A Comparison of Approximate Dynamic Programming Techniques on Benchmark Energy Storage Problems: Does Anything Work?

Daniel R. Jiang, Thuy V. Pham, Warren B. Powell, Daniel F. Salas, and Warren R. Scott

***Abstract*—As more renewable, yet volatile, forms of energy like solar and wind are being incorporated into the grid, the problem of finding optimal control policies for energy storage is becoming increasingly important. These sequential decision problems are often modeled as stochastic dynamic programs, but when the state space becomes large, traditional (exact) techniques such as backward induction, policy iteration, or value iteration quickly become computationally intractable. Approximate dynamic programming (ADP) thus becomes a natural solution technique for solving these problems to near–optimality using significantly fewer computational resources. In this paper, we compare the performance of the following: various approximation architectures with approximate policy iteration (API), approximate value iteration (AVI) with structured lookup table, and direct policy search on a benchmarked energy storage problem (i.e., the optimal solution is computable).**

I. Introduction

In this paper, we investigate the effectiveness of several techniques that fall under the realm of approximate dynamic programming (ADP) on a simple energy storage and allocation problem (previously described in [1] and [2]): we seek to optimally control (profit maximization) a storage device that interacts with both the grid and an uncertain energy supply (i.e., wind) in order to meet demand. In our benchmarks, we consider a stochastic wind supply, stochastic electricity prices (from the grid), and a deterministic demand. We use this problem class because it can be simplified through discretization (and possibly dimensionality reduction) to obtain benchmark problems that can be solved optimally. The idea is to use these benchmark problems to provide insights into the performance of a variety of ADP strategies (for an overview of traditional methods in ADP, see e.g., [3], [4], [5]). A precise formulation of this problem is given in Section III.

Algorithmically, we consider solution techniques that are variants of approximate policy iteration (API) and approximate value iteration (AVI). The basis for both of these algorithms is a value function approximation (VFA) (the value function is also known as the cost–to–go function) and thus, by altering the approximation architecture, we arrive at a family of ADP algorithms.

For API, we test several methods typically found in the machine learning (ML) literature to approximate the value function: support vector regression (SVR), Gaussian process regression (GPR), local parametric methods (LPR), and a clustering method called Dirichlet cloud with radial basis functions (DCR). In the case of AVI, we consider lookup table techniques that exploit the structural properties of the problem at hand: monotonicity (the use of the natural concavity in this problem was studied previously in [2]). Although lookup table itself can be a very limited method, we find that the additional knowledge of problem structure makes it an extremely effective solution method, even when compared to more advanced statistical estimation methods.

This paper reports on the performance of a variety of approximation methods that have been developed in the approximate dynamic programming community, tested using a series of optimal benchmark problems drawn from a relatively simple energy storage application. These suggest that methods based on Bellman error minimization, using both approximate value iteration and approximate policy iteration, work surprisingly poorly if we use approximation methods drawn from machine learning. Pure table lookup also works poorly. By contrast, a simple cost function approximation estimated using policy search works remarkable well, hinting that the problem is not the approximation architecture (though this method does not scale to more complex policies). In addition, lookup table methods that exploit convexity or monotonicity (if applicable) work extremely well, but do not scale to complex state–of–the–world variables. The implications for many current ADP algorithms are not encouraging, which signals the need for further work in this area.

The paper is organized as follows. In Section II, we give a brief literature review. Section III provides the mathematical formulation for the problem and discusses its inherent structure. Next, Sections IV–V give an overview of the algorithmic techniques that we employ, followed by numerical work (including previous work) in Sections VI–X. We conclude in Section XI.

II. Literature Review

The problem of energy storage, and its closely related problems in inventory and asset management, has been widely studied. For example, in [6], the authors derive, under an assumption on the distribution of wind energy, an analytical solution to an energy commitment problem in the presence of storage. The mathematical formulation is similar regardless of the exact application; [7] and [8], for example, present different techniques (including optimal switching and ADP) to study control policies of natural gas storage facilities. Moreover, [9] and [10] study the optimization of a hydro–electric reservoir, with the additional complication of bidding day–ahead. The second paper, [10], uses a method based on

stochastic dual dynamic programming (SDDP). SDDP and its related methods use Benders cuts, but the theoretical work in this area uses the assumption that random variables only have a finite set of outcomes [11] (and thus difficult to scale to larger problems).

Taking a slightly different point of view, [12] considers the capacity value of energy storage by solving a dynamic program. Broader works include [13], [14], [15], and [16], all of which solve related problems that involve storage and an generic asset or commodity.

Simple, scalar storage (or inventory) problems can be easily solved using backward dynamic programming (see [17]), but these methods quickly become intractable as we add additional state–of–the–world variables, leading us to consider the use of approximate dynamic programming. [1] uses approximate policy iteration with parametric linear model (i.e., basis functions), least–squares temporal difference (LSTD), and Bellman error minimization to solve the same energy allocation problem that we consider here. [18] takes an alternative approach to the policy evaluation step and uses neural networks (in this paper, we use nonparametric models). [2] uses the natural concavity of the value functions to speed up the convergence of a TD(1) algorithm (see [19]). [16] takes a similar approach of exploiting concavity for a generic problem with a scalar resource, but within an approximate value iteration framework. Moreover, [20] considers a simple storage problem motivated by mutual fund management and solves it using a lookup table approach exploiting concavity. Also taking advantage of structure, [21] exploits the monotonicity in the value functions in a lookup table approach to solving an optimal bidding and storage problem. In both the cases of [20] and [21], pure lookup table without structure does not work in practice within reasonable time constraints. [7] solves the natural gas storage control problem through the discretization of a continuous time model and applying a basis function approximation of the value function. One of few works to consider a nonparametric approximation of the value function, [15] employs Dirichlet process mixture models to cluster states and then uses a convex model within each cluster.

As can be seen from the literature, it is generally the case that a specific algorithm is applied to a specific application. The contribution of this paper is to empirically compare the effectiveness of several popular ADP methods on *common set* of problems derived from a energy storage application.

III. Mathematical Formulation

We now formulate the energy storage and allocation problem as a Markov decision process (MDP). Let $t \in \mathbb{N}$ be a discrete time index representing the decision epochs of the MDP (in this problem, t could be measured in hours or days). Over a finite horizon from $t = 0$ to $t = T$, our goal is to find a policy that maximizes expected profits. Let $R_t \in \mathscr{R} = [0, R_{\max}]$ be the level of energy in storage at time t, that has charge and discharge efficiencies denoted by β^c and β^d, respectively, with both β^c and β^d in $(0, 1)$. Also, let γ^c and γ^d be the maximum amount of energy that can be charged or discharged, respectively, from the storage device. For example, suppose that our storage device is a 1 MW battery (meaning that it can charge and discharge at a rate of 1 MW) and we make allocation decisions every hour. In this case, we have that $\gamma^c = \gamma^d = 1$ MWh.

Let E_t be the amount of energy available from wind at time t and P_t be the spot price of electricity. Finally, suppose D_t is the amount of demand that must be satisfied at time t. To allow for different models (either deterministic or stochastic), we also define E_t^S, P_t^S, and D_t^S to be the state variables associated with the respective processes at time t. As an example, if E_t is modeled as a Markov process, then $E_t^S = E_t$ and if D_t is modeled as a deterministic process, then $D_t^S = \{\}$. Hence, the *state variable* for the problem is $S_t = (R_t, E_t^S, P_t^S, D_t^S)$. To abbreviate, let $W_t = (E_t^S, P_t^S, D_t^S) \in \mathscr{W}$ and $S_t = (R_t, W_t)$. Throughout this paper, we operate under the assumption that the process W_t is independent of R_t. Next, we define the *exogenous information*, $\hat{W}_{t+1}$, to be the change in W_t:

$$W_{t+1} = W_t + \hat{W}_{t+1},$$

which of course is model dependent (the specific processes we use for benchmarking are defined in Section VI).

The decision problem is that, while anticipating the future value of storage, we must combine energy from the following three sources in order to fully satisfy the demand:

1) energy currently in storage, constrained by γ^c, γ^d, and R_t (represented by a decision x_t^{rd});
2) newly available wind energy, constrained by E_t (represented by a decision x_t^{wd});
3) and energy from the grid, at a spot price of P_t (represented by a decision x_t^{gd}).

Additional allocation decisions are x_t^{wr}, amount of wind energy to store; x_t^{rg}, the amount of energy to sell to the grid at price P_t; and x_t^{gr}, the amount of energy to buy from the grid and store. These allocation decisions are summarized by the six–dimensional, nonnegative decision vector,

$$x_t = (x_t^{\text{wd}}, x_t^{\text{gd}}, x_t^{\text{rd}}, x_t^{\text{wr}}, x_t^{\text{gr}}, x_t^{\text{rg}})^{\mathsf{T}} \geq 0, \tag{1}$$

and the constraints are as follows:

$$x_t^{\text{wd}} + \beta^d x_t^{\text{rd}} + x_t^{\text{gd}} = D_t, \tag{2}$$

$$x_t^{\text{rd}} + x_t^{\text{rg}} \leq R_t, \tag{3}$$

$$x_t^{\text{wr}} + x_t^{\text{gr}} \leq R_{\max} - R_t, \tag{4}$$

$$x_t^{\text{wr}} + x_t^{\text{wd}} \leq E_t, \tag{5}$$

$$x_t^{\text{wr}} + x_t^{\text{gr}} \leq \gamma^c, \tag{6}$$

$$x_t^{\text{rd}} + x_t^{\text{rg}} \leq \gamma^d. \tag{7}$$

The first constraint guarantees that demand is fully satisfied; (3) and (4) are storage capacity constraints; (5) states that the maximum amount of energy used from wind is bounded by E_t; and finally, (6) and (7) constrain the decisions to within the storage transfer rates. Let us denote the feasible set, determined by the constraints (1)–(7), by $\mathcal{X}(S_t)$. Suppressing the dependence on S_t for ease of notation, define $\mathcal{X}_t = \mathcal{X}(S_t)$. See Figure 1 below for an illustrative summary of the problem, annotated with the components of x_t.

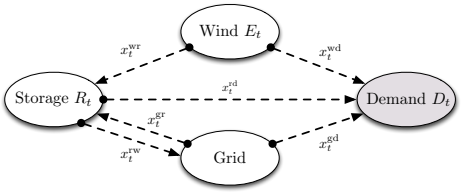

Fig. 1. Illustration of the Energy Storage/Allocation Problem

Let $\phi = (0, 0, -1, \beta^c, \beta^c, -1)^\mathsf{T}$ be a vector containing the flow coefficients of a decision x_t with respect to the storage device. Then, the *transition function* is:

$$R_{t+1} = R_t + \phi^\mathsf{T} x_t. \tag{8}$$

Note that there is no dependence on any random information, allowing us to easily take advantage of the *post–decision* formulation of this problem, to be made clear below.

Now, we define the *contribution function*. For a given state S_t and decision x_t, we define:

$$C(S_t, x_t) = P_t \cdot (D_t + \beta^d x_t^{\text{rg}} - x_t^{\text{gr}} - x_t^{\text{gd}}),$$

the profit realized at time t (we get paid for satisfying demand and for selling to the grid, but we must pay for any energy that originates from the grid). Using Bellman's optimality equation, we define value functions through the following set of recursive equations. Let $V_T^*(S_T) = 0$ and for $t \leq T-1$,

$$V_t^*(S_t) = \max_{x_t \in \mathcal{X}_t} \big[C(S_t, x_t) + \mathbf{E}(V_{t+1}^*(S_{t+1})|S_t)\big], \tag{9}$$

where it is understood that S_{t+1} depends on both S_t and x_t. For simulation and computational purposes, it is often troublesome to deal with an expectation operator within a maximum operator. As described in detail in [3], this can be remedied by appealing to the post–decision formulation of Bellman's equation. Essentially, the post–decision state $S_t^x \in \mathscr{S}$ is the state immediately after the decision x_t is made but before any new information has arrived, where $\mathscr{S}$ is the post–decision state space. The canonical form of the post–decision state is simply $S_t^x = (S_t, x_t)$, but oftentimes, it can be written in a more condensed way. Mathematically, it must be the case that $S_{t+1}|S_t^x \overset{d}{=} S_{t+1}|S_t, x_t$ (equal in distribution). Let $R_t^x = R_{t+1}$ as defined in (8); for our problem, due to the fact that R_{t+1} depends solely on R_t and x_t, the post–decision state is given by:

$$S_t^x = (R_t^x, E_t^S, P_t^S, D_t^S) = (R_t^x, W_t).$$

We define the post–decision value function as $V_t^x(S_t^x) = \mathbf{E}(V_{t+1}^*(S_{t+1})|S_t^x)$, which gives us the following two relations:

$$V_t^*(S_t) = \max_{x_t \in \mathcal{X}_t} \big[C(S_t, x_t) + V_t^x(S_t^x)\big] \tag{10}$$

and

$$V_{t-1}^x(S_{t-1}^x) = \mathbf{E}\Big[\max_{x_t \in \mathcal{X}_t} \big[C(S_t, x_t) + V_t^x(S_t^x)\big] \,\Big|\, S_{t-1}^x\Big]. \tag{11}$$

Equation (10) is useful for simulating a policy induced by a set of value functions and equation (11) is used in the simulation steps of the various ADP algorithms.

In [2], the concavity of the post–decision value functions is exploited as the VFAs are learned by the ADP algorithm. For this paper, in addition to the API variants, we consider for comparison a more recent algorithm that takes advantage of monotonicity, called *Monotone–ADP* (see [22]). To do so, we give the following proposition.

Proposition 1. *For each time $t \leq T-1$, the post–decision value function $V_t^x(R_t^x, W_t)$ is nondecreasing in R_t^x.*

Proof. We proceed by induction. Since $V_T^*(S_T) = 0$, it is clear that $V_{T-1}^x(S_{T-1}^x) = 0$ by definition and hence satisfies monotonicity. Assume that $V_t^x(S_t^x)$ satisfies the monotonicity property (induction hypothesis) and consider (11). At time $t-1$, fix two states $S_{t-1}^x = (R_{t-1}^x, W_{t-1})$ and $\bar{S}_{t-1}^x = (\bar{R}_{t-1}^x, W_{t-1})$, with both $R_{t-1}^x, \bar{R}_{t-1}^x \in \mathscr{R}$, such that $R_{t-1}^x < \bar{R}_{t-1}^x$. Let $\epsilon = \bar{R}_{t-1}^x - R_{t-1}^x$.

Denote $S_t = (R_t, W_t) = (R_{t-1}^x, W_t)$ and $\bar{S}_t = (\bar{R}_t, W_t) = (\bar{R}_{t-1}^x, W_t)$, with S_t^x and $\bar{S}_t^x$ be the corresponding post–decision states. As before, let $\mathcal{X}_t = \mathcal{X}(S_t)$, but also let $\bar{\mathcal{X}}_t = \mathcal{X}(\bar{S}_t)$. We aim to show that the following inequality holds for any outcome of $W_t|W_{t-1}$ (note that $W_t|S_{t-1}^x \overset{d}{=} W_t|\bar{S}_{t-1}^x \overset{d}{=} W_t|W_{t-1}$, so the distribution of the exogenous information is the same in both situations):

$$\begin{aligned}\max_{x_t \in \mathcal{X}_t} \big[C(S_t, x_t) + V_t^x(S_t^x)\big] \\ \leq \max_{x_t \in \bar{\mathcal{X}}_t} \big[C(\bar{S}_t, x_t) + V_t^x(\bar{S}_t^x)\big].\end{aligned}$$

Note the differing feasible sets $\mathcal{X}$ and $\bar{\mathcal{X}}_t$. Denote the optimal solution to the left hand side of the inequality as x_t and the optimal value of the objective as F. Now, there are two cases:

1) $x_t \in \bar{\mathcal{X}}_t$. Using this same decision on the right hand side as well, we see that since $R_t < \bar{R}_t$, we have $R_t^x < \bar{R}_t^x$. Using $C(S_t, x_t) = C(\bar{S}_t, x_t)$ and the induction hypothesis, we conclude that $C(\bar{S}_t, x_t) + V_t^x(\bar{S}_t^x) \geq F$. Since there exists a feasible solution, namely x_t, in the new decision space $\bar{\mathcal{X}}_t$ that achieves the objective value greater than or equal to F, the inequality is verified.
2) $x_t \notin \bar{\mathcal{X}}_t$. To get from $\mathcal{X}_t$ to $\bar{\mathcal{X}}_t$, constraint (3) is relaxed by ϵ and constraint (4) is tightened by ϵ. Therefore, it must be the case that constraint (4) is violated by x_t:

$$\begin{aligned}x_t^{\text{wr}} + x_t^{\text{gr}} &> R_{\max} - \bar{R}_t \\ &= R_{\max} - R_t - \epsilon.\end{aligned}$$

To construct a feasible solution $\bar{x}_t \in \bar{\mathcal{X}}_t$ from x_t, let us simply decrease $x_t^{\text{wr}} + x_t^{\text{gr}}$ until (4) is satisfied. That is, choose $\bar{x}_t^{\text{wr}}$ and $\bar{x}_t^{\text{gr}}$ such that $\bar{x}_t^{\text{wr}} + \bar{x}_t^{\text{gr}} = R_{\max} - \bar{R}_t$. It is clear that:

$$(x_t^{\text{wr}} + x_t^{\text{gr}}) - (\bar{x}_t^{\text{wr}} + \bar{x}_t^{\text{gr}}) \leq \epsilon,$$

and thus, from the resource transition function (8), we see that $\bar{R}_t^x \geq R_t^x$. Also, $C(S_t, x_t) = C(\bar{S}_t, \bar{x}_t)$, so by the induction hypothesis, we have shown the existence of a feasible solution $\bar{x}_t$ in $\bar{X}_t$ such that $C(\bar{S}_t, \bar{x}_t) + V_t^x(\bar{S}_t^x) \geq F$, the original inequality is verified.

Because this is true for any realization of W_t, monotonicity holds in expectation as well and the proof is complete. □

Monotonicity often exists in the state–of–the–world dimensions as well, but this depends‘ on the model of the random processes used. We show how to take advantage of this structural property in Section V.

IV. Approximate Policy Iteration

Exact policy iteration involves two main steps: policy evaluation and policy improvement (see, e.g., [5]). The (exact) policy evaluation step can be completed using a matrix inversion (solving Bellman's optimality equations), but this is often intractable. One option for approximating the policy evaluation step is to apply exact value iteration for a large number of iterations, but even this is difficult for complex problems with large state spaces and impossible when the problem admits a continuous state space. In our implementation of the approximate policy evaluation step (for a finite horizon model), we take a simulation approach where we first generate observations of the value function for the fixed policy and then fitting a model to the observations.

Consider a fixed policy $\pi = (\pi_1, \pi_2, \ldots, \pi_T)$ and a general approximation architecture Q, which takes a set of samples $\mathcal{Z} = \left\{(x_i, y_i)_{i=1}^M\right\}$ (with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$) and produces a model $Q(\mathcal{Z}, \cdot)$ that maps from $\mathcal{X}$ to $\mathcal{Y}$. Figure 2 provides the precise steps taken to perform approximate policy evaluation, given π, Q, and the number of samples desired, M. The idea is that we simulate the policy π from various initial states, keeping track of both the (post–decision) states $S_t^{x,m}$ that we visit and contributions $C_t^m = C(S_t^m, x_t^m)$ that we receive. From this, we produce a set of samples $\mathcal{Z}_t$ (see Step 5 of Figure 2) that is used by Q to produce an approximation.

Approximate Policy Evaluation
(Inputs: policy π, approximation Q, sample size M)

Step 0. Set $m = 1$.
Step 1. Select an initial post–decision state $S_0^{x,m}$.
Step 2. For $t = 1, 2, \ldots, (T-1)$:

Step 2a. Sample W_t and set pre–decision state:
$$S_t^m = (R_0^{x,m}, W_t).$$

Step 2b. Apply policy to receive decision and compute contribution:
$$x_t^m = \pi_t(S_t^m);\ C_t^m = C(S_t^m, x_t^m).$$

Step 2d. Compute next post–decision state S_t^x using (8).

Step 3. Compute observations of the time dependent value function. For each t, set:
$$v_t^m = \sum_{\tau=t}^{T-1} C_\tau^m.$$

Step 4. If $m < M$, increment m and return to **Step 1**.
Step 5. Denote the set of samples by:
$$\mathcal{Z}_t = \left\{(S_t^{x,m}, v_t^m)_{m=1}^M\right\}.$$

Using the approximation model, return $\overline{V}_t^x(\cdot) = Q(\mathcal{Z}_t, \cdot)$.

Fig. 2. Approximate Policy Evaluation Step for API

With the policy evaluation step defined, we define the API algorithm by essentially replacing the exact policy evaluation step in traditional policy iteration with the approximate version. As mentioned above, the algorithm iterates the two steps of policy evaluation and improvement, shown in Figure 3.

A. Choices of Approximation Architecture Q

In this section, we give a brief introduction for each of the following approximation architectures Q (for a detailed treatment, see the corresponding literature). The motivation for choosing nonparametric estimators is that they are popular in the statistics and machine learning community, but have received relatively limited attention in the ADP and RL communities. The more traditional technique of LSTD was tested on the same problem in [1]. Assume that the notation is self–contained for each of the following sub–sections; in addition, for purposes of presentation, we have removed the subscript and superscript from the notation $V_t^x(s)$ and use $V(s)$ instead.

Approximate Policy Iteration
(Inputs: approximation Q, sample size M, iterations N)

Step 0. Set initial policy π^0; set $n = 1$.
Step 1. Use Approximate Policy Evaluation with arguments (π^{n-1}, Q, M) to compute $\overline{V}_t^{x,n-1}$ for each t.
Step 2. Policy improvement step:
$$\pi_t^n(S_t) = \arg\max_{x_t \in \mathcal{X}_t}\left[(C(S_t, x_t) + \overline{V}_t^{x,n-1}(S_t^x)\right].$$

Step 3. If $n < N$, increment n and return to **Step 1**.

Fig. 3. Approximate Policy Iteration Algorithm

Support vector regression (SVR), originally introduced in [23], is an extension of the well–known support vector machine (SVM) algorithm for classification to the problem of regression. Also see [24] for an overview of SVR and implementations. Given a linear model, $V(s) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(s) = \langle \theta_f, \phi_f(s)\rangle$, where $\mathcal{F}$ is a set of features, ϕ_f are basis functions, and θ_f are weights. Let the training dataset be represented by (s_m, y_m) for $m = 1, 2, \ldots, M$. The essential idea of SVR is to choose a hyperplane defined by the weights θ_f, so that *most* of the training pairs fall within ϵ of the hyperplane while keeping the hyperplane as "flat" or as "simple" as possible, by minimizing $\|\theta\| = \langle\theta, \theta\rangle$ (given two models that explain the training data, we prefer the simpler one that is less affected by noise in s_m). The optimization problem can be written as follows:

$$\text{minimize } \frac{1}{2}\|\theta\|^2 + \eta \cdot \sum_{m=1}^{M}\left(\xi_m + \xi_m^*\right)$$
$$\text{subject to } \begin{cases} y_m - \langle\theta, \phi(s_m)\rangle & \le \epsilon + \xi_m \\ -y_m - \langle\theta, \phi(s_m)\rangle & \le \epsilon + \xi_m^* \\ \xi_m, \xi_m^* & \ge 0. \end{cases}$$

In the numerical work, we leverage the oft–used and versatile Gaussian radial basis kernel. SVR is implemented using `svm` of the R package `e1071` (with $\lambda = 10$ and $\epsilon = 0.01$).

Gaussian process regression (GPR) is a Bayesian machine learning technique (see [25] for a thorough description) that allows us to model the unknown value function by a Gaussian process indexed by elements s of the state space $\mathscr{S}$. A Gaussian process $V \sim \mathcal{GP}(m, k)$, specified by a mean function $m(s) = \mathbf{E}[V(s)]$ and covariance function $k(s, s') = \text{Cov}[V(s), V(s')]$, is a (possibly infinite) collection of random variables such that any finite set of them is jointly Gaussian. For the prior, a typical choice of mean function is $m(s) = 0$ (note that the posterior mean is not necessarily zero). In our work, we choose k to be the Gaussian radial basis

function, $k(s,s') = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|s-s'\|^2}{2\sigma^2}\right)$, and assume we observe $y = V(s) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. The essential step in GPR is computing the posterior Gaussian process by conditioning on the observed values. We implement GPR using `gausspr` of the R package `kernlab`.

Local Polynomial Regression (LPR), or more specifically, locally weighted scatterplot smoothing (LOESS), is a nonparametric technique used for smooth functions [26]. As before, let $(\mathbf{S}, \mathbf{y})$ be the training set and suppose we are interested in estimating the value of $V(s)$. Let $\theta(s) = (V(s), V'(s), \ldots, V^{(l)}(s))$ and $U(u) = (1, u, u^2/2!, \ldots, u^l/l!)$. For any $s_i \in \mathbf{S}$ near s, the value $V(s_i)$ can be approximated using (Taylor expansion) $\theta(s)^\intercal U(s_i - s)$. The LOESS estimator for θ is defined by:

$$\hat{\theta}(s) = \underset{\theta \in \mathbb{R}^{l+1}}{\arg\min} \sum_{s_i} \Big[y_i - \theta^\intercal U(s_i - s) \Big]^2 K\left(\frac{s_i - s}{h}\right).$$

In our numerical work, we use a second–order local polynomial fit ($l = 2$). LOESS is implemented using `loess` of the R package `stats`.

Dirichlet Cloud – Radial Basis Functions (DCR) is a method, developed in [27], that performs local regressions on clusters of data. As each training point is processed, a cluster for the point is chosen and the local (low–degree) polynomial functions is updated recursively. Once again, we use the Gaussian radial basis function; using the notation from [27], let $\phi(r) = \frac{1}{\sqrt{2\pi}} \exp(-r^2/2)$. Let N_c be the total number of clusters, c_i be the centroid of the i–th cluster, and p_i be the polynomial fitted to the i–th cluster. The model can be summarized by the following equation; for a new state s:

$$V(s) = \frac{\sum_{i=1}^{N_c} p_i(s)\phi(\|s - c_i\|)}{\sum_{i=1}^{N_c} \phi(\|s - c_i\|)},$$

a weighted average of the predictions of each of the individual clusters. For a detailed description of when and how new clusters are created and the precise equations for the fitting of local polynomials, see [27]. This method is implemented in R.

As we can see, LPR and DCR are similarly motivated by local approximations, while SVR and GPR are significantly different: SVR is a more sophisticated version of the basis function technique, while GPR is a Bayesian method of modeling a function as a random process.

V. Approximate Value Iteration with Monotonicity Preservation

We now move away from API and consider another main ADP technique, approximate value iteration (AVI). The version of AVI for finite horizon problems that we consider is a forward simulation method that iteratively updates the VFA based on each new observation. A weakness of this method is that it requires a lookup table representation of the state space, something that the API methods do not require. Nevertheless, in this paper, all problems are discretized in order to compare to optimal benchmark. We present a version of the AVI that exploits the monotone structure of the problem (see Proposition 1), called Monotone–ADP (MADP) [22].

First, we define some notation. Let $\overline{V}_t^{x,n}(s)$ be the estimate of the (post–decision) value function evaluated at $s \in \mathscr{S}$ at iteration n of the algorithm. The state that the algorithm visits on iteration n and time t is denoted $S_t^{x,n}$. Also, let α_t^n be a possibly stochastic stepsize sequence, with $\alpha_t^n(s) = \alpha_t^n \cdot \mathbf{1}_{\{S_t^{x,n} = s\}}$.

Consider two states $s = (r, w) \in \mathscr{S}$ and $s' = (r', w') \in \mathscr{S}$, with $r, r' \in \mathscr{R}$ and $w, w' \in \mathscr{W}$. We say that $s \preccurlyeq s'$ if and only if $r \leq r'$ and $w = w'$ — the necessary conditions to invoke Proposition 1. Now we define the monotonicity preservation operator, Π_M (see Figure 4 for an illustration). In the following definition, suppose that v is the previous estimate of the value of a particular state s and that z_t^n is a new observation of the value of the currently visited state $S_t^{x,n}$. We define:

$$\Pi_M(S_t^{x,n}, z_t^{x,n}, s, v) = \begin{cases} z_t^{x,n} & \text{if } s = S_t^{x,n}, \\ z_t^{x,n} \vee v & \text{if } S_t^{x,n} \preccurlyeq s,\ s \neq S_t^{x,n}, \\ z_t^{x,n} \wedge v & \text{if } s \preccurlyeq S_t^{x,n},\ s \neq S_t^{x,n}, \\ v & \text{otherwise.} \end{cases}$$

The precise description of the algorithm is given in Figure 5.

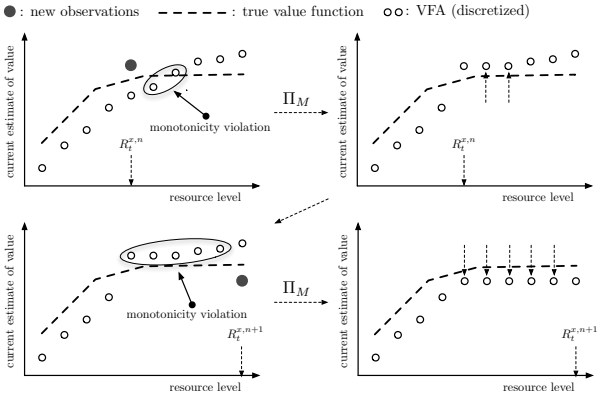

Fig. 4. Illustration of the Monotonicity Preservation Operator in the Resource Dimension (i.e., for a fixed t and fixed outcome of W_t)

We remark that Monotone–ADP is a provably convergent algorithm under certain technical conditions (see [22]). Although we do not describe the details of the convergence theory in this paper, it can be easily checked that the problem at hand, after discretization, satisfies the conditions for convergence.

VI. Benchmark Problems

The problems that we use as optimal benchmarks to the proposed algorithms originated from [2]. For all of the benchmark problems, we choose $R_{\max} = 30$, $\beta^c = \beta^d = 1$, $\gamma^c = \gamma^d = 5$, and $T = 100$. The deterministic demand is assumed to have a seasonal structure:

$$D_t = \left\lfloor \max\left\{0, 3 - 4\sin\left(\tfrac{2\pi t}{T}\right)\right\} \right\rfloor.$$

We now define two parameters that determine the support of the price processes, $P_{\min} = 30$ and $P_{\max} = 70$. Moreover, [2] defines a discrete distribution called the *pseudonormal distribution*, characterized by five parameters, μ, σ^2, a, b, and Δ. Let X be pseudonormally distributed (written $X \sim \mathcal{PN}(\mu, \sigma^2, a, b, \Delta)$). The support of X is defined to be $\mathcal{X} =$

Monotone–ADP Algorithm

Step 0a. Initialize $\overline{V}_t^{x,0}(s) = 0$ for each $t \leq T-1$ and $s \in \mathcal{S}$.
Step 0b. Set $\overline{V}_T^{x,n}(s) = 0$ for each $s \in \mathcal{S}$ and $n \leq N$.
Step 0c. Set $n = 1$.
Step 1. Select an initial state $S_0^{x,n} = (R_0^{x,n}, W_0)$.
Step 2. For $t = 0, \ldots, (T-1)$:

Step 2a. Sample S_{t+1}^n and get a noisy observation:

$$\hat{v}_t^{x,n}(S_t^{x,n}) = \max_{x_{t+1}} \{C(S_{t+1}^n, x_{t+1}) + \overline{V}_{t+1}^{x,n-1}(S_{t+1}^x)\}.$$

Step 2b. Smooth new observation with previous value:

$$z_t^{x,n}(S_t^{x,n}) = (1-\alpha_t^{n-1}(S_t^{x,n})) \cdot \overline{V}_t^{x,n-1}(S_t^{x,n}) + \alpha_t^{n-1}(S_t^{x,n}) \cdot \hat{v}_t^{x,n}(S_t^{x,n}).$$

Step 2c. Enforce monotonicity. For each $s \in \mathcal{S}$:

$$\overline{V}_t^{x,n}(s) = \Pi_M(S_t^{x,n}, z_t^{x,n}, s, \overline{V}_t^{x,n-1}(s)).$$

Step 2d. Choose the next state $S_{t+1}^{x,n}$.
Step 3. If $n < N$, increment n and return **Step 1**.

Fig. 5. Monotone–ADP Algorithm using Post–Decision States

$\{a, a+\Delta, a+2\Delta, \ldots, b-\Delta\}$ and for $x_i \in \mathcal{X}$, we have $\mathbf{P}(X = x_i) = f(x_i;\mu,\sigma^2)/\sum_{x_j \in \mathcal{X}} f(x_j;\mu,\sigma^2)$, where $f(\cdot;\mu,\sigma^2)$ is the pdf of a normal random variable with mean μ and variance σ^2. Three types of price processes are considered. Let $\epsilon_t^P \sim \mathcal{PN}(\mu_P, \sigma_P^2, -8, 8, 1)$, $\epsilon_t^J \sim \mathcal{PN}(0, 50^2, -40, 40, 1)$ (for jumps), and $u_t \sim \mathcal{U}(0,1)$ be i.i.d. random variables.

1) **Sinusoidal**.

$$P_t = \min\left\{\max\left\{40 - 10\sin\left(\frac{5\pi t}{2T}\right) + \epsilon_t, P_{\min}\right\}, P_{\max}\right\}.$$

Note that since the P_t is independent of history, $P_t^S = \{\}$.

2) **Markov chain**. Let $P_0 = P_{\min}$ and

$$P_{t+1} = \min\left\{\max\left\{P_t + \epsilon_{t+1}, P_{\min}\right\}, P_{\max}\right\}.$$

In this case, $P_t^S = P_t$.

3) **Markov chain with jumps**. Let $P_0 = P_{\min}$ and

$$P_{t+1} = \min\Big\{\max\Big\{P_t + \epsilon_{t+1} + \mathbf{1}_{\{u_{t+1} \leq p\}}\epsilon_{t+1}^J, P_{\min}\Big\}, P_{\max}\Big\}.$$

Again, $P_t^S = P_t$.

We consider a Markov chain model for the wind process, E_t. Define $E_{\min} = 1$ and $E_{\max} = 7$. The support of E_t are the values between $E_{\min}$ and $E_{\max}$ discretized at a level of a parameter ΔE. Let ϵ_t^E i.i.d. random variables that can be either uniformly or pseudonormally distributed, $\mathcal{PN}(\mu_E, \sigma_E^2, -3, 3, \Delta E)$.

$$E_{t+1} = \min\left\{\max\left\{E_t + \epsilon_{t+1}, E_{\min}\right\}, E_{\max}\right\}.$$

Lastly, suppose that R_t^x takes values between 0 and $R_{\max}$, discretized at a level ΔR.

Table I summarizes the stochastic benchmark problems; for ϵ_t^E and ϵ_t^P, since a, b, and Δ are defined the same way across all problems, we use $\mathcal{PN}(\mu, \sigma^2)$ as shorthand.

VII. Numerical Results

Due to the more complex nature of the various approximation architectures, there are more computational issues associated with the API algorithms than the AVI algorithm.

TABLE I
Parameter Choices for Stochastic Benchmark Problems [2]

Problem	ΔR	ΔE	ϵ_t^E	Price Process	ϵ_t^P
S1	0.5	0.5	$\mathcal{U}(-1,1)$	Sinusoidal	$\mathcal{PN}(0,25^2)$
S2	0.5	0.5	$\mathcal{PN}(0,0.5^2)$	Sinusoidal	$\mathcal{PN}(0,25^2)$
S3	0.5	0.5	$\mathcal{PN}(0,1.0^2)$	Sinusoidal	$\mathcal{PN}(0,25^2)$
S4	0.5	0.5	$\mathcal{PN}(0,1.5^2)$	Sinusoidal	$\mathcal{PN}(0,25^2)$
S5	1	1	$\mathcal{U}(-1,1)$	MC + Jump	$\mathcal{PN}(0,0.5^2)$
S6	1	1	$\mathcal{U}(-1,1)$	MC + Jump	$\mathcal{PN}(0,1.0^2)$
S7	1	1	$\mathcal{U}(-1,1)$	MC + Jump	$\mathcal{PN}(0,2.5^2)$
S8	1	1	$\mathcal{U}(-1,1)$	MC + Jump	$\mathcal{PN}(0,5.0^2)$
S9	1	1	$\mathcal{PN}(0,0.5^2)$	MC + Jump	$\mathcal{PN}(0,5.0^2)$
S10	1	1	$\mathcal{PN}(0,1.0^2)$	MC + Jump	$\mathcal{PN}(0,5.0^2)$
S11	1	1	$\mathcal{PN}(0,1.5^2)$	MC + Jump	$\mathcal{PN}(0,5.0^2)$
S12	1	1	$\mathcal{PN}(0,2.0^2)$	MC + Jump	$\mathcal{PN}(0,5.0^2)$
S13	1	1	$\mathcal{PN}(0,0.5^2)$	MC + Jump	$\mathcal{PN}(0,1.0^2)$
S14	1	1	$\mathcal{PN}(0,1.0^2)$	MC + Jump	$\mathcal{PN}(0,1.0^2)$
S15	1	1	$\mathcal{PN}(0,1.5^2)$	MC + Jump	$\mathcal{PN}(0,1.0^2)$
S16	1	1	$\mathcal{PN}(0,0.5^2)$	MC	$\mathcal{PN}(0,1.0^2)$
S17	1	1	$\mathcal{PN}(0,1.0^2)$	MC	$\mathcal{PN}(0,1.0^2)$

The main difficulty arises in the policy improvement step (given in **Step 2** of Figure 3 but the computational cost is actually realized in **Step 2b** of Figure 2). Due to the existence of local optima when solving **Step 2** of Figure 3, the maximization problem is solved using grid–search, a computationally expensive method. Because of these limitations, we are able to use approximately 12.5% of the state space for policy evaluation purposes and 10 policy improvement steps. On the other hand, the MADP algorithm uses matrix operations to manipulate a simple lookup table representation of states and can finish $5 \cdot 10^6$ iterations within 2 days of computation time.

For a given post–decision VFA, $\overline{V}_t^x$, we define the approximate policy as $X_t^\pi(S_t) = \arg\max_{x_t \in \mathcal{X}_t}\big[C(S_t, x_t) + \overline{V}_t^x(S_t^x)\big]$. To compute the value of the policy, we generate 1000 sample paths of the wind and price processes, and for each sample path, we follow the policy and sum the contributions. The *value of the policy* is then the average contribution over the 1000 sample paths. The percent of optimality is defined to be the value of the approximate policy divided by the value of the optimal (backward dynamic programming) policy. See [3, Section 4.94] for a detailed description of determining a policy's value.

The results are given in Figure 6. SVR and MADP generate the highest quality policies, but it is noteworthy to see that SVR does not use problem specific information while MADP does. Despite this, when considering the relative simplicity of the energy storage problem when compared to other real–world problems, results of 90% are not necessarily encouraging (GPR, LPR, and DCR often perform significantly worse than 90%). This suggests that care needs to be taken when combining API with a general purpose approximation architecture—not any approximation method will work.

VIII. Exploiting Concavity

It needs to be pointed out that neither SVR nor MADP perform at the level of the ADP algorithm of [2] (98–99% optimality, see Figure 7), which exploits the piecewise linear concave nature of the value functions; the algorithm of [2]

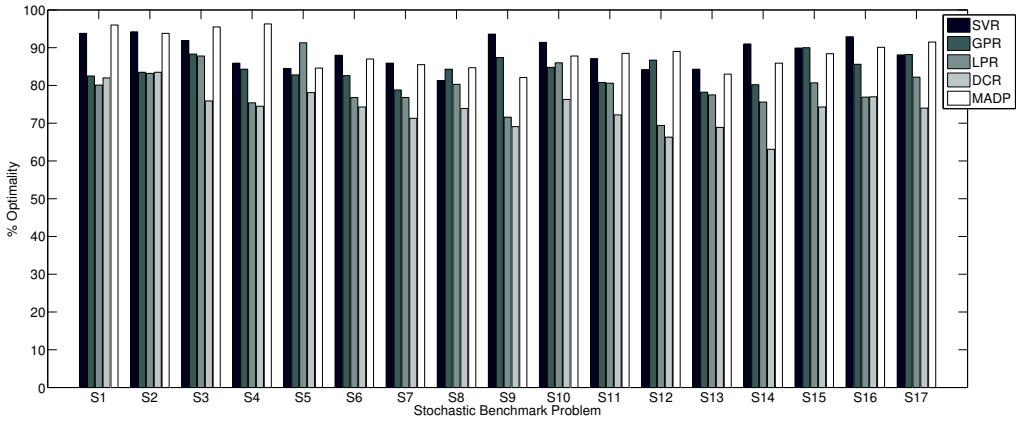

Fig. 6. Benchmark Results

also uses a specific backward pass designed with this energy storage application in mind.

Although experiments are not shown in this paper, we want to stress that, while convergence theory exists, *unstructured* lookup table with AVI does not work for any reasonably large problems (the convergence rate is far too slow to be of any practical use). [20] and [21] show the benefits of taking advantage of structure, respectively. We would also like to note that AVI can also be used with other approximations beyond lookup table (with or without structure), such as basis functions, but it is shown in [28] that there is often a lack of a fixed–point.

Our results in this paper suggest that structured, problem–specific lookup table techniques also outperform other, more general approaches, such as API paired with a generic approximation technique. At this point, the numerical results suggest that structured lookup table is consistently effective on moderately sized problems, unlike any other methods that we tested.

The caveat, of course, is that lookup table techniques do not scale to larger state spaces due to the requirement of storing a value estimate for every state. Not only that, it is typically the case that structure exists in only 1 or 2 dimensions of higher dimensional state variable (state–of–the–world variables quickly add dimensionality and there is no guarantee that they contain structure).

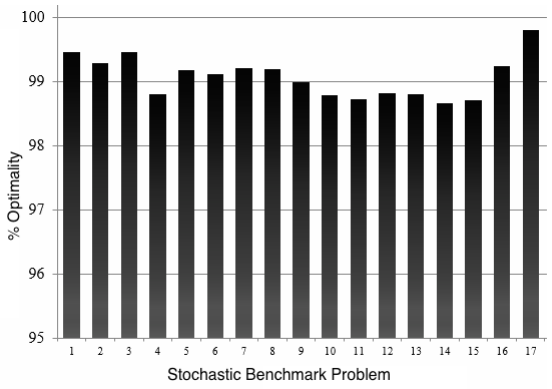

Fig. 7. Results from ADP Exploiting Concavity, [2]

IX. Direct Policy Search

In this section, we review the somewhat surprising result from [1] that direct policy search (over a low–dimensional parametrized space of policies) yields better results than API based algorithms. Several versions of API are discussed in the original paper [1]; here, we only reproduce the results for best performing version, API with instrumental variables (IVAPI). The other version considered in [1] is least squares API (LSAPI). Quadratic basis functions are used for the approximation. Direct policy search is implemented using the knowledge gradient for continuous parameters (KGCP, see [29]). The structure of the policy is:

$$X^{\pi}(S_t|\theta) = \arg\max_{x_t \in \mathcal{X}_t}[C(S_t, x_t) + \phi(S_t^x)^{\intercal}\theta],$$

where ϕ is the vector of basis functions and θ is a vector of weights (the parameter of the policy). Note that although the second term resembles a VFA, the policy search technique has no notion of minimizing the distance between $\phi(S_t^x)^{\intercal}\theta$ to $V_t^x(S_t^x)$. The reproduced results are shown below in Figure 8. Although direct policy search seems robust in this application, we emphasize that this type of direct search does not easily scale to higher dimensional parameter spaces.

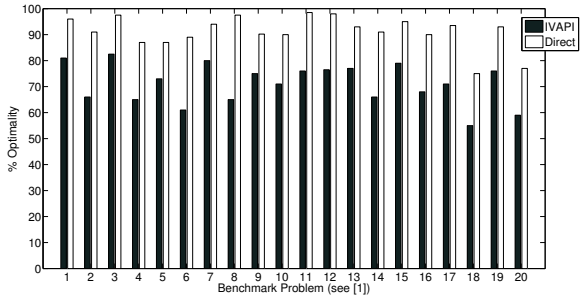

Fig. 8. Direct Policy Search vs. IVAPI, [1]

X. API Sampling Distribution

In the implementations of API discussed in this paper, the sampling distribution used for **Step 1** of the approximate policy evaluation step of Figure 2 is chosen to be a uniform distribution over the state space. One hypothesis to explain

API's relatively poor performance is that instead of sampling uniformly, we can sample from the distribution of states visited under the optimal policy (say, given a deterministic initial state, S_0^x). In most cases, this distribution is unknown; however, we are able to test this hypothesis on a simple problem with a computable optimal policy. We consider a version of our energy storage problem where the state variable is the scalar resource state, R_t^x combined with a quadratic approximation. When sampling uniformly, we consistently achieve policies that are **90%–95%** optimal, but when sampling from the optimal policy's state distribution, we obtain policies that are anywhere from **40%–70%** optimal. The primary reason that we observed for such low–quality policies is that the optimal policy visits some states with very low to zero probability, causing the quadratic approximation to be very accurate for a portion of the state space but at the same time very poor in other portions of the state space. This often leads to policy oscillations (or *chattering*, see [30] for a discussion on this issue). Besides these preliminary observations, the issue of the correct sampling distribution remains a work in progress.

XI. Conclusion

In this paper, we describe a simple finite–horizon energy storage and allocation problem that is subject to stochastic prices and wind supply, with the purpose of comparing the performance of several ADP algorithms. We consider API algorithms that take advantage of the following approximation architectures: SVR, GPR, LPR, and DCR. In addition, we test an AVI algorithm that exploits the known monotonicity of the problem, MADP. We draw the following conclusions from this and related papers:

- API performs decently well with SVR, but poorly with the other approximation architectures that we considered. However, given the simplicity of the problem, even the results from SVR are not too encouraging.
- Pure lookup table AVI performs poorly in practice, despite convergence theory (see [21], [20]).
- Structured lookup table AVI (concavity or monotonicity, but especially concavity) works extremely well, but is limited to a low–dimensional state–of–the–world variable (see [2]).
- Direct policy search also displays superior performance compared to API based methods (see [1]), but cannot scale to policies requiring a large number of parameters. In particular, direct policy search is generally not suitable for time–dependent policies.

From this, we can conclude that none of these techniques work reliably in a way that would scale to more complex problems. Therefore, we believe that new theory and methodology need to be developed in order to reliably solve real–world sequential decision problems, which are becoming increasingly difficult.

References

[1] W. Scott and W. B. Powell, "Approximate Dynamic Programming for Energy Storage with New Results on Instrumental Variables and Projected Bellman Errors," *(working paper)*, 2012.

[2] D. Salas and W. B. Powell, "Benchmarking a Scalable Approximation Dynamic Programming Algorithm for Stochastic Control of Multidimensional Energy Storage Problems," *(working paper)*, 2013.

[3] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. Wiley, 2011.

[4] F. L. Lewis and D. Vrabie, "Learning and Adaptive Dynamic Programming for Feedback Control," *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32–50, 2009.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro–Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[6] J. H. Kim and W. B. Powell, "Optimal Energy Commitments with Storage and Intermittent Supply," *Operations Research*, vol. 59, no. 6, pp. 1347–1360, 2011.

[7] R. Carmona and M. Ludkovski, "Valuation of energy storage: An optimal switching approach," *Quantitative Finance*, pp. 1–29, 2010.

[8] M. Thompson, M. Davison, and H. Rasmussen, "Natural gas storage valuation and optimization: A real options application," *Naval Research Logistics*, vol. 56, no. 3, pp. 226–238, 2009.

[9] G. Pritchard, B. Philpott, and J. Neame, "Hydroelectric reservoir optimization in a pool market," vol. 461, pp. 445–461, 2005.

[10] N. Löhndorf, D. Wozabal, and S. Minner, "Optimizing Trading Decisions for Hydro Storage Systems Using Approximate Dual Dynamic Programming," *Operations Research*, vol. 61, no. 4, pp. 810–823, 2013.

[11] A. Philpott and Z. Guan, "On the convergence of stochastic dual dynamic programming and related methods," *Operations Research Letters*, vol. 36, no. 4, pp. 450–455, 2008.

[12] R. Sioshansi, S. H. Madaeni, and P. Denholm, "A Dynamic Programming Approach to Estimate the Capacity Value of Energy Storage," *IEEE Transactions on Power Systems*, vol. 29, no. 1, pp. 395–403, 2013.

[13] J. M. Nascimento and W. B. Powell, "An Optimal Approximate Dynamic Programming Algorithm for the Lagged Asset Acquisition Problem," *Mathematics of Operations Research*, vol. 34, no. 1, pp. 210–237, 2009.

[14] N. Secomandi, "Optimal Commodity Trading with a Capacitated Storage Asset," *Management Science*, vol. 56, no. 3, pp. 449–467, 2010.

[15] L. Hannah and D. Dunson, "Approximate dynamic programming for storage problems," *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[16] J. M. Nascimento and W. B. Powell, "An Optimal Approximate Dynamic Programming Algorithm for Concave, Scalar Storage Problems With Vector-Valued Controls," *IEEE Transactions on Automatic Control*, vol. 58, no. 12, pp. 2995–3010, 2013.

[17] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 1994.

[18] D. Liu and Q. Wei, "Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems for Discrete-Time Nonlinear Systems," vol. 25, no. 3, pp. 621–634, 2014.

[19] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press, 1998.

[20] J. M. Nascimento and W. B. Powell, "Dynamic Programming Models and Algorithms for the Mutual Fund Cash Balance Problem," *Management Science*, vol. 56, no. 5, pp. 801–815, 2010.

[21] D. R. Jiang and W. B. Powell, "Optimal hour-ahead bidding in the real-time electricity market with battery storage using approximate dynamic programming," *arXiv preprint arXiv:1402.3575*, 2014.

[22] ——, "An Approximate Dynamic Programming Algorithm for Monotone Value Functions," *arXiv preprint arXiv:1401.1590*, 2014.

[23] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," *Advances in neural information processing systems*, no. 9, pp. 155–161, 1997.

[24] A. J. Smola and B. Scholkopf, "A Tutorial on Support Vector Regression," 2003.

[25] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[26] W. Cleveland and S. Devlin, "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting," *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 596–610, 1988.

[27] A. A. Jamshidi and W. B. Powell, "A recursive local polynomial approximation method using Dirichlet clouds and radial basis functions," *(working paper)*, 2013.

[28] D. De Farias and B. Van Roy, "On the Existence of Fixed Points for Approximate Value Iteration and Temporal-Difference Learning," *Journal of Optimization theory and Applications*, vol. 105, no. 3, pp. 589–608, 2000.

[29] W. Scott, P. I. Frazier, and W. B. Powell, "The Correlated Knowledge Gradient for Simulation Optimization of Continuous Parameters using Gaussian Process Regression," *SIAM Journal on Optimization*, vol. 21, no. 3, p. 996, 2011.

[30] D. P. Bertsekas, "Approximate Policy Iteration : A Survey and Some New Methods," *Journal of Control Theory and Applications*, no. June, 2011.