Markov Decision Processes Lecture Notes 06

Policy Iteration

Authors: Benjamin Van Roy *October 20, 2025*

1 Policy Evaluation

Consider an MDP $(\mathcal{S}, \mathcal{A}, P)$, reward function r, and discount factor $\gamma \in [0, 1)$. Recall from Theorem 5 of Lecture 04 that, for any stationary policy π, $V_{\pi,\gamma}$ is the unique solution to the system of equations

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \right) \qquad \forall s \in \mathcal{S}. \tag{1}$$

This is a system of *linear* equations and can be written more concisely by defining a matrix P_π and vector r_π. In particular, let elements of the matrix be $P_{\pi ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) P_{ass'}$ and components of the vector be $r_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) r(s, a)$. Then the linear system of equations (1) can be written as

$$V = r_\pi + \gamma P_\pi V. \tag{2}$$

The process of computing $V_{\pi,\gamma}$ is sometimes called *policy evaluation*. This is because $V_{\pi,\gamma}$ expresses the performance of the policy π for each possible starting state. The function $V_{\pi,\gamma}$ is the unique solution to (2) and hence can be computed by solving a system of $|\mathcal{S}|$ linear equations. This can be done by inverting a matrix to obtain $V_{\pi,\gamma} = (I - \gamma P_\pi)^{-1} r_\pi$ or by more computationally efficient algorithms for solving linear systems of equations.

Figure 1 plots two value functions for the queueing example of Lecture 04, which allows up to twenty customers to wait in the queue. These functions are obtained by solving (2) for two different policies: one that only ever uses the fast mode of service and one that only ever uses the slow mode of service.

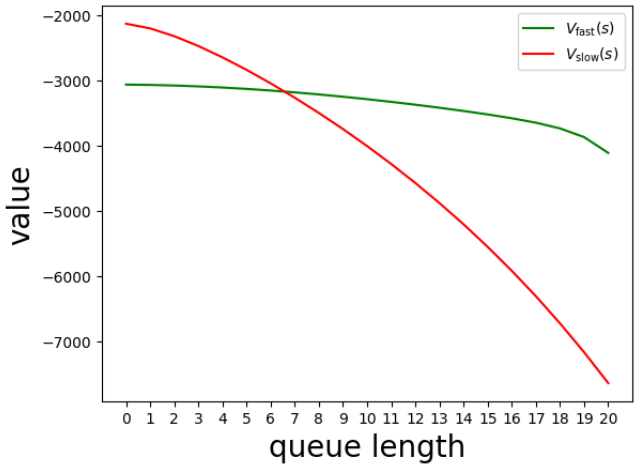


Figure 1: Discounted value functions for fast-only and slow-only policies in the queueing system.

2 Policy Iteration

In principle, one could find an optimal policy by evaluating each stationary deterministic policy and choosing one that dominates all others. For example, we could populate Figure 1 with all such curves, and one will attain the maximum among curves simultaneously across states. However, the computational requirements of this process would be prohibitive, even for state spaces of modest size.

Policy iteration (Algorithm 1) provides a more efficient process of searching through policies. While value iteration is initialized with a value function, policy iteration is initialized with a policy. Each iteration entails first evaluating the iterate π_k to obtain the value function $V_{\pi_k,\gamma}$. Then, the next iterate π_{k+1} is chosen to be greedy with respect to $V_{\pi_k,\gamma}$.

Algorithm 1 discounted policy iteration

choose an initial policy π_0
for $k = 0, \ldots, K-1$ **do**
  evaluate $V_{\pi_k,\gamma}$
  let π_{k+1} be a deterministic stationary policy that is greedy with respect to $V_{\pi_k,\gamma}$

Figure 2 plots value functions generated by applying policy iteration to the queueing system. The initial policy is chosen to be the one that always uses the slow mode of service. Each policy π_k turns out to be characterized by a threshold: the slow mode of service is chosen when the queue length is less than the threshold and the fast mode when the queue length equals or exceeds the threshold. As can be seen from the plot of thresholds, the policy converge in just six iterations. It is common in practical problems for policy iteration to generate near-optimal policies within a small number of iterations. However, relative to value iteration, the computation required per iteration is much larger.

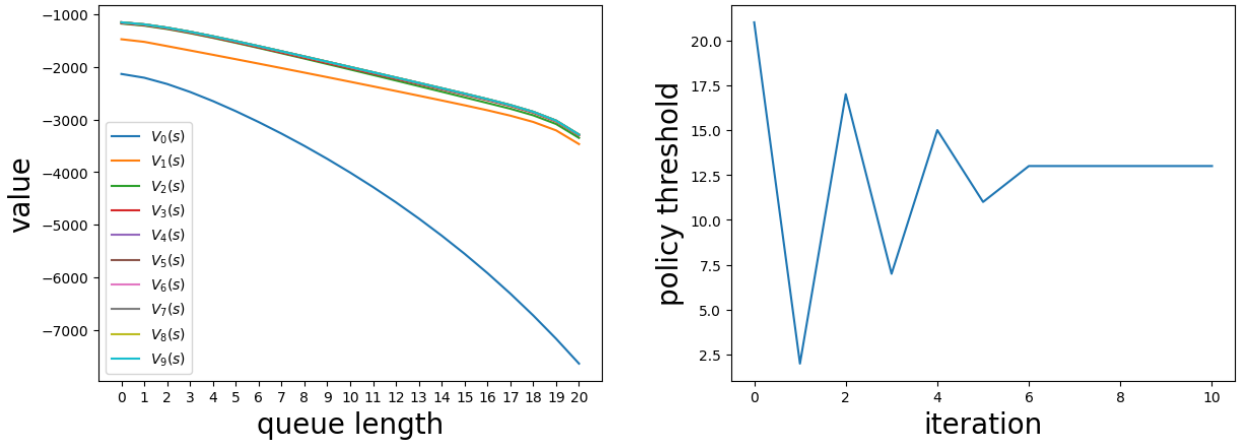


Figure 2: Policy iteration for the queueing system. The threshold indicates the minimum queue length at which the fast mode of service is applied

Let us now develop a deeper understanding of why each policy iteration ought to improve a policy and why this process converges. To do so, we first establish an abstract property of Bellman operators.

Theorem 1. (monotonicity of T) *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 $V, V' \in \mathbb{R}^{\mathcal{S}}$, if $V \leq V'$ then $TV \leq TV'$.*

Proof. First note that, for all $a \in \mathcal{A}$ and $s, s' \in \mathcal{S}$, $\gamma P_{ass'} V(s') \leq \gamma P_{ass'} V'(s')$ because the discount factor γ and the probability $P_{ass'}$ are nonnegative. Hence, $\gamma \sum_{s' \in \mathcal{S}} P_{ass'} V(s') \leq \gamma \sum_{s' \in \mathcal{S}} P_{ass'} V'(s')$. It follows

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

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

The result follows. □

An analogous result applies to T_π. We omit the proof, which would be very similar to the proof of Theorem 1.

Theorem 2. (monotonicity of T_π) *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 policies π and $V, V' \in \mathbb{R}^{\mathcal{S}}$, if $V \leq V'$ then $T_\pi V \leq T_\pi V'$.*

Each iteration of the policy iteration algorithm applies a policy improvement step. The policy improvement theorem establishes that each policy improvement step can not reduce the expected discounted return from any state and, unless the policy is already optimal, improves the expected discounted return from one or more states.

Theorem 3. (policy improvement) *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 stationary policies π, if π' is greedy with respect to $V_{\pi,\gamma}$ then*

1. *$V_{\pi,\gamma} \leq V_{\pi',\gamma}$,*
2. *if $V_{\pi,\gamma} = V_{\pi',\gamma}$ then $V_{\pi,\gamma} = V_{*,\gamma}$.*

Proof. We drop γ from the subscripts to reduce clutter. Note that

$$V_\pi = T_\pi V_\pi \leq T V_\pi = T_{\pi'} V_\pi,$$

where the first equality follows from Bellman's equation, the inequality follows from monotonicity, and the final equality follows from the fact that π' is greedy with respect to π. It follows from that

$$V_\pi \leq T_{\pi'} V_\pi \leq T_{\pi'}^2 V_\pi \leq T_{\pi'}^3 V_\pi \leq \cdots \leq V_{\pi'},$$

where the inequalities starting with the second follow from monotonicity. Further, if $V_\pi = V_{\pi'}$ then this sequence of inequalities implies that $V_\pi = T_{\pi'} V_\pi = T V_\pi$. The equation $V_\pi = T V_\pi$ is the Bellman equation, whose unique solution is V_*. Hence, if $V_\pi = V_{\pi'}$ then $V_\pi = V_*$. □

We can use the policy improvement theorem to establish finite-iteration convergence of policy iteration.

Theorem 4. (policy iteration) *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 stationary policy π_0, there exists a finite k such that $V_{\pi_k,\gamma} = V_{*,\gamma}$.*

Proof. We drop γ from the subscripts to reduce clutter. By the policy improvement theorem, for all k, $V_{\pi_k} \leq V_{\pi_{k+1}}$ and, unless $V_{\pi_k} = V_*$, $V_{\pi_k} \neq V_{\pi_{k+1}}$. It follows that V_{π_k} is a monotonically nondecreasing sequence of functions and that, if the sequence converges, it converges to V_*. Since each policy π_k is a deterministic stationary policy and there are a finite number of deterministic stationary policies, the sequence must converge. The result follows. □

3 Approximate Policy Iteration

When an MDP has too many states – take the game of Tetris, for example – it is not possible to store one value per state. In such cases, one might instead approximate the value function using a parameterized family of functions. In particular, suppose $\tilde{V}_\theta : \mathcal{S} \to \mathbb{R}$ is a function parameterized by a vector $\theta \in \mathbb{R}^K$. Then, while we can not store a value function $V_{\pi,\gamma}$, we might approximate it reasonably well by selecting a suitable vector θ such that $\tilde{V}_\theta \approx V_{\pi,\gamma}$.

Algorithm 2 presents an approximate version of policy iteration that generates iterates via approximate policy evaluation. This leaves open the question of how θ is chosen in each iteration. We will discuss that further later.

Algorithm 2 discounted approximate policy iteration

choose an initial policy π_0
for $k = 0, \ldots, K-1$ **do**
  identify θ such that $\tilde{V}_\theta \approx V_{\pi_k,\gamma}$
  let π_{k+1} be a deterministic stationary policy that is greedy with respect to $\tilde{V}_\theta$

Figure 3 plots iterates generated from applying approximate policy iteration to our queueing system. The parameterized family of functions is taken to be

$$\tilde{V}_\theta(s) = \theta_0 + \theta_1 s + \theta_2 s^2.$$

This affords a quadratic approximation to any value function. Each iterate in the figure is generated by minimizing squared error:

$$\theta \in \arg\min_{\theta \in \mathbb{R}^3} \sum_{s \in \mathcal{S}} (V_{\pi_k,\gamma}(s) - \tilde{V}_\theta(s))^2.$$

The value functions converge. The policy threshold converges but to a value of 9, which differs from the optimal value. As shown in Figure 2, the optimal threshold is 13. That being said, the performance of a policy with threshold of 9 is almost as good as one with a threshold of 13. More broadly, approximate policy iteration with suitable parameterized functions often can generate effective policies.

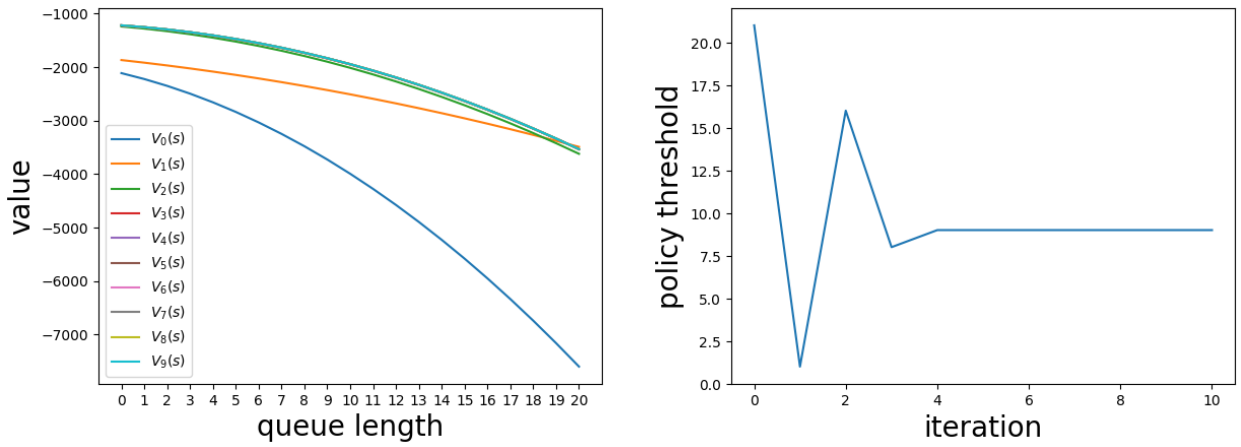


Figure 3: Approximate policy iteration for the queueing system, with quadratic function approximation. The threshold indicates the minimum queue length at which the fast mode of service is applied

The aforementioned implementation of approximate policy iteration requires minimizing a sum of squared errors across the state space. While this results in a value function that can be encoded in terms of only a

few parameters, the computational requirements scale with the number of states. This becomes prohibitive when the state space is large, as is the case, for example, in the game of Tetris.

In Lecture 04, we introduced rollouts as a way of avoiding this dependence of compute on the number of states. But rollouts, especially if they are iterated, give rise to prohibitive online computational requirements. By *online* here we mean computation that must occur as actions are selected and executed in a real environment.

Rollouts can, however, be combined with approximate policy iteration to offer an algorithm that can be applied effectively with large state spaces. In particular, consider using rollouts in each policy evaluation step as follows. First, randomly sample N initial states. Then, from each of these states, simulate a rollout $S_0^n, A_0^n, S_1^n, A_1^n, \ldots$ with actions selected by π_k. Then, solve a least squares problem of the form

$$\theta \in \arg\min_{\theta \in \mathbb{R}^3} \sum_{n=1}^{N} \left(\sum_{t=0}^{\infty} \gamma^t r(S_t^n, A_t^n) - \tilde{V}_\theta(S_0^n) \right)^2.$$

As N grows, θ converges to a vector that minimized $\mathbb{E}_{\pi_k}[(V_{\pi_k,\gamma}(S_0) - \tilde{V}_\theta(S_0))^2]$. This procedure may require many simulations (large N) and thus a lot of computation. The good news, however, is that the number of simulations required does not grow with the number of states $|\mathcal{S}|$.

4 Gain/Bias

The concepts of policy evaluation, policy improvement, and policy iteration extend naturally from expected discounted return to gain and bias.

4.1 Policy Evaluation

Evaluation amounts to computing the gain and bias of a policy π:

$$\lambda_\pi(s) = \lim_{T \to \infty} \mathbb{E}_{s,\pi} \left[\frac{1}{T} \sum_{t=0}^{T-1} r(S_t, A_t) \right] \tag{3}$$

$$V_\pi(s) = \lim_{\gamma \uparrow 1} \mathbb{E}_{s,\pi} \left[\sum_{t=0}^{\infty} \gamma^t (r(S_t, A_t) - \lambda_\pi(S_t)) \right]. \tag{4}$$

4.1.1 Unichain Case

Let us first consider computation of these functions for a unichain MDP. That is, an MDP for which the Markov process induced by any stationary policy has a single recurrent class. In this case, $\lambda_\pi(s)$ does not depend on s, and thus we can treat λ_π as a scalar rather than a function.

One might consider solving the equation

$$V = T_\pi V - \lambda \mathbf{1}.$$

Each solution takes the form $(\lambda_\pi, V_\pi + \alpha \mathbf{1})$ for some $\alpha \in \mathbb{R}$. Any value of α provides a solution. The reason for this multiplicity of solutions is that $P_\pi \mathbf{1} = \mathbf{1}$, and therefore, if (V, λ) solves $V = T_\pi V - \lambda \mathbf{1}$ then

$$V + \alpha \mathbf{1} = T_\pi V + \alpha P_\pi \mathbf{1} - \lambda \mathbf{1} = T_\pi (V + \alpha \mathbf{1}) - \lambda \mathbf{1}.$$

A solution of the form $(\lambda_\pi, V_\pi + \alpha \mathbf{1})$ suffices for policy improvement because the constant shift of $\alpha \mathbf{1}$ does not impact choice of a greedy policy. However, one can also hone in on (λ_π, V_π) by instead solving an augmented linear system of equations, as presented in the following result.

Theorem 5. *Fix a finite-state finite-action unichain MDP $(\mathcal{S}, \mathcal{A}, P)$ and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. For any stationary policy π and solution $(\lambda \in \mathbb{R}, V \in \mathbb{R}^{\mathcal{S}}, J \in \mathbb{R}^{\mathcal{S}})$ to*

$$V = T_\pi V - \lambda \mathbf{1} \tag{5}$$
$$V = (I - P_\pi) J, \tag{6}$$

$\lambda = \lambda_\pi$ and $V = V_\pi$.

Equation (6) serves to rule out $V_\pi + \alpha \mathbf{1}$ for $\alpha \neq 0$. To see why, note that $I - P_\pi$ has a right eigenvector of $\mathbf{1}$ with eigenvalue 0. Hence, the range of $I - P_\pi$ rules out constant offsets. At the same time, V_π is within the range because,

$$(I - P_\pi) V_\pi = (I - P_\pi) \lim_{\gamma \uparrow 1} \sum_{t=0}^{\infty} \gamma^t P_\pi^t (r_\pi - \lambda_\pi \mathbf{1}) = r_\pi - \lambda_\pi \mathbf{1},$$

and it follows that

$$(I - P_\pi) \sum_{t=0}^{\infty} P_\pi^t V_\pi = \sum_{t=0}^{\infty} P_\pi^t (I - P_\pi) V_\pi = \lim_{\gamma \uparrow 1} \sum_{t=0}^{\infty} \gamma^t P_\pi^t (r_\pi - \lambda_\pi \mathbf{1}) = V_\pi.$$

Hence, $(\lambda_\pi, V_\pi, \sum_{t=0}^{\infty} P_\pi^t V_\pi)$ solves (5-6).

4.1.2 Multichain Case

Let us now relax the unichain assumption so that λ_π is a function of state. In this case, λ_π and V_π can be obtained by solving further augmented linear system of equations, as presented in the next result.

Theorem 6. *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$ and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. For any stationary policy π and solution $(\lambda \in \mathbb{R}^{\mathcal{S}}, V \in \mathbb{R}^{\mathcal{S}}, J \in \mathbb{R}^{\mathcal{S}})$ to*

$$\lambda = P_\pi \lambda \tag{7}$$
$$V = T_\pi V - \lambda \tag{8}$$
$$V = (I - P_\pi) J. \tag{9}$$

$\lambda = \lambda_\pi$ and $V = V_\pi$.

Equation (7) ensures that the gain from any state is equal to the expectation of the gain starting at the next state. This equation was not needed for the unichain case because it is satisfied by all constant functions and therefore would not have imposed any additional constraint on the set of solutions.

4.2 Policy Improvement

The following result pertains to a policy improvement step that applies to multichain MDPs.

Theorem 7. (policy improvement) *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 stationary policy π. For each $s \in \mathcal{S}$, let $\mathcal{A}_*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ass'} \lambda_\pi(s')$. Choose a deterministic stationary policy π' such that*

$$\pi'(s) \in \max_{a \in \mathcal{A}_*(s)} \left(r(s,a) - \lambda_\pi(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V_\pi(s') \right),$$

with preference for $\pi'(s) = \pi(s)$. If $\pi' \neq \pi$ then one of the following to statements holds:

1. *$\lambda_\pi \leq \lambda_{\pi'}$ and there exists $s \in \mathcal{S}$ such that $\lambda_\pi(s) < \lambda_{\pi'}(s)$.*
2. *$\lambda_\pi = \lambda_{\pi'}$, $V_\pi \leq V_{\pi'}$, and there exists $s \in \mathcal{S}$ such that $V_\pi(s) < V_{\pi'}(s)$.*

Unichain MDPs can be viewed a special case where the gain function λ_π does not depend on the state. In that case, the policy improvement step simplifies because, for all $s \in \mathcal{S}$, $\mathcal{A}_*(s) = \mathcal{A}$.

4.3 Policy Iteration

Repeated application of policy evaluation and improvement gives us a policy iteration algorithm.

Algorithm 3 gain-bias policy iteration

choose an initial policy π_0
for $k = 0, \ldots, K-1$ **do**
evaluate $(\lambda_{\pi_k}, V_{\pi_k})$
for $s \in \mathcal{S}$, $\mathcal{A}_*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ass'} \lambda_{\pi_k}(s')$
choose a deterministic stationary policy π_{k+1} such that

$$\pi_{k+1}(s) \in \max_{a \in \mathcal{A}_*(s)} \left(r(s,a) - \lambda_{\pi_k}(s) + \sum_{s' \in \mathcal{S}} P_{ass'} V_{\pi_k}(s') \right)$$

with preference for $\pi_{k+1}(s) = \pi_k(s)$

Theorem 8. (policy iteration) *Fix a finite-state finite-action MDP $(\mathcal{S}, \mathcal{A}, P)$ and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. For any initial stationary policy π_0, there exists a finite k such that iterates generated by Algorithm 3 satisfy $\lambda_{k+1} = \lambda_k$.*

Together with Theorem 7, the fact that there are a finite number of deterministic stationary policies implies that there is some k after which subsequent iterates are all equal to π_k. It is easy to see that under this policy π_k, the gain λ_{π_k} and bias V_{π_k} satisfy the Bellman equations of Theorem 9 from Lecture 04 notes. It follows from that theorem that $\lambda_{\pi_k} = \lambda_*$ and π_k is gain-optimal.

Figure 4 plots bias functions generated by applying policy iteration to the queueing system, as well as associated thresholds. The initial policy is chosen to be the one that always uses the slow mode of service. Note that bias functions do not progress monotonically. But, as show in Figure 4, the bias functions do.

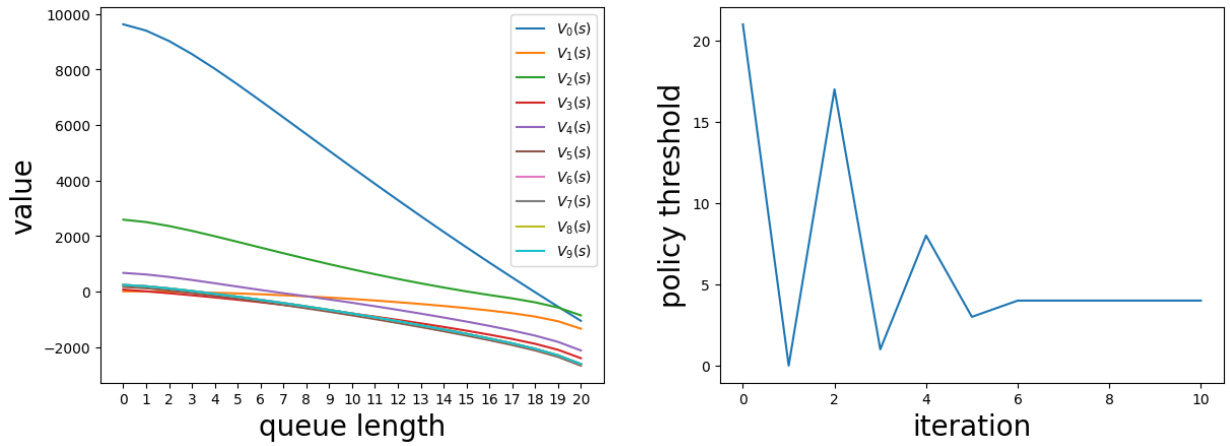
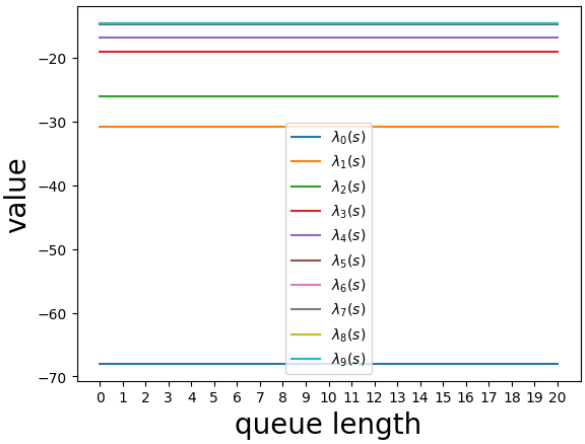


Figure 4: Bias functions from policy iteration for the queueing system. The threshold indicates the minimum queue length at which the fast mode of service is applied


value
queue length
−20
−30
−40
−50
−60
−70
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
$\lambda_0(s)$
$\lambda_1(s)$
$\lambda_2(s)$
$\lambda_3(s)$
$\lambda_4(s)$
$\lambda_5(s)$
$\lambda_6(s)$
$\lambda_7(s)$
$\lambda_8(s)$
$\lambda_9(s)$