1. Introduction

Sequential decision problems are an important concept in many fields, including operations research, economics, and finance. For a small, tractable problem, the backward dynamic programming (BDP) algorithm can be used to compute the optimal value functions, from which we get an optimal decision making policy (Puterman 1994). However, the state space for many real-world applications can be immense, making this algorithm very computationally intensive. Hence, we often must turn to the field of approximate dynamic programming, which seeks to solve these problems via approximation techniques. One way to obtain a better approximation is to exploit structural properties (which are known but problem-dependent) of the optimal value functions,
and doing so often accelerates the convergence of ADP algorithms. In this paper, we focus on the case where the optimal value functions are monotone in one or more dimensions.

The following list is a small sample of real-world applications spanning the aforementioned disciplines that satisfy the special property of monotone value functions.

- The value functions from the problem of maximizing revenue using battery storage while bidding hourly in the electricity market can be shown to satisfy monotonicity in the resource, bid, and remaining battery lifetime (see Section 7).

- The problem of mutual fund cash balancing, described in Nascimento and Powell (2010), is faced by fund managers who must decide on the amount of cash to hold, taking into account various market characteristics and investor demand. The value functions turn out to be monotone in the interest rate and the portfolio’s rate of return.

- A classical model of search unemployment in economics describes a situation where at each period, a worker has a decision of accepting a wage offer or continuing to search for employment. The resulting value functions can be shown to be increasing with wage (see Section 10.7 of Stockey and Lucas, Jr. (1989) and McCall (1970)).

- Dynamic portfolio choice/allocation can be formulated as an MDP so that the value functions are monotone in the current wealth (see Binsbergen and Brandt (2007)).

- The problem of batch servicing of customers at a service station as discussed in Papadaki and Powell (2002) features value functions that are monotone in the number of customers. Similarly, the related problem of multiproduct batch dispatch studied in Papadaki and Powell (2003b) can be shown have monotone value functions in the multidimensional state variable that contains the number of products awaiting dispatch, for each product class.

This paper makes the following contributions. We describe and prove the convergence of an algorithm, called Monotone-ADP (M-ADP) for learning monotone value functions by preserving monotonicity after each update. We also provide empirical results for the algorithm in the context of a problem related to battery storage and bidding in the electricity market as experimental evidence that exploiting monotonicity dramatically improves the rate of convergence.

Section 2 gives a literature review, followed by the problem formulation and algorithm description in Section 3 and Section 4. Next, Section 5 provides the assumptions necessary for convergence and Section 6 states and proves the convergence theorem, with several proofs of lemmas and propositions postponed until the Appendix. Section 7 describes an example application with empirical results. We conclude in Section 8.
2. Literature Review

General monotone functions (not necessarily a value function) have been extensively studied in the academic literature. The statistical estimation of monotone functions is known as isotonic or monotone regression and has been studied as early as 1955; see Ayer et al. (1955) or Brunk (1955). The main idea of isotonic regression is to minimize a weighted error under the constraint of monotonicity (see Barlow et al. (1972) for a thorough description). The problem can be solved in a variety of ways, including the Pool Adjacent Violators Algorithm (PAVA) described in Ayer et al. (1955). More recently, Mammen (1991) built upon this previous research by describing an estimator that combines kernel regression and PAVA to produce a smooth regression function. Additional studies from the statistics literature include: Mukerjee (1988), Ramsay (1998), Dette et al. (2006). Although these approaches are outside the context of dynamic programming, the fact that they were developed and well–studied highlights the pertinence of monotone functions.

From the operations research literature, monotone value functions and conditions for monotone optimal policies are broadly described in Puterman (1994) [Section 4.7] and some general theory is derived therein. Similar discussions of the topic can be found in Ross (1983), Stockey and Lucas, Jr. (1989), Müller (1997), and Smith and McCardle (2002). The algorithm that we describe in this paper was first used in Papadaki and Powell (2002) as a heuristic to solve the stochastic batch service problem, where the value functions can be shown to be monotone. However, the convergence of the algorithm was not analyzed and the state variable was considered to be a scalar. Finally, in Papadaki and Powell (2003a), the authors prove the convergence of the Discrete On–Line Monotone Estimation (DOME) algorithm, which takes advantage of a monotonicity preserving step to iteratively estimate a discrete monotone function. DOME, though, was not designed for dynamic programming and the proof of convergence requires independent observations across iterations, which is an assumption that cannot be made for Monotone–ADP.

Another common property of value functions, especially in resource allocation problems, is convexity/concavity. Rather than using a monotonicity preserving step as Monotone–ADP does, algorithms such as the Successive Projective Approximation Routine (SPAR) of Powell et al. (2004), the Lagged Acquisition ADP Algorithm of Nascimento and Powell (2009), and the Leveling Algorithm of Topaloglu and Powell (2003) use a concavity preserving step, which is the same as maintaining monotonicity in the slopes. The proof of convergence for our algorithm, Monotone–ADP, is a modified version of ideas found in Bertsekas and Tsitsiklis (1996) and Nascimento and Powell (2009).
3. Mathematical Formulation

We consider a generic problem with a finite time–horizon, \( t = 0, 1, 2, \ldots, T \). Let \( S \) be the state space under consideration, where \(|S| < \infty\), and let \( A \) be the set of actions or decisions available at each time step. Let \( S_t \in S \) be the random variable representing the state at time \( t \) and \( a_t \in A \) be the action taken at time \( t \). For a state \( S_t \in S \) and an action \( a_t \in A \), let \( C_t(S_t, a_t) \) be a contribution or reward received in period \( t \). Let \( A^\pi : S \rightarrow A \) be the decision function for a policy \( \pi \) from the class \( \Pi \) of all admissible policies. Our goal is to maximize the expected total contribution, giving us the following objective function:

\[
\sup_{\pi \in \Pi} E \left[ \sum_{t=0}^{T} C_t(S_t, A^\pi(S_t)) \right],
\]

where we seek a policy to choose the actions \( a_t \) sequentially based on the states \( S_t \) that we visit. For \( t \in \mathbb{N} \), let \( W_t \in W \) be a discrete time stochastic process that represents the exogenous information in our problem. With this, we can explicitly define the state transition function \( S^{M} \):

\[
S_{t+1} = S^{M}(S_t, a_t, W_{t+1}).
\]

The optimal policy can be expressed through a set of optimal value functions using the well–known Bellman’s equation:

\[
V^*_t(S_t) = \sup_{a_t \in A} \left\{ C_t(S_t, a_t) + E(V^*_{t+1}(S_{t+1})|S_t) \right\} \text{ for } t = 0, 1, 2, \ldots, T - 1,
V^*_T(S_T) = 0,
\]

with the understanding that \( S_{t+1} \) transitions from \( S_t \) according to (2) and thus, the second term within the supremum depends on the action \( a_t \) as well (see Powell (2011) for a detailed discussion regarding the model). Also, for simplicity, we assume that the terminal contribution is zero for every state.

Suppose that a state \( S_t \in S \) can be written in the form \( S_t = (M_t, I_t) \) with \( M_t \in \mathcal{M} \) and \( I_t \in \mathcal{I} \), where \(|\mathcal{M}| < \infty \) and \(|\mathcal{I}| < \infty \). The dimensions of the state \( S_t \) are separated into two groups; the dimensions that satisfy a monotonicity property for the optimal value functions are placed in \( \mathcal{M} \), while the remaining dimensions are placed in \( \mathcal{I} \). Thus, because \( \mathcal{I} \) is finite, we have a set of monotone functions, \( V^*_t(\cdot, i) \), defined over \( \mathcal{M} \) and indexed by the variables \( i \in \mathcal{I} \) and \( t \leq T \). Suppose the following monotonicity property holds. For each \( t \leq T \), \( m_1, m_2 \in M \) with \( m_1 \leq m_2 \), \( i \in \mathcal{I} \), and \( a \in A \),

\[
C_t(m_1, i, a) \leq C_t(m_2, i, a).
\]

In addition, for each \( t \leq T \), \( m_1, m_2 \in M \) with \( m_1 \leq m_2 \) and \( i \in \mathcal{I} \),

\[
V^*_t(m_1, i) \leq V^*_t(m_2, i).
\]
In other words, $V^*_t(\cdot, i)$ is a discrete nondecreasing function for every $i$. We remark that the algorithm (described below) can be easily altered to handle nonincreasing value functions as well.

We note that it is not necessary for the set $\mathcal{M}$ to admit a total ordering and any binary relation, denoted $\leq$, can be chosen. For example, when $\mathcal{M}$ is multidimensional and componentwise inequalities are used to determine if a monotonicity relationship exists between two given states, observing one state does not always provide information regarding the other. For many problems, $\mathcal{M}$ is one dimensional, so potentially, observing a single $m \in \mathcal{M}$ can provide information regarding every $m' \in \mathcal{M}$.

The following proposition provides a simple condition that can be used to verify monotonicity in the value functions.

**Proposition 1.** Let the current state be $S_t = (M_t, I_t)$. Suppose the following assumptions hold.

(i) For every $m_1, m_2 \in \mathcal{M}$ (the portion of the state space that satisfies monotonicity) with $m_1 \leq m_2$, $i \in I$ (the portion of the state space that does not satisfy monotonicity), $a \in A$, and $w \in W$, the state transition function satisfies

\[
(m_1', i') = S^M((m_1, i), a, w),
\]

\[
(m_2', i') = S^M((m_2, i), a, w),
\]

where $m_1' \leq m_2'$ and $i' \in I$.

(ii) For each $t \leq T$, $m_1, m_2 \in \mathcal{M}$ with $m_1 \leq m_2$, $i \in I$, and $a \in A$,

\[
C_t(m_1, i, a) \leq C_t(m_2, i, a).
\]

(iii) For each $t \leq T$, $M_t$ and $W_{t+1}$ are independent.

(iv) $A$ is compact.

Then, the value functions $V^*_t$ satisfy the monotonicity property of (5).

**Proof:** See Appendix.

There are other similar ways to check for monotonicity; for example, see Proposition 4.7.3 of Puterman (1994) or Theorem 9.11 of Stockey and Lucas, Jr. (1989) for conditions on the transition probabilities. We choose to provide the above proposition due to its relevance to our example application to be described later.

Now we introduce some additional useful notation. Suppose we have an arbitrary (not necessarily optimal) set of value functions $V_t(\cdot)$ defined over $\mathcal{S}$. Since $|\mathcal{S}| < \infty$, we can collect values of the $(T + 1) \cdot |\mathcal{S}|$ couples of the form $(t, s)$ with $t \leq T$ and $s \in \mathcal{S}$ into a vector $V$, so that the component of $V$ at $(t, s)$ is given by $V_t(s)$. For the sake of clarity, we describe the components of the vector $V$ by supplying two arguments instead of one, but still consider $V$ as a vector. In this way, the
single vector $V$ contains all of the information stored in the value functions $V_t(\cdot, \cdot)$ for $t \leq T$. We emphasize that these steps are taken purely for notational reasons and allows us to define a single dynamic programming operator $H$, as introduced in Bertsekas and Tsitsiklis (1996), that acts on all states $s$ for every time $t$. $H$ operates on a vector $V$ to produce another vector of the same dimension, $HV$, where the specific component of $HV$ at $(t, S_t)$ is written as:

$$
(HV)_t(S_t) = \begin{cases} 
\sup_{a_t \in A} \{ C_t(S_t, a_t) + E(V_{t+1}(S_{t+1})|S_t) \} & \text{for } t = 0, 1, 2, \ldots, T-1, \\
0 & \text{for } t = T.
\end{cases}
$$

(6)

Note that the state transition function and the dependence of $S_{t+1}$ on $a_t$ and $W_{t+1}$ is omitted for succinctness. Also, notice that the values of $V_{t+1}$ used on the right-hand side is fully contained within the argument $V$. In addition, $V_T$ does not change when $H$ is applied, and in this case, $V_T(s) = 0$ for all $s \in S$. The next proposition is similar to Assumption 4.4 of Bertsekas and Tsitsiklis (1996), but we can show that these statements always hold true for our more specific problem setting.

**Lemma 1.** The following statements are true for $H$.

(i) $V \leq V' \implies HV \leq HV'$.

(ii) $V^*$ uniquely satisfies $HV^* = V^*$.

(iii) $HV - \eta e \leq H(V - \eta e) \leq H(V + \eta e) \leq HV + \eta e$, where $e$ is a vector of ones and $\eta > 0$.

*Proof:* See Appendix.

4. Algorithm

In this section, we formally describe the Monotone–ADP algorithm. Let $\overline{V}^n$ be the approximation of $V^*$ at iteration $n$ and $S^n_t = (M^n_t, I^n_t)$ be the state that is visited at time $t$ in iteration $n$. The observation of the optimal value function at time $t$, iteration $n$, and state $S^n_t$ is denoted $\hat{v}^n_t(S^n_t)$ and is calculated using the estimates of the value functions from iteration $n-1$. Before presenting the description of the ADP algorithm, we provide some definitions. We start with $\Pi_M$, the monotonicity preserving projection operator. Suppose $z^n_t$ is the observed value of a state $(M^n_t, I^n_t)$.

**Definition 1.** $\Pi_M$ considers $(M^n_t, z^n_t)$ as the “reference point” to which an arbitrary state $s = (m, I^n_t)$ with value $v$ is compared. The definition of $\Pi_M$ is as follows:

$$
\Pi_M(M^n_t, z^n_t, m, v) = \begin{cases} 
z^n_t & \text{if } m = M^n_t, \\
z^n_t \lor v & \text{if } m \geq M^n_t, m \neq M^n_t, \\
z^n_t \land v & \text{if } m \leq M^n_t, m \neq M^n_t, \\
v & \text{otherwise},
\end{cases}
$$

(7)

where, in the algorithm, the output of $\Pi_M$ becomes the updated value of the state $(m, I^n_t)$.
If $z^n_t$ is larger than the current value $v$ where $m \geq M^n_t$, then we increase $v$ to $z^n_t$ (since $z^n_t = z^n_t \lor v$) in order to satisfy the monotonicity requirement of (5). When $v$ is larger, monotonicity is satisfied and nothing changes. For $m \leq M^n_t$, a similar adjustment to $v$ is made whenever $z^n_t$ is smaller than $v$. See Figure 1 for an example showing a sequence of two observations and the resulting projections in the Cartesian plane.

![Figure 1](image-url)  
Figure 1 Example Illustrating the Projection Operator

We now introduce, for each $t$, a (possibly stochastic) stepsize sequence $\alpha^n_t \leq 1$ used for smoothing in new observations. The algorithm only directly updates values (i.e., not including updates from the projection operator) for states that are visited, so for each $s \in S$, let

$$\alpha^n_t(s) = \alpha^n_t \cdot 1_{\{s = S^n_t\}}.$$  \hfill (8)

Let $w^n_t(S^n_t)$ represent the noise associated with an observation of the value function at $S^n_t$ at time $t$ in iteration $n$:

$$w^n_t(S^n_t) = \hat{v}^n_t(S^n_t) - (HV^{n-1})_t(S^n_t).$$  \hfill (9)

**Definition 2.** Let us denote the history of the algorithm up until iteration $n$ by the filtration

$$\mathcal{F}^n = \sigma\{(S^m_t, w^m_t(S^m_t), \alpha^m_t)_{m \leq n, t \leq T}\}.$$  \hfill (10)

A precise description of the algorithm is given in Figure 2. Notice from the description that if the monotonicity property (5) is satisfied at iteration $n - 1$, then the fact that the projection operator $\Pi_M$ is applied to all elements of $\mathcal{M}$ ensures that the monotonicity property is satisfied again at time $n$. 
Step 0a. Initialize $\tilde{V}_t^0 \in [0, V_{\text{max}}]$ for each $t \leq T - 1$ such that monotonicity is satisfied within $\tilde{V}_t^0$, as described in (5).

Step 0b. Set $\tilde{V}_T^n(s) = 0$ for each $s \in S$ and $n \leq N$.

Step 0c. Set $n = 1$.

Step 1. Select an initial state $S_0^n = (M_0^n, I_0^n)$.

Step 2. For $t = 0, 1, \ldots, (T - 1)$:

Step 2a. Sample a noisy observation:
$$\hat{v}_t^n(S^n_t) = (H \tilde{V}_t^{n-1})(S^n_t) + w^n_t(S^n_t).$$

Step 2b. Smooth in the new observation with previous value:
$$z^n_t(S^n_t) = (1 - \alpha^{n-1}_t(S^n_t)) \cdot \tilde{V}_t^{n-1}(S^n_t) + \alpha^{n-1}_t(S^n_t) \cdot \hat{v}_t^n(S^n_t).$$

Step 2c. Perform monotonicity projection operator. For each $m \in M$:
$$\tilde{V}_t^n(m, I^n_t) = \Pi_M(M^n_t, z^n_t, m, V_t^{n-1}(m, I^n_t)).$$

Step 2d. Choose the next state $S^n_{t+1}$ given $F^{n-1}$.

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

Figure 2 Monotone Value Function ADP Algorithm

We close this section by making the point that a convergence proof can be obtained without maintaining monotonicity (see Bertsekas and Tsitsiklis (1996)), but the algorithms can be extremely slow, even on low dimensional problems. Maintaining monotonicity can dramatically improve the rate of convergence (see Section 7 for empirical results), but leaves open the question of whether the algorithm is still provably convergent. We establish this in the remainder of the paper.

5. Assumptions

We begin by providing some technical assumptions that are needed for convergence analysis.

Assumption 1. For all $s \in S$ and $t \leq T$,
$$\sum_{n=1}^{\infty} P(S^n_t = s | F^{n-1}) = \infty \quad a.s.$$  

By the Extended Borel–Cantelli Lemma (see Breiman (1992)), any scheme for choosing states that satisfies the above condition will visit every state infinitely often with probability one.
ASSUMPTION 2. There exists a constant $C_{\text{max}} > 0$ such that for all $s \in S$, $t \leq T$, and $a \in A$,

$$|C_t(s, a)| \leq C_{\text{max}}.$$ 

Furthermore, for all $s \in S$ and $t \leq T$, there exists $a \in A$ such that $C_t(s, a) \geq 0$.

Assumption 2 easily implies that there exists a constant $V_{\text{max}} > 0$ such that for all $s \in S$ and $t \leq T$,

$$0 \leq V_t^*(s) \leq V_{\text{max}}. \quad (11)$$

ASSUMPTION 3. Let $s_1 = (m_1, i)$ and $s_2 = (m_2, i)$ for $m_1 \leq m_2$. For $f : \mathcal{M} \times \mathcal{I} \rightarrow \mathbb{R}$ such that for each $i \in \mathcal{I}$, $f(\cdot, i)$ is a nondecreasing function over $\mathcal{M}$, suppose that

$$\mathbb{E}(f(S_{t+1})|S_t = s_1) \leq \mathbb{E}(f(S_{t+1})|S_t = s_2). \quad (12)$$

The above assumption can be satisfied either using a condition on the transition function, as in Proposition 1, or a condition on the transition probabilities. The next three assumptions are standard ones made on the observations $\hat{v}^n_t$, the noise $w^n_t$, and the stepsize sequence $\alpha^n_t$; see Bertsekas and Tsitsiklis (1996) (e.g. Assumption 4.3 and Proposition 4.6) for additional details.

ASSUMPTION 4. The observations we receive are bounded (same constant $V_{\text{max}}$):

$$0 \leq \hat{v}^n_t(S^n_t) \leq V_{\text{max}} \quad \text{a.s.}$$

ASSUMPTION 5. The following hold regarding the noise sequence $w^n_t$:

(i) $\mathbb{E}\{w^{n+1}_t(s)|\mathcal{F}^n\} = 0$,

(ii) $\mathbb{E}\{(w^{n+1}_t(s))^2|\mathcal{F}^n\} \leq A + B\|\mathbf{V}^n\|_\infty^2$.

ASSUMPTION 6. For each $t \leq T$, suppose $\alpha^n_t \in [0, 1]$ is $\mathcal{F}^n$-measurable and

(i) $\sum_{n=0}^{\infty} \alpha^n_t = \infty \quad \text{a.s.,}$

(ii) $\sum_{n=0}^{\infty} (\alpha^n_t)^2 < \infty \quad \text{a.s.}$
6. Convergence Analysis of the Monotone–ADP Algorithm

We are now ready to show the convergence of the algorithm. Note that although there is a significant similarity between this algorithm and the Discrete On–Line Monotone Estimation (DOME) algorithm described in Papadaki and Powell (2003a), but the proof technique for the DOME algorithm cannot be directly extended to our problem due to differences in the assumptions.

Instead, a modified version of the proof technique found in Bertsekas and Tsitsiklis (1996) and Nascimento and Powell (2009) is used. In the latter, the authors prove convergence of a purely exploitative ADP algorithm given concave, piecewise–linear value functions for the lagged asset acquisition problem. We cannot exploit certain properties inherent to that problem, but in our algorithm we assume exploration of all states, a requirement that can be avoided when we are able to assume concavity. Furthermore, a significant difference in this proof is that we consider the case where $\mathcal{M}$ may not be a total ordering. Specifically, we allow the case where the monotonicity property covers multiple dimensions (i.e., the relation on $\mathcal{M}$ is the componentwise inequality), which was not allowed in Nascimento and Powell (2009).

**Theorem 1.** Under Assumptions 1–6, for each $t \leq T$ and $s \in \mathcal{S}$, the estimates $\bar{V}_t^n(s)$ produced by the Monotone–ADP Algorithm of Figure 2, converge to the optimal value functions $V^*_t(s)$ almost surely.

Before providing the proof for this convergence result, we present some preliminary definitions and results. First, we define two deterministic bounding sequences, $U^k$ and $L^k$. The two sequences $U^k$ and $L^k$ can be thought of, jointly, as a sequence of “shrinking” rectangles, with $U^k$ being the upper bounds and $L^k$ being the lower bounds. The central idea to the proof is showing that the estimates $\bar{V}^n$ enter (and stay) in smaller and smaller rectangles, for a fixed $\omega \in \Omega$ (we assume that the $\omega$ does not lie in a discarded set of probability zero). We can then show that the rectangles converge to the point $V^*$, which in turn implies the convergence of $\bar{V}^n$ to the optimal value functions. This idea is attributed to Bertsekas and Tsitsiklis (1996) and is illustrated in Figure 3.

The sequences $U^k$ and $L^k$ are written recursively. Let

\[
U^0 = V^* + V_{\text{max}} \cdot e,
\]

\[
L^0 = V^* - V_{\text{max}} \cdot e,
\]

and let

\[
U^{k+1} = \frac{U^k + HU^k}{2},
\]

\[
L^{k+1} = \frac{L^k + HL^k}{2}.
\]
Lemma 2. For all $k \geq 0$, we have that
\[
HU^k \leq U^{k+1} \leq U^k,
\]
\[
HL^k \geq L^{k+1} \geq L^k.
\]
Furthermore,
\[
U^k \to V^*,
\]
\[
L^k \to V^*.
\]

Proof: The proof of this lemma is given in Bertsekas and Tsitsiklis (1996) (see Lemma 4.5 and Lemma 4.6). The properties of $H$ given in Lemma 1 are used for this result.

Lemma 3. The bounding sequences satisfy the monotonicity property; that is, given $k \geq 0$, $t \leq T$, $s_1 = (m_1, i) \in \mathcal{S}$, $s_2 = (m_2, i) \in \mathcal{S}$ such that $m_1 \leq m_2$, we have
\[
U^k_t(s_1) \leq U^k_t(s_2),
\]
\[
L^k_t(s_1) \leq L^k_t(s_2).
\]

Proof: See Appendix.

Definition 3. For a fixed $t$ and $s$, let $N^\Pi^-(t, s)$ be the last iteration for which the state $s$ was increased at time $t$:
\[
N^\Pi^-(t, s) = \max\{n : z^n_t(s) < \overline{V}_t^n(s)\}.
\]
Similarly,
\[
N^\Pi^+(t, s) = \max\{n : z^n_t(s) \geq \overline{V}_t^n(s)\}.
\]
The random variables $N^\Pi^-(t, s)$ and $N^\Pi^+(t, s)$ are not necessarily finite. Let $N^\Pi$ be large enough so that for iterations $n \geq N^\Pi$, any state increased (decreased) finitely often by the projection operator...
\( \Pi_M \) is no longer affected by \( \Pi_M \). In other words, if some state is increased (decreased) by \( \Pi_M \) on an iteration after \( N^\Pi \), then that state is increased (decreased) by \( \Pi_M \) infinitely often. We can write:

\[
N^\Pi = \max \left( \left\{ N^{\Pi-}(t,s) : t \leq T, s \in \mathcal{S}, N^{\Pi-}(t,s) < \infty \right\} \cup \left\{ N^{\Pi+}(t,s) : t \leq T, s \in \mathcal{S}, N^{\Pi+}(t,s) < \infty \right\} \right) + 1. \tag{18}
\]

We now define, for each \( t \), two random subsets \( \mathcal{S}_t^- \) and \( \mathcal{S}_t^+ \) of the state space \( \mathcal{S} \), similar to Nascimento and Powell (2009), where \( \mathcal{S}_t^- \) contains states that are increased by the projection operator \( \Pi_M \) finitely often and \( \mathcal{S}_t^+ \) contains states that are decreased by the projection operator finitely often. The role that these two sets play in the proof is as follows:

- We first show convergence for states that are projected finitely often (\( s \in \mathcal{S}_t^- \) or \( s \in \mathcal{S}_t^+ \)).
- Next, relying on the fact that convergence already holds for states that are projected finitely often, we use an induction–like argument to extend the property to states that are projected infinitely often (\( s \in \mathcal{S} \setminus \mathcal{S}_t^- \) or \( s \in \mathcal{S} \setminus \mathcal{S}_t^+ \)). This step requires the definition of a tree structure that arranges the set of states and its partial ordering in an intuitive way.

Let

\[
\mathcal{S}_t^- = \{ s \in \mathcal{S} : z^n_t(s) < \bar{V}^n_t(s) \text{ finitely often} \} = \{ s \in \mathcal{S} : z^n_t(s) \geq \bar{V}^n_t(s) \forall n \geq N^\Pi \}, \tag{19}
\]

and similarly,

\[
\mathcal{S}_t^+ = \{ s \in \mathcal{S} : z^n_t(s) > \bar{V}^n_t(s) \text{ finitely often} \} = \{ s \in \mathcal{S} : z^n_t(s) \leq \bar{V}^n_t(s) \forall n \geq N^\Pi \}. \tag{20}
\]

**Lemma 4.** For any \( \omega \in \Omega \), the random sets \( \mathcal{S}_t^- \) and \( \mathcal{S}_t^+ \) are nonempty.

**Proof:** See Appendix.

**Definition 4.** For \( s \in \mathcal{S} \), let \( \mathcal{N}_t^- (s) \) be a random set representing the iterations for which \( s \) was increased by the projection operator. Similarly, let \( \mathcal{N}_t^+ (s) \) represent the iterations for which \( s \) was decreased.

We now provide several remarks regarding the projection operator \( \Pi_M \). The value of a state \((m, i)\) can only be increased by \( \Pi_M \) if we visit a “lower” state, i.e., \( M^n_t \leq m \) and \( I^n_t = i \). This statement is obvious from the second condition of (7). Similarly, the value of the state can only be decreased by \( \Pi_M \) if the visited state is “larger,” i.e., \( M^n_t \geq m \) and \( I^n_t = i \). Intuitively, it can be useful to imagine that, in some sense, the values of states can be “pushed up” from the “left” and “pushed down” from the “right.”

Finally, due to our assumption that \( \mathcal{M} \) is only a partial ordering, the update process (from \( \Pi_M \)) becomes more difficult to analyze than in the total ordering case. To facilitate the analysis of the process, we introduce the notions of lower (upper) immediate neighbors and lower (upper) update trees.
**Definition 5.** For \( s = (m, i) \in S \), we define the set of *lower immediate neighbors* \( S_L(s) \) in the following way:

\[
S_L(s) = \{ s' = (m', i) : \begin{align*}
&m' \leq m, \\
&m' \neq m, \\
&m'' \in M, m'' \neq m, m'' \neq m', m' \leq m'' \leq m \}. 
\]

(21)

In other words, there does not exist \( m'' \) in between \( m' \) and \( m \). The set of *upper immediate neighbors* \( S_U(s) \) is defined in a similar way:

\[
S_U(s) = \{ s' = (m', i) : \begin{align*}
&m' \geq m, \\
&m' \neq m, \\
&m'' \in M, m'' \neq m, m'' \neq m', m' \geq m'' \geq m \}. 
\]

(22)

The intuition for the next lemma is that if some state \( s \) is increased by \( \Pi_M \), then it must have been caused by visiting a lower state. In particular, *either* the visited state was one of the lower immediate neighbors *or* one of the lower immediate neighbors was also increased by \( \Pi_M \). In either case, one of the lower immediate neighbors has the same value as \( s \). This lemma is crucial later in the proof.

**Lemma 5.** Suppose the value of \( s \) is increased by \( \Pi_M \) on some iteration \( n \):

\[
\overline{V}^n_t(s) > \overline{V}^{n-1}_t(s). 
\]

Then, there exists another state \( s' \in S_L(s) \) (in the set of lower immediate neighbors) whose value is equal to the newly updated value:

\[
\overline{V}^n_t(s') = \overline{V}^n_t(s). 
\]

**Proof:** See Appendix.

**Definition 6.** For a fixed \( \omega \in \Omega \), let \( s = (m, i) \in S \setminus S_t^\omega \), meaning that \( s \) is increased by \( \Pi_M \) infinitely often: \(|N^-_t(s)| = \infty \). A *lower update tree* \( T^-_t(s) \) represents a partial organization of states \( s' = (m', i) \) such that \( m' \leq m \). Suppose node \( j \) represents state \( s_j \). \( T^-_t(s) \) has the following properties:

(i) The root node of \( T^-_t(s) \) represents the state \( s \).

(ii) \( j \) does not have any child nodes if \( s_j \in S_t^- \) (\( j \) is a leaf node).

(iii) The child of a node \( j \) where \( s_j \in S \setminus S_t^- \) are precisely the set of states \( S_L(s_j) \).

The tree \( T^-_t(s) \) is unique and can easily be built by starting with the root node and successively applying the rules. The *upper update tree* \( T^+_t(s) \) is defined in a similar way.
Note that it cannot be the case for some state $s' \in \mathcal{S} \setminus \mathcal{S}_t$ with $\mathcal{S}_L(s') = \emptyset$, as for it to be increased infinitely often, there must exist at least one “lower” state whose observations cause the monotonicity violations. Because of this observation, every leaf node of $T_t^-(s)$ is an element of $\mathcal{S}_t^-$. The reason for discontinuing the tree at states in $\mathcal{S}_t^-$ is that in the convergence proof, we use an induction–like argument up the tree, starting with states in $\mathcal{S}_t^-$. Lastly, we remark that it is possible for multiple nodes to stand for the same state. Since this idiosyncrasy has no bearing on the proof, we allow it to happen, in order not to consider a more complicated construct (such as a directed graph).

As an illustrative example, consider the case with $\mathcal{M} = \{0, 1, 2\}^2$ and $\mathcal{I} = \emptyset$ where component-wise inequalities are used. Assume that for a particular sample path $\omega \in \Omega$, $s = m = (m_1, m_2) \in \mathcal{S}_t^-$ if and only if $m_1 = 0$ or $m_2 = 0$ (lower boundary of the square). See Figure 4.

The next lemma is a useful technical result used in the convergence proof.

**Lemma 6.** For any $s \in \mathcal{S}$,

$$\lim_{m \to \infty} \left[ \prod_{n=1}^{m} (1 - \alpha_t^n(s)) \right] = 0 \quad \text{a.s.}$$

**Proof:** See Appendix.

With these preliminaries in mind (other elements will be defined as they arise), we begin the convergence analysis.

**Proof of Theorem 1:** As previously mentioned, to show that the sequence $V_t^n(s)$ (almost surely) converges to $V_t^*(s)$ for each $t$ and $s$, we need to argue that $V_t^n(s)$ eventually enters every rectangle (or “interval,” when we discuss a specific component of the vector $V^n$), defined by the sequence
$L_k^t(s)$ and $U_k^t(s)$. Recall that the estimates of the value function produced by the algorithm are indexed by $n$ and the bounding rectangles are indexed by $k$. Hence, we aim to show that for each $k$, we have that for $n$ sufficiently large, it is true that $\forall s \in S$,

$$L_k^t(s) \leq \overline{V}_t^n(s) \leq U_k^t(s).$$  \hfill (23)

Following this step, an application of (16) in Lemma 2 completes the proof. We show the second inequality of (23) and remark that the first can be shown in a completely symmetric way. The goal is then to show that $\exists N_k^t < \infty$ a.s. such that $\forall n \geq N_k^t$ and $\forall s \in S$,

$$\overline{V}_t^n(s) \leq U_k^t(s).$$  \hfill (24)

Choose $\omega \in \Omega$. For ease of presentation, the dependence of the random variables on $\omega$ is omitted. We use backward induction on $t$ to show this result, which is the same technique used in Nascimento and Powell (2009). The inductive step is broken up into two cases, $s \in S_t^-$ and $s \in S \setminus S_t^-$. 

**Base case, $t = T$.** Since for all $s \in S$, $k$, and $n$, we have that (by definition)

$$\overline{V}_T^n(s) = U_T^k(s) = 0,$$

we can arbitrarily select $N_k^T$ and another integer $N_{T-}^k$ (to be used in the induction step where (24) holds for $n \geq N_{T-}^k$ and $s \in S_T^-$). Suppose we choose $N_T^k = N_{T-}^k \geq \Pi$. 

**Induction hypothesis, $t+1$.** Assume for $t+1 \leq T$ that $\forall k \geq 0$

(i) $\exists N_{t+1}^k < \infty$ such that $N_{t+1}^k \geq \Pi$ and $\forall n \geq N_{t+1}^k$ and $\forall s \in S$,

$$\overline{V}_{t+1}^n(s) \leq U_{t+1}^k(s).$$

(ii) $\exists N_{t+1}^{k-} < \infty$ such that $N_{t+1}^{k-} \geq \Pi$ and $\forall n \geq N_{t+1}^{k-}$ and $\forall s \in S_{t+1}^-$,

$$\overline{V}_{t+1}^n(s) \leq U_{t+1}^k(s).$$

**Inductive step from $t+1$ to $t$.** The remainder of the proof concerns this inductive step and is broken up into two cases, $s \in S_t^-$ and $s \in S \setminus S_t^-$. 

**Case 1: $s \in S_t^-$.** To prove this case, we induct forwards on $k$. Note that we are still inducting backwards on $t$, so the induction hypothesis for $t+1$ still holds. Within the induction on $k$, we essentially follow the proof of Proposition 4.6 of Bertsekas and Tsitsiklis (1996). 

**Base case, $k = 0$ (within induction on $t$).** By (11) and (13), we have that

$$U_T^0(s) \geq V_{\text{max}}.$$
But by Assumption 4, the updating equation (Step 2b of Figure 2), and the initialization of \( \mathcal{V}^0_t(s) \in [0, V_{\max}] \), we can easily see that

\[
\mathcal{V}^n_t(s) \in [0, V_{\max}],
\]

which means that

\[
\mathcal{V}^n_t(s) \leq U^0_t(s),
\]

for any \( n \) and \( s \), so we can choose \( N^0_t \) arbitrarily. Let us choose \( N^0_{t+1} \geq N^0_t + 1 \geq \mathcal{N}^\tau \) so that certain properties can be used later. Note that we have \( N^0_{t+1} \) from the induction hypothesis for \( t + 1 \).

**Induction hypothesis, \( k \) (within induction on \( t \)).** Assume for \( k \geq 0 \) that \( \exists N_{t+1}^k < \infty \) such that

\[
N_t^{k-} \geq N_{t+1}^k \geq N^\Pi \text{ and } \forall n \geq N_t^{k-} \text{ and } \forall s \in S_t^-, \nabla^0_t(s) \leq U^k_t(s).
\]

Before we begin the inductive step from \( k \) to \( k+1 \), we define some additional sequences and state a few useful lemmas.

**Definition 7.** The positive incurred noise (since iteration \( m \)) sequence \( W_{t,m}^n(s) \), for a particular \( t \leq T, s \in S, \) and \( m \geq 1 \), is defined as follows:

\[
W_{t,m}^m(s) = 0,
\]

\[
W_{t,m}^{n+1}(s) = \left[(1 - \alpha^n_t(s)) \cdot W_{t,m}^n(s) + \alpha^n_t(s) \cdot w_{t+1}^n(s)\right]^+ \text{ for } n \geq m.
\]

**Lemma 7.** For all \( m \geq 1, t \leq T, \) and \( s \in S, \) under Assumptions 4, 5, and 6,

\[
\lim_{n \to \infty} W_{t,m}^n(s) = 0 \quad \text{a.s.}
\]

**Proof:** See the proof of Lemma 6.2 in Nascimento and Powell (2009).

To reemphasize the presence of \( \omega \), we note that the following definition and the subsequent lemma both use the realization \( N_{t}^{k^-}(\omega) \) from the fixed \( \omega \) chosen at the beginning of the proof.

**Definition 8.** The other auxiliary sequence that we define is \( X_{t}^n(s) \), for a particular \( t \leq T \) and \( s \in S, \) is defined as follows:

\[
X_{t}^{n,k^-}(s) = U_t^k(s),
\]

\[
X_{t}^{n+1}(s) = (1 - \alpha_t^n(s)) \cdot X_{t}^n(s) + \alpha_t^n(s) \cdot (HU_t^k)(s) \text{ for } n \geq N_t^{k-}.
\]
Lemma 8. For $n \geq N_{k^*}^-$ and $s \in S_{k^*}$,

$$\nabla_t^n(s) \leq X_t^n(s) + W_t^n s_{k^*}.$$ 

Proof: See Appendix.

Inductive step from $k$ to $k + 1$. If $U_t^k(s) = (HU)^k_t(s)$, then by Lemma 2, we see that $U_t^k(s) = U_t^{k+1}(s)$ so $\nabla_t^n \leq U_t^k(s) \leq U_t^{k+1}(s)$ and the proof is complete. Similarly, if $U_t^k(s) = U_t^{k+1}(s)$, then the proof is trivially complete. So we assume that

$$U_t^k(s) < U_t^{k+1}(s).$$

In this case, we can define

$$\delta^k = \min_{s \in S_{k^*}} \left( \frac{U_t^k(s) - (HU)^k_t(s)}{4} \right) > 0. \quad (29)$$

Choose $N^U \geq N_{k^*}$ such that

$$\prod_{n=N_{k^*}^-}^{N^U-1} (1 - \alpha_t^n(s)) \leq \frac{1}{4} \quad (30)$$

and for all $n \geq N^U$,

$$W_t^n s_{k^*} \leq \delta^k. \quad (31)$$

$N^U$ clearly exists because both sequences converge to zero, by Lemma 6 and Lemma 7. Recursively using the definition of $X_t^n(s)$, we get that

$$X_t^n(s) = \beta_t^n(s) \cdot U_t^k(s) + (1 - \beta_t^n(s)) \cdot (HU)^k_t(s), \quad (32)$$

where $\beta_t^n = \prod_{s_{k^*}^-}^{n-1} (1 - \alpha_t^s(s))$. For $n \geq N^U$, we know that $\beta_t^n(s) \leq \frac{1}{4}$ so we can write

$$\beta_t^n(s) + \eta = 1/4, \quad (33)$$

for some $\eta \geq 0$. Hence, for $n \geq N^U$,

$$X_t^n(s) = \beta_t^n(s) \cdot U_t^k(s) + (1 - \beta_t^n(s)) \cdot (HU)^k_t(s)$$

$$= (\beta_t^n(s) + \eta) \cdot U_t^k(s) - \eta \cdot U_t^k(s) + (1 - \beta_t^n(s) - \eta) \cdot (HU)^k_t(s) + \eta (HU)^k_t(s)$$

$$= \frac{1}{4} \cdot U_t^k(s) + \frac{3}{4} \cdot (HU)^k_t(s) - \eta [U_t^k(s) - (HU)^k_t(s)]$$

$$\leq \frac{1}{4} \cdot U_t^k(s) + \frac{3}{4} \cdot (HU)^k_t(s) \quad (34)$$

$$= \frac{1}{2} \cdot [U_t^k(s) + (HV)^k_t(s)] - \frac{1}{4} \cdot [U_t^k(s) - (HU)^k_t(s)]$$

$$\leq U_t^{k+1}(s) - \delta^k,$$
where (34) follows from (28). Choose \(N_i^{(k+1)-}\geq N^U\), so that for \(n\geq N_i^{(k+1)-}\geq N^U\geq N_i^{k-}\) we can apply Lemma 8 to get
\[
\nabla^n_i(s) \leq X^n_i(s) + W_i^{n,N_i^{k-}}(s) \\
\leq (U_i^{k+1}(s) - \delta^k) + \delta^k \\
= U_i^{k+1}(s).
\]
Thus, the inductive step from \(k\) to \(k+1\) is complete.

Case 2: \(s \in S \setminus S^-\). Recall that we are still in the inductive step from \(t+1\) to \(t\). As previously mentioned, the proof for this case relies on an induction–like argument over the tree \(T^-_t(s)\).

We reiterate the following two facts about \(T^-_t(s)\):
(i) Every leaf node of \(T^-_t(s)\) represents an element of \(S^-\) (every node representing an element of \(S \setminus S^-\) has at least one child node).
(ii) The parent of every node besides root node is a node that represents an element of \(S \setminus S^-\).

The following lemma provides the essential step of the proof.

**Lemma 9.** Consider some \(k \geq 0\) and a node \(j\) of \(T^-_t(s)\) representing a state \(s_j \in S \setminus S^-\) and let the \(C_j \geq 1\) children of \(j\) be denoted by the set \(\{s_{j,1}, s_{j,2}, \ldots, s_{j,C_j}\}\). Suppose that for each \(s_{j,c}\) where \(1 \leq c \leq C_j\), we have that \(\exists N_i^{k,u}(s_{j,c}) < \infty\) such that \(\forall n \geq N_i^{k,u}(s_{j,c})\),
\[
\nabla^n_i(s_{j,c}) \leq U_i^k(s_{j,c}).
\]
(35)

Then, \(\exists N_i^{k,u}(s_j) < \infty\) such that \(\forall n \geq N_i^{k,u}(s_j)\),
\[
\nabla^n_i(s_j) \leq U_i^k(s_j).
\]
(36)

**Proof:** See Appendix.

From Case 1 we know that for each of the states \(s_i \in S^-\) represented by the leaf nodes, \(s_i\) satisfies (35). Thus, we can apply Lemma 9 to get \(N_i^{k,u}\) of \(s'\), where \(s'\) is any state such that all of the children of \(s'\) are leaf nodes. After this application of Lemma 9, \(s'\) satisfies the condition (35) and becomes eligible to be one of the children \(s_{j,c}\) from the statement of Lemma 9, where \(s_j\) is the parent of \(s'\). As soon as each sibling node of \(s'\) becomes eligible, we apply Lemma 9 again, in order to continue moving up the tree. Again, the fact that every leaf node is eligible to begin with (as they are elements of the set \(S^-\)) guarantees we can continue this process until eventually, we reach the root node and receive the value \(N_i^{k,u}(s)\) such that when \(n \geq N_i^{k,u}(s)\),
\[
\nabla^n_i(s) \leq U_i^k(s).
\]

To complete the proof, we must then choose \(N_i^k\) so that the above argument holds; thus, we select an \(N_i^k\) so that \(N_i^k \geq N_i^{k,u}(s')\) for every \(s' \in S \setminus S^-\).
7. Example Application: Bidding in the Electricity Market with Storage

Here we explain a simplified version of a real-world problem that illustrates an application where the value functions are monotone but not convex/concave. Consider the situation where a company, utilizing battery storage, places bids in an hour-ahead market in order to trade electricity and make profits (often called battery arbitrage). The bid consists of a buy (low) bid and a sell (high) bid, such that when the price falls below the buy bid, the company is obligated to buy one unit of electricity at the current price, and when the price exceeds the sell bid, the company is obligated to sell one unit of electricity at the current price. In this empirical study, we examine two variations of the bidding problem, one with a 3-dimensional state variable and another with additional 4th dimension, the lifetime of the battery.

Before providing the problem details, we remark that although a complete numerical study is not the main focus of this paper, the results below do indeed show that Monotone–ADP provides benefits in this specific, yet certainly nontrivial, problem setting. Future numerical work includes investigating a variety of different application classes.

7.1. Variation 1

Assume the exogenous price process $P_t$ is a nonnegative, discrete time, stochastic process. Furthermore, assume that $P_t$ has finite support, allowing us to compute exact expectations in the simulations (in a true application, Monte Carlo sampling can be used to approximate the expectations for price processes with infinite support). We consider the case where $P_t$ can be written as the sum of a deterministic function of $t$ and i.i.d. noise:

$$P_t = S(t) + \epsilon_t.$$  \hfill (37)

The amount of energy in the battery at time $t$ is represented by $R_t \in \{0, 1, 2, \ldots, R_{\text{max}}\}$ and the two-dimensional bidding decision made at time $t$ is $b_t = (b^-_t, b^+_t) \in B = \{(b^-, b^+) : b_{\min} \leq b^- \leq b^+ \leq b_{\max}\}$. Although this is a continuous decision space, we discretize it to compute an optimal policy.

The bidding process is as follows: at time $t$, a bid $b_t$ is submitted and becomes active for the hour $(t+1, t+2]$. At this point, in the real problem, a spot price is revealed and a transaction takes place every 5 minutes within the hour; however, to keep the example simple, let us assume that we receive a single price, $P_{t+2}$, at the end of the next hour. The bid $b_t$, in conjunction with the revealed spot price $P_{t+2}$, determines whether the battery is charged ($-1$), discharged ($+1$), or kept idle ($0$). This can be described by defining the function:

$$q(P_{t+2}; b_t) = \mathbf{1}_{\{b^-_t \leq P_{t+2}\}} - \mathbf{1}_{\{b^+_t \geq P_{t+2}\}}.$$  \hfill (38)
Thus, the transition from $R_t$ to $R_{t+1}$, denoted by the function $g$, depends on $q$ (and forces the next value to be between 0 and $R_{\text{max}}$):

$$R_{t+1} = g(R_t, b_{t-1}, P_{t+1}) = \left[\min(R_t - q(P_{t+1}, b_{t-1}), R_{\text{max}})\right]^+.$$  

The state variable is $S_t = (R_t, b_{t-1}^-, b_{t-1}^+)$ and transitions according to:

$$S_{t+1} = S^{M}(S_t, b_t, P_{t+1}) = (g(R_t, b_{t-1}, P_{t+1}), b_t).$$

Consider

$$h(r, q) = q \cdot \left(1 - 1_{\{r=0\}} \cdot 1_{\{q=1\}}\right),$$

a function that takes in a resource value $r$ and a battery action $q$. It returns zero if we are discharging and we do not have the resource (i.e. if both indicator functions produce one) and returns the input value of $q$ otherwise (indicating idle, selling, or buying with no efficiency loss). Since the bidding decision does not affect the current hours profits, the contribution/reward function for time $t$ is defined as the expected profit in the next hour $(t + 1, t + 2]$ from making the bid $b_t$:

$$C_t(S_t, b_t) = \mathbb{E}(P_{t+2} \cdot h(R_{t+1}, q(P_{t+2}, b_t)) | S_t).$$

The optimal value functions $V_t^*(S_t)$ are defined as in (3), with the action space $A$ replaced with the bid space $B$.

**Proposition 2.** For Variation 1 of the bidding problem, we have that for each $t \leq T$, the optimal value function $V_t^*(R_t, b_{t-1}^-, b_{t-1}^+)$ is nondecreasing in each of its three dimensions.

**Proof:** Proposition 1 can be applied (details omitted).

In order to produce an optimal solution via backward dynamic programming (BDP), we discretized the bid (action) space so that instead of $B$, we use:

$$B^D = \{b_{\text{min}}, b_{\text{min}} + \delta, b_{\text{min}} + 2\delta, \ldots, b_{\text{max}} - \delta, b_{\text{max}}\},$$

where $\delta = (b_{\text{max}} - b_{\text{min}})/(D - 1)$. Figure 5 shows one of the optimal value functions (for $t = 12$) and visually demonstrates the monotonicity to which Proposition 2 refers. The monotone structure in the two bid dimensions (for a fixed $R_t$) is evident from Figure 5a, while the monotone structure in the resource dimension is clear from Figure 5b (the sequence of increasing surfaces corresponds to the sequence of increasing resource levels).

Since Monotone–ADP is simply a version of Approximate or Asynchronous Value Iteration (AVI) with an additional projection step, comparing these two algorithms allows us to directly observe the effects of $\Pi_M$. The version of AVI that we implement is exactly Figure 2 with the Step 2c removed.
In the experiments we conducted for Variation 1 of the bidding problem, we fixed the price process \( P_t \) and the values of \( R_{\text{max}}, T, b_{\text{min}}, \) and \( b_{\text{max}} \) but varied the level of discretization \( D \). By keeping the problem setting essentially constant but increasing the size of both the state and action spaces, we are able to observe the difference in the empirical convergence speed between the two algorithms.

A typical size for the batteries under consideration is \( R_{\text{max}} = 6 \) MWh, meaning that the battery can be charged at 1 MW for 6 hours before reaching capacity. We consider the problem of consecutively bidding for \( T = 24 \) hours with \( b_{\text{min}} = 15 \) and \( b_{\text{max}} = 85 \). The price process \( P_t \) has the form taken in (37). It is defined by a sinusoidal (to represent the hour–of–day effects on price) deterministic component:

\[
S(t) = 15 \cdot \sin\left(\frac{2\pi t}{24}\right) + 50,
\]

and \( \epsilon_t \in \{0, \pm 1, \pm 2, \ldots, \pm 20\} \), a sequence of mean zero i.i.d. random variables distributed according to the discrete pseudonormal distribution (with \( \sigma_r^2 = 49 \) \(^1\)). We allow the value of \( D \) to vary between 30 and 150.

7.3. Numerical Results for Variation 1

Figure 6 gives a quick visual comparison between the approximate value functions (generated by AVI and M–ADP) and the optimal value function. We remark that after \( N = 1000 \) iterations, the value function approximation in Figure 6b obtained by exploiting monotonicity has a very similar shape to that of the optimal value functions in Figure 5a, while the result in Figure 6a has no such resemblance.
We follow the method in Powell (2011) for evaluating the policies generated by the algorithms. By the principle of dynamic programming, for a particular set of value functions \( V \), the set of decision functions can be written as

\[
A_t(S_t) = \arg \max_{b_t \in B} \left[ C_t(S_t, b_t) + \mathbb{E}(V_{t+1}(S_{t+1}) | S_t) \right].
\]

For a sample path \( \omega \in \Omega \), let

\[
F(V, \omega) = \sum_{t=0}^{T} P_{t+2}(\omega) \cdot h \left( R_{t+2}(\omega), q \left( P_{t+2}(\omega), A_t(S_t(\omega)) \right) \right)
\]

be a sample outcome of the profit. To evaluate the value of a policy (in this case, the expected profit generated by the policy), we take a test set of \( L = 1000 \) sample paths \( \hat{\Omega} \) and consider the empirical mean:

\[
F(V) = \frac{1}{L} \sum_{\omega \in \hat{\Omega}} F(V, \omega).
\]

Table 1 summarizes a series of experiments performed on the bidding problem with varying levels of discretization. We report the percent of optimality by calculating the value of the policies generated by the approximation algorithms and comparing the result to \( F(V^*) \), where \( V^* \) is computed using backward dynamic programming. Although this version of AVI is also a convergent algorithm (see Proposition 4.6 of Bertsekas and Tsitsiklis (1996)), its performance is markedly worse, especially when the state space is large. Because it exploits the monotone structure, M–ADP has the ability to quickly attain the general shape of the value functions. Figure 7 illustrates this by showing the approximations at early iterations of the algorithm.

We now observe that when \( D \geq 70 \), the state space exceeds 15000 (i.e., the number of iterations used by the approximation algorithms). Since BDP requires visiting the entire state space, we see that M–ADP can achieve near-optimality using just a fraction of the number of observations (and
hence, optimizations) that BDP requires. For $D = 150$ and $N = 15000$, M–ADP visits at most $15000/79725 = 18.8\%$ of the number of states as BDP does, but achieves a solution that is 93.0\% optimal, compared to 80.1\% of optimal when we do not exploit monotonicity.

Table 2 compares the computation times of M–ADP to that of computing the optimal solution via BDP. To speed up the computation of the optimal solution, we leveraged the power of parallel computing within each time step, which cannot be easily done for the ADP algorithm due to its iterative nature. Assuming that we want to achieve 90\% optimality and that parallel computing is available, Table 2 shows that for the problems with smaller state/action spaces (i.e. $D = 30$ or $D = 50$), it is advantageous to simply compute the optimal solution rather than run M–ADP for $N \approx 3000$ (needed for 90\% optimality) iterations. However, for a problem with a larger state/action space, even in the presence of parallel computing, M–ADP can achieve 90\% optimality with a significantly faster computation time.

| Iterations | Algorithm | $D = 30$ | $D = 50$ | $D = 70$ | $D = 90$ | $D = 110$ | $D = 130$ | $D = 150$ |
|------------|-----------|----------|----------|----------|----------|----------|----------|----------|
| $N = 1000$ | M–ADP     | 80.6\%   | 80.8\%   | 79.3\%   | 77.6\%   | 78.7\%   | 80.5\%   | 78.9\%   |
|            | AVI       | 65.0\%   | 77.4\%   | 71.3\%   | 54.5\%   | 63.2\%   | 73.2\%   | 56.2\%   |
| $N = 3000$ | M–ADP     | 92.7\%   | 90.9\%   | 89.5\%   | 91.1\%   | 89.5\%   | 88.4\%   | 89.6\%   |
|            | AVI       | 68.0\%   | 63.0\%   | 75.6\%   | 70.9\%   | 74.7\%   | 80.4\%   | 72.4\%   |
| $N = 5000$ | M–ADP     | 94.8\%   | 95.5\%   | 94.5\%   | 94.3\%   | 91.0\%   | 93.0\%   | 86.5\%   |
|            | AVI       | 76.4\%   | 78.5\%   | 75.1\%   | 81.4\%   | 84.4\%   | 83.9\%   | 79.6\%   |
| $N = 7000$ | M–ADP     | 96.7\%   | 92.2\%   | 93.8\%   | 91.2\%   | 94.1\%   | 92.6\%   | 90.0\%   |
|            | AVI       | 81.2\%   | 71.2\%   | 73.3\%   | 80.5\%   | 79.7\%   | 85.5\%   | 76.1\%   |
| $N = 9000$ | M–ADP     | 96.3\%   | 93.8\%   | 94.4\%   | 93.7\%   | 93.8\%   | 94.2\%   | 91.3\%   |
|            | AVI       | 80.3\%   | 74.4\%   | 74.1\%   | 76.1\%   | 74.9\%   | 83.2\%   | 74.9\%   |
| $N = 11000$| M–ADP     | 97.7\%   | 93.0\%   | 97.0\%   | 93.4\%   | 95.7\%   | 94.6\%   | 92.7\%   |
|            | AVI       | 85.7\%   | 71.8\%   | 84.3\%   | 85.9\%   | 81.9\%   | 88.7\%   | 77.9\%   |
| $N = 13000$| M–ADP     | 99.9\%   | 96.1\%   | 98.0\%   | 95.4\%   | 94.7\%   | 93.8\%   | 94.2\%   |
|            | AVI       | 87.8\%   | 82.9\%   | 85.2\%   | 90.8\%   | 79.8\%   | 87.8\%   | 79.0\%   |
| $N = 15000$| M–ADP     | 99.0\%   | 97.8\%   | 96.9\%   | 97.2\%   | 94.5\%   | 94.0\%   | 93.0\%   |
|            | AVI       | 92.3\%   | 82.2\%   | 83.8\%   | 91.5\%   | 83.4\%   | 88.4\%   | 80.1\%   |

Table 1  % Optimal of Policies from the M–ADP and AVI Algorithms for Variation 1

| Algorithm                          | $D = 30$ | $D = 50$ | $D = 70$ | $D = 90$ | $D = 110$ | $D = 130$ | $D = 150$ |
|------------------------------------|----------|----------|----------|----------|----------|----------|----------|
| M–ADP (single CPU, $N = 1000$)     | 78.8     | 113.7    | 158.0    | 172.6    | 258.8    | 295.2    | 347.9    |
| BDP (parallel, 8 CPUs)             | 129.6    | 364.9    | 808.1    | 1350.7   | 2264.5   | 2920.0   | 3415.5   |
| BDP (single CPU)                   | 1036.8   | 2919.2   | 6465.1   | 10805.6  | 18116.0  | 23360.0  | 27324.0  |

Table 2  Comparison of Computation Times (in minutes) of M–ADP vs BDP for Variation 1
For the purpose of demonstrating Monotone–ADP on a higher dimensional state space, let us consider the same problem, with the added notion of battery lifetime. Our model assumes that the lifetime of the battery, \( L_t \in \mathcal{L} = \{0, 1, \ldots, L_{\text{max}}\} \), decreases after every discharge performed (battery lifetimes are typically quoted in units of charge–discharge cycles). Furthermore, we assume that the degraded quality of the battery is reflected solely by a smaller revenue when discharging represented by some fraction of the actual price: \( \beta(L_{t-1}) \cdot P_t \), where \( \beta : \mathcal{L} \to [0, 1] \), the aging function, is increasing in \( L_{t-1} \). Therefore, a simplifying assumption we are making is that we still charge and discharge at unit increments (i.e. \( R_t \) continues to fluctuate by \( \pm 1 \)). \( L_t \) transitions according to the equation:

\[
L_{t+1} = g_2(L_t, b_{t-1}, P_{t+1}) \\
= \left[ L_t - 1_{\{b_{t-1} \geq P_{t+1}\}} \right]^+, \\
\]

so that \( L_{t+1} = L_t \) whenever the battery is not discharging.
For Variation 2, the state variable is $S_t = (R_t, L_t, b_{t-1}, b_{t-1}^r)$ and the new state transition function can be written as:

$$S_{t+1} = S^M(S_t, b_t, P_{t+1})$$

$$= (g_1(R_t, b_{t-1}, P_{t+1}), g_2(L_t, b_{t-1}, P_{t+1}), b_t).$$

The new contribution function is defined as:

$$C_t(S_t, b_t) = E((\beta(L_{t+1}) \cdot 1_{q(P_{t+2}, b_t)=1} + 1_{q(P_{t+2}, b_t)\neq 1}) \cdot P_{t+2} \cdot h(R_{t+1}, q(P_{t+2}, b_t))| S_t),$$

and the optimal value functions are defined as before.

**Proposition 3.** For Variation 2 of the bidding problem, we have that for each $t \leq T$, the optimal value function $V^*_t(R_t, L_t, b_{t-1}, b_{t-1}^r)$ is nondecreasing in each of its four dimensions.

**Proof:** Proposition 1 can be applied (details omitted).

### 7.5. Parameter Values for Variation 2

We explore several versions of the second problem by altering the parameter values. Due to the practicality of a smaller action space and the fact that changing the discretization level does not significantly change the problem, we fix the discretization level to $D = 30$. We also fixed the deterministic part of the price process, $S(t)$ to the sinusoidal function used in Variation 1; see (39). Moreover, $b_{\min} = 15$, $b_{\max} = 85$, and $\text{supp}(\epsilon_t) = \{0, \pm 1, \pm 2, \ldots, \pm 20\}$ remain unchanged from Variation 1. We considered both the cases where the battery aged mattered and did not matter (by setting $\beta(l) = 1$), in effect introducing an irrelevant state variable. When aging mattered, the aging function we chose was $\beta(l) = (l/L_{\max})^{1/6}$, which provides a roughly linear decline in efficiency from 100% to around 70%, followed by a much steeper decline. Lastly, in Problem 4, we considered a uniform distribution for the noise, while the remaining problems used pseudonormal noise, as was done in Variation 1. The different problems that were created are summarized in Table 3.

| Problem | $T$ | $R_{\max}$ | $L_{\max}$ | $\beta(l)$ | Dist. of $\epsilon_t$ |
|---------|-----|------------|------------|-------------|----------------------|
| Problem A | 24  | 6          | 8          | 1           | Pseudonormal          |
| Problem B | 24  | 6          | 8          | (l/8)$^{1/6}$ | Pseudonormal          |
| Problem C | 36  | 6          | 8          | 1           | Pseudonormal          |
| Problem D | 24  | 12         | 12         | (l/12)$^{1/6}$ | Uniform              |
| Problem E | 24  | 12         | 12         | (l/12)$^{1/6}$ | Pseudonormal          |
| Problem F | 36  | 18         | 18         | (l/18)$^{1/6}$ | Pseudonormal          |

**Table 3** Parameter Choices for Variation 2 Test Problems
7.6. Numerical Results for Variation 2

The results for Variation 2 are given in Table 4. Because the sizes of the state spaces for the problems of Variation 2 are larger than those of Variation 1, the value of monotonicity preservation becomes more pronounced (in general, the results from both Variation 1 and 2 show that larger state spaces benefit more from $\Pi_M$). This is especially evident in Problem $F$, where after $N = 1000$ iterations, M–ADP achieves 45.9% optimality while traditional AVI does not even reach 10%. These numerical results suggest that the convergence rate of the ADP algorithm is substantially increased through the use of the monotonicity preserving operation.

A comparison of computation times is given in Table 5. Once again, we notice that, especially for the larger problems, it is possible to produce a reasonably good policy using M–ADP in a fraction of the time it takes to compute the optimal policy (even in the presence of parallel computing). Even in the case of this small problem, an optimal solution requires days to compute (Problem $F$ needed 5 days on 8 CPU’s). For a true, real–world, application, we cannot expect to be able to compute an optimal solution in any reasonable amount of time. Thus, M–ADP becomes a feasible alternative for structured problems with monotone value functions.

| Iterations | Algorithm | Problem A $|S| = 22320$ | B $|S| = 22320$ | C $|S| = 22320$ | D $|S| = 66960$ | E $|S| = 66960$ | F $|S| = 150660$ |
|------------|-----------|----------------|----------------|----------------|----------------|----------------|----------------|
| $N = 1000$ | M–ADP 58.9% | 67.8% | 73.5% | 60.7% | 56.8% | 45.9% |
|            | AVI 53.7%  | 45.7% | 66.6% | 23.4% | 24.8% | 7.8%  |
| $N = 3000$ | M–ADP 74.6% | 79.5% | 72.9% | 58.5% | 55.1% | 69.0% |
|            | AVI 53.5%  | 58.9% | 73.1% | 36.0% | 43.3% | 32.1% |
| $N = 5000$ | M–ADP 83.7% | 82.8% | 87.2% | 73.8% | 66.1% | 64.1% |
|            | AVI 60.7%  | 67.3% | 82.1% | 43.8% | 52.6% | 49.0% |
| $N = 7000$ | M–ADP 87.2% | 91.1% | 91.1% | 73.1% | 79.4% | 72.3% |
|            | AVI 65.8%  | 72.4% | 80.2% | 50.5% | 54.7% | 57.0% |
| $N = 9000$ | M–ADP 89.4% | 93.6% | 93.3% | 76.2% | 74.9% | 86.6% |
|            | AVI 70.2%  | 75.8% | 85.3% | 46.7% | 58.8% | 57.3% |
| $N = 11000$| M–ADP 91.5% | 90.5% | 96.0% | 79.0% | 83.2% | 87.0% |
|            | AVI 73.6%  | 81.8% | 88.1% | 52.3% | 59.0% | 68.7% |
| $N = 13000$| M–ADP 93.8% | 89.9% | 96.8% | 79.8% | 83.7% | 88.5% |
|            | AVI 76.3%  | 83.1% | 87.8% | 49.8% | 68.2% | 57.8% |
| $N = 15000$| M–ADP 94.5% | 93.9% | 97.7% | 84.0% | 87.3% | 90.9% |
|            | AVI 76.7%  | 86.6% | 88.3% | 59.4% | 67.9% | 70.2% |
| $N = 17000$| M–ADP 95.8% | 96.4% | 97.8% | 82.7% | 86.8% | 91.4% |
|            | AVI 78.0%  | 85.1% | 90.7% | 62.2% | 72.0% | 70.6% |
| $N = 19000$| M–ADP 96.9% | 94.1% | 97.1% | 85.5% | 88.2% | 91.2% |
|            | AVI 82.3%  | 84.2% | 89.3% | 61.6% | 75.3% | 77.7% |
| $N = 21000$| M–ADP 95.0% | 98.4% | 98.1% | 90.5% | 87.8% | 92.7% |
|            | AVI 81.1%  | 87.7% | 90.0% | 61.0% | 73.7% | 76.3% |
| $N = 23000$| M–ADP 96.9% | 95.9% | 97.3% | 90.8% | 88.5% | 91.2% |
|            | AVI 81.4%  | 87.5% | 90.5% | 62.3% | 75.0% | 77.0% |
| $N = 25000$| M–ADP 97.0% | 98.5% | 98.5% | 89.7% | 90.4% | 94.8% |
|            | AVI 86.4%  | 89.4% | 92.1% | 60.0% | 75.1% | 76.0% |

Table 4 % Optimal of Policies from the M–ADP and AVI Algorithms for Variation 2
8. Conclusion

In this paper, we formulated a general sequential decision problem with the property of monotone value functions. We then formally described an ADP algorithm, Monotone–ADP, first proposed in Papadaki and Powell (2002), that exploits the structure of the value functions by performing a monotonicity preserving operation at each iteration to increase the information gained from each random observation. Under the assumption that all states are visited infinitely often, we prove the convergence of the estimates produced by Monotone–ADP to the optimal value functions. The proof uses an argument adapted from Bertsekas and Tsitsiklis (1996) and Nascimento and Powell (2009). However, in Nascimento and Powell (2009), where concavity was assumed, pure exploitation could be used, but only in one dimension. This paper requires a full exploration policy, but exploits monotonicity in multiple dimensions, dramatically accelerating the rate of convergence. We then presented an example application related to battery arbitrage and bidding in the electricity market. Empirical results for this example problem show that Monotone–ADP produces a significantly better policy than traditional approximate value iteration and can often do so much faster than backward dynamic programming. In a true application where the optimal solution cannot be easily computed due to a large state space, we expect that, utilizing of monotonicity, when it exists, can bring significant advantages.

| Algorithm                  | A    | B    | C    | D    | E    | F    |
|----------------------------|------|------|------|------|------|------|
| M–ADP (single CPU, N = 1000) | 125.4| 107.7| 194.6| 114.9| 115.9| 190.0|
| BDP (parallel, 8 CPUs)     | 1764.0| 3049.3| 2340.8| 2411.0| 2531.9| 7121.2|
| BDP (single CPU)           | 14112.0| 24394.7| 18726.5| 19448.1| 20255.1| 56970.0|

Table 5  Comparison of Computation Times (in minutes) of M–ADP vs BDP for Variation 2
Appendix. Proofs

A. Proof of Proposition 1:

This is easily shown using backwards induction starting from the base case of $V_T^*$, which satisfies (5) by definition. Consider $s_1 = (m_1, i) \in S$ and $s_2 = (m_2, i) \in S$ where $m_1 \leq m_2$. Given that $V_{t+1}^*$ satisfies (5), applying (i) then (iii) and using the monotonicity of the conditional expectation, we see that for any $a \in A$,

$$
E(V_{t+1}^*(SM(m_1, i, a, W_{t+1}))|S_t = s_1) = E(V_{t+1}^*(SM(m_2, i, a, W_{t+1}))|I_t = i) \tag{40}
$$

$$
\leq E(V_{t+1}^*(SM(m_2, i, a, W_{t+1}))|I_t = i) \tag{41}
$$

$$
= E(V_{t+1}^*(SM(m_2, i, a, W_{t+1}))|S_t = s_2). \tag{42}
$$

Summing with the inequality in (ii) and using the fact that the resulting inequality is true for all elements $a$ of the compact set $A$ (in particular, it is true for the maximizing action), we can see that $V_{t}^*(s_1) \leq V_{t}^*(s_2)$.

B. Proof of Lemma 1:

(i) and (iii) are true by the definition of $H$. (ii) follows from the well–known fact that our finite horizon MDP has a unique solution, which can be seen from the fact that the optimal value functions can be written recursively using Bellman’s Equation, starting from the deterministic terminal value.

C. Proof of Lemma 3:

We note that by definition and (5), $U^0$ and $L^0$ satisfy this property. By the definition of $H$, (4), and Assumption 3, it is easy to see that if $U^k$ satisfies the property, then $HU^k$ does as well. Thus, by the definition of $U^{k+1}$ being the average of the two, we see that $U^{k+1}$ also satisfies the monotonicity property.

D. Proof of Lemma 4:

We will consider $S_t^-$. Since $S$ is finite, there exists a state $s = (m, i)$ such that there does not exist $m' \in M$ where $m' \leq m$. An increase from the projection operator must originate from a violation of monotonicity during an observation of a state $s' = (m', i)$ where $m' \leq m$ and $m' \neq m$, but such a state does not exist. Thus, $s \in S_t^-$. 
E. Proof of Lemma 5:
Let \( s = (m, i) \) and \( s' = (m', i) \) (since \( s' \) represents more than one state in the equations below, let this notation hold for any state that \( s' \) represents). Define:

\[
A = \{ s'' = (m'', i) : m'' \leq m, m'' \neq m \}, \\
B = \bigcup_{s' \in S_L(s)} \{ s'' = (m'', i) : m'' \leq m' \}.
\]

We argue that \( A \subseteq B \). Choose \( s_1 = (m_1, i) \in A \) and suppose for the sake of contradiction that \( s_1 \notin B \). In particular, this means that \( s_1 \notin S_L(s) \) because \( S_L(s) \subseteq B \). By Definition 5, it follows that there must exist \( s_2 = (m_2, i) \) such that \( m_1 \leq m_2 \leq m \) where \( m_2 \neq m_1 \) and \( m_2 \neq m \). It now follows that \( s_2 \notin S_L(s) \) because if it were, \( s_1 \) would be an element of \( B \). This argument can be repeated to produce other states \( s_3, s_4, \ldots \), each different from the rest, such that

\[
m_1 \leq m_2 \leq m_3 \leq m_4 \leq \ldots \leq m,
\]

where each state \( s_k \) is not an element of \( S_L(s) \). However, because \( S \) is a finite set, eventually we reach a point where we cannot produce another state to satisfy Definition 5 and we will have that the final state, call it \( s_K \), is an element of \( S_L \). Here, we reach a contradiction because (43) (specifically \( m_1 \leq m_K \)) implies that \( s_1 \in B \). Thus, \( s_1 \in B \) and we have shown that \( A \subseteq B \).

Because the value of \( s \) was increased, a violation of monotonicity must have occurred during the observation of \( S^n_t = (M^n_t, i) \in A \). As this implies \( S^n_t \in B \), we know that \( M^n_t \leq m' \) for some \( s' = (m', i) \in S_L(s) \). We can write

\[
\nabla^{n-1}_t (S^n_t) \leq \nabla^{n-1}_t (s') \leq \nabla^{n-1}_t (s) < z^n_t (S^n_t) = \nabla^n_t (s),
\]

meaning that \( \Pi_M \) acts on \( s' \) and we have

\[
\nabla^n_t (s') = z^n_t (S^n_t) = \nabla^n_t (s),
\]

the desired result.

F. Proof of Lemma 6:
We first notice that the product inside the limit is nonnegative because the stepsizes \( \alpha^n_t \leq 1 \). Also the sequence is monotonic; therefore, the limit exists. Now,

\[
\prod_{n = N^k_t}^{m} (1 - \alpha^n_t (s)) = \exp \left[ \sum_{n = N^k_t}^{m} \log (1 - \alpha^n_t (s)) \right] \\
= \exp \left[ - \sum_{n = N^k_t}^{m} \alpha^n_t (s) \right],
\]
where the inequality follows from \( \log(1 - x) \leq -x \). Since \( s \) is visited infinitely often and

\[
\sum_{n=N_k^{-t}}^{\infty} \alpha_t^n = \infty \quad \text{a.s.,}
\]

the result follows by appropriately taking limits.

**G. Proof of Lemma 8:**

First, we show an inequality needed later in the proof. Since \( n \geq N_k^{-t} \), we know that \( n \geq N_k^{-t} + 1 \) by the induction hypothesis for \( k \). Now by the induction hypothesis for \( k+1 \), we know that

\[
V_{n+1}^t(s) \leq U_{t+1}^k(s) \quad (44)
\]

holds for all \( s \in S \). Using the definition of \( H \) in (6),

\[
(HV^n)_t(s) = \sup_{a_t \in A} \{ C_t(s, a_t) + E(V_{t+1}^n(S_{t+1})|S_t = s) \}
\]

\[
\leq \sup_{a_t \in A} \{ C_t(s, a_t) + E(U_{t+1}^k(S_{t+1})|S_t = s) \} \quad (45)
\]

where the inequality follows from (44). With this inequality in mind, we continue on to show the statement of the lemma, by inducting forwards on \( n \).

*Base case, \( n = N_k^{-t} \).* By the induction hypothesis for \( k \) (which we can legitimately use in this proof because of the placement of the lemma after the induction hypothesis), we have that

\[
V_{N_k^{-t}}^t(s) \leq U_t^k(s)
\]

Combined with the fact that

\[
W_{N_k^{-t}, N_k^{-t}}^t(s) = 0 \quad \text{and} \quad U_t^k(s) = X_{N_k^{-t}}^t(s),
\]

we see that the statement of the lemma holds for the base case.

*Inductive hypothesis, \( n \).* Suppose the statement of the lemma holds for \( n \).

*Inductive step from \( n \) to \( n+1 \).* Suppose \( S_{t+1}^n = s \), meaning a direct update happened on iteration \( n+1 \) and \( \alpha_t^n(s) = \alpha_t^n \). Thus, we have that

\[
V_{t+1}^{n+1}(s) = z_t^{n+1}(s)
\]

\[
= (1 - \alpha_t^n) \cdot V_t^n(s) + \alpha_t^n \cdot \hat{v}_t^{n+1}(s)
\]

\[
= (1 - \alpha_t^n) \cdot V_t^n(s) + \alpha_t^n \cdot [ (HV^n)_t(s) + w_t^{n+1}(s) ]
\]

\[
\leq (1 - \alpha_t^n) \cdot (X_t^n(s) + W_t^{n,N_k^{-t}}(s)) + \alpha_t^n \cdot [ (HV^n)_t(s) + w_t^{n+1}(s) ] \quad (46)
\]

\[
\leq (1 - \alpha_t^n) \cdot (X_t^n(s) + W_t^{n,N_k^{-t}}(s)) + \alpha_t^n \cdot [ (HU^k)_t(s) + w_t^{n+1}(s) ] \quad (47)
\]

\[
= X_t^{n+1}(s) + W_t^{n+1,N_k^{-t}}(s).
\]

where (46) follows from the induction hypothesis for \( n \), (47) follows from (45).
Now suppose that $S_{n+1} \neq s$. This means that the stepsize $\alpha_{n+1}(s) = 0$ and thus,

$$
X^{n+1}(s) = X^n(s),
W^{n+1,N^k}(s) = W^n,N^k(s).
$$

(48)

Because we assumed that $s \in S^-$ and $n+1 \geq N^k$ (induction hypothesis for $k$), we know that the projection operator did not increase the value of $s$ on this iteration (a decrease is possible); hence,

$$
\nabla V^{n+1}(s) \leq \nabla V^n(s) \leq X^n(s) + W^{n,N^k}(s) \leq X^{n+1}(s) + W^{n+1,N^k}(s)
$$

by the induction hypothesis for $n$ and (48).

### H. Proof of Lemma 9:

We break the proof into several steps, as was done in Nascimento and Powell (2009).

**Step 1.** We show that $\exists N^k(s) < \infty$ such that $\forall n \geq N^k(s)$,

$$
\nabla V^n(s) \leq U^k(s) + W^{n,N^k}(s).
$$

(49)

To show this, let

$$
N^k(s) = \min \left( n \in S^- : n \geq \max_c N^k(u)(s) \right),
$$

which exists because $s \in S^- \setminus S^+$ and is increased infinitely often. This means that $\Pi_M$ increased the value of state $s$ on iteration $\bar{n} = N^k(w(s))$. We will show (49) using induction.

**Base case, $n = \bar{n}$.** Using Lemma 5, for some $c \in \{1, 2, \ldots, C_J\}$, we have:

$$
\nabla V^n(s) \leq U^k(s) + W^{\bar{n},\bar{n}}(s).
$$

where the second inequality follows from monotonicity within $U^k$ (see Lemma 3) and the fact that $W^{\bar{n},\bar{n}} = 0$.

**Induction hypothesis, $n$.** Suppose (49) is true for $n$ where $n \geq \bar{n}$.

**Inductive step from $n$ to $n+1$.** Consider the cases:

(I) Suppose $n+1 \in N^-(s)$. The proof for this is exactly the same as for the base case, except we use $W^{n+1,\bar{n}} \geq 0$. Again, this step depends heavily on Lemma 5 and on the fact that every child node represents a state that satisfies (35).

(II) Suppose $n+1 \notin N^-(s)$. There are two cases to consider:
(A) Suppose $S_t^{n+1} = s_j$. Then,
\[
\bar{V}_t^{n+1}(s_j) = z_t^{n+1}(s_j) = (1 - \alpha_t^n) \cdot \bar{V}_t^n(s_j) + \alpha_t^{n+1} \cdot \tilde{v}_t^{n+1}
\]
\[
\leq (1 - \alpha_t^n) \cdot \left( U_t^k(s_j) + W_t^{n,N_t,k,W}(s_j) \right) + \alpha_t^n \cdot \tilde{v}_t^{n+1} \quad (50)
\]
\[
= (1 - \alpha_t^n) \cdot \left( U_t^k(s_j) + W_t^{n,\tilde{n},\tilde{n}}(s_j) \right) + \alpha_t^n \cdot w_t^{n+1}(s_j) + \alpha_t^n \cdot (H \bar{V}_t^n)(s_j)
\]
\[
\leq (1 - \alpha_t^n) \cdot \left( U_t^k(s_j) + W_t^{n,N_t,k,W}(s_j) \right) + \alpha_t^n \cdot w_t^{n+1}(s_j) + \alpha_t^n \cdot U_t^k(s_j) \quad (51)
\]
\[
\leq U_t^k(s_j) + W_t^{n+1,\tilde{n}}(s_j),
\]
where (50) follows from the induction hypothesis for $n$, (51) follows from (45) and Lemma 1.

(B) Suppose $S_t^{n+1} \neq s_j$. This implies $\alpha_t^{n+1}(s_j) = 0$ and therefore, $W_t^{n+1,\tilde{n}}(s_j) = W_t^{n+1,\tilde{n}}(s_j)$. Because the value of $s_j$ was not increased at $n + 1$,
\[
\bar{V}_t^{n+1}(s_j) \leq \bar{V}_t^n(s_j)
\]
\[
\leq U_t^k(s_j) + W_t^{n,\tilde{n}}(s_j)
\]
\[
= U_t^k(s_j) + W_t^{n+1,\tilde{n}}(s_j).
\]

Step 2. By Lemma 7, we know that $W_t^{n,\tilde{n}}(s_j) \rightarrow 0$ and thus, $\exists N_t^{k,\epsilon}(s_j) < \infty$ such that $\forall n \geq N_t^{k,\epsilon}(s_j)$,
\[
\bar{V}_t^n(s_j) \leq U_t^k(s_j) + \epsilon. \quad (52)
\]

Let $\epsilon = U_t^k(s_j) - V_t^*(s_j) > 0$. Since $U_t^k(s_j) \searrow V_t^*(s_j)$, we also have that $\exists k' > k$ such that,
\[
U_t^{k'}(s_j) - V_t^*(s_j) < \epsilon/2.
\]

Combining with the definition of $\epsilon$, we have
\[
U_t^{k'}(s_j) - U_t^k(s_j) > \epsilon/2.
\]

Applying (52), we know that $\exists N_t^{k,\epsilon/2}(s_j) < \infty$ such that $\forall n \geq N_t^{k,\epsilon/2}(s_j)$,
\[
\bar{V}_t^n(s_j) \leq U_t^{k'}(s_j) + \epsilon/2 \leq U_t^k(s_j) - \epsilon/2 + \epsilon/2 \leq U_t^k(s_j).
\]

Choose $N_t^{k,u}(s_j)$ such that $N_t^{k,u}(s_j) \geq N_t^{k,\epsilon/2}(s_j)$ to conclude the proof.
Endnotes

1. See http://www.castlelab.princeton.edu/Datasets/Time-dependent_storage/readme.pdf for the definition.

References

Ayer, M, HD Brunk, GM Ewing. 1955. An empirical distribution function for sampling with incomplete information. The Annals of Mathematical Statistics 26(4) 641–647. URL http://www.jstor.org/stable/10.2307/2236377.

Barlow, R. E., D.J. Bartholomew, J. M. Bremner, H. D. Brunk. 1972. Statistical inference under order restrictions; the theory and application of isotonic regression. Wiley, New York.

Bertsekas, D. P., J. N. Tsitsiklis. 1996. Neuro–Dynamic Programming. Athena Scientific, Belmont, MA.

Binsbergen, Jules H., Michael W. Brandt. 2007. Solving dynamic portfolio choice problems by recursing on optimized portfolio weights or on the value function? Computational Economics 29(3-4) 355–367. doi:10.1007/s10614-006-9073-z. URL http://link.springer.com/10.1007/s10614-006-9073-z.

Breiman, L. 1992. Probability. Society of Industrial and Applied Mathematics, Philadelphia, PA.

Brunk, H. D. 1955. Maximum Likelihood Estimates of Monotone Parameters. The Annals of Mathematical Statistics URL http://projecteuclid.org/euclid.aoms/1177728420.

Dette, H., N. Neumeyer, K. F. Pilz. 2006. A simple nonparametric estimator of a strictly monotone regression function. Bernoulli 12(3) 469–490. URL http://projecteuclid.org/euclid.bj/1151525131.

Mammen, E. 1991. Estimating a Smooth Monotone Regression Function. The Annals of Statistics URL http://www.jstor.org/stable/10.2307/2242080.

McCall, JJ. 1970. Economics of Information and Job Search. The Quarterly Journal of Economics 84(1) 113–126. URL http://www.jstor.org/stable/10.2307/1879403.

Mukerjee, H. 1988. Monotone nonparametric regression. The Annals of Statistics URL http://projecteuclid.org/euclid.aos/1176350832.

Müller, A. 1997. How does the value function of a Markov decision process depend on the transition probabilities? Mathematics of Operations Research URL http://mor.journal.informs.org/content/22/4/872.short.

Nascimento, J., W. Powell. 2010. Dynamic Programming Models and Algorithms for the Mutual Fund Cash Balance Problem. Management Science 56(5) 801–815. doi:10.1287/mnsc.1100.1143. URL http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.1100.1143.

Nascimento, J. M., W. B. Powell. 2009. An Optimal Approximate Dynamic Programming Algorithm for the Lagged Asset Acquisition Problem. Mathematics of Operations Research 34(1) 210–237. doi:10.1287/moor.1080.0360. URL http://mor.journal.informs.org/cgi/doi/10.1287/moor.1080.0360.
Papadaki, K. P., W. B. Powell. 2003a. A Discrete On–Line Monotone Estimation Algorithm. *Operations Research working papers, LSEOR 03.73*.

Papadaki, Katerina P., Warren B. Powell. 2002. Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem. *European Journal of Operational Research* **142**(1) 108–127. doi:10.1016/S0377-2217(01)00297-1. URL http://linkinghub.elsevier.com/retrieve/pii/S0377221701002971.

Papadaki, Katerina P., Warren B. Powell. 2003b. An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem. *Naval Research Logistics* **50**(7) 742–769. doi:10.1002/nav.10087. URL http://doi.wiley.com/10.1002/nav.10087.

Powell, Warren, Andrzej Ruszczynski, Huseyin Topaloglu. 2004. Learning Algorithms for Separable Approximations of Discrete Stochastic Optimization Problems. *Mathematics of Operations Research* **29**(4) 814–836. doi:10.1287/moor.1040.0107. URL http://mor.journal.informs.org/cgi/doi/10.1287/moor.1040.0107.

Powell, Warren B. 2011. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. 2nd ed. Wiley.

Puterman, Martin L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York.

Ramsay, J. O. 1998. Estimating smooth monotone functions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **60**(2) 365–375. doi:10.1111/1467-9868.00130. URL http://doi.wiley.com/10.1111/1467-9868.00130.

Ross, Sheldon M. 1983. *Introduction to Stochastic Dynamic Programming*. Academic Press, New York.

Smith, James E., Kevin F. McCardle. 2002. Structural Properties of Stochastic Dynamic Programs. *Operations Research* **50**(5) 796–809. doi:10.1287/opre.50.5.796.365. URL http://or.journal.informs.org/cgi/doi/10.1287/opre.50.5.796.365.

Stockey, N., R. E. Lucas, Jr. 1989. *Recursive Methods in Economic Dynamics*. Harvard University Press, Cambridge, Massachusetts and London, England.

Topaloglu, H., W. B. Powell. 2003. An Algorithm for Approximating Piecewise Linear Concave Functions from Sample Gradients. *Operations Research Letters* URL http://www.sciencedirect.com/science/article/pii/S0167637702001876.