4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2024) Ian Steenstra
Northeastern University Due Oct 4, 2024

Exercise 3: Dynamic Programming

1. **1 point.** (RL2e 3.25 – 3.29) *Fun with Bellman.*
Written:

   (a) Give an equation for v_* in terms of q_*.

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s, a)$$

   (b) Give an equation for q_* in terms of v_* and the four-argument p.

$$q_*(s, a) = \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_*(s') \right]$$

   (c) Give an equation for π_* in terms of q_*.

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} q_*(s, a)$$

   (d) Give an equation for π_* in terms of v_* and the four-argument p.

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_*(s') \right]$$

   (e) Rewrite the four Bellman equations for the four value functions (v_π, v_*, q_π, q_*) in terms of the three-argument function p (Equation 3.4) and the two-argument function r (Equation 3.5).

$$\begin{aligned}
v_\pi(s) &= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[r(s, a) + \gamma v_\pi(s') \right] \\
v_*(s) &= \max_a \sum_{s'} p(s'|s, a) \left[r(s, a) + \gamma v_*(s') \right] \\
q_\pi(s, a) &= \sum_{s'} p(s'|s, a) \left[r(s, a) + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \\
q_*(s, a) &= \sum_{s'} p(s'|s, a) \left[r(s, a) + \gamma \max_{a'} q_*(s', a') \right]
\end{aligned}$$

2. **1 point.** (RL2e 4.5, 4.10) *Policy iteration for action values.*
Written:

   (a) How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 80 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.

   Policy iteration for action values

   (i) Initialization
   $Q(s, a) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

(ii) Policy Evaluation
Repeat
- $\Delta \leftarrow 0$
- For each $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
  - $q \leftarrow Q(s,a)$
  - $Q(s,a) \leftarrow \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma \sum_{a'} \pi(a'|s')Q(s',a')\right]$
  - $\Delta \leftarrow \max(\Delta, |q - Q(s,a)|)$

until $\Delta < \theta$ (a small positive number)

(iii) Policy Improvement
- *policy-stable* $\leftarrow$ *true*
- For each $s \in \mathcal{S}$:
  - $b \leftarrow \pi(s)$
  - $\pi(s) \leftarrow \arg\max_a Q(s,a)$
  - If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
- If *policy-stable*, then stop and return Q and π; else go to (ii)

(b) What is the analog of the value iteration update Equation 4.10 for action values, $q_{k+1}(s,a)$?

$$q_{k+1}(s,a) = \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma \max_{a'} q_k(s',a')\right]$$

3. **2 points.** *Policy iteration by hand.*
Written: Consider an undiscounted MDP having three states, x, y, z. State z is a terminal state. In states x and y there are two possible actions: b and c. The transition model is as follows:

- In state x, action b moves the agent to state y with probability 0.8 and makes the agent stay put (at state x) with probability 0.2.
- In state y, action b moves the agent to state x with probability 0.8 and makes the agent stay put (at state y) with probability 0.2.
- In either state x or state y, action c moves the agent to state z with probability 0.1 and makes the agent stay put with probability 0.9.

The reward model is as follows:

- In state x, the agent receives reward -1 regardless of what action is taken and what the next state is.
- In state y, the agent receives reward -2 regardless of what action is taken and what the next state is.

Answer the following questions:

(a) What can be determined *qualitatively* about the optimal policy in states x and y (i.e., just by looking at the transition and reward structure, *without* running value/policy iteration to solve the MDP)?
The MDP's structure suggests that reaching the terminal state 'z' is desirable, as it is the only state without negative rewards. Action 'c', despite mostly keeping the agent in its current state, is the only action that offers a chance (10%) to transition to 'z'. On the other hand, action 'b' keeps the agent cycling between states 'x' and 'y', both of which have negative rewards, making it a less attractive option. Therefore, we can reasonably expect that the optimal policy will favor taking action 'c' more frequently to increase the likelihood of reaching the goal state 'z.

(b) Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states x and y. Assume that the initial policy has action c in both states.
Iteration 1:
- **Policy:** $\pi(x) = c$, $\pi(y) = c$
- **Policy Evaluation (System of Equations):**

$$V(x) = -1 + 0.9V(x) + 0.1V(z)$$
$$V(y) = -2 + 0.9V(y) + 0.1V(z)$$
$$V(z) = 0 \quad \text{(Terminal state)}$$

Solving this system, we get: $\mathbf{V(x) = -10}$, $\mathbf{V(y) = -20}$

- **Policy Improvement:**
  - **State x:**
    - Expected return for 'b': $-1 + 0.8(-20) + 0.2(-10) = -19$
    - Expected return for 'c': $-1 + 0.9(-10) + 0.1(0) = -10$
    - $\pi(\mathbf{x})$ remains 'c'
  - **State y:**
    - Expected return for 'b': $-2 + 0.8(-10) + 0.2(-20) = -14$
    - Expected return for 'c': $-2 + 0.9(-20) + 0.1(0) = -20$
    - $\pi(\mathbf{y})$ changes to 'b'

Iteration 2:

- **Policy:** $\pi(x) = c$, $\pi(y) = b$
- **Policy Evaluation (System of Equations):**

$$V(x) = -1 + 0.9V(x) + 0.1V(z)$$
$$V(y) = -2 + 0.8V(x) + 0.2V(y)$$
$$V(z) = 0$$

  Solving, we get: $\mathbf{V(x) = -10}$, $\mathbf{V(y) = -18}$
- **Policy Improvement:**
  - **State x:**
    - Expected return for 'b': $-1 + 0.8(-18) + 0.2(-10) = -17.4$
    - Expected return for 'c': $-1 + 0.9(-10) + 0.1(0) = -10$
    - $\pi(\mathbf{x})$ remains 'c'
  - **State y:**
    - Expected return for 'b': $-2 + 0.8(-10) + 0.2(-18) = -13.6$
    - Expected return for 'c': $-2 + 0.9(-18) + 0.1(0) = -18.2$
    - $\pi(\mathbf{y})$ remains 'b'

Optimal Policy and Value Function:

Since the policy did not change in Iteration 2, it has converged.

- $\pi^*(\mathbf{x}) = \mathbf{c}$
- $\pi^*(\mathbf{y}) = \mathbf{b}$
- $\mathbf{V}^*(\mathbf{x}) = \mathbf{-10}$
- $\mathbf{V}^*(\mathbf{y}) = \mathbf{-18}$

(c) What happens to policy iteration if the initial policy has action b in both states? Does discounting help? Does the optimal policy depend on the discount factor (in this particular MDP)?

Even if we initially set the policy to always choose action 'b', policy iteration would eventually converge to an optimal policy—it may just take a few extra interactions however. That's because starting with 'b' keeps us stuck in a loop of bad rewards, which is further from the best strategy of trying to reach that terminal state, 'z'. Additionally, adding a discount factor doesn't really change the optimal policy in this case. The negative rewards in 'x' and 'y' make us want to get to the zero-reward terminal state as quickly as possible anyway, so the discount doesn't have much impact on the best way to do that.

4. **2 points.** *Implementing dynamic programming algorithms.*
Code/plot: For all algorithms, you may use any reasonable convergence threshold (e.g., $\theta = 10^{-3}$). We implement the 5×5 grid-world in Example 3.5 for you and please read the code in Jupyter Notebook for more details.

(a) Implement *value iteration* to output both the optimal state-value function and optimal policy for the given MDP (i.e., the 5×5 grid-world). Print out the optimal value function and policy for the 5×5 grid-world using your implementation (v_* and π_* are given in Figure 3.5). Please use the threshold value $\theta = 1e^{-4}$ and $\gamma = 0.9$.

See code & Figure 1 (Problem 4(a)).

(b) Implement *policy iteration* to output both the optimal state-value function and optimal policy for the given MDP (i.e., the 5×5 grid-world). Print out the optimal value function and policy for the 5×5 grid-world using your implementation (v_* and π_* are given in Figure 3.5). Please use the threshold value $\theta = 1e^{-4}$ and $\gamma = 0.9$.

See code & Figure 2 (Problem 4(b)).

5. **3 points.[5180]** (RL2e 4.7) *Jack's car rental problem.*

(a) **Code/plot:** Replicate Example 4.2 and Figure 4.2. The implementation for Jack's car rental problem is given in the Jupyter Notebook. Please complete the policy iteration implementation to solve for the optimal policy and value function. Reproduce the plots shown in Figure 4.2 (The plotting functions are also given), showing the policy iterates and the final value function – your plots do not have to be in exactly the same style, but should be similar (See the Figure below).

See Figure 3 (Problem 6(a)).

(b) **Code:** Re-solve Jack's car rental problem with the following changes.
Written: Describe how you will change the reward function (i.e. compute_reward_modified function in the JackCarRental class) to reflect the following changes.
My plan is to update the reward function to better incentivize minimizing costs and maximizing rental income. The moving cost will be calculated by first subtracting 1 from the absolute number of cars moved between the lots (representing one free move), then multiplying the result by -2 to reflect the $2 cost per car move. A storage penalty of $4 per car will be applied to each lot when the number of cars exceeds 10, encouraging a balanced distribution. Rental income will be calculated as $10 for each car rented, limited by the average customer requests at each lot. The final reward will be the sum of the move cost, storage cost, and rental income.
Plot: Similar to part (a), produce plots of the policy iterates and the final value functions.
See Figure 4 (Problem 6(b).
Written: How does your final policy differ from Q5(a)? Explain why the differences make sense. The updated reward function promotes a more cost-conscious and demand-driven policy. The free car move discourages excessive transfers, while the storage penalty limits stockpiling cars at either lot. Unlike the original policy, which favored a balanced distribution, the new policy may prioritize moving cars to the lot with higher demand (lot2) to maximize rental income, showcasing how subtle reward adjustments can significantly alter agent behavior.

6. **1 point.** (RL2e 4.4) *Fixing policy iteration.*
Written:

(a) The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good. This is okay for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed.

i. **Initialization**
$V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$

ii. **Policy Evaluation**
Repeat
$\Delta \leftarrow 0$
For each $s \in S$:
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

iii. **Policy Improvement**
policy_improved $\leftarrow$ **false**
For each $s \in S$:
$a \leftarrow \pi(s)$
$\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
If $a \neq \pi(s)$ and $\sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] > V(s)$, then *policy_improved* $\leftarrow$ **true**
If *policy_improved*, **then stop** and **return** V and π; **else go to** 2

Update: I added a different flag to make sure there was actual improvement in the performance of the policy. However, it should be noted that this should also check that it equates to a better value for each state as well.

(b) Is there an analogous bug in value iteration? If so, provide a fix; otherwise, explain why such a bug does not exist.

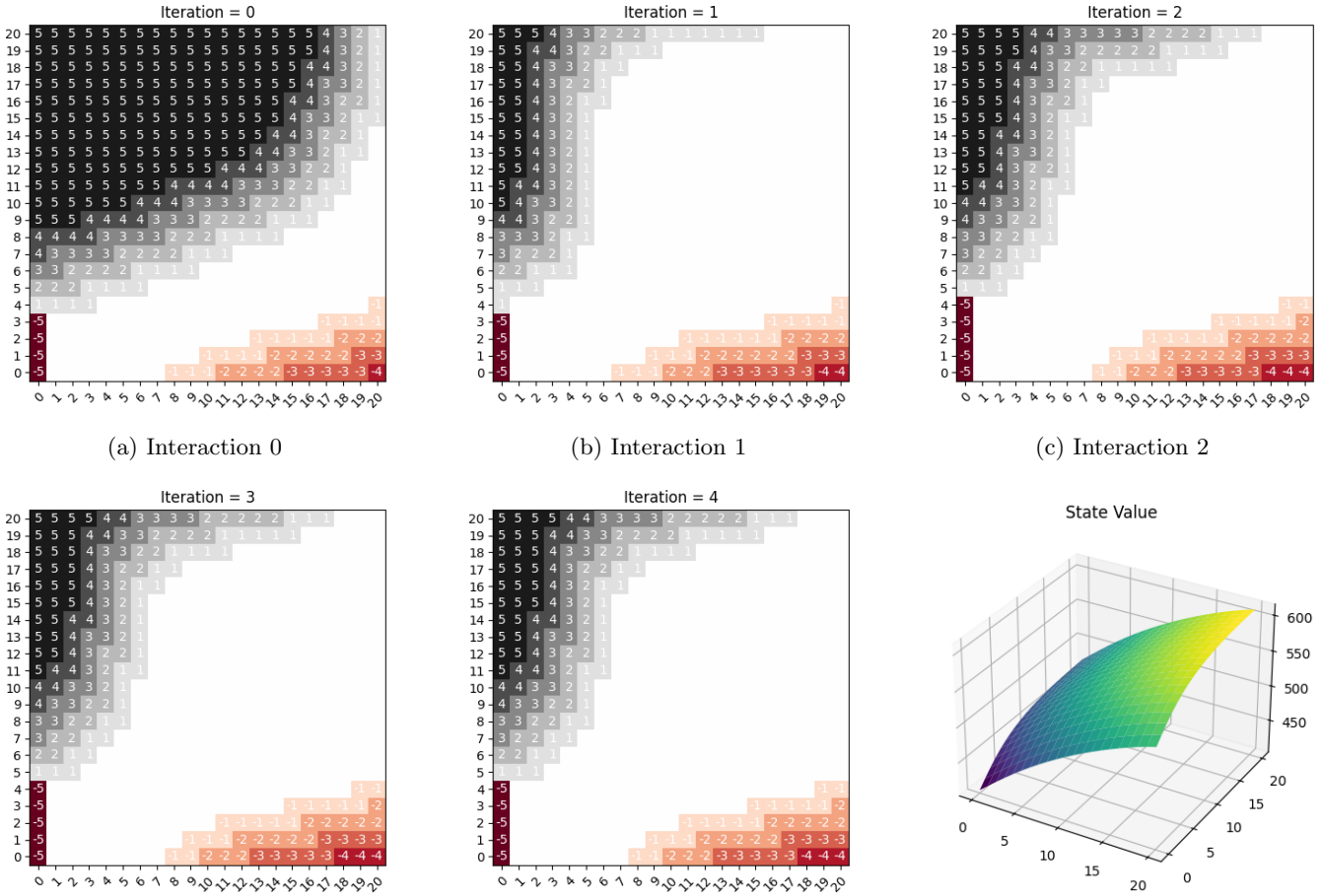
The bug arises when extracting the policy: multiple actions might have the same maximum expected return for a state. While not preventing convergence, this can introduce arbitrariness if we extract a deterministic policy. To address this, we can either represent the optimal policy as a stochastic policy or use a tie-breaking rule when selecting among actions with equal values.

```
============================
==  Optimal State Value    ==
============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
============================
============================
==     Optimal Policy      ==
============================
[0, 0] = ['east']
[0, 1] = ['north', 'south', 'west', 'east']
[0, 2] = ['west']
[0, 3] = ['north', 'south', 'west', 'east']
[0, 4] = ['west']
----------------------------
[1, 0] = ['north', 'east']
[1, 1] = ['north']
[1, 2] = ['north', 'west']
[1, 3] = ['west']
[1, 4] = ['west']
----------------------------
[2, 0] = ['north', 'east']
...
[4, 2] = ['north', 'west']
[4, 3] = ['north', 'west']
[4, 4] = ['north', 'west']
----------------------------
```

Figure 1: Problem 4(a): Optimal State Value & Optimal Policy

```
============================
==  Optimal State Value    ==
============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
============================
============================
==     Optimal Policy      ==
============================
[0, 0] = ['east']
[0, 1] = ['north', 'south', 'west', 'east']
[0, 2] = ['west']
[0, 3] = ['north', 'south', 'west', 'east']
[0, 4] = ['west']
----------------------------
[1, 0] = ['north', 'east']
[1, 1] = ['north']
[1, 2] = ['north', 'west']
[1, 3] = ['west']
[1, 4] = ['west']
----------------------------
[2, 0] = ['north', 'east']
...
[4, 2] = ['north', 'west']
[4, 3] = ['north', 'west']
[4, 4] = ['north', 'west']
----------------------------
```

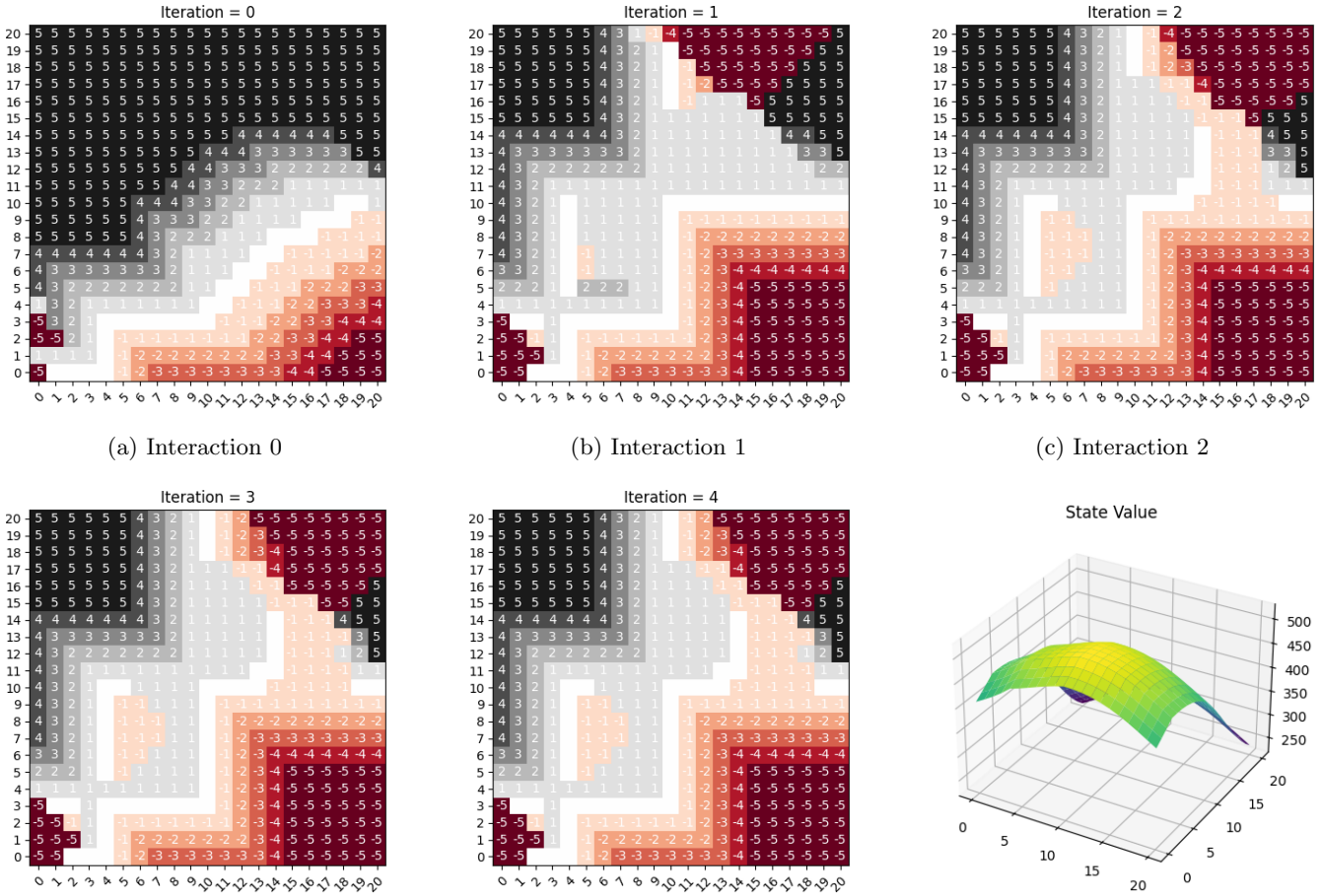
Figure 2: Problem 4(b): Optimal State Value & Optimal Policy



(a) Interaction 0 (b) Interaction 1 (c) Interaction 2

(d) Interaction 3 (e) Interaction 4 (f) State Value

Figure 3: Problem 6(a): Evolution of the system across policy interactions & final state value.



(a) Interaction 0 (b) Interaction 1 (c) Interaction 2

(d) Interaction 3 (e) Interaction 4 (f) State Value

Figure 4: Problem 6(b): Evolution of the system across policy interactions & final state value.