Markov Decision Processes and Reinforcement Learning

Day 2 — Optimal Value Functions and Classical Computational Methods

Luiz Felipe Martins

Department of Mathematics and Statistics
Cleveland State University

January 27, 2026

Outline

1. Review of Previous Presentation
2. Optimal value functions
3. Tabular Solution Methods
   - Dynamic Programming

Definition of Markov Decision Process

- A trajectory of an MDP consists of three stochastic sequences:
  - $S_0, S_1, S_2, \ldots$: the states visited by the environment.
  - $A_1, A_2, A_2, \ldots$ the actions chosen by the agent.
  - $R_0, R_1, R_2, \ldots$ the rewards awarded to the agent.
- Model is specified by two deterministic functions:

$$p(s' \mid s, a) \quad \text{and} \quad R(s, s', a).$$

- Model evolution:

$$\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) = p(s' \mid s, a)$$

$$R_{t+1} = r(S_t, S_{t+1}, A_t)$$

Rewards, returns and state value functions

- The **reward** received at time $t+1$ (after action A_t is selected) is denoted by R_{t+1}.
- The **total return** (or simply return) for a run is:

$$\sum_{t=0}^{T} \gamma^t R_{t+1}$$

- The expected return of a run is:

$$\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_{t+1}\right]$$

Policies and value functions

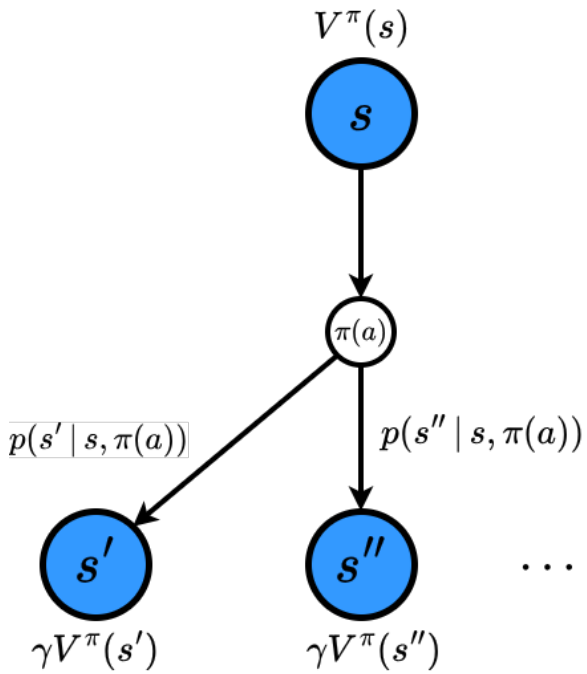
- A **deterministic policy** associates to each state s and action $a = \pi(s)$, where a is in the set of admissible actions $\mathscr{A}(s)$
- The state value function for policy π is:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^{t} R_{t+1} \,|\, S_0 = s\right]$$

- We assume:
  - For episodic tasks, $P(T < \infty) = 1$ and $0 < \gamma \leq 1$.
  - For continuing tasks, $T = \infty$ and $0 < \gamma < 1$.

Bellman equation for a deterministic policy

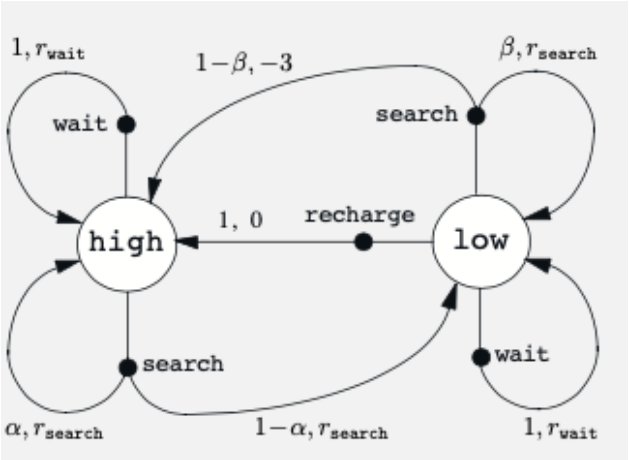
Backup diagram:



$$V^\pi(s) = \sum_{s' \in \mathcal{S}} p(s', s \mid \pi(a))(r(s, s', \pi(a)) + \gamma V^\pi(s'))$$

For episodic tasks, set $V^\pi(s) = 0$ if s is a terminal state

Value function computation example



- Policy: $\pi(\texttt{high}) = \texttt{search}, \quad \pi(\texttt{low}) = \texttt{search}$
- Bellman equations:

$$V^\pi(\texttt{h}) = \alpha(r_\texttt{s} + \gamma V^\pi(\texttt{h})) + (1-\alpha)(r_\texttt{s} + \gamma V^\pi(\texttt{l}))$$
$$V^\pi(\texttt{l}) = (1-\beta)(-3 + \gamma V^\pi(\texttt{h})) + \beta(r_\texttt{s} + \gamma V^\pi(\texttt{l}))$$

Solving the linear system

$$V^\pi(\mathtt{h}) = \alpha(r_\mathtt{s} + \gamma V^\pi(\mathtt{h})) + (1-\alpha)(r_\mathtt{s} + \gamma V^\pi(\mathtt{l}))$$
$$V^\pi(\mathtt{l}) = (1-\beta)(-3 + \gamma V^\pi(\mathtt{h})) + \beta(r_\mathtt{s} + \gamma V^\pi(\mathtt{l}))$$

Rewrite in "linear algebra" style:

$$(I - \gamma P)V = b$$

Where:

$$P = \begin{bmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{bmatrix}, \quad \begin{bmatrix} r_s \\ -3(1-\beta) + r_s\beta \end{bmatrix}$$

Numerical solution

Parameters:

$$\alpha = 0.8, \quad \beta = 0.3$$

$$r_{\texttt{search}} = 15, \quad r_{\texttt{wait}} = 10, \quad r_{\texttt{empty}} = -3.0$$

For $\gamma = 0.9$:

$$A = I - \gamma P = \begin{bmatrix} 0.28 & -0.18 \\ -0.63 & 0.73 \end{bmatrix}, \quad b = \begin{bmatrix} 15. \\ 2.4 \end{bmatrix}$$

Solution:

$$\begin{bmatrix} V(\texttt{high}) \\ V(\texttt{low}) \end{bmatrix} = \begin{bmatrix} 125.07692308 \\ 111.23076923 \end{bmatrix}$$

Solution strategies for a linear system

$$V = b + \gamma QV$$

- **Linear algebra solution**. Write the system in the form:

$$(I - \gamma P)V = b \quad \rightarrow \quad V = (I - \gamma P)^{-1} b$$

  Don't use inversion, use an LU decomposition (Gaussian elimination).
- **Gauss-Jacobi Iteration**.
  - Choose V_0 arbitrarily.
  - Iterate:

$$V_{n+1} = b + \gamma QV_n$$

- This is an example of **fixed point iteration**.

Jacobi iteration for the RR model

```
Initial approximation: [0. 0.]
Step 1: [15.   2.4]
Step 2: [26.232 12.498]
Step 3: [36.13668 22.30062]
Step 4: [45.0325212 31.1872758]
Step 5: [53.03712491 39.19105282]
.................
Step 99: [125.07332253 111.22716869]
Step 100: [125.07368259 111.22752874]
```

Gauss-Seidel Iteration

- Similar to Jacobi iteration, but *use the already updated values in each step*:
  - Initialize V_0 arbitrarily.
  - Iterate:

$$V_{n+1}(1) = b_1 + \gamma(p_{11}V_n(1) + p_{12}V_n(2)]$$
$$V_{n+1}(2) = b_2 + \gamma(p_{21}V_{n+1}(1) + p_{12}V_n(2)$$

- Results in slightly faster convergence.

Gauss-Seidel iteration for the RR model

```
Initial approximation: [0. 0.]
Step 1: [15.    11.85]
Step 2: [27.933   23.19729]
Step 3: [39.2872722  33.41424979]
Step 4: [49.30140095 42.48173004]
Step 5: [58.14372009 50.50061077]
.........................
Step 79: [125.07002966 111.22451454]
Step 80: [125.07083397 111.22524433]
```

Optimal Value Function

- The optimal value function is:

$$V^\star(s) = \max_\pi V^\pi(s) = \max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_{t+1} \,|\, S_0 = s \right]$$

  where π ranges over all admissible (randomized) policies.
- A policy $\pi^\star$ is optimal if:

$$V^{\pi^\star}(s) = V^\star(s) \quad \text{for all } s \in \mathscr{S}$$

- Under the finiteness assumptions we made, an optimal policy always exists.

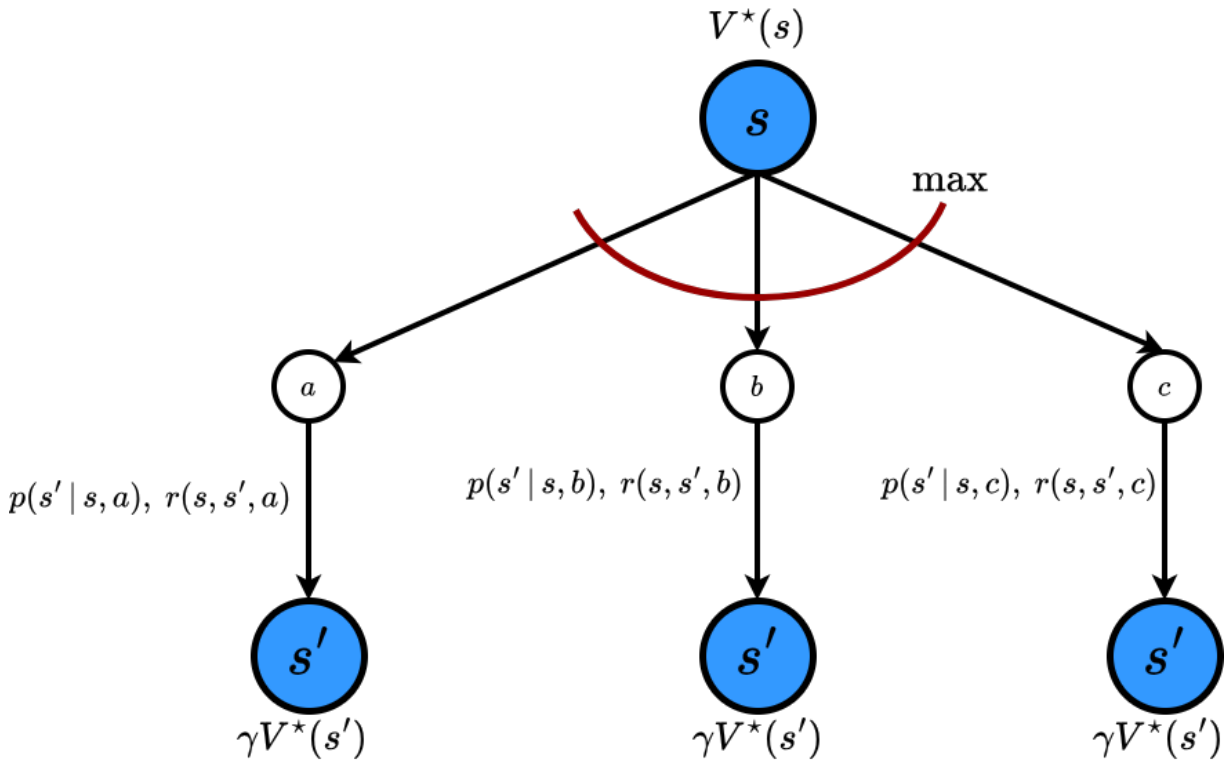
How to live the best life

- There is an oracle that can tell us what to do to maximize our happiness. However, the oracle will only tell us what is the best we can do starting tomorrow.
- We can ask a series of questions to the oracle:
  - If I take action a_j today, what is the best outcome I can expect from tomorrow onward?
- We can then find:

$$\max\{R(a_j) + \mathbb{E}[\text{rewards from tomorrow onward} \,|\, \text{action } a_j \text{ chosen}]\}$$

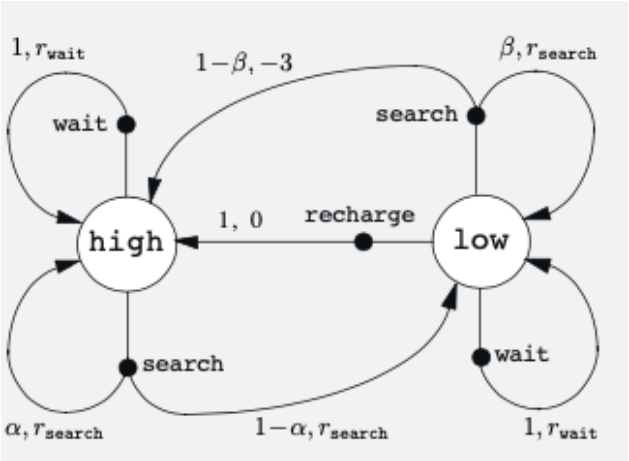
Optimal Bellman equation

Backup diagram:



$$V^\star(s) = \max_{a \in \mathscr{A}(s)} \left\{ \sum_{s' \in \mathscr{S}} p(s, s' \mid a) \left[r(s, s', a) + \gamma V^\star(s') \right] \right\}$$

Example — Recycling Robot



$$V^\star(\texttt{high}) = \max\big\{\alpha(r_{\texttt{search}} + \gamma V^\star(\texttt{high})) + (1-\alpha)(r_{\texttt{search}} + \gamma V^\star(\texttt{low})),\\ r_{\texttt{wait}} + \gamma V^\star(\texttt{high})\big\}$$

$$V^\star(\texttt{low}) = \max\big\{(1-\beta)(-3 + \gamma V^\star(\texttt{high})) + \beta(r_{\texttt{search}} + \gamma V^\star(\texttt{low})),\\ r_{\texttt{wait}} + \gamma V^\star(\texttt{low}),\\ 0 + \gamma V^\star(\texttt{high})\big\}$$

Verification Theorem

Theorem

Suppose $V^\star$ is a solution of the optimal Bellman equation:

$$V^\star(s) = \max_{a \in \mathscr{A}(s)} \left\{ \sum_{s' \in \mathscr{S}} p(s, s' \,|\, a) \left[r(s, s', a) + \gamma V^\star(s') \right] \right\}.$$

Then, for any randomized policy $\pi(a \,|\, s)$ we have:

$$V^\star(s) \geq V^\pi(s) \quad \text{for all } s \in \mathscr{S}$$

Furthermore, an optimal (deterministic) policy is given by:

$$\pi^\star(s) = \arg\max \left\{ \sum_{s' \in \mathscr{S}} p(s, s' \,|\, a) \left[r(s, s', a) + \gamma V^\star(s') \right] \;\middle|\; a \in \mathscr{A}(s) \right\}$$

What are tabular solution methods

- In **tabular solution methods**, we build *complete* tables of the optimal value function and optimal policy.
- These methods can only be used to solve relatively small problems.
- It is also assumed that we have complete information about transition probabilities and rewards (*model based RL*).
- It is important to learn them, because:
  - The theory of these methods is *very* well established, so we know very well how they behave.
  - They are essential to develop intuition.
  - They contain the main ideas used in more advanced algorithms.
  - In practice, they provide a good way to build test cases for complex algorithms.

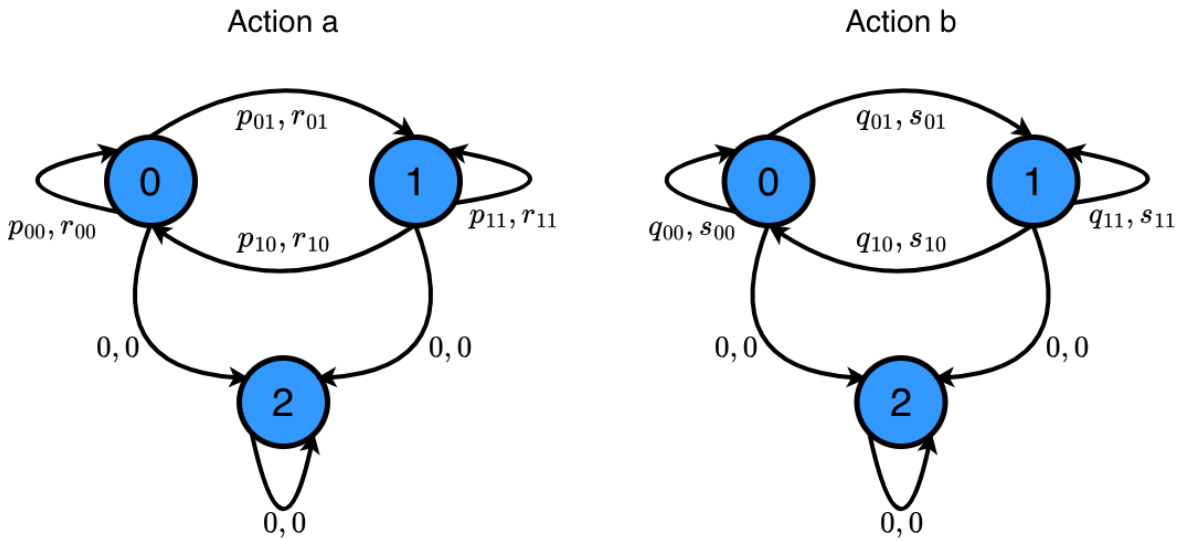
Value Iteration algorithm

- Let $V_0(s)$ be arbitrary for $s \in \mathscr{S}$.
- Iterate:

$$V_{n+1}(s) = \max_{a \in \mathscr{A}(s)} \left\{ \sum_{s' \in \mathscr{S}} p(s, s' \mid a) \left[r(s, s', a) + \gamma V_n(s') \right] \right\},$$

  except that we let $V_{n+1}(s) = 0$ if s is a terminal state in episodic tasks.
- Assume that:
  - $0 < \gamma < 1$ for continuing tasks.
  - $0 < \gamma \leq 1$ and a terminal state is reached with probability 1 for any policy.
- Then $\lim_{n \to \infty} V_n(s) = V^\star(s)$ for all $s \in \mathscr{S}$.

Value Iteration Example — Tri-state MDP



Optimal Bellman equation for $\gamma = 1$:

$$V^\star(0) = \max\{p_{00}(r_{00} + V^\star(0)) + p_{01}(r_{01} + V^\star(1)),$$
$$q_{00}(s_{00} + V^\star(0)) + q_{01}(s_{01} + V^\star(1))\}$$
$$V^\star(1) = \max\{p_{10}(r_{10} + V^\star(0)) + p_{11}(r_{11} + V^\star(1)),$$
$$q_{10}(s_{10} + V^\star(0)) + q_{11}(s_{11} + V^\star(1))\}$$

Tri-state MDP — Value Iteration

$$
\begin{aligned}
V_{n+1}(0) = \max\{&p_{00}(r_{00} + V_n(0)) + p_{01}(r_{01} + V_n(1)),\\
&q_{00}(s_{00} + V_n(0)) + q_{01}(s_{01} + V_n(1))\}\\
V_{n+1}(1) = \max\{&p_{10}(r_{10} + V_n(0)) + p_{11}(r_{11} + V_n(1)),\\
&q_{10}(s_{10} + V_n(0)) + q_{11}(s_{11} + V_n(1))\}
\end{aligned}
$$

Parameters for testing:

$$
\begin{array}{llll}
p_{00} = 0.2 & p_{01} = 0.7 & p_{10} = 0.5 & p_{11} = 0.3\\
q_{00} = 0.1 & q_{01} = 0.6 & q_{10} = 0.4 & q_{11} = 0.3\\
\\
r_{00} = 10 & r_{01} = 15 & r_{10} = 8 & r_{11} = 12\\
s_{00} = 13 & s_{01} = 13 & s_{10} = 15 & s_{11} = 20
\end{array}
$$

Value Iteration — Results

n	$V_n(0)$	$V_n(1)$
0	0.00	0.00
1	12.50	16.00
2	26.20	25.80
⋮	⋮	⋮
38	71.24	63.57
39	71.25	63.57
40	71.25	63.57

- Convergence is not very fast.
- Optimal policy is found by a one-step lookahead. In this case we get:

$$\pi(0) = a, \quad \pi(1) = b$$

Gauss-Seidel Iteration

$$V_{n+1}(0) = \max\{p_{00}(r_{00} + V_n(0)) + p_{01}(r_{01} + V_n(1)),$$
$$q_{00}(s_{00} + V_n(0)) + q_{01}(s_{01} + V_n(1))\}$$
$$V_{n+1}(1) = \max\{p_{10}(r_{10} + V_{n+1}(0)) + p_{11}(r_{11} + V_n(1)),$$
$$q_{10}(s_{10} + V_{n+1}(0)) + q_{11}(s_{11} + V_n(1))\}$$

n	$V_n(0)$	$V_n(1)$
0	0.00	0.00
1	12.50	21.00
2	29.70	34.18
⋮	⋮	⋮
26	71.24	63.57
27	71.25	63.57
28	71.25	63.57

- Easier to code, uses less memory.
- Slightly faster convergence.

Policy Improvement Theorem

Theorem

Let π be an arbitrary policy and $V^{\pi}(\cdot)$ the value function of π. Define a new policy π' by:

$$\pi'(s) = \arg\max \left\{ \sum_{s' \in \mathscr{S}} p(s' \mid s, a) \left[r(s, s', a) + \gamma V^{\pi}(s') \right] \mid a \in \mathscr{A}(s) \right\}$$

Then,

$$V^{\pi'}(s) \geq V^{\pi}(s) \text{ for all } s \in \mathscr{S},$$

with strict inequality for at least one s unless policy π is optimal.

- Suppose we have an approximation to a value function $V(\cdot)$, which could be a policy value function, or an optimal value function.
- We say that a policy is **greedy** with respect to $V(\cdot)$ if:

$$\pi(s) = \arg\max \left\{ \sum_{s' \in \mathscr{S}} p(s' \mid s, a) \left[r(s, s', a) + \gamma V(s') \right] \mid a \in \mathscr{A}(s) \right\}$$

  for all $s \in \mathscr{S}$.
- We can say:
  - An optimal policy $\pi^\star$ is greedy with respect to the optimal value function $V^\star$.
  - If π is any policy and π' is greedy with respect to V^π, then π' is an improvement over π.

Policy Iteration Algorithm

- Let π_0 be a randomized initial policy, with corresponding value function $V_0^{\pi}(\cdot)$.
- Iterate for $n = 0, 1, 2, \ldots$:
  - Compute a new policy:

$$\pi_{n+1}(s) = \arg\max \left\{ \sum_{s' \in \mathscr{S}} p(s' \mid s, a) \left[r(s, s', a) + \gamma V_n^{\pi}(s') \right] \mid a \in \mathscr{A}(s) \right\}$$

  - Compute the value function of π_{n+1}, $V^{\pi_{n+1}}(\cdot)$
- Stopping criteria:
  - $\pi_{n+1}(s) = \pi_n(s)$ for all $s \in \mathscr{S}$
  - $\max_{s \in \mathscr{S}} |V^{\pi_{n+1}}(s) - V^{\pi_n}(s)| < \epsilon_{\text{tol}}$
  - Maximum number of iterations exceeded.

Policy Iteration example

n	$\pi_n(0)$	$\pi_n(1)$	$V_n(0)$	$V_n(1)$
0	b	a	33.121	34.515
1	a	b	71.250	63.571
2	a	b	71.250	63.571

- In general, policy iteration converges much faster than value iteration.
- In fact, for a finite MDP, policy improvement will "converge" in a finite number of steps, since the total number of deterministic policies is finite.
- This assumes that:
  - Transition probabilities are known.
  - The whole state is observed.
  - Rewards are a discrete subset of $\mathbb{R}$.
  - $V_n(s)$ is computed exactly.

Approximate Policy Iteration

- Policy iteration starts with a policies π_n that are far from optimal.
- It is wasteful to compute $V^{\pi_n}(s)$ exactly in the beginning.
- Instead, we compute only an *approximation* $\tilde{V}^{\pi_n}(s)$ in each step. For example, use a Jacobi or Gauss-Seidel iteration with a small number of relaxations.
- We don't even need a good approximation!
- Stopping condition:
  - Policy stabilizes: $\pi_{n+1}(s) = \pi_n(s)$ for all states s *and*
  - Value function estimates stabilize: $\tilde{V}^{\pi_{n+1}}(s) \approx \tilde{V}^{\pi_n}(s)$
- Compute a high-precision approximation of $V^{\pi_n}(s)$ for the final policy.
- Optionally, do one more policy improvement step to guarantee we have an optimal policy.

Approximate Policy Iteration Example

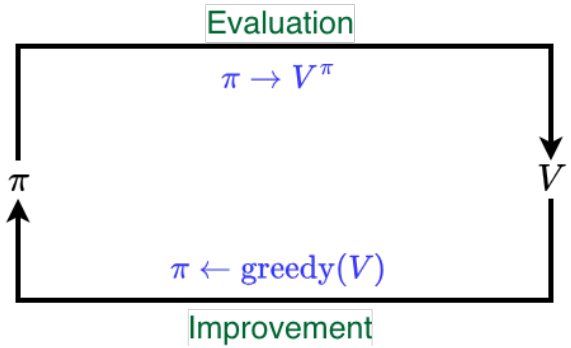
Approximate Policy Iteration for Tri-state MDP with 10 Gauss-Seidel relaxations for each policy improvement step:

n	$\pi_n(0$	$\pi_n(1)$	$\tilde{V}_n(0)$	$\tilde{V}_n(1)$	$\max_s \lvert\tilde{V}_n(s) - \tilde{V}_{n-1}(s)\rvert$
0	b	a	32.59054893	34.02505034	–
1	a	b	70.18751040	62.82240404	3.760×10^{1}
2	a	b	71.22266853	63.55216067	1.035×10^{0}
3	a	b	71.24929693	63.57093292	2.663×10^{-2}
4	a	b	71.24998191	63.57141582	6.850×10^{-4}

High precision value function for final policy, obtained with 200 Gauss-Seidel relaxations:

$$V(0) = 71.25000000, \quad V(1) = 63.57142857$$

Generalized Policy Iteration



- Policy iteration alternates two stages:
  - **Evaluation**: Given a policy π, evaluate to V^π.
  - **Improvement**: Given a value function V, compute a policy π greedy with respect to V.
- **Generalized policy iteration** (GPI) algorithm: interleaves these two processes arbitrarily.
  - Value iteration: policy improvement step is performed after $V(s)$ is updated for each state.
  - Asynchronous DP: the two processes are carried out in parallel.