Markov Decision Processes Lecture Notes 05

Value Iteration

Authors: Benjamin Van Roy *October 13, 2025*

1 Bellman Operators

Bellman operators offer concise notation for expressing Bellman's equation as well as steps of planning algorithms. For a fixed policy π, we have a Bellman operator $T_\pi : \mathbb{R}^{\mathcal{S}} \mapsto \mathbb{R}^{\mathcal{S}}$, defined by

$$(T_\pi V)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \right),$$

for all $s \in \mathcal{S}$. Then, there is the optimal Bellman operator $T : \mathbb{R}^{\mathcal{S}} \mapsto \mathbb{R}^{\mathcal{S}}$, defined by

$$(TV)(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \right).$$

Each of these operators also depends on the MDP $(\mathcal{S}, \mathcal{A}, P)$ and the discount factor $\gamma \in [0,1]$, though our notation does not explicitly convey that dependence.

With this notation, Bellman equations satisfied by discounted value functions can be written concisely as

$$V_{\pi,\gamma} = T_\pi V_{\pi,\gamma} \qquad \text{and} \qquad V_{*,\gamma} = T V_{*,\gamma}. \tag{1}$$

Similarly, for total value functions, we have

$$V_{\pi,\tau} = T_\pi V_{\pi,\tau-1} \qquad \text{and} \qquad V_{*,\tau} = T V_{*,\tau-1}. \tag{2}$$

for $\tau = 1, 2, \ldots, T$. We will discuss Bellman equations for gain/bias later.

2 Total Value Iteration

The value iteration algorithm generates value functions via a process sometimes referred to as *backward induction*. To produce total value functions, this involves initializing with $\tilde{V}_0 = 0$ and generating iterates $k = 1, 2, \ldots$ according to $\tilde{V}_{k+1} = T\tilde{V}_k$. Each iterate satisfies the Bellman equation (1) by construction. Hence, $V_{*,T} = \tilde{V}_T$. Algorithm 2 more concretely presents this procedure for computing $V_{*,0}, \ldots, V_{*,T}$.

Algorithm 1 total value iteration

```
$\tilde{V}_0(s) \leftarrow 0$ for all $s \in \mathcal{S}$
for $k = 1, \ldots, T$ do
    $\tilde{V}_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}}(r(s,a) + \sum_{s' \in \mathcal{S}} P_{ass'} \tilde{V}_k(s'))$ for all $s \in \mathcal{S}$
```

A similar procedure computes $V_{\pi,0}, \ldots, V_{\pi,T}$. In particular, initialize with $\tilde{V}_0 = 0$ and generate iterates $k = 1, 2, \ldots$ according to $\tilde{V}_{k+1} = T_\pi \tilde{V}_k$.

2.1 Example: Shortest Path

Let us work through the special case of a shortest path problem. Such a problem is characterized by an undirected graph $(\mathcal{V}, \mathcal{E})$, origin and destination nodes $v_{\text{orgn}}, v_{\text{dest}} \in \mathcal{V}$, and distances $d : \mathcal{E} \to [0, d_{\max}]$.

To formulate this in terms of an MDP $(\mathcal{S}, \mathcal{A}, P)$, let $\mathcal{S} = \mathcal{V}$, $\mathcal{A} = \mathcal{V}$, and $P_{ass'} = \mathbf{1}_a(s')$. Take the reward function to be

$$r(s,a) = \begin{cases} -d(s,a) + \mathbf{1}_a(v_{\text{dest}})|\mathcal{V}|d_{\max} & \text{if } (s,a) \in \mathcal{E}, s \neq v_{\text{dest}} \\ 0 & \text{if } s = a = v_{\text{dest}} \\ -\infty & \text{otherwise.} \end{cases}$$

Figure 1 provides an instance of a graph. Taking the origin and destination to be states 1 and 7, value iteration delivers the numbers in Table 1. The upper-leftmost cell provides the minimal total value starting at the initial state, which is 60. This the reward of 70 for reaching the destination minus the distances to node 3 then 4 then 7, which is $3 + 2 + 5 = 10$.

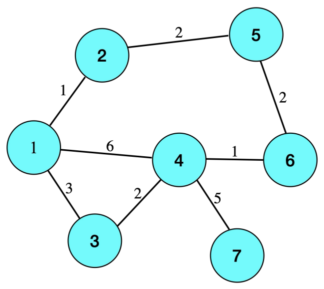

Figure 1: A shortest path problem.

state	$V_{*,7}$	$V_{*,6}$	$V_{*,5}$	$V_{*,4}$	$V_{*,3}$	$V_{*,2}$	$V_{*,1}$	$V_{*,0}$
1	60	60	60	60	60	59	-1	0
2	60	60	60	60	58	-2	-1	0
3	63	63	63	63	63	63	-2	0
4	65	65	65	65	65	65	65	0
5	62	62	62	62	62	-3	-2	0
6	64	64	64	64	64	64	-1	0
7	0	0	0	0	0	0	0	0

Table 1: Each column provides the optimal total value function for a particular horizon.

2.2 Example: Linear-Quadratic Control

The computational requirements per value iteration scale with the numbers of states and actions. However, special structure in some MDPs and rewards function enables dramatically more efficient computation. There are even some problems with infinite state and action spaces where value iteration can be efficiently applied. One example is linear-quadratic control.

While our point applies to linear-quadratic control more broadly, to keep our algebra simple, we will restrict attention in this section to scalar state and action variables. In particular, dynamics evolve according to

$$f(x, u, w) = ax + bu + w,$$

for fixed scalars $a, b \in \mathbb{R}$, where $w \sim \mathcal{N}(0, \sigma^2)$. we take the reward function to be

$$r(x, u) = -x^2 - cu^2,$$

where c is a positive scalars.

What enables an efficient implementation of value iteration in spite of the infinite state and action spaces is special structure retained by the value function. In particular, each kth iterate takes the form

$$V_k(x) = \sigma^2 \sum_{k'=0}^{k-1} \eta_{k'} + \eta_k x^2,$$

for some scalar η_k. To see why, note that $\eta_0 = 0$ and, for each k,

$$\begin{aligned}
V_{k+1}(x) =&(TV_k)(x) \\
=& \max_{u \in \mathbb{R}} \int \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} (r(x,u) + V_k(ax + bu + w)) dw \\
=& \max_{u \in \mathbb{R}} \int \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \left(-x^2 - cu^2 + \sigma^2 \sum_{k'=0}^{k-1} \eta_{k'} + \eta_k (ax + bu + w)^2 \right) dw \\
=& \sigma^2 \sum_{k'=0}^{k} \eta_{k'} - x^2 + \eta_k a^2 x^2 + \max_{u \in \mathbb{R}} ((\eta_k b^2 - c)u^2 + 2\eta_k abxu).
\end{aligned}$$

Setting the derivative to zero leads to

$$u = -\frac{\eta_k ab}{\eta_k b^2 - c} x,$$

and therefore,

$$\begin{aligned}
V_{k+1}(x) =& \sigma^2 \sum_{k'=0}^{k} \eta_{k'} - x^2 + \eta_k a^2 x^2 + (\eta_k b^2 - c) \left(\frac{\eta_k ab}{\eta_k b^2 - c} \right)^2 x^2 - 2 \frac{\eta_k^2 a^2 b^2}{\eta_k b^2 - c} x^2 \\
=& \sigma^2 \sum_{k'=0}^{k} \eta_{k'} - x^2 + a^2 \left(\eta_k - \frac{\eta_k^2 b^2}{\eta_k b^2 - c} \right) x^2 \\
=& \sigma^2 \sum_{k'=0}^{k} \eta_{k'} + \eta_{k+1} x^2,
\end{aligned}$$

for some η_{k+1}.

Given the special structure of V_k, we need to compute only the scalar η_k in each iteration. In particular,

$$\eta_{k+1} \leftarrow -1 + a^2 \left(\eta_k - \frac{\eta_k^2 b^2}{\eta_k b^2 - c} \right).$$

Hence, computational requirements do not scale with the number of states (which is infinite).

3 Discounted Value Iteration

To produce a discounted value function, we initializing with some $\tilde{V}_0$ and generating iterates $k = 1, 2, \ldots$ according to $\tilde{V}_{k+1} = T\tilde{V}_k$. As we will establish, if the discount factor is in $[0, 1)$ and there are finite numbers of states and actions, this sequence is guaranteed to converge on $V_{*,\gamma}$.

Algorithm 2 discounted value iteration

```
initialize with some $\tilde{V}_0$
for $k = 1, 2, 3, \ldots$ do
    $\tilde{V}_{k+1}(s) \leftarrow \max_{a\in\mathcal{A}}(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ass'}\tilde{V}_k(s'))$ for all $s \in \mathcal{S}$
```

A similar procedure computes $V_{\pi,0}, \ldots, V_{\pi,T}$. In particular, generate iterates $k = 1, 2, \ldots$ according to $\tilde{V}_{k+1} = T_\pi \tilde{V}_k$.

3.1 Convergence Analysis

Convergence follows from the fact that, with a discount factor $\gamma \in [0,1)$, the Bellman operator T is a contraction mapping with respect to the maximum norm.

Theorem 1. (contraction mapping) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0,1)$. For all $V, V' \in \mathbb{R}^{\mathcal{S}}$,*

$$\|TV - TV'\|_\infty \leq \gamma \|V - V'\|_\infty.$$

Proof. For all $s \in \mathcal{S}$,

$$\begin{aligned}(TV)(s) - (TV')(s) =& \max_{a\in\mathcal{A}}\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ass'}V(s')\right) - \max_{a\in\mathcal{A}}\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ass'}V'(s')\right) \\ \leq& \max_{a\in\mathcal{A}}\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ass'}V(s')\right) - \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{ass'}V'(s')\right) \\ =& \gamma \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P_{ass'}(V(s') - V'(s')) \\ \leq& \gamma \max_{a\in\mathcal{A}} \max_{s'\in\mathcal{S}} |V(s') - V'(s')| \\ =& \gamma \|V - V'\|_\infty.\end{aligned}$$

The result follows. □

And as we mentioned earlier the contraction property leads to a convergence result.

Theorem 2. (convergence) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0,1)$. For any $\tilde{V}_0$, the sequence $\tilde{V}_0, \tilde{V}_1, \tilde{V}_2, \cdots$ generated according to $\tilde{V}_{k+1} = T\tilde{V}_k$ converges on $V_{*,\gamma}$.*

Proof. Recall that $V_{*,\gamma} = TV_{*,\gamma}$. For each k, we have

$$\|V_{*,\gamma} - \tilde{V}_{k+1}\|_\infty = \|TV_{*,\gamma} - T\tilde{V}_k\|_\infty \leq \gamma \|V_{*,\gamma} - \tilde{V}_k\|_\infty.$$

It follows that

$$\|V_{*,\gamma} - \tilde{V}_{k+1}\|_\infty \leq \gamma^k \|V_{*,\gamma} - \tilde{V}_0\|_\infty,$$

and therefore, $\|V_{*,\gamma} - \tilde{V}_{k+1}\|_\infty$ vanishes. □

Analogous results pertain to T_π. We state them without proof.

Theorem 3. (contraction mapping) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor $\gamma \in [0,1)$, and a stationary policy π. For all $V, V' \in \mathbb{R}^{\mathcal{S}}$,*

$$\|T_\pi V - T_\pi V'\|_\infty \leq \gamma \|V - V'\|_\infty.$$

Theorem 4. (convergence) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor $\gamma \in [0,1)$, and a stationary policy π. For any $\tilde{V}_0$, the sequence $\tilde{V}_0, \tilde{V}_1, \tilde{V}_2, \cdots$ generated according to $\tilde{V}_{k+1} = T_\pi \tilde{V}_k$ converges on $V_{\pi,\gamma}$.*

3.2 Example: Queueing

Recall our queueing example from Lecture 04, which allows a maximum of twenty customers waiting in the queue. If a customer arrives when the queue is full, that customer abandons and this incurs a cost of 500. This is large relative to the cost of service, which is 30 for the fast mode and 0 for the slow mode. Each waiting customer incurs one unit of cost per time step.

Over each time step, a customer arrives with probability 0.5. If a customer is present then their service is completed over the next time step with probability 0.7 or 0.4, depending on whether the fast mode or slow mode of service is deployed.

Expected discounted return, with a discount factor of $\gamma = 0.99$, is maximized by a policy that uses the slow mode of service when there are no more than 12 waiting customers and, otherwise, the fast mode.

Figure 2 plots the value function $\tilde{V}_k$ produced for values of k that are multiples of 25. We initialize with $\tilde{V}_0 = 0$ and all rewards are negative, so the values decrease as k increases. The sequence appears to converge, as differences between iterates are vanishing.

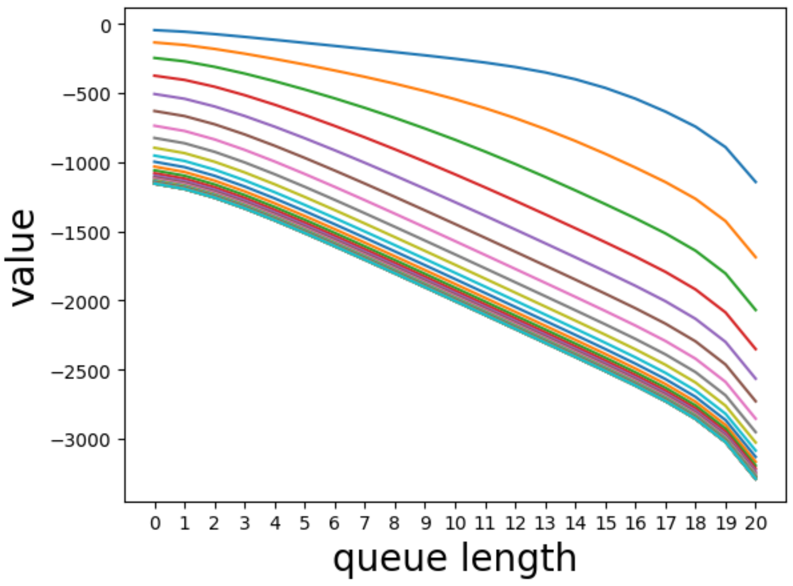

Figure 2: Optimal discounted value functions for a queueing system.

3.3 Example: Linear-Quadratic Control

We considered ealier in these notes total value iteration for a simple linear-quadratic control problem. Iterates took the form $\tilde{V}_k = \sigma^2 \sum_{k'=0}^{k-1} \eta_{k'} + \eta_k x^2$, with η_k updated according to

$$\eta_{k+1} \leftarrow -1 + a^2 \left(\eta_k - \frac{\eta_k^2 b^2}{\eta_k b^2 - c} \right).$$

Discounted value iteration generate iterates of the form $\tilde{V}_k = \sigma^2 \sum_{k'=0}^{k-1} \gamma^{k-k'} \eta_{k'} + \eta_k x^2$, with a modified update formula:

$$\eta_{k+1} \leftarrow -1 + \gamma a^2 \left(\eta_k - \frac{\eta_k^2 b^2}{\eta_k b^2 - c/\gamma} \right).$$

One may be tempted to solve a fixed point equation of the form

$$\eta_* = -1 + \gamma a^2 \left(\eta_* - \frac{\eta_*^2 b^2}{\eta_* b^2 - c/\gamma} \right) \tag{3}$$

to obtain the optimal discounted value function. Indeed, there is a scalar η_* that solves this equation for which

$$V_{*,\gamma}(x) = \frac{\gamma \sigma^2 \eta_*}{1-\gamma} + \eta_* x^2.$$

However, Equation (3) is equivalent to a quadratic equation and typically has two solutions, one positive and the other negative. It is the negative solution that provides the parameter η_* for the optimal discounted value function.

This example highlights technical complexities that can arise with infinite state spaces. While much insight developed for finite state spaces extends to infinite state spaces, the extensions often require technical assumptions.

4 Distributed and Asynchronous Computation

Once useful feature of discounted value iteration is that it can be applied in a distributed and asynchronous manner. Suppose, for example, you have access to a large number of CPUs. Say, enough to allocate one per state. Then, you could apply each CPU to update the value of one state according to $V(s) := (TV)(s)$. Even if some CPUs update more frequently than others, this mode of asynchronous value iteration will converge on the optimal value function. The following theorem captures this robustness of value iteration to asynchronous value iteration.

Theorem 5. (convergence) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$. If $\mathcal{S}_0, \mathcal{S}_1, \dots$ is a sequence of subsets of $\mathcal{S}$ such that each element $s \in \mathcal{S}$ appears infinitely often then, for any $\tilde{V}_0$, the sequence $\tilde{V}_0, \tilde{V}_1, \tilde{V}_2, \cdots$ generated according to*

$$V_{k+1}(s) = \begin{cases} (TV_k)(s) & s \in \mathcal{S}_k \\ V_k(s) & s \notin \mathcal{S}_k. \end{cases} \tag{4}$$

converges on $V_{*,\gamma}$.

Proof. Recall that $V_{*,\gamma} = TV_{*,\gamma}$. Since T is a contraction mapping, for each V and $s \in \mathcal{S}$,

$$|V_{*,\gamma}(s) - (TV)(s) \leq \gamma \|V_{*,\gamma} - V\|_\infty.$$

It follows that, for all $s \in \mathcal{S}_k$,

$$|V_{*,\gamma}(s) - V_{k+1}(s)| \leq \gamma \|V_{*,\gamma} - V_k\|_\infty.$$

Because each state appears infinitely often in the sequence $\mathcal{S}_0, \mathcal{S}_1, \dots$, we can choose indices $k_0 = 0, k_1, k_2, k_3, \dots$ such that within each interval $k_i, \dots, k_{i+1} - 1$, each state appears at least once. It is easy to see then that, for all i and $s \in \mathcal{S}$

$$|V_{*,\gamma}(s) - V_{k_{i+1}}(s)| \leq \gamma \|V_{*,\gamma} - V_{k_i}\|_\infty.$$

The result follows. □

This result can be generalized to show that value iteration is tolerant to communication delays. In particular, if it takes time to relay updated values from one CPU to another so that each uses outdated values, value iteration still converges on the optimal discounted value function. Bertsekas and Tsitsiklis [1989] elaborate on this and provides more comprehensive theoretical results.

Even with a single processor, asynchronous application of value iteration can accelerate convergence. A special case of this is Gauss-Seidel value iteration, which updates a the value function at a single state at a time. To contrast, the standard version of value iteration that updates values at all states in parallel is called Gauss-Jacobi value iteration.

Figure 3 plots iterates generated by Gauss-Seidel versus Gauss-Jacobi value iteration. The latter makes more progress with each sweep through the state space.

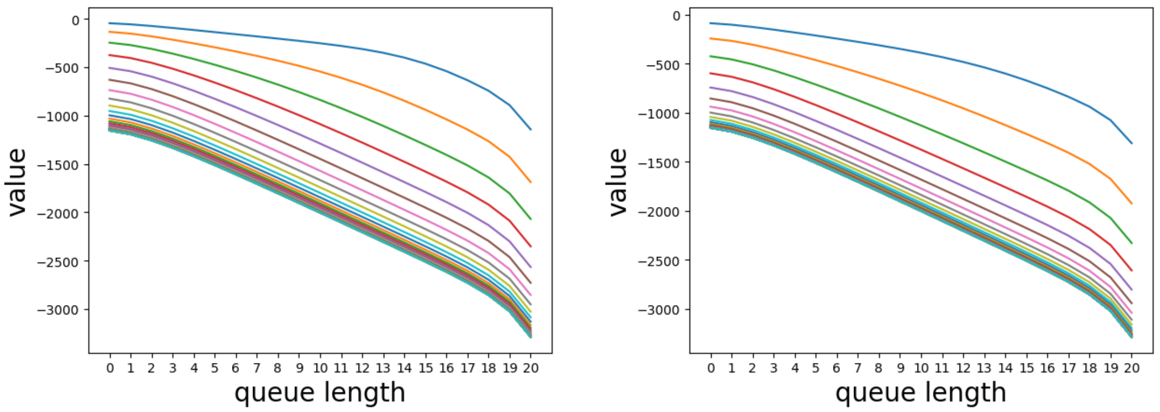

Figure 3: Gauss-Jacobi (left) vs Gauss-Seidel (right) value iteration for the queueing system. Iterates 1 through 1000, with every 25th iterate plotted.

For some problems, the difference in convergence rate between Gauss-Seidel and Gauss-Jacobi value iteration can be dramatic. An example of this is the river swim environment, illustrated in Figure 4. In this environment, a large reward is earned when arriving at the rightmost state. But a small cost is incurred for each rightward transition.

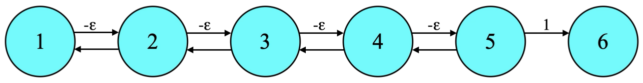

Figure 4: A river swim environment.

Figure 5 plots iterates produced by Gauss-Seidel and Gauss-Jacobi value iteration. The former converges after one sweep through the state space. The latter requires as many iterations as there are states.

5 Gain and Bias via Value Iteration

There are versions of value iteration designed to compute gain and bias functions. One version, which is studied in [Bertsekas, 2011], generates a sequence of total value functions:

$$J_{k+1} \leftarrow TJ_k.$$

Then, gain and bias functions are given by

$$\tilde{\lambda}_{k+1} = J_{k+1} - J_k \qquad \text{and} \qquad \tilde{V}_k = J_k - k\tilde{\lambda}_k.$$

Figure 6 plots gain and bias functions generated in this way for the queueing system.

As established in [Bertsekas, 2011], when periodicity is not a concern, this leads to a gain/bias optimal policy.

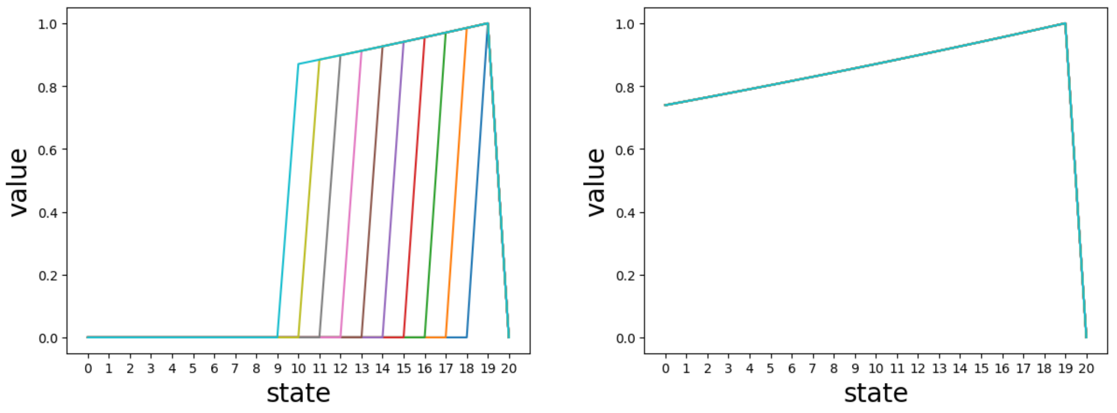

Figure 5: Gauss-Jacobi (left) vs Gauss-Seidel (right) value iteration for river swim. Iterates 1 through 10.

Theorem 6. (convergence) *Fix a finite-state finite-action MDP* $(\mathcal{S}, \mathcal{A}, P)$*, a reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$*, and an initialization* $J_0 \in \mathbb{R}^{\mathcal{S}}$*. Let* $J_1, J_2, \ldots$ *be a sequence such that* $J_{k+1} = TJ_k$ *for all* k*. Let* $\tilde{V} = \lim_{k\to\infty}(J_k - k\lambda_*)$*. If every gain optimal stationary policy is aperiodic then* $\tilde{\lambda} = \lim_{k\to\infty}(J_{k+1} - J_k)$ *and* $\tilde{V} = \lim_{k\to\infty}(J_k - k\lambda_*)$ *satisfy the optimality equations:*

$$\lambda(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P_{ass'}\lambda(s') \qquad \forall s \in \mathcal{S} \tag{5}$$

$$V(s) = \max_{a\in\mathcal{A}_*(s)} \left(r(s,a) - \lambda_*(s) + \sum_{s'\in\mathcal{S}} P_{ass'}V(s') \right) \qquad \forall s \in \mathcal{S}, \tag{6}$$

where $\mathcal{A}_*(s)$ *is the set of actions that attains the maximum in (5).*

It follows from this result that $\tilde{\lambda} = \lambda_*$ and any policy that attains that maximum in (6) is gain and bias optimal.

It would be nice to have a version of value iteration that works even in the face of periodicity. While we haven't seen an analysis of this algorithm in the literature, we conjecture that the following will do the job. First, let $\gamma_0, \gamma_1, \gamma_2, \ldots$ be a decreasing sequence of scalars in $(0, 1]$ for which $\sum_k \gamma_k = \infty$ and $\sum_k \gamma_k^2 < \infty$. Then, let

$$J_{k+1} \leftarrow \gamma_k J_k + (1 - \gamma_k)TJ_k,$$

and

$$\tilde{\lambda}_{k+1} = J_{k+1} - J_k \qquad \text{and} \qquad \tilde{V}_k = J_k - k\tilde{\lambda}_k.$$

References

Dimitri Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena scientific, 2011.

D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, 1989.

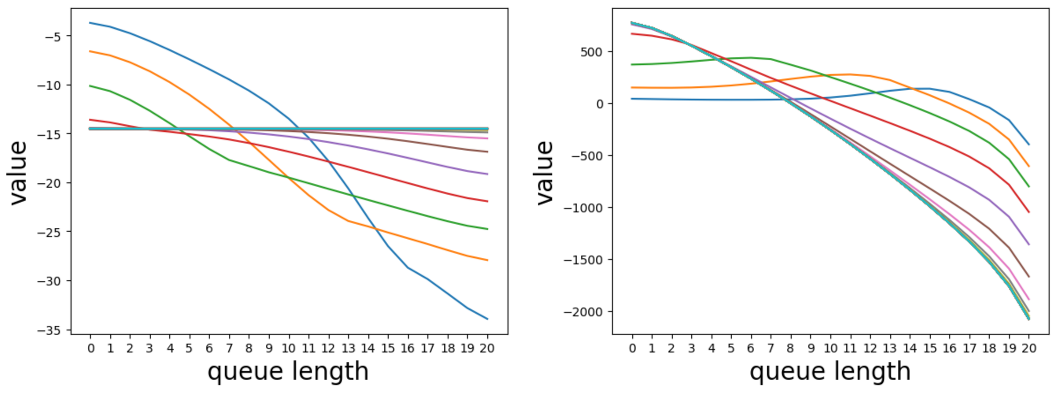

Figure 6: Gain and bias functions for a queueing system. Every 25th iterate is plotted.