CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2021) Robert Platt
Northeastern University Due October 12, 2021

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_*.

$$v_*(s) = \max_a q_*(s,a)$$

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

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

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

$$\pi_*(a|s) = arg\max_a q_*(s,a)$$

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

$$\pi_*(a|s) = arg\max_a \sum_{s',r} p(s',r|s,a)\,(r + \gamma v_*(s'))$$

(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).

$$p(s'|s,a) \doteq \Pr\{S_t{=}s' \mid S_{t-1}{=}s, A_{t-1}{=}a\} = \sum_{r\in\mathcal{R}} p(s',r|s,a). \tag{3.4}$$

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1}{=}s, A_{t-1}{=}a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a), \tag{3.5}$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\,(r(s',a) + \gamma v_\pi(s'))$$

$$v_*(s) = \max_a \sum_{s'} p(s'|s,a)\,(r(s',a) + \gamma v_\pi(s'))$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s')\, q_\pi(s',a')$$

$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_a \pi(a'|s') q_*(s',a')$$

2. **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.

Since the condition works for policy iteration according to the policy change to get an optimal one, but more to concern about is the iteration times or the value iteration result. Once we got an optimal value, that should be okay to stop our loop as a policy stable.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   ~~$V(s) \in \mathbb{R}$~~ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
   $Q(s,a) \in \mathbb{R}$
2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

If $q_{\text{old-action}}(s,a) \neq q_\pi(s,a)$, then policy-stable $\leftarrow$ false

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

Value iteration does not have analogous bug. The value iteration is stable or not is not depends on the policy function but the value function. The policy is optimal as long as the result is optimal. Therefore, such bug does not exist.

3. **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 (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
   $Q(s,a) \in \mathbb{R}$
2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

$q \leftarrow Q(s,a)$
$Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)[r + \gamma Q(s',a')]$
$\Delta \leftarrow \max(\Delta, |q - Q(s,a)|)$

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

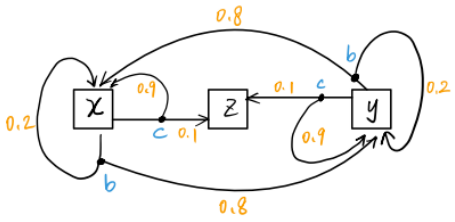
$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_k(s',a'))$$

4. **2 points.** (AIMA 17.10) *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)?

We wish to arrive at state z as soon as possible as a result. The only action that reaches state z is through action b succeeds with low probability, so the agent should minimize the in the process. This suggests that the agent should try action b in state x and try action a to get to state x rather than directly to state in state y.

(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.

$V_\pi(s) = \sum_a \pi(s|a)\, p(s', r|s, a)(r + \gamma V_\pi(s'))$

$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)(r + \gamma V_\pi(s'))$

Initial action is c:

$V(x) = 0.1 \times (-1) + 0.9 \times (-1 + \gamma V(x)) = -1 + 0.9 V(x)$ $\Rightarrow V(x) = -10$

$V(y) = 0.1 \times (-2) + 0.9 \times (-2 + \gamma V(y)) = -2 + 0.9 V(y)$ $V(y) = -20$

$\pi(a) = 0.8 \times (-20) + (0.2) \times 10 = -18$

$\pi(b) = 0.1 \times 0 \times 0.9 \times (-10) = -9$

so action b is preferred for state x

update:

$\pi(a) = 0.8\ (-10) + 0.2(-20) = -12$

$\pi(b) = 0.1(0) + 0.9(-20) = -18$

2nd evaluation:

$V(x) = -1 + 0.1(0) + 0.9V(x)$
$V(y) = -2 + 0.8V(x) + 0.2V(y)$
$\Rightarrow \begin{cases} V(x) = -10 \\ V(y) = -15 \end{cases}$

$\pi(a) = 0.8(-15) + 0.2(-10) = -14$
$\pi(b) = 0.1(0) + 0.9(-15) = -13.5$

so action b is preferred for state x

update:

$\pi(a) = 0.8(-10) + 0.2(-15) = -11$
$\pi(b) = 0.1(0) + 0.9(-15) = -13.5$

so action a is preferred for state x

Therefore, in state y, the preferred action is b to move to state x. And state x is preferred with action a to move to state z.

(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)?

initial policy for both is action b:

$V_\pi(x) = -1 + 0.2V_\pi(x) + 0.8V_\pi(y)$
$V_\pi(y) = -2 + 0.8V_\pi(x) + 0.2V_\pi(y)$
$\Rightarrow \begin{cases} V_\pi(x) \\ V_\pi(y) \end{cases}$ is unsolvable

but with discounting factor,

$V_\pi(x) = -1 + 0.2\gamma V_\pi(x) + 0.8\gamma V_\pi(y)$
$V_\pi(y) = -2 + 0.8\gamma V_\pi(x) + 0.2\gamma V_\pi(y)$

$\Rightarrow \begin{cases} V_\pi(x) = \frac{-2 - 0.4\gamma}{1 - 0.6\gamma^2 - 0.4\gamma} = -1 + 0.9V_\pi(x) \\ V_\pi(y) = \frac{-1 + 0.2\gamma^2 - 1.2\gamma}{0.4 - 0.6\gamma + 0.2\gamma^2} = -2 + 0.9V_\pi(y) \end{cases}$ from (b)

when $V_\pi(x) < -10$, then $\pi(x)$ is action c.
when $V_\pi(y) < -20$, then $\pi(y)$ is action c.

If the policy has action b in both state, it is unsolvable. With the discounting factor, the result is changed. The value range is from negative infinite to -2. By changing the gamma value, we van control the optimal policy for state y and x. So optimal policy is depended on the discount factor.

5. **2 points.** *Implementing dynamic programming algorithms.*
Code/plot: For all algorithms, you may use any reasonable convergence threshold (e.g., $\theta = 10^{-3}$).

(a) Implement the 5×5 grid-world in Example 3.5 (i.e., implement its dynamics function, to be used in dynamic programming). Implement *iterative policy evaluation* and verify that you obtain the value function for the equiprobable random policy, given in Figure 3.2.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

```
[[ 3.3  8.8  4.4  5.3  1.5]
 [ 1.5  3.   2.3  1.9  0.6]
 [ 0.1  0.7  0.7  0.4 -0.4]
 [-1.  -0.4 -0.4 -0.6 -1.2]
 [-1.9 -1.3 -1.2 -1.4 -2. ]]
```

(b) Implement *value iteration* to output both the optimal state-value function and optimal policy for a given MDP (dynamics function). Use your implementation to verify the optimal value function and policy for the 5×5 grid-world (v_* and π_* are given in Figure 3.5).

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

```
[[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]]  ['↑/→', '↑', '↑/←', '↑/←', '↑/←']
```

(c) Implement *policy iteration* to output both the optimal state-value function and optimal policy for a given MDP (dynamics function). You should include the fix you proposed in Q2(a). Use your implementation to verify the optimal value function and policy for the 5×5 grid-world (v_* and π_* are given in Figure 3.5). Consider reusing your code from part (a) in policy iteration.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

```
[[22.  24.4 22.  18.4 16.6]
 [19.8 22.  19.8 16.6 14.9]
 [17.8 19.8 17.8 14.9 13.4]
 [16.  17.8 16.  13.4 12.1]
 [14.4 16.  14.4 12.1 10.9]]
[['→' '↓' '←' '↓' '←']
 ['↑' '↑' '↑' '↑' '↑']
 ['↑' '↑' '↑' '↑' '↑']
 ['↑' '↑' '↑' '↑' '↑']
 ['↑' '↑' '↑' '↑' '↑']]
```

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

(a) **Code/plot:** Replicate Example 4.2 and Figure 4.2. Implement the dynamics function for Jack's car rental problem, and use your policy iteration implementation (ideally from Q5) to solve for the optimal policy and value function. Reproduce the plots shown in Figure 4.2, 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.

> **Example 4.2: Jack's Car Rental** Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number. Suppose λ is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns. To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night. We take the discount rate to be $\gamma = 0.9$ and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight. Figure 4.2 shows the sequence of policies found by policy iteration starting from the policy that never moves any cars.

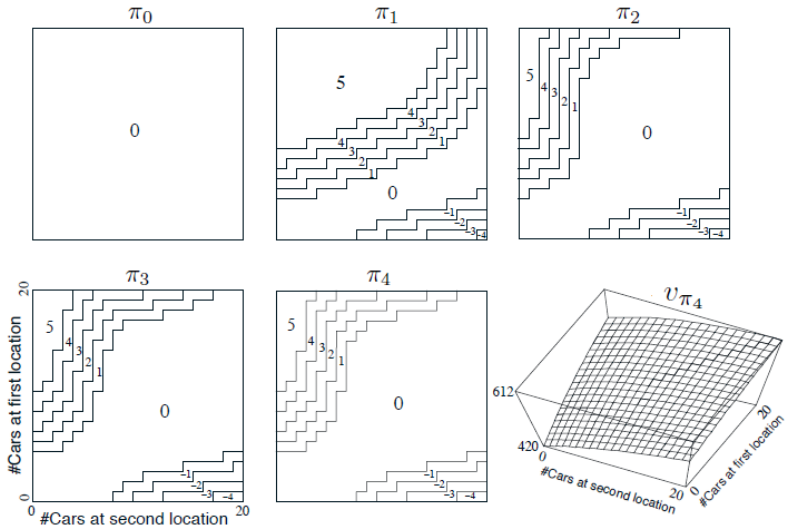


Figure 4.2: The sequence of policies found by policy iteration on Jack's car rental problem, and the final state-value function. The first five diagrams show, for each number of cars at each location at the end of the day, the number of cars to be moved from the first location to the second (negative numbers indicate transfers from the second location to the first). Each successive policy is a strict improvement over the previous policy, and the last policy is optimal. ■

(b) **Code:** Re-solve Jack's car rental problem with the following changes.
Written: Describe how you will change the dynamics function to reflect the following changes.
Plot: Similar to part (a), produce plots of the policy iterates and their respective value functions.
Written: How does your final policy differ from Q6(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 Q6:

- 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.
  - Consider pre-computing and saving your transition and reward functions so that you only have to do it once (assuming it is done correctly). This step could be more costly than policy iteration.
  - Even computing the transition and reward functions might require some care to ensure that it is efficient. One observation is that, once the cars have been moved, the two locations evolve independently until the end of the next day. Therefore, we do not need to consider their transitions (and the myriad of request/return possibilities at each location) jointly.
  - In particular, consider writing an "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 pre-computed 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.
- Useful Python tools: `scipy.stats.poisson`, `matplotlib.pyplot.imshow` (visualizing policy)

7. **[Extra credit.] 2 points.** (AIMA 17.6) *Proving convergence of value iteration.*
Written:

(a) Show that, for any functions f and g,

$$\left|\max_a f(a) - \max_a g(a)\right| \leq \max_a |f(a) - g(a)|$$

Hint: Consider the cases $\max_a f(a) - \max_a g(a) \geq 0$ and $\max_a f(a) - \max_a g(a) < 0$ separately.

case 1: $\max_a f(a) - \max_a g(a) \geq 0$

$$\begin{aligned} |\max_a f(a) - \max_a g(a)| &= \max_a f(a) - \max_a g(a) \\ &= f(a^*) - \max_a g(a) \quad \text{if } a^* \text{ is max value} \\ &\leq f(a^*) - g(a^*) \\ &\leq \max |f(a) - g(a)| \end{aligned}$$

case 2: $\max_a f(a) - \max_a g(a) < 0$

$$\begin{aligned} |\max_a f(a) - \max_a g(a)| &= -(\max_a f(a) - \max_a g(a)) \\ &= \max_a g(a) - \max_a f(a) \\ &= g(a^*) - \max_a f(a) \\ &\leq g(a^*) - f(a^*) \\ &\leq \max |g(a) - f(a)| \\ &= \max |f(a) - g(a)| \end{aligned}$$

Therefore, for all functions f, g,
$|\max_a f(a) - \max_a g(a)| \leq \max |f(a) - g(a)|$.

(b) Let V_k be the k'th iterate of value iteration (assume the two-array version for simplicity, i.e., there is no in-place updating). Let $\mathcal{B}$ denote the *Bellman backup operator*, i.e., the value iteration update can be written as (in vector form):

$$V_{k+1} \leftarrow \mathcal{B}V_k$$

which is equivalent to applying the following assignment for all $s \in \mathcal{S}$:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V_k(s')\right]$$

Let V_i and V_i' be any two value function vectors. Using part (a), show that:

$$\|\mathcal{B}V_i - \mathcal{B}V_i'\|_\infty \leq \gamma \|V_i - V_i'\|_\infty$$

(The ℓ_∞-norm of a vector v is the maximum absolute value of its elements: $\|v\|_\infty \triangleq \max\{|v_1|, |v_2|, \ldots, |v_n|\}$)

$$\begin{aligned}
|BV_i(s) - BV_i'(s)| &= \left|\left[R(s) + \gamma \max_a \sum_{s'} p(s'|s,a) V_i(s')\right] - \left[R(s) + \gamma \max_a \sum_{s'} p(s'|s,a) V_i'(s')\right]\right| \\
&= \gamma \left|\max_a \sum_{s'} p(s'|s,a) V_i(s') - \max_a \sum_{s'} p(s'|s,a) V_i'(s')\right| \\
&\leq \gamma \max_a \left|\sum_{s'} p(s'|s,a) V_i(s') - \sum_{s'} p(s'|s,a) V_i'(s')\right| \\
&= \gamma \left|\sum_{s'} p(s'|s,a) V_i(s') - \sum_{s'} p(s'|s,a) V_i'(s')\right|_\infty \\
&= \gamma \left|\sum_{s'} p(s'|s,a)\left(V_i(s') - V_i'(s')\right)\right|_\infty \\
\|BV_i(s) - BV_i'(s)\|_\infty &= \max_s |BV_i(s) - BV_i'(s)| \\
&\leq \gamma \max_s \left|\sum_{s'} p(s'|s,a)\left(V_i(s') - V_i'(s')\right)\right| \\
&\leq \gamma \max_s |V_i(s) - V_i'(s)| \\
&= \gamma \|V_i - V_i'\|_\infty
\end{aligned}$$

Therefore, $\|BV_i(s) - BV_i'(s)\|_\infty \leq \gamma \|V_i - V_i'\|_\infty$.

(c) The result from part (b) shows that the Bellman backup operator is a *contraction* by a factor of γ. Prove that value iteration always converges to the optimal value function v_*, assuming that $\gamma < 1$. To do this, first assume that the Bellman backup operator $\mathcal{B}$ has a fixed point x, i.e., $\mathcal{B}x = x$. (The proof of this is beyond this scope of this course; see Banach fixed-point theorem for details.) You should then show three things: value iteration converges to this assumed fixed point x, the fixed point is unique, and this unique fixed point is the optimal value function v_*.