4180/5180: Reinforcement Learning and Sequential Decision Making (Spring 2026) Robert Platt & Chris Amato
Northeastern University Due Feb 9, 2026

Exercise 3: Dynamic Programming

Please remember the following policies:

- Exercise due at **11:59 PM EST Feb 9, 2026**.
- 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. Please do **not** zip them into a single file.
- You can make as many submissions as you wish, but only the latest one will be considered. Please do **not** make any submissions after the deadline unless you would like the submission to be considered as a late submission.
- 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. Please also submit a pdf of the Jupyter Notebook (.ipynb file).
- 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.
- Generative AI can only be used for finding bugs or polishing English—all code and solutions should be written up by the student.
- **Notations: RL2e is short for the reinforcement learning book 2nd edition. x.x means the Exercise x.x in the book.**
- Please note that question 5 is required for 5180 students and extra credit for 4180 students.

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.

1)

a) $V^*(s) = \arg\max_a q^*(s,a)$

b) $q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'/s,a)\, V^*(s')$

c) $\pi^*(s) = \arg\max_a q^*(s,a)$

d) $\pi^*(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s'} P(s'/s,a)\, V^*(s') \right)$

2) transitions $x \xrightarrow{b} y$ (0.7) or stay at x (0.3)

transition $y \xrightarrow{b} x$ (0.7) or stay at y (0.3)

transition $x \text{ or } y \xrightarrow{c} z$ (0.2) or stays at xy (0.8)

Rewards : $r(x) = -1$, $r(y) = -2$

a) Staying at state y is worst since it always gets (-2) rewards. From state y with action b it has more libely to go state x (0.7). so its better to do it. From state x with action b has 0.7 chance to go State b which is unattractive. Action c gives 0.2 chance to be in State z which is terminal state. It might

be good if we want to stop accumulating negative rewards for example, when in state x

So I would say it by, in x prefer action c to avoid worse state y and terminate. In state y prefer b

b) initialization step is already in question so

init: $V(x), V(y) \in R \quad V(z) \doteq 0 \;,\; \pi(x) = c \;,\; \pi(y) = c$

1- policy evaluation

$\Delta = 0 \;,\; v = V(s)$

$V(s) = \sum p(s', r \mid s, \pi(s)) [r + \gamma V(s')] \;,\; \Delta = \max(\Delta, v - V(s))$

$\pi_0 = (c, c) \;,\; V_0(z) = 0 \qquad \gamma = 1$

for x: $V(z) = 0$

$V(x) = 0.8 \times [-1 + 1 \times V(x)] + 0.2 [-1 + 1 \times V(z)]$

$V(x) = -0.8 + 0.8 V(x) + (-0.2 + (0.2 \times 0))$

$V(x) = -0.8 + 0.8 V(x) - 0.2 = -1 + 0.8 V(x)$

$V(x) - 0.8 V(x) = -1 \;,\; 0.2 V(x) = -1$ thus, $V(x) = -5$

for y:

$V(y) = 0.8 \times [-2 + 1 V(y)] + 0.2 [-2 + 1 V(z)]$

$V(y) = -1.6 + 0.8\,V(y) + (-0.4 + 0)$

$V(y) = -1.6 - 0.4 + 0.8\,V(y)$

$V(y) = -2 + 0.8\,V(y)$, $0.2\,V(y) = -2$

$V(y) = -10$

thus $V(x) = -5$, $V(y) = -10$, $V(z) = 0$

Policy Improvement

policy-stable = true

for each $S \in \{x, y$ old-action $= \pi(S)$

$\pi(S) = \underset{a}{\operatorname{argmax}} \sum P(s', r \mid s, a)\,[r + \gamma V(s')]$

if old-action $\neq \pi(S)$ then policy-stable = false

At x: $V(x) = -5$

if action = a

$\sum P(s' \mid x, a)\,[-1 + 1\,V(s')] = 0.8\,(-1 + V(x)) + 0.2\,(-1 + V(z))$

$= -5$

if action b:

$0.7\,(-1 + V(y)) + 0.3\,(-1 + V(x)) = 0.7(-1-10) + 0.3(-1-5) = -9.5$

So $\pi(x) = a$ (no change needed)

At y: $V(y) = -10$

if action = c

$$0.8\,(-2 + V(y)) + 0.2\,(-2 + V(z)) = -10$$

if action = b

$$0.7\,(-2 + V(x)) + 0.3\,(-2 + V(y)) = -8.5$$

So $\pi(y) = b$ (policy changes since $-8.5 \geq -10$)

New policy: $\pi(y) = b$, $\pi(x) = c$

then evaluate again, then improve again till convergence

C) (My interpretation here (change later))

if initial policy was b for both states then we would get more negative rewards. The discount factor would help reduce the negative reward little bit but in policy improvement we would try to get policy that maximizes our expected reward so for state x we would change the pol's. In this example with different discount factors could change if its a small value in long term

it ~~would~~ could change the optimal policy.

3) Code and plot is in .ipynb file as well.

4)

a) fixed psuedo code!

(the fix is in 3rd step policy improvement)

(I will only show policy improvement part since the rest is same)

Psuedo code!

// Policy Improvemen

Policy-stable = true

for each $s \in S$

old-action = $\pi(s)$

// Fix starts here

$A^* = \arg\max \sum_{s,r} p(s', r | s, a)[r + \gamma V(s')]$

if old-action $\in A^*$: // Avoiding it when its tie

$\pi(s) =$ old-action

else:

$\pi(s) =$ Tiebreak(A*)

if old-action $\neq \pi(s)$:

policy-stable = false

if policy-stable, then stop and return $V \approx v^*$ and $\pi \approx \pi^*$

else: go to 2

b) No, value iteration part is stopping when $\Delta < \theta$ (threshold). Not if greedy policy is stable. So tie switching between equally good actions doesn't prevent convergence of the value. (for $\gamma < 1$)

5. **3 points.** (This question is **required** for 5180 students and extra credit for 4180 students)(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 to Figure 4.2.

   (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.
   Plot: Similar to part (a), produce plots of the policy iterates and the final value functions.
   Written: How does your final policy differ from Q5(a)? Explain why the differences make sense.

   - One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still costs 2, as do all cars moved in the other direction.
   - In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then a total additional cost of 4 must be incurred to use a second parking lot (independent of how many cars are kept there). (Each location has a separate overflow lot, so if both locations have > 10 cars, the total additional cost is 8.)

   These sorts of nonlinearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming.

Some clarification and guidance for completing Q5 and understanding the environment implementation:

- The description of Jack's car rental problem in Example 4.2 is detailed, but some extra details are needed to reproduce the results shown in Figure 4.2. Assume the following daily schedule for the problem:
  - 6 PM: "End of day": Close of business; this is when move actions are decided.
    From the description: "The state is the number of cars at each location at the end of the day."
  - 8 PM: Cars to be moved (if any) have arrived at their new location, including (in part b) by the employee going from location 1 to 2. The new location may have max $20 + 5$ cars after the move.
  - 8 PM – 8 AM: Overnight parking; in part b, need to pay \$4 for each location that has > 10 cars.
  - 8 AM: "Start of day": Open of business; one location may have up to 25 cars.
  - 9 AM: All requests come in at this time (before any returns).
  - 5 PM: All cars are returned at this time, i.e., a returned car cannot be rented out on the same day.
  - 5:59 PM: Excess cars (> 20) are removed at each location; each location has max 20 cars.
- Because of the somewhat larger state space and numerous request/return possibilities, a number of enhancements will likely be necessary to make dynamic programming efficient.
  - The four-argument *dynamics function* $p(s', r|s, a)$ is the most general form, but also the most inefficient form. In this case, using the three-argument *transition function* $p(s'|s, a)$ (Equation 3.4) and the two-argument *reward function* $r(s, a)$ will be much more efficient. Use the Bellman equations for v_π and v_* in terms of $p(s'|s, a)$ and $r(s, a)$, derived in Q1(e), to replace the relevant lines in policy iteration.
  - The **compute_expected_return** function already computes the $p(s'|s, a)$ and $r(s, a)$ for your. Therefore, you only have to implement the incremental update of the expected return given s and a.
  - In particular, the **open_to_close** function that computes, for a single location, the probability of ending the day with $s_{\text{end}} \in [0, 20]$ cars, given that the location started the day with $s_{\text{start}} \in [0, 20 + 5]$ cars. The function should also compute the average reward the location experiences during the day, given that the location started the day with s_{start} cars. This "open to close" function can be precomputed for all 26 possible starting numbers of cars for each location. Then, to compute the joint dynamics between the two locations, all that is necessary is to consider the (deterministic) overnight dynamics, and then combine the appropriate "open to close" dynamics for each location. The function is implemented for you. But the description above will help you understand the implementation.