performance. Most of these optimization algorithms can be classified into two categories: gradient-based/direct-search [20, 15] and global stochastic search [13, 1, 7]. Both classes of optimization algorithms face limitations: gradient-based and direct-search algorithms settle for local optima, while global stochastic search algorithms such as genetic algorithms typically require many function evaluations for convergence, and even then there is no assurance that the global optimum has been found.

In principle, one can formulate production optimization as a nonlinear optimal control problem and find a global optimum using dynamic programming (DP) [2]. The key idea in DP is to decompose the optimization problem into a sequence of sub-problems, each representing optimization of a control action at one point in time. These sub-problems are related through the value function, which maps the system state to the net present value of future revenues. Once the value function is computed, the optimal control action at each time can be found by solving a sub-problem at that time. In general, these sub-problems are much simpler than the original optimization problem; in many cases, a global optimum of each sub-problem can be efficiently computed.

For problems of practical scale, the computational requirements of DP become prohibitive due to the curse of dimensionality. In particular, time and memory requirements typically grow exponentially with the number of state variables. Approximate dynamic programming (ADP) aims to address this computational burden by efficiently approximating the value function (see [3, 25, 19] for more on ADP). The result is an approximate value function, which is typically represented by a linear combination of a set of predefined basis functions. ADP algorithms provide methods for computing the coefficients associated with these basis functions. ADP has been successfully applied across a broad range of domains such as asset pricing [23, 18, 24], transportation logistics [21], revenue management [26, 10, 27], portfolio management [14, 12], and even to games such as backgammon [22].

In this paper, we introduce a new ADP algorithm for petroleum reservoir production optimization. We apply our algorithm to single-

phase and multiphase problems with simple or general well and production constraints and compare performance to that achieved using various baseline strategies. As we will show, ADP performs well relative to the baseline results for the problems considered.

Performance aside, it is worth noting that for reservoir production problems, our ADP algorithm offers some advantages relative to alternative optimization techniques. For example, ADP can readily handle general (nonlinear) constraints, such as maximum watercut, which are not straightforward to handle with some algorithms. In addition, although our proposed ADP algorithm does include non-deterministic components (specifically constraint sampling, as described below), the ADP algorithm is otherwise non-stochastic, yet it searches for the global optimum. This distinguishes it from other global search algorithms that have been applied to production optimization.

This paper is organized as follows. First, we formulate a dynamic optimization model for the reservoir production problem. Second, we describe our ADP algorithm for reservoir production, focusing on how we construct and select basis functions and how we compute the coefficients of these basis functions. Third, we discuss the computational demands of the ADP algorithm in terms of the number of simulations required. Fourth, we present simulation results and compare the performance of the ADP algorithm to various baselines. We conclude with a summary.

## Dynamic Optimization Model

We represent the discrete system of reservoir flow equations as a system of ordinary differential equations:

$$d\mathbf{x}(t)/dt = F\left(\mathbf{x}(t), \mathbf{u}(t)\right), \tag{1}$$

subject to an initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and instantaneous constraints $S(\mathbf{x}(t), \mathbf{u}(t)) \leq 0$. Here $\mathbf{x}(t)$ and $\mathbf{u}(t)$ denote the reservoir states and the control action at time $t$, respectively. Typically, $\mathbf{x}$ includes the pressure and saturation of each grid block and $\mathbf{u}$ encodes BHPs and/or flow rates of the injection/production wells (or well groups). The constraints $S(\mathbf{x}(t), \mathbf{u}(t)) \leq 0$ restrict the state and control action at time $t$. Constraints in reservoir production problems typically include upper/lower bounds on BHPs, upper/lower bounds on flow rates of components (oil/water/gas), and maximum watercut or gas cut of production wells or well groups.

We also assume that at time $t$, a payoff accumulates at a rate given by an instantaneous payoff function $L(\mathbf{x}(t), \mathbf{u}(t))$ that depends on the state $\mathbf{x}(t)$ and control action $\mathbf{u}(t)$. A widely used instantaneous payoff function is

$$L(\mathbf{x}(t), \mathbf{u}(t)) = \text{revenue from producing oil} - (\text{cost for producing water} + \text{cost for injecting water}). \tag{2}$$

Our objective is to maximize the net present value (NPV) of future payoffs $\int_0^T e^{-\alpha t} L\left(\mathbf{x}(t), \mathbf{u}(t)\right) dt$, where $\alpha \geq 0$ is the continuous-time discount rate.

In most reservoir production problems, the termination time $T$ is large enough so that the difference between the NPV of the cumulative profit, $\int_0^T e^{-\alpha t} L\left(\mathbf{x}(t), \mathbf{u}(t)\right) dt$, and its infinite horizon approximation, $\int_0^\infty e^{-\alpha t} L\left(\mathbf{x}(t), \mathbf{u}(t)\right) dt$, is sufficiently small. Moreover, it is well known that there exists a stationary policy $\mathbf{u}^*(t) = \mu^*\left(\mathbf{x}(t)\right)$ maximizing the infinite horizon discounted objective. As we will see later, the existence of a stationary globally optimal policy $\mu^*$ simplifies our ADP algorithms. For this reason, we will focus on solving the infinite horizon dynamic optimization problem:

$$\begin{aligned} \max_{\{u(t), t \geq 0\}} \quad & \int_{t=0}^{\infty} e^{-\alpha t} L(\mathbf{x}(t), \mathbf{u}(t)) dt \\ \text{s.t.} \quad & d\mathbf{x}(t)/dt = F\left(\mathbf{x}(t), \mathbf{u}(t)\right) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & S\left(\mathbf{x}(t), \mathbf{u}(t)\right) \leq \mathbf{0}. \end{aligned} \tag{3}$$

## Approximate Dynamic Programming Algorithms for Reservoir Production

In this section, we develop an optimization algorithm based on Approximate Dynamic Programming (ADP) for the dynamic optimization model presented above. We first review Dynamic Programming (DP) and Approximate Dynamic Programming (ADP), then describe how we construct basis functions for reservoir production problems, then introduce a linear programming (LP) based approach to compute the basis-function coefficients, then discuss the computation of globally-optimal controls (or near globally-optimal controls), and finally describe two advanced techniques – adaptive basis function selection and bootstrapping – to improve algorithm performance. Throughout this section, we assume that we have access to a reservoir simulator, which can numerically solve $d\mathbf{x}(t)/dt = F\left(\mathbf{x}(t), \mathbf{u}(t)\right)$.

### Review of Dynamic Programming

Dynamic programming (DP) offers a class of optimization algorithms that decompose a dynamic optimization problem into a sequence of simpler sub-problems. At each time $t$, an optimal control action $\mathbf{u}^*(t)$ is computed by solving one sub-problem. These sub-problems are coordinated across time through a value function denoted by $J^*$. The value function captures the future impact of the

current action; the algorithm must balance immediate payoff against future possibilities. In our model, the value function is defined by:

$$\begin{array}{rcll} J^*(\mathbf{x}_0) & = & \max\limits_{u(t),t\geq 0} & \int_{t=0}^{\infty} e^{-\alpha t} L(\mathbf{x}(t),\mathbf{u}(t))dt \\ & & \text{s.t.} & d\mathbf{x}(t)/dt = F\left(\mathbf{x}(t),\mathbf{u}(t)\right) \\ & & & \mathbf{x}(0) = \mathbf{x}_0 \\ & & & S\left(\mathbf{x}(t),\mathbf{u}(t)\right) \leq \mathbf{0}. \end{array} \tag{4}$$

Note that $J^*(\mathbf{x}_0)$ is the maximum net present value starting from state $\mathbf{x}_0$ at time 0. Further, given the time homogeneity of our model, the value $J^*(\mathbf{x}(t))$ at any time $t$ and for any state $\mathbf{x}(t)$ is the maximum net present value of payoffs that can be accumulated starting at that time and state.

Given the value function $J^*$ and current state $\mathbf{x}(t)$, an optimal control action at time $t$ can be selected by solving the following optimization problem [2]:

$$\begin{array}{ll} \max\limits_{\mathbf{u}} & L\left(\mathbf{x}(t),\mathbf{u}\right) + F\left(\mathbf{x}(t),\mathbf{u}\right)^T \nabla J^*\left(\mathbf{x}(t)\right) \\ \text{s.t.} & S(\mathbf{x}(t),\mathbf{u}) \leq \mathbf{0}, \end{array} \tag{5}$$

where $\nabla J^*$ is the gradient of $J^*$. Note that this problem decouples the choice of control action at time $t$ from that at all other times. It is in this sense that DP decomposes the dynamic optimization problem into simpler sub-problems through use of the value function. As we will see later, for reservoir production problems, the sub-problem (5) can be solved efficiently.

In light of the relative ease of generating optimal control actions given the value function, the challenge in optimizing reservoir production using DP (or ADP) reduces to the computation of the value function. As discussed in [12], the optimal value function for a continuous-time optimal control problem can in principle be computed by solving the Hamilton-Jacobi-Bellman (HJB) equation:

$$\max_{\mathbf{u}:\, S(\mathbf{x},\mathbf{u})\leq \mathbf{0}} \left\{ L(\mathbf{x},\mathbf{u}) + F(\mathbf{x},\mathbf{u})^T \nabla J(\mathbf{x}) - \alpha J(\mathbf{x}) \right\} = 0, \tag{6}$$

which is a nonlinear partial differential equation, or alternatively by solving an infinite dimensional linear program (LP):

$$\begin{array}{ll} \min\limits_{J} & \int J(\mathbf{x})\pi(d\mathbf{x}) \\ \text{s.t.} & L(\mathbf{x},\mathbf{u}) + F(\mathbf{x},\mathbf{u})^T \nabla J(\mathbf{x}) - \alpha J(\mathbf{x}) \leq 0 \quad \forall \mathbf{x},\mathbf{u} \text{ such that } S(\mathbf{x},\mathbf{u}) \leq \mathbf{0}, \end{array} \tag{7}$$

where $\pi$ is a probability measure chosen with exhaustive support such that that the integral of $J^*$ is finite. This LP poses an infinite number of decision variables and an infinite number of constraints since there is one decision variable $J(\mathbf{x})$ per state $\mathbf{x}$ and there is one constraint $S(\mathbf{x},\mathbf{u}) \leq 0$ per state-action pair $(\mathbf{x},\mathbf{u})$.

In most reservoir production problems of practical interest, solving the HJB equation (6) or the LP (7) exactly is impossible, as this would require computing and storing $J^*$ for each state in a continuous state space. Even if we were to discretize the state space by quantizing each state variable into multiple discrete values, the number of discrete states would grow exponentially as a function of $D$, the dimension of the state space, again making storage and computation impractical.

In a few special cases, the value function assumes a special structure which facilitates efficient computation and storage. This is the case, for example, with single-phase reservoir models with no constraints on control actions and an instantaneous payoff function that is quadratic in state and control action. In this context, the value function is itself quadratic and the problem reduces to one of linear-quadratic control (LQ). Such problems are well known to be tractable. For more realistic problems, involving for example multiphase flow, the reservoir dynamics are nonlinear and computing the value function exactly is infeasible.

**Approximate Value Function**

One way to overcome the curse of dimensionality is to approximate the value function. As is common in ADP, we will approximate the value function of our dynamic optimization model in terms of a linear combination of a selected set of basis functions:

$$J^*(\mathbf{x}) \approx \tilde{J}(\mathbf{x}) = \sum_{k=0}^{K} r_k \phi_k(\mathbf{x}), \tag{8}$$

where $\phi_k(\cdot)$, $k = 0, \cdots, K$ is a set of basis functions, and the $r_k$'s are their respective coefficients. Replacing the exact value function $J^*$ with the approximation $\tilde{J}$ when solving each sub-problem (5) results in a control strategy $\tilde{\mathbf{u}} = \tilde{\mu}(\mathbf{x})$. Of course $\tilde{\mu}$ is not in general equal to $\mu^*$, and thus is not in general a global optimum. However, if $\tilde{J}$ is "close" to $J^*$, the use of controls $\tilde{\mu}$ will provide performance that is "close" to the global optimal. ADP algorithms aim to obtain a near-optimal strategy $\tilde{\mu}$ through appropriate (application-specific) approximation of the value function.

Our ADP algorithm proceeds as follows:

1. Construct a set of basis functions $\phi_k(\cdot)$, $k = 0, \cdots, K$.
2. Compute the coefficients $r_k$ of the basis functions.
3. Compute near-optimal control actions $\tilde{\mathbf{u}}$ using the approximate value function $\tilde{J}$ by solving sub-problem (5).

In general, each step of the above procedure requires a custom design, namely one that exploits the structure of the problem under study. In the remainder of this section, we will discuss our implementation of each of these key steps. In addition, we introduce two additional techniques, adaptive basis function selection and bootstrapping, to further enhance the performance of our proposed ADP algorithm. The overall procedure is illustrated in Figure 1.

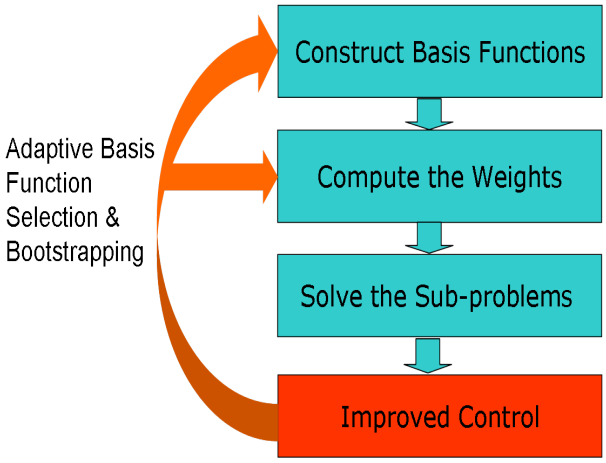



Figure 1: Flowchart of proposed ADP algorithm for reservoir production problems.

**Basis Function Construction**

We now describe the construction of the basis functions $\phi_k(\cdot)$ for reservoir production problems. Although there are studies focusing on constructing basis functions automatically (see, e.g., [17]), in most applications, in order to achieve a more accurate approximation of the value function, problem-dependent approaches are used. In our case, the value function $J^*$ is the net present value of the future total profit under the optimal control strategy. Hence, the basis functions should encode information about aspects of the states that strongly impact future profit. We have observed that the following two categories of functions are correlated to future profit, which has motivated us to choose basis functions accordingly:

1. Functions reflecting the "global status" of the reservoir, such as the average oil, water and gas saturations and the average pressure. A systematic way to capture the global status is through proper orthogonal decomposition (POD) of global snapshots. We will discuss how to construct basis functions based on POD later.
2. Functions reflecting the status of the near-well regions, such as the average oil saturation near production wells, watercut, average pressure near injection wells, etc.

In addition to the values described above, it is also useful to include higher-order polynomial functions of these values, as discussed below.

POD identifies a low-dimensional subspace that captures most of the system variability. It is used in model-order reduction for reservoir simulation (e.g., [5]) and other application areas, so it seems reasonable to apply POD for the construction of ADP basis functions. Our use of POD for basis construction proceeds as follows.

We first run reservoir simulations under a prescribed baseline strategy $\mu_0$ and record "snapshots" of the states. That is, we use the reservoir simulator to solve

$$d\mathbf{x}/dt = F\left(\mathbf{x}, \mu_0(\mathbf{x})\right),$$

and sample the state trajectory $\mathbf{x}(t)$ at $t = t_0, t_1, t_2, \cdots, t_{L-1}$. We define $\mathbf{x}_p(t)$ and $\mathbf{x}_S(t)$ to be vectors containing the pressure and saturation components of $\mathbf{x}(t)$, respectively (for an oil-water problem). The sample means are denoted by $\bar{\mathbf{x}}_p$ and $\bar{\mathbf{x}}_s$.

The normalized snapshot matrices for pressure and saturation are

$$\begin{aligned} X_p &= [\mathbf{x}_p(t_0) - \bar{\mathbf{x}}_p, \mathbf{x}_p(t_1) - \bar{\mathbf{x}}_p, \cdots, \mathbf{x}_p(t_{L-1}) - \bar{\mathbf{x}}_p] \\ X_S &= [\mathbf{x}_S(t_0) - \bar{\mathbf{x}}_S, \mathbf{x}_S(t_1) - \bar{\mathbf{x}}_S, \cdots, \mathbf{x}_S(t_{L-1}) - \bar{\mathbf{x}}_S], \end{aligned}$$

respectively. The subspace can now be characterized using singular value decomposition (SVD). Specifically, after applying SVD, we decompose $X_p$ and $X_S$ as $X_p = U_p \Sigma_p V_p^T$ and $X_S = U_S \Sigma_S V_S^T$, respectively, where $\Sigma_p$ and $\Sigma_S$ are diagonal matrices made up of singular values in descending order. Let $U_p(1:N_p)$ and $U_S(1:N_S)$ denote the first $N_p$ and $N_S$ columns of $U_p$ and $U_S$, respectively. We define the projection matrix $\mathbf{\Phi}$ as

$$\mathbf{\Phi} = \begin{bmatrix} U_p(1:N_p) & \mathbf{0} \\ \mathbf{0} & U_S(1:N_S) \end{bmatrix}. \tag{9}$$

Note that $span(\mathbf{\Phi})$ is a subspace that captures most of the variation in pressure and saturation.

Now, we construct basis functions that are polynomials defined on $span(\mathbf{\Phi})$. Specifically, letting $\varphi_i$ denote the $i$th column of $\mathbf{\Phi}$, the basis functions take the form

$$\phi_k(\mathbf{x}) = M_k \prod_{i=1}^{I} \left(\varphi_{k_i}^T \mathbf{x}\right)^{m_i},$$

where $m_i$ are nonnegative integers, $M_k$ is a normalization constant and $I$ is the number of terms in the basis function $\phi_k$. We say $\phi_k$ is a one-term polynomial if $I = 1$.

This approach is based on the assumption that the reservoir dynamics under the baseline strategy $\mu_0$ are sufficiently similar to those under the optimal strategy $\mu^*$. This is not always the case. One way to address this issue involves bootstrapping, as we will describe later.

It is evident that we can potentially construct many (polynomial) functions reflecting either the status of the entire reservoir or the status of near-well regions. Thus, there are many candidate basis functions. However, the use of too many basis functions can result in onerous computational requirements or performance loss due to overfitting. Thus, in practice, only a subset of all of these candidate basis functions are used to approximate the value function. We will later propose an adaptive basis function selection scheme, which chooses an effective subset of basis functions.

Finally, no matter how we construct and choose basis functions, we always include a constant basis function $\phi_0(\mathbf{x}) = 1$ to achieve better approximation results and numerical stability. We also normalize all basis functions such that $|\phi_k(\mathbf{x}_0)| = 1$ for $k = 0, \cdots, K$, where $\mathbf{x}_0$ is the initial state.

**Computation of Coefficients**

As discussed above, for a given set of basis functions, we need to compute coefficients to approximate the value function. A number of ADP algorithms have been proposed for this purpose. Examples include approximate value iteration, approximate policy iteration, temporal-difference learning, Bellman error minimization and approximate linear programming (ALP) (see [3, 25, 19]). In this paper, we use smoothed reduced linear programming (SRLP) with $L_1$ regularization. This is a variant of ALP proposed in [9]. Compared with other ADP algorithms, the main advantage of this procedure (in our context) is that it provides "optimal" solutions in much less time than alternative approaches. In the remainder of this section, we describe the SRLP algorithm with $L_1$ regularization. A key aspect of this algorithm – constraint sampling – is discussed in the Appendix.

Theoretically, the value function can be obtained by solving the infinite dimensional LP (7). However, this is not feasible because there are an infinite number of decision variables and an infinite number of constraints. This technical difficulty can be addressed by constraining the function to the span of the basis functions. In particular, we consider solving

$$\begin{aligned} \min_{\mathbf{r}} \quad & \int \sum_{k=0}^{K} r_k \phi_k(\mathbf{x}) \pi(d\mathbf{x}) \\ \text{s.t.} \quad & L(\mathbf{x},\mathbf{u}) + F(\mathbf{x},\mathbf{u})^T \sum_{k=0}^{K} r_k \nabla \phi_k(\mathbf{x}) - \alpha \sum_{k=0}^{K} r_k \phi_k(\mathbf{x}) \le 0 \quad \forall \mathbf{x}, \mathbf{u} \text{ such that } S(\mathbf{x},\mathbf{u}) \le \mathbf{0}, \end{aligned} \tag{10}$$

which is known as the approximate linear program (ALP). However, there are still an infinite number of constraints in the ALP (10), so it cannot be solved exactly. We overcome this issue through constraint sampling; that is, only $\mathcal{M}$, a finite set of sampled states and control actions, is used to constrain variables. This results in what is known as the reduced linear program (RLP). Let $\mathcal{M} = \left\{(\mathbf{x}^{(1)}, \mathbf{u}^{(1)}), \cdots, (\mathbf{x}^{(M)}, \mathbf{u}^{(M)})\right\}$, where $M$ is the cardinality of $\mathcal{M}$. The RLP takes the form

$$\begin{aligned} \min_{\mathbf{r}} \quad & \frac{1}{M} \sum_{m=1}^{M} \sum_{k=0}^{K} r_k \phi_k(\mathbf{x}^{(m)}) \\ \text{s.t.} \quad & L(\mathbf{x}^{(m)},\mathbf{u}^{(m)}) + F(\mathbf{x}^{(m)},\mathbf{u}^{(m)})^T \sum_{k=0}^{K} r_k \nabla \phi_k(\mathbf{x}^{(m)}) - \alpha \sum_{k=0}^{K} r_k \phi_k(\mathbf{x}^{(m)}) \le 0 \quad \forall m = 1, \cdots, M. \end{aligned} \tag{11}$$

The way in which constraints are sampled significantly impacts the performance of the resulting approximation, and our approach to constraint sampling is nontrivial. Roughly speaking, we sample the state/action pairs based on a randomized baseline strategy and then

bootstrap this process. Details are provided in the Appendix. We choose the number of sampled states $M$ to be 40 to 60 times the size of $K$, the number of basis functions. It has been observed in some studies [9] that RLP (11) might be in some sense over-constrained. We have also observed that the $L_1$ regularization of $\mathbf{r}$ can improve performance. These observations motivate the smoothed reduced linear program (SRLP) [9] with $L_1$ regularization:

$$\begin{aligned}
\min_{\mathbf{r},\mathbf{s}} \quad & \frac{1}{M}\sum_{m=1}^{M}\sum_{k=0}^{K} r_k\phi_k(\mathbf{x}^{(m)}) + \epsilon||\mathbf{r}||_1 \\
\text{s.t.} \quad & L(\mathbf{x}^{(m)},\mathbf{u}^{(m)}) + F(\mathbf{x}^{(m)},\mathbf{u}^{(m)})^T\sum_{k=0}^{K} r_k\nabla\phi_k(\mathbf{x}^{(m)}) - \alpha\sum_{k=0}^{K} r_k\phi_k(\mathbf{x}^{(m)}) \le s^{(m)} \quad \forall m = 1,\cdots,M. \\
& s^{(m)} \ge 0 \quad \forall m = 1,\cdots,M. \\
& \sum_{m=1}^{M} s^{(m)} \le \theta,
\end{aligned} \tag{12}$$

where $\mathbf{s} = \left[s^{(1)}, s^{(2)}, \cdots, s^{(M)}\right]$. There are two parameters to be specified in (12), $\theta$ and $\epsilon$. The variable $\theta \ge 0$ characterizes the extent to which the constraints of the RLP are relaxed, and $\epsilon \ge 0$ characterizes a penalty on the magnitude of $\mathbf{r}$, in the sense of the $L_1$ norm. We have observed that the selection of $\theta$ and $\epsilon$ influences $\mathbf{r}_{srlp}$, the solution of (12). We use line search to choose the $\theta$ and $\epsilon$ that provide the maximum net present value. The progression of the algorithms we have described, from the infinite-dimensional LP (7) to the SRLP (12), is summarized in Figure 2.

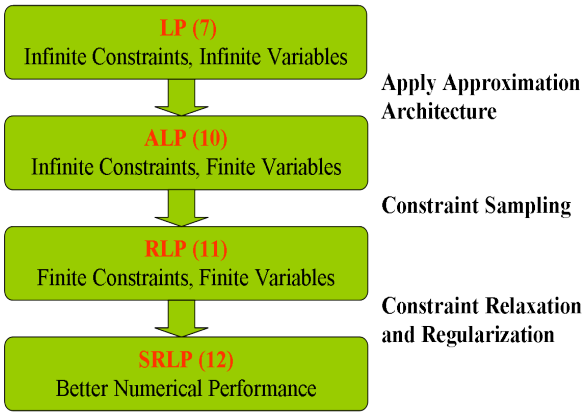



Figure 2: Progression of algorithms from the infinite-dimensional LP to the SRLP.

An alternative approach to computing coefficients is temporal-difference (TD) learning, which is described in detail in the Appendix. TD learning outperforms SRLP in the primary production case, as we will demonstrate in the results below. However, for general multiphase cases, the performance of the SRLP approach is superior.

**Solving Sub-Problems**

Once the approximate value function $\tilde{J}$ is available, control actions $\tilde{\mathbf{u}}$ can be obtained by solving (5), with $J^*$ replaced by $\tilde{J}$. A useful observation in reservoir production problems can significantly simplify this step. Consider the reservoir dynamics $d\mathbf{x}/dt = F(\mathbf{x},\mathbf{u})$ and constraints $S(\mathbf{x},\mathbf{u}) \le 0$. In general, $F(\mathbf{x},\mathbf{u})$ and $S(\mathbf{x},\mathbf{u})$ are nonlinear functions of the state $\mathbf{x}$. However, because well transmissibility does not depend on BHP or flow rate, $F$ is affine in the control action $\mathbf{u}$ for fixed $\mathbf{x}$ and, for common constraints arising in practice, such as watercut and maximum/minimum flow rates, $S$ is also affine in the control action $\mathbf{u}$ for fixed $\mathbf{x}$.

This means that

$$\begin{aligned}
F(\mathbf{x},\mathbf{u}) &= F_1(\mathbf{x}) + F_2(\mathbf{x})\mathbf{u} \\
S(\mathbf{x},\mathbf{u}) &= S_1(\mathbf{x}) + S_2(\mathbf{x})\mathbf{u}.
\end{aligned}$$

Thus, the sub-problem at time $t$ is

$$\max_{\mathbf{u}} \quad L\left(\mathbf{x}(t),\mathbf{u}\right) + \left[\nabla\tilde{J}\left(\mathbf{x}(t)\right)\right]^T F_2(\mathbf{x}(t))\mathbf{u} \tag{13}$$

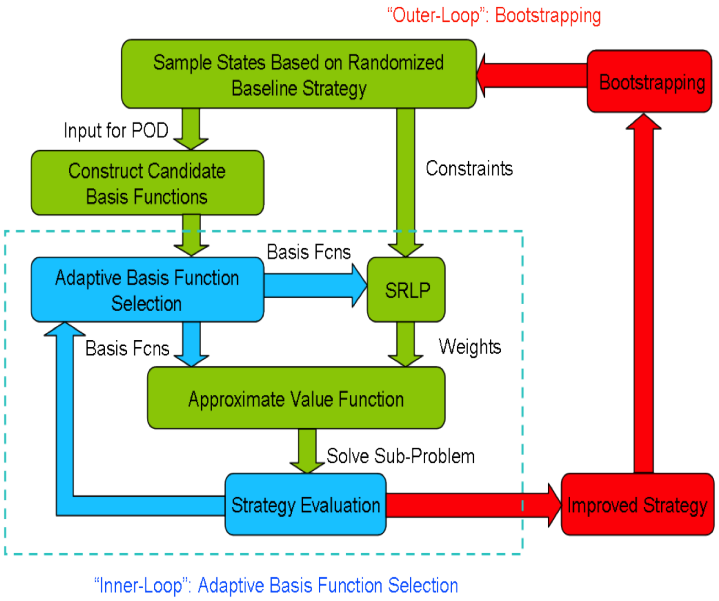



Figure 3: Architecture of the overall ADP algorithm.

$$\text{s.t.} \qquad S_1(\mathbf{x}(t)) + S_2(\mathbf{x}(t))\mathbf{u} \leq \mathbf{0}.$$

This problem (13) is easy to solve if the instantaneous payoff function $L$ is concave in $\mathbf{u}$. For example, with the instantaneous payoff function $L$ defined in (2), the sub-problem at each time reduces to a low-dimensional linear program.

### Adaptive Basis Function Selection and Bootstrapping

The ADP algorithm described thus far is "open-loop" in the sense that once a new control $\tilde{\mathbf{u}}$ is available, it does not get used to further improve the result. In this subsection, we introduce two advanced techniques, adaptive basis function selection and bootstrapping, to "close the loop" and enhance the performance of the ADP algorithm.

***Adaptive Basis Function Selection*** As discussed earlier, in practical implementations, we need to select a subset of basis functions from the large set of candidates. One way to effectively choose basis functions is through use of adaptive basis function selection. Specifically, we initialize the set of basis functions $\mathcal{F}$ as a set containing only the constant basis function $\phi_0(\cdot) = 1$. At each iteration, we choose a new basis function and add it to $\mathcal{F}$. Our rule in selecting the new basis function is that, compared to other candidate basis functions, including the new basis function in $\mathcal{F}$ results in the largest increase in the net present value. We continue to add basis functions until the addition of any new basis function leads to a decrease in net present value. A detailed description of this algorithm is given in the Appendix.

***Bootstrapping*** Another technique to further improve the performance of our ADP algorithm is bootstrapping. Specifically, to construct basis functions based on POD and sample the constraints for SRLP (12), we need to sample states and state/action pairs based on a known strategy $\mu$. Theory [8] suggests that, to obtain the best results, we should use an optimal control policy when sampling. Since an optimal policy is not available (the objective of the ADP algorithm is to find a policy close to optimal), we sample using a baseline strategy $\mu_0$, which inevitably leads to some performance loss.

Bootstrapping has been proposed to address this issue. Specifically, we start from a baseline strategy $\mu_0$, then we apply the above-described ADP algorithm to compute an approximate value function $\tilde{J}_1$, which generates a new strategy $\mu_1$. Then, we sample states and state/action pairs based on $\mu_1$, and apply the ADP algorithm again to obtain a new approximate value function $\tilde{J}_2$ and a policy $\mu_2$. We repeat this procedure until it no longer increases net present value. The Appendix provides a more detailed description of the bootstrapping algorithm.

Figure 3 illustrates the architecture of our ADP algorithm with the incorporation of adaptive basis function selection and bootstrapping.

## Computational Requirements for the ADP Algorithm

As is the case for other production optimization algorithms, most of the computation in ADP is consumed by the reservoir simulations. Thus it is appropriate to assess the computational demands for ADP in terms of the required number of simulations. As shown in Figure 3, the ADP algorithm uses the reservoir simulator to sample states and evaluate the strategy $\tilde{\mu}$. As discussed above, in general, SRLP (12) requires $40K$ to $60K$ state samples, where $K$ is the number of basis functions, and each state sample requires one simulation. For example, in Case 2 below, we use 39 basis functions and 2000 state samples. Hence, the number of simulations is 2001, with one extra simulation for strategy evaluation.

Adaptive basis function selection requires a large number of additional simulations. For this approach, assume there are $\tilde{K}$ candidate basis functions and we are allowed to use up to $K_{max}$ basis functions. One obvious upper bound on the number of required simulations is $60K_{max} + \tilde{K}K_{max}$. However, this bound is far from tight for the following reasons: first, as is observed in Case 3 below, the number of basis functions ultimately used ($K$) tends to be much less than $K_{max}$. Second, the ADP algorithm does not need to reevaluate the strategy if the computed coefficient of the new basis function is zero, which happens frequently since coefficients are solved by SRLP with $L_1$ regularization. Thus the number of required simulations in practice is much less than the theoretical bound. For Case 3 below, $\tilde{K} = 412$, $K_{max} = 50$, so the theoretical bound is around 23600. However, for this case $K = 7$ and we only perform about 2600 simulations.

For the ADP algorithm with bootstrapping, assuming the algorithm applies the bootstrapping $B$ times, the required number of simulations will be $B$ times the above results. In practice, it has been observed that the performance tends to degrade after several iterations of bootstrapping (see [11]; this is also observed in Case 1). Hence, $B \leq 10$ in most cases. For Case 1 below, where each bootstrapping iteration requires 1001 simulations, we observe that the ADP algorithm bootstraps four times and the total number of simulations is 4004.

It is evident that ADP run time scales with the number of (candidate) basis functions. In general, the use of more candidate basis functions in adaptive basis function selection will improve algorithm performance though it requires more computation. Thus, there is a trade-off between performance and computational effort. It is important to note that both state sampling and adaptive basis function selection can be implemented in parallel. Thus, by using multiple processors, the elapsed time for the ADP algorithm need not be excessive.

One way to design the candidate basis function pool is to consider only one-term polynomials. In this case, the number of candidate basis functions is $\tilde{K} = O(N_p + N_S + N_w)$, where $N_w$ is the number of wells/well groups and $N_p$ and $N_S$ are determined during POD. However, in some cases, the "cross terms" (i.e., functions with more than one term) are useful in approximating the value function. Of course, adding all such cross terms will result in an intractably large candidate basis function pool. One way to address this issue is to add only the cross terms between the first few columns of $\mathbf{\Phi}$ to the candidate basis function pool. This approach is used in Case 3 below.

We note finally that it is not always necessary to use adaptive basis function selection or bootstrapping. In Case 2 below, for example, useful results are obtained without using either of these procedures.

## Simulation Results

In this section, we apply the ADP algorithm to three example cases and compare our results against various baseline strategies. These examples involve single-phase primary production, waterflood with bound constraints, and waterflood with general (nonlinear) constraints. All examples involve two-dimensional reservoir models. In our implementation we use Stanford's General Purpose Research Simulator, GPRS [4, 16] for all simulations and CPLEX, a commercial LP solver, to solve the SRLP (12). Other components of the ADP algorithm are implemented in MATLAB.

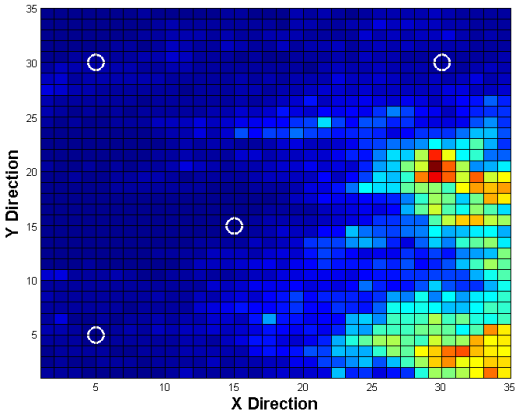
### Case 1: Primary Production with Penalty Term

In the first example, we apply the ADP algorithm to a single-phase primary production case. The reservoir dynamics are described by a linear flow equation in this case. The geological model is a modified portion of the SPE 10 model (see [6]). The $x$-direction permeability field is shown in Figure 4(a). The model contains $35 \times 35$ grid blocks and has four production wells, each of which is controlled by a bounded BHP. We simulate for 6000 days with 200 control periods. The instantaneous profit function is

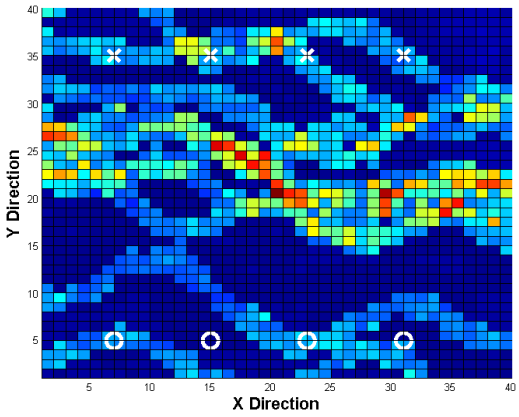
$$L(\mathbf{x}, \mathbf{u}) = \text{revenue from producing oil} + \lambda \sum_{i=1}^{4} \log(\mathbf{u}_i - LB_i),$$

where $\mathbf{u}_i$ is the BHP for producer $i$ and $LB_i$ is its lower bound. The logarithm term penalizes BHPs that are close to the lower bound. Including this term (in this form) enables an analytical solution of the global optimum using DP. The parameters for Case 1 are summarized in Table 1.

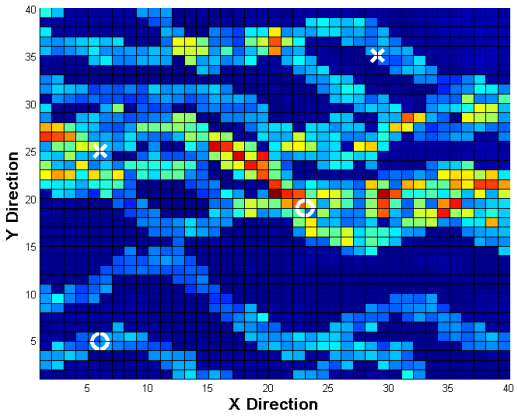
We apply the ADP algorithm with fixed basis functions and bootstrapping. The basis functions are constructed based on POD. Specifically, POD characterizes a subspace of dimension 13 that captures more than $99.9999\%$ of the variation in pressure (in the sense of singular values). We choose basis functions as one-term polynomials in that subspace. In each iteration of bootstrapping, we resample 1000 states to reconstruct the basis functions and better constrain the SRLP (12). Hence, for this case, the ADP algorithm requires 4004 simulations in total. Bootstrapping is terminated at the fourth iteration since at this point performance starts to degrade.




(a) Case 1




(b) Case 2



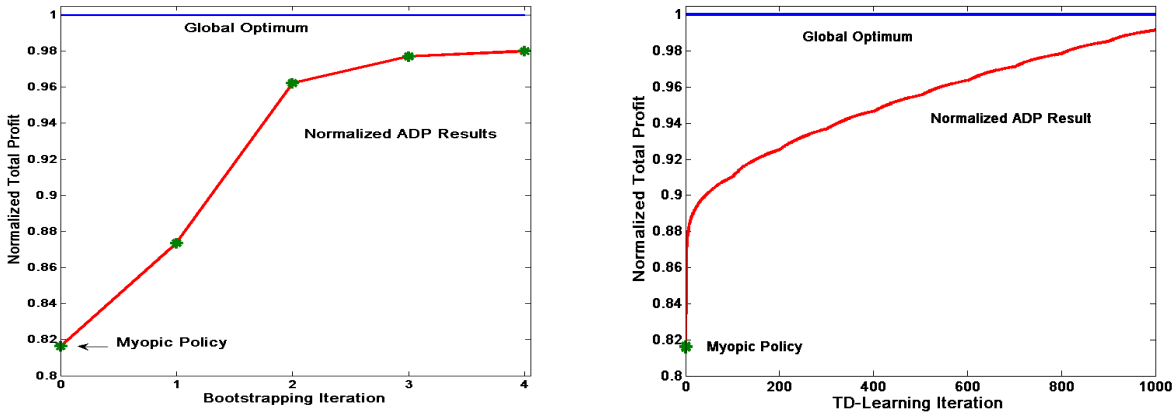

(c) Case 3

Figure 4: Geological model and well configurations for the three cases. Injection and production wells are represented as white crosses and white circles, respectively. Grid blocks are colored to indicate value of permeability in $x$-direction (red and blue indicate high and low permeability, respectively).

Table 1: Parameters for Case 1

| | | | |
|---|---|---|---|
| BHP range for Producer 1 | 2500-5000 psi | BHP range for Producer 2 | 2400-5000 psi |
| BHP range for Producer 3 | 2700-5000 psi | BHP range for Producer 4 | 2600-5000 psi |
| Initial pressure | 4500 psi | Oil price | \$42.93/STB |
| Logarithm penalty coefficient $\lambda$ | $1 \times 10^4$ | Discount rate $\alpha$ | $5 \times 10^{-4}$ |

The result using the ADP algorithm is presented in Figure 5(a), where the normalization is with respect to the global optimum. We observe that the result from the ADP algorithm is within $2\%$ of the global optimum. The "myopic policy" in Figure 5(a) is a policy that aims at maximizing the instantaneous profit at each time. It is evident that the result of the ADP algorithm is significantly better than that using the myopic policy.




(a) SRLP (b) TD Learning

Figure 5: Performance of ADP for primary production problem using (a) SRLP and (b) TD learning.

For this case, the use of TD learning to compute coefficients yields slightly better performance. This is illustrated in Figure 5(b), where the ADP optimum is now within $1\%$ of the global optimum. Each iteration of TD learning requires only one simulation, so $1000$ total simulations are performed. We have observed, however, that the use of SRLP to compute the coefficients tends to outperform TD learning for multiphase models.

Although this case is quite simple, it represents one of the few cases where the global optimum can be obtained analytically. It is significant that, for a case where the global optimum is known, ADP does indeed provide a result that is very close to the global optimum.

**Case 2: Water Injection with Bounded BHPs**

In the second example, we apply the ADP algorithm to an oil-water model with bounded BHPs. The reservoir model contains $40 \times 40$ blocks, with four producers and four injectors. The model is simulated for $3000$ days and there are $10$ control periods. The permeability field in the $x$-direction is shown in Figure 4(b). The instantaneous profit in this case is computed using (2). The parameters used for Case 2 are summarized in Table 2.

Table 2: Parameters for Case 2

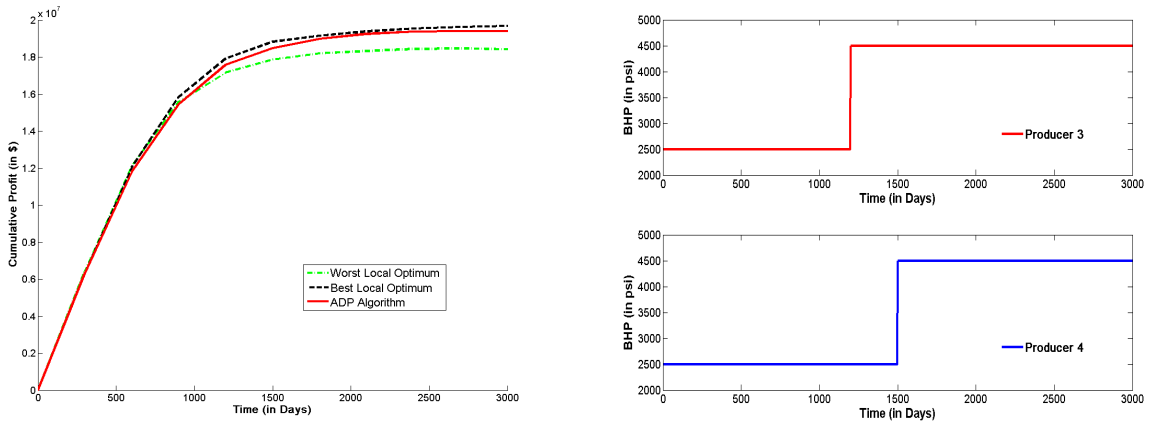
| | | | |
|---|---|---|---|
| BHP range for producers | 2500-4500 psi | BHP range for injectors | 6000-9000 psi |
| Initial pressure | 5080 psi | Initial $S_w$ | 0.15 |
| Oil price | \$80/STB | Water production cost | \$36/STB |
| Water injection cost | \$18/STB | Discount rate $\alpha$ | $1 \times 10^{-3}$ |

In this case, we compare the performance of our proposed ADP algorithm with that of the gradient-based method described in [20].

The performance of the gradient-based method depends on the starting point. For this case, we ran the gradient-based method from 200 randomly generated starting points and thus obtained 200 local optima. We recorded the best and the worst of these local optimum.

For the ADP algorithm, we use a fixed set of basis functions. Neither adaptive basis function selection nor bootstrapping are applied for this case. The subspace characterized by POD is of dimension 22 (with $N_p = 8$ and $N_S = 14$). We select the following basis functions: (1) the constant basis function $\phi_0 = 1$, (2) one-term polynomials of $\bar{S}_w$, $\bar{S}_o$, $\bar{p}$ and oil-water ratio up to fourth order, and (3) 22 one-term polynomials defined on subspace $span(\mathbf{\Phi})$ up to fourth order. Thus, there are 39 basis functions in total. We sample 2000 states to constrain the SRLP (12), and set $\epsilon = 1 \times 10^{-4}$ and $\theta = 500$. The total number of simulations performed for this case is around 2000.

The optimization results are shown in Figure 6(a). The performance of ADP is within $1\%$ of the best local optimum found by the gradient method and $6\%$ better than the worst local optimum. In Figure 6(b), we illustrate the BHP schedules for Producers 3 and 4 computed by the ADP algorithm. Here we observe a "bang-bang" control sequence (this is also observed for the gradient-based method), meaning the producer BHPs abruptly switch from one bound (2500 psi) to the other (4500 psi).




(a) Cumulative profit from ADP and gradient algorithms

(b) Controls for Producers 3 and 4 determined by ADP

Figure 6: Optimization results for oil-water model with bound controls.

It is of interest to note that, for the gradient-based algorithm, around $70\%$ of all of the local optima computed tend to be the best local optima (which we conjecture to be the global optimum). Thus for this case the problem of finding local optima that are significantly inferior to the global optimum does not seem to be an issue. It is nonetheless significant that ADP achieves a result that is very close to the global optimum. It is worth reiterating that some degree of underperformance relative to the global optimum is to be expected with ADP, since the algorithm does involve some approximation.

**Case 3: Water Injection with General Constraints**

In the third example, we consider an oil-water model with general constraints. The reservoir model contains $40 \times 40$ blocks; flow is driven by two injectors and two producers. We simulate for 3285 days and there are 12 control periods. The $x$-direction permeability field is shown in Figure 4(c). The instantaneous profit in this case is specified by (2). The constraints and parameters are given in Table 3.

Table 3: Parameters for Case 3

| | | | |
|---|---|---|---|
| BHP range for producers | 1500-6000 psi | BHP range for injectors | 2000-8000 psi |
| Maximum field water injection rate | 338 STB/day | Minimum field oil production rate | 113 STB/day |
| Maximum field liquid production rate | 394 STB/day | Maximum watercut | 0.6 |
| Oil price | \$50/STB | Water production cost | \$10/STB |
| Water injection cost | \$5/STB | Initial pressure | 4896psi |
| Initial $S_w$ | 0.15 | Discount rate $\alpha$ | $1 \times 10^{-3}$ |

For reservoir production problems with general constraints, it can be difficult to find a feasible control strategy without applying an optimization technique. However, for the sub-problems (13), feasible control strategies can be found as follows. We randomly sample parameters $\mathbf{g}$ and replace the unknown linear function $L(\mathbf{x}(t), \mathbf{u}) + [\nabla J^*(\mathbf{x}(t))]^T F_2(\mathbf{x}(t))\mathbf{u}$ by $\mathbf{g}^T\mathbf{u}$. At time $t$ with state $\mathbf{x}(t)$, we then solve the following LP to obtain a feasible control at time $t$:

$$\begin{aligned} \max_{\mathbf{u}} \quad & \mathbf{g}^T\mathbf{u} \\ \text{s.t.} \quad & S_1(\mathbf{x}(t)) + S_2(\mathbf{x}(t))\mathbf{u} \leq \mathbf{0}. \end{aligned} \tag{14}$$

If (14) is infeasible at time $t$, we restart this procedure from time 0. We use this approach to generate 100 feasible control strategies, and choose the best one as the baseline for Case 3.

Adaptive basis function selection is used for this case. The coefficients are computed based on SRLP (12). Specifically, we partition $\mathbf{\Phi} = [\mathbf{\Phi}_1, \mathbf{\Phi}_2]$, where $\mathbf{\Phi}_1$ includes the first seven left singular vectors. The candidate basis function pool includes (1) a constant basis function $\phi_0 = 1$, (2) one-term polynomials of $\bar{S}_w$, $\bar{S}_o$, $\bar{p}$ and oil-water ratio of the entire reservoir up to third order, (3) one-term polynomials of the average oil saturations in near-producer regions up to third order, (4) all the polynomials defined on $span(\mathbf{\Phi}_1)$ up to third order, and (5) all the other one-term polynomials defined on $span(\mathbf{\Phi})$ up to third order. There are thus a total of $\tilde{K} = 412$ candidate basis functions. We use SRLP (12) to compute the coefficients and set $\theta = 0$ and $\epsilon = 0.005$.

The maximum number of basis functions is prescribed as $K_{max} = 50$, so we sample 2000 states to constrain the SRLP. The adaptive basis function selection algorithm terminates after having added six basis functions. Due to $L_1$ regularization in SRLP, the total number of strategy evaluations in the course of adaptive basis function selection is about 600.

Optimization results are presented in Figure 7(a). The ADP algorithm achieves a 19% improvement compared with the baseline strategy. We plot the tight constraints over time in Figures 7(b), 7(c) and 7(d). It is evident that the maximum liquid production rate is tight at the beginning of simulation, the lower bounds on the producer BHPs are tight at the end of the simulation, and the maximum water injection rate is tight at all times. The other constraints are not tight over the course of the simulation, which suggests that they could be relaxed for this example. In any event, this example demonstrates that our ADP algorithm does indeed satisfy bound and nonlinear constraints throughout the simulation.

## Concluding Remarks

In this paper, we developed an optimization algorithm for reservoir production based on Approximate Dynamic Programming (ADP). We discussed the general algorithm, described how to construct ADP basis functions, and proposed two additional techniques, adaptive basis function selection and bootstrapping, to enhance algorithm performance. Results were presented for primary production and waterflooding problems. Both bound constraints and general constraints were considered. Compared with baseline strategies (or in one case results from a gradient-based algorithm), the performance of the ADP algorithm was found to be very good.

In future work, we plan to refine and further test our ADP implementation. This will include testing on larger three-dimensional problems and three-phase cases.

## Acknowledgements

We are grateful to the industry sponsors of the Stanford Smart Fields Consortium for partial funding of this work. We thank the Stanford Center for Computational Earth and Environmental Science for providing distributed computing resources. We also thank David Echeverría Ciaurri and Huanquan Pan for useful discussions and suggestions.

## References

[1] V. Artus, L. J. Durlofsky, J. Onwunalu, and K. Aziz. Optimization of nonconventional wells under uncertainty using statistical proxies. *Computational Geosciences*, 10:389–404, 2006.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 1995.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[4] H. Cao. *Development of Techniques for General Purpose Simulators*. PhD thesis, Stanford University, 2002.

[5] M. A. Cardoso and L. J. Durlofsky. Use of reduced-order modeling procedures for production optimization. *SPE Journal*, 15(2):426–435, June 2010.

[6] M. A. Christie and M. J. Blunt. Tenth SPE comparative solution project: A comparison of upscaling techniques. *SPE Reservoir Evaluation and Engineering*, 4:308–317, 2001.

[7] A. S. Cullick, D. Heath, K. Narayanan, J. April, and J. Kelly. Optimizing multiple-field scheduling and production strategy with reduced risk. SPE paper 84239 presented at the 2003 SPE Annual Technical Conference and Exhibition, Denver, Colorado.

[8] D. P. de Farias and B. Van Roy. On constraint sampling in linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, August 2004.

[9] V. V. Desai, V. F. Farias, and C. C. Moallemi. The smoothed approximate linear program. In *Advances in Neural Information Processing Systems 22*. MIT Press, 2009.

[10] V. F. Farias and B. Van Roy. An approximate dynamic programming approach to network revenue management. `www.stanford.edu/~bvr/psfiles/adp-rm.pdf`. Working paper.

[11] V. F. Farias and B. Van Roy. Tetris: A study of randomized constraint sampling. In G. Calafiore and F. Dabbene, editors, *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer-Verlag, 2006.

[12] J. Han and B. Van Roy. Control of diffusions via linear programming. To appear in *Stochastic Programming*.

[13] T. J. Harding, N. J. Radcliffe, and P. R. King. Optimization of production strategies using stochastic search methods. SPE paper 35518 presented at the 1996 European 3-D Reservoir Modeling Conference, Stavanger, Norway.

[14] M. Haugh, L. Kogan, and Z. Wu. Portfolio optimization with position constraints: an approximate dynamic programming approach. `www.columbia.edu/~mh2078/ADP_Dual_Oct06.pdf`. Working paper.

[15] O. J. Isebor. Constrained production optimization with an emphasis on derivative-free methods. Master's thesis, Stanford University, 2009.

[16] Y. Jiang. *Techniques for Modeling Complex Reservoirs and Advanced Wells*. PhD thesis, Stanford University, 2007.

[17] P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[18] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: A simple least-square approach. *Review of Financial Studies*, 14(1):113–147, 2001.

[19] W. B. Powell. *Approximate Dynamic Programming*. John Wiley and Sons, 2007.

[20] P. Sarma, L. J. Durlofsky, K. Aziz, and W. H. Chen. Efficient real-time reservoir management using adjoint-based optimal control and model updating. *Computational Geosciences*, 10:3–36, 2006.

[21] H. P. Simão, A. George, W. B. Powell, T. Gifford, J. Nienow, and J. Day. Approximate dynamic programming captures fleet operations for schneider national. *Interfaces*, 40(5):342–352, 2010.

[22] G. J. Tesauro. Temporal difference learning and TD–Gammon. *Communications of the ACM*, 38:58–68, 1995.

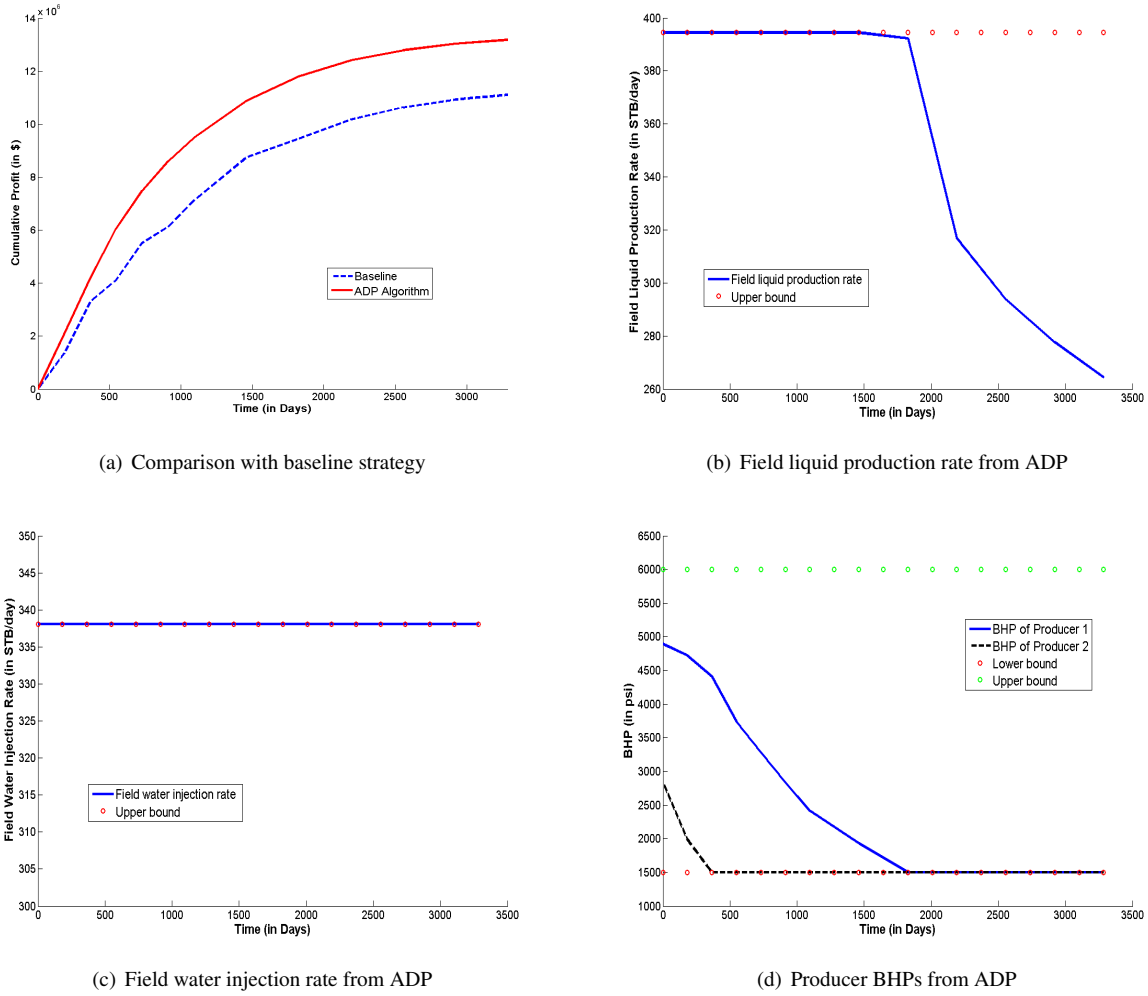
[23] J. N. Tsitsiklis and B. Van Roy. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, 1999.

[24] J. N. Tsitsiklis and B. Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, July 2001.

[25] B. Van Roy. Neuro-dynamic programming: Overview and recent trends. In E. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer, 2001.

[26] D. Zhang and D. Adelman. Dynamic bid-prices in revenue management. *Operations Research*, 55(4):647–661, 2007.

[27] D. Zhang and D. Adelman. An approximate dynamic programming approach to network revenue management with customer choice. *Transportation Science*, 43:381–394, 2009.




(a) Comparison with baseline strategy

(b) Field liquid production rate from ADP

(c) Field water injection rate from ADP

(d) Producer BHPs from ADP

Figure 7: Optimization results for oil-water model with general controls.

# Appendix

### Constraint Sampling Algorithm

A key issue in the development of SRLP is constraint sampling. To ensure that the LP approach can be implemented in a practical LP solver, we note that in RLP (11), a finite set of sampled states $\mathcal{M}$ rather than the whole state space is used to constrain the LP problem. Define $\mathbf{R}_{ALP}$ and $\mathbf{R}_{RLP,\mathcal{M}}$ as the feasible sets of ALP (10) and RLP (11) with state samples $\mathcal{M}$, respectively, and observe $\mathbf{R}_{ALP} \subseteq \mathbf{R}_{RLP,\mathcal{M}}$. To guarantee that the solution of RLP (11) is "close" to the solution of ALP (10), we expect intuitively that states should be sampled in such a way that $\mathbf{R}_{RLP,\mathcal{M}}$ provides a close approximation of $\mathbf{R}_{ALP}$.

From de Farias et al. [8], the optimal constraint sampling strategy is to sample the state set $\mathcal{M}$ based on the optimal strategy $\mu^*$. However, $\mu^*$ is what we aim to derive and is not available. Thus, in practice, we sample $\mathcal{M}$ based on the baseline strategy $\mu_0$. In general, we cannot assume that $\mu_0$ is close to $\mu^*$ and, hence, the states sampled based on $\mu_0$ might be far from optimal. As we have described, this problem can be resolved by bootstrapping.

Another problem associated with the constraint sampling strategy is as follows. Since the reservoir dynamics are deterministic, if we sample states based on a fixed strategy (e.g., the baseline strategy $\mu_0$), then we tend to sample similar states in different sampling iterations. We randomize the sampling strategy to resolve this problem. The state sampling algorithm is described in Algorithm 1.

In addition to initially choosing baseline strategy $\mu_0$ as the sampling strategy $\mu$, we also need to determine randomization magnitude $\eta$ to implement the constraint sampling algorithm. There exist some tradeoffs here: if $\eta$ is too small, the sampling strategy is not randomized enough, and we tend to sample similar states over different iterations. As a result, the SRLP (12) is not well-constrained. On the other hand, if $\eta$ is too large, the sampling strategy deviates significantly from $\mu$, which implies that, when $\mu$ is near-optimal, the states are not sampled based on the best strategy. In practice, we choose $\eta$ based on a line search.

**Algorithm 1** Constraint Sampling Algorithm

**Require:** Number of samples $M$, time step $\Delta$, initial state $\mathbf{x}_0$, randomization magnitude $\eta$ and a sampling strategy $\mu$.
**Ensure:** Return a set of sampled state/action pairs $\mathcal{M}$.
**for** $m = 1$ to $M$ **do**
Set initial state $\mathbf{x}(0) := \mathbf{x}_0$ and set horizon $N \sim Geom\left(e^{-\alpha\Delta}\right)$, where $Geom(q)$ refers to a geometric distribution with parameter $q$.
**for** $n = 0$ to $N-1$ **do**
Set randomized greedy control as $\mathbf{u}(n\Delta) = \mu\left(\mathbf{x}(n\Delta)\right) + \eta\tilde{\mathbf{u}}$, where $\tilde{\mathbf{u}}$ is a $length(\mathbf{u})$ dimensional vector whose components are i.i.d. uniformly distributed over interval $[-1, 1]$.
Set the control as $\mathbf{u}(n\Delta)$ on time interval $[n\Delta, (n+1)\Delta]$ and run reservoir simulator to obtain $\mathbf{x}((n+1)\Delta)$.
**end for**
Set the $m$th sample as $\mathbf{x}^{(m)} = \mathbf{x}(N\Delta)$ and $\mathbf{u}^{(m)} = \mathbf{u}((N-1)\Delta)$.
**end for**
**return** $\mathcal{M} = \left(\mathbf{x}^{(1)}, \mathbf{u}^{(1)}\right), \cdots, \left(\mathbf{x}^{(M)}, \mathbf{u}^{(M)}\right)$

It is also worth noting that in Algorithm 1, sampling one state/action pair is independent from sampling another. Hence, using parallel computing, the constraint sampling algorithm 1 can run much faster.

**TD-learning Algorithm**

An alternative approach for computing the coefficients of the basis functions is temporal-difference (TD) learning. The TD($\lambda$) algorithm for reservoir production problems is summarized in Algorithm 2.

**Algorithm 2** TD-learning Algorithm

**Require:** Set of basis functions $\{\phi_0 = 1, \phi_1, \cdots, \phi_K\}$, parameters $0 \leq \lambda \leq 1$ and $\gamma_0$, maximum number of iterations $I$, time step $\Delta$, horizon $T$, and initial state $\mathbf{x}_0$ for reservoir simulation.
**Ensure:** Return the coefficient vector $\mathbf{r}$.
Initialize $\mathbf{Z}_{cum} = 0$, $\mathbf{r} = 0$ and $i = 1$ {$\mathbf{Z}_{cum}$ is the cumulative eligibility vector }
**for** $i = 1, 2, \cdots, I$ **do**
Set $\mathbf{Z} = 0$ and $\gamma = \gamma_0/i$.
Run a simulation from time 0 to time $T$, with step size $\Delta$. At each step, generate control **greedy** to the current approximate value function $\tilde{J}(\mathbf{x}; \mathbf{r})$ and compute the temporal difference $d_t = L(\mathbf{x}, \mathbf{u})\Delta + e^{-\alpha\Delta}\tilde{J}(\mathbf{x}_{new}; \mathbf{r}) - \tilde{J}(\mathbf{x}, \mathbf{r})$ and update $\mathbf{Z} := \mathbf{Z} + e^{-\alpha t} d_t \phi(\mathbf{x})$.
Normalize $\mathbf{Z} = (1 - e^{-\alpha\Delta})\mathbf{Z}$.
Update $\mathbf{Z}_{cum} = \lambda\mathbf{Z}_{cum} + \mathbf{Z}$ and $\mathbf{r} := \mathbf{r} + \gamma\mathbf{Z}_{cum}$.
Test the performance of $\tilde{J}(\mathbf{x}; \mathbf{r})$. If the performance satisfies certain criterion, terminate the loop; otherwise, set $i := i + 1$.
**end for**
**return** $\mathbf{r}$.

**Adaptive Basis Function Selection Algorithm**

Adaptive basis function selection algorithm is described in Algorithm 3. In this algorithm, we treat parameters $\epsilon$, $\eta$ and $\theta$ as fixed, though they can be tuned in the SRLP step. Tuning these parameters can potentially improve algorithm performance, though it also consumes more time. Finally, we observe that at each step of Algorithm 3, the adaptive basis function selection iterates over all of the candidate basis functions to choose the optimal basis function to add to $\mathcal{F}$, which can be time consuming. However, similar to the constraint sampling algorithm, parallel computation can significantly accelerate this algorithm.

**Bootstrapping Algorithm**

The bootstrapping algorithm is described in Algorithm 4.

**Algorithm 3** Adaptive Basis Function Selection

**Require:** Set of sampled states/actions $\mathcal{M}$, set of candidate basis functions $\{\phi_0 = 1, \phi_1, \cdots, \phi_L\}$, parameters $\epsilon$, $\eta$, $\theta$ for SRLP, time step $\Delta$, horizon $T$ and initial state $\mathbf{x}_0$ for reservoir simulation.
**Ensure:** Return a set of basis functions $\mathcal{F}$.
  Set $\mathcal{F} = \{\phi_0 = 1\}$; set evaluated profit $J^{(0)} = -\infty$ and $n = 0$.
  **repeat**
    Update $n := n + 1$ and set $J^r = -\infty$ and $l^* = -1$ $\{J^r$ means the best profit of the round.$\}$
    **for** $l = 1$ to $L$ **do**
      **if** $\mu_l \notin \mathcal{F}$ **then**
        Define $\mathcal{F}' = \mathcal{F} \bigcup \{\phi_l\}$.
        Selecting $\mathcal{F}'$ as the set of basis functions and using parameters $\epsilon$, $\eta$ and $\theta$, solve SRLP (12) to obtain the coefficients and, hence, the approximate value function $\tilde{J}$.
        **if** the coefficient of $\phi_l$ is not 0 **then**
          Run the reservoir simulator under policy greedy to $\tilde{J}$, with time step $\Delta$ until time $T$. Record the profit from the simulation as $J^e$ $\{J^e$ means the evaluated profit.$\}$
          **if** $J^e > J^r$ **then**
            Set $J^r = J^e$ and $l^* = l$.
          **end if**
        **end if**
      **end if**
    **end for**
    Set $J^{(n)} = J^r$.
    **if** $J^{(n)} > J^{(n-1)}$ **then**
      $\mathcal{F} := \mathcal{F} \bigcup \{\mu_{l^*}\}$.
    **end if**
  **until** $J^{(n)} \leq J^{(n-1)}$
  **return** $\mathcal{F}$.

**Algorithm 4** ADP Algorithm with Bootstrapping

**Require:** Baseline strategy $\mu_0$.
**Ensure:** Return a near-optimal strategy $\mu^{**}$.
  Set $k := 0$ and $J^{(0)} = -\infty$.
  **repeat**
    Sample states and state/action pairs based on (randomized) strategy $\mu_k$. Construct a large set of basis functions based on the sampled states. Set $k := k + 1$.
    Apply an ADP algorithm (either with or without adaptive basis function selection) to obtain an approximate value function $\tilde{J}_k$ and its greedy policy $\mu_k$.
    Run the reservoir simulation under the strategy $\mu_k$ and record the profit from the simulation as $J^{(k)}$.
  **until** $J^{(k)} \leq J^{(k-1)}$
  Set $\mu^{**} = \mu_{k-1}$.
  **return** $\mu^{**}$.