4180/5180: Reinforcement Learning and Sequential Decision Making (Spring 2025) Robert Platt
Northeastern University Due Feb 13, 2025

Exercise 3: Dynamic Programming

Please remember the following policies:

- Exercise due at **11:59 PM EST Feb 13, 2025**.
- Submissions should be made electronically on Canvas. Please ensure that your solutions for both the written and programming parts are present. You can upload multiple files in a single submission, or you can zip them into a single file. You can make as many submissions as you wish, but only the latest one will be considered.
- For **Written** questions, solutions should be typeset.
- The PDF file should also include the figures from the **Plot** questions.
- For both **Plot** and **Code** questions, submit your source code in Jupyter Notebook (.ipynb file) along with reasonable comments of your implementation. Please make sure the code runs correctly.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution and code yourself. Also, you *must* list the names of all those (if any) with whom you discussed your answers at the top of your PDF solutions page.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 10% per day late. Submissions later than two days will not be accepted.
- Contact the teaching staff if there are medical or other extenuating circumstances that we should be aware of.
- **Notations: RL2e is short for the reinforcement learning book 2nd edition. x.x means the Exercise x.x in the book.**

1. **1 point.** (RL2e 3.25 – 3.29) *Fun with Bellman.*
   Written: Write the Bellman equations for the value functions in terms of the three-argument function $p(s'|s,a)$ (Equation 3.4) and the two-argument function $r(s,a)$ (Equation 3.5).
   (a) Give an equation for v_* in terms of q_*.
   (b) Give an equation for q_* in terms of v_* and p.
   (c) Give an equation for π_* in terms of q_*.
   (d) Give an equation for π_* in terms of v_* and p.

2. **3 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 0.7 and makes the agent stay put (at state x) with probability 0.3.
   - In state y, action b moves the agent to state x with probability 0.7 and makes the agent stay put (at state y) with probability 0.3.
   - In either state x or state y, action c moves the agent to state z with probability 0.2 and makes the agent stay put with probability 0.8.

   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)?

(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.

(c) What happens to policy iteration if the initial policy has action b in both states? Does a discounting factor (for example $\gamma = 0.9$) help? Does the optimal policy change with different γ in this particular MDP?

3. **2 points.** *Implementing dynamic programming algorithms.*
Code: 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 $\theta = 1e^{-3}$ and $\gamma = 0.8$.

   (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 $\theta = 1e^{-3}$ and $\gamma = 0.8$.

4. **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.

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

5. **3 point.** (RL2e 4.8, 4.9) *Gamber's problem.*

   (a) **Written:** Why does the optimal policy for the gambler's problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy?

   (b) **Code, Plot:** Implement value iteration for the gambler's problem and solve it for $p_h = 0.25$ and $p_h = 0.55$. In programming, you may find it convenient to introduce two dummy states corresponding to termination with capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically, as in Figure 4.3 (shown below). Are your results stable as $\theta \to 0$?

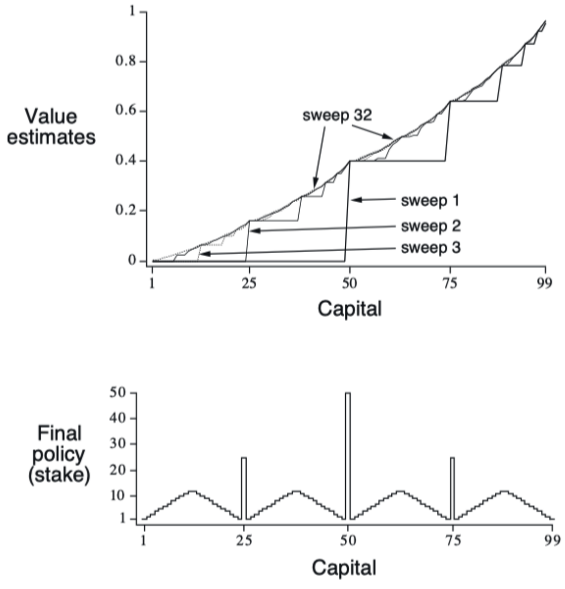

Figure 4.3: The solution to the gambler's problem for $p_h = 0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.