Stanford CME 241 (Winter 2025) - Assignment 1

Due: Sunday, January 19 @ 11:59 PM PST on Gradescope.

Assignment instructions:

- Make sure each of the subquestions have answers
- Ensure that group members indicate which problems they're in charge of
- Show work and walk through your thought process where applicable
- Empty code blocks are for your use, so feel free to create more under each section as needed
- Document code with light comments (i.e. 'this function handles visualization')

Submission instructions:

- When complete, fill out your publicly available GitHub repo file URL and group members below, then export or print this .ipynb file to PDF and upload the PDF to Gradescope.

Link to this ipynb file in your public GitHub repo (replace below URL with yours):

https://github.com/JunyoungJeong-acct/RL-book/blob/master/assignments/assignment1.ipynb

Group members (replace below names with people in your group):

- Junyoung Jeong
- Elliot Porter
- Andrew Sung

Imports

In [4]:
```python
import numpy as np
import abc
from dataclasses import dataclass
from typing import Mapping, Dict
from rl.distribution import Categorical, FiniteDistribution
from rl.markov_process import FiniteMarkovProcess
```

Question 1: Snakes and Ladders (Led by Andrew)

In the classic childhood game of Snakes and Ladders, all players start to the left of square 1 (call this position 0) and roll a 6-sided die to represent the number of squares they can move forward. The goal is to reach square 100 as quickly as possible. Landing on the bottom rung of a ladder allows for an automatic free-pass to climb, e.g. square 4 sends you directly to 14; whereas landing on a snake's head forces one to slide all the way to the tail, e.g. square 34 sends you to 6. Note, this game can be viewed as a Markov Process, where the outcome is only depedent on the current state and not the prior trajectory. In this question, we will ask you to both formally describe the Markov Process that describes this game, followed by coding up a version of the game to get familiar with the RL-book libraries.

Problem Statement

How can we model this problem with a Markov Process?

Subquestions

Part (A): MDP Modeling

Formalize the state space of the Snakes and Ladders game. Don't forget to specify the terminal state!

Part (B): Transition Probabilities

Write out the structure of the transition probabilities. Feel free to abbreviate all squares that do not have a snake or ladder.

Part (C): Modeling the Game

Code up a `transition_map: Transition[S]` data structure to represent the transition probabilities of the Snakes and Ladders Markov Process so you can model the game as an instance of `FiniteMarkovProcess`. Use the `traces` method to create sampling traces, and plot the graph of the distribution of time steps to finish the game. Use the image below for the locations of the snakes and ladders.

Snakes and Laddders

Part (A) Answer

Answer:

Suppose that there are n players playing the game. The state space of this game $\mathcal{S} = \{S \in \mathbb{N}^n\}$ consists of a $n \times 1$ vector with entries representing position p_i for each player $i \in \{1, \cdots, n\}$. I assume that the game ends when one of the players finishes the game, so a terminal state is whenever at least one one of the players reaches position 100 - corresponding to a state vector with at least 1 entry of 100. Furthermore, I assume that all players essentially play simultaneously - essentially they all roll the dice at the same time and if multiple players win in the same round they are all winners.

In math, the set of terminal states $\mathcal{T} = \{S \in \mathcal{S} \mid \{S\} \cap \{100\} \neq \emptyset\}$ - essentially is the position of at least one player 100. However, if we wanted to track the ranking of all players and not just the winner, we could let state 100 be an absorbing state, and add additional entries to our vector to track the ranking of the players.

Therefore, a sufficient set of positions that comprise the entries of our state vector would be $\mathbb{N} \cap [0, 100]$. However, some integers are not positions because they immediately transport the player to another positions, so we can remove them without loss. Specifically, using the image provided in Part (C):

```
\begin{align}
    \begin{array}{|c|c|}
        \hline
        \text{Removed State}:\text{Mapping},
        \hline
        1:38,
        4:14,
        8:30,
        21:42,
        28:78,
        32:10,
        36:6,
        48:26,
        50:67,
        62:18,
        71:92,
        80:99,
        88:24,
        95:56,
        97:78,
        \hline
    \end{array}
\end{align}
```

The states in the first column of the table above will be removed from the state space.

Part (B) Answer

I assume that rolling the die to determine the change in position of a player corresponds to a draw from $Unif[\{1, 2, 3, 4, 5, 6\}]$, and that each player's draw is independent of the other players. Let us define a function of the order of positions as $\sigma(j)$. For example, $\sigma(1) = 38$ since there is a ladder on position 1 leading to position 38. This notation allows us to concisely to define the structure of transition policies as $\Pr(S' = \sigma(S + i)) = 6^{-n}$ for any $i \in \{1, 2, 3, 4, 5, 6\}^n$. Essentially, we draw n draws from the uniform, add it to every players current position and each unique (maintaining order) change in position will have a chance of occuring with probability 6^{-n}. Note that this is different from change in state, since we may need to add together repeat states from the snakes and ladders aspect of this game.

Part (C) Answer

In [5]:
```python
# I coded the game for 2 players
# I am not quite sure how to do a variable number of players as the states class do

# Define the state class for snakes and ladders
@dataclass(frozen=True)
class SLState:
    position1: int
    position2: int

    def state1(self) -> int:
        return position1
    def state2(self) -> int:
        return position2

class SLFiniteMP(FiniteMarkovProcess[SLState]):

    def __init__(
        self
    ):
        super().__init__(self.get_transition_map())

    def get_transition_map(self) -> \
            Mapping[SLState, FiniteDistribution[SLState]]:
        d: Dict[SLState, Categorical[SLState]] = {}
        # Dictionary mapping order on board -> position of player
        SL_dict = {1:38, 4:14, 8:30, 21:42, 28:78, 32:10, 36:6, 48:26,
                   50:67, 62:18, 71:92, 80:99, 88:24, 95:56, 97:78}
        positions = np.zeros(2)
        # For every player, every position, and every realization of the die
        for p1 in range(100):
            for p2 in range(100):
                state_probs_map: Mapping[SLState, float] = {
                    SLState(SL_dict.get(p1 + i + 1,p1 + i + 1),SL_dict.get(p2 + j +
                    1/36 if (p1 and p2) < 100
                    else 0
                    for i in range(5)
                    for j in range(5)
                }
                d[SLState(p1,p2)] = Categorical(state_probs_map)
        return d

# if __name__ == '__main__':
#     n = 2

#     Snakes = SLFiniteMP()

#     print("Transition Map")
#     print("--------------")
#     print(Snakes)

#     print("Stationary Distribution")
#     print("-----------------------")
#     Snakes.display_stationary_distribution()
```

Question 2: Markov Decision Processes (Led by Junyoung)

Consider an MDP with an infinite set of states $\mathcal{S} = \{1, 2, 3, \ldots\}$. The start state is $s = 1$. Each state s allows a continuous set of actions $a \in [0, 1]$. The transition probabilities are given by:

$$\mathbb{P}[s+1 \mid s, a] = a, \mathbb{P}[s \mid s, a] = 1 - a \text{ for all } s \in \mathcal{S} \text{ for all } a \in [0, 1]$$

For all states $s \in \mathcal{S}$ and actions $a \in [0, 1]$, transitioning from s to $s + 1$ results in a reward of $1 - a$ and transitioning from s to s results in a reward of $1 + a$. The discount factor $\gamma = 0.5$.

Problem Statement

How can we derive a mathematical formulation for the value function and the optimal policy? And how do those functions change when we modify the action space?

The decision maker (agent) solves the following problem.

$$V^\pi(s) = R(s_0, a_0) + \gamma \mathbb{E}_0 \left[\sum_1^\infty \sum_1^\infty \pi(s_t, a_t) R(s_t, a_t) \right] \tag{1}$$

Subquestions

Part (A): Optimal Value Function

Using the MDP Bellman Optimality Equation, calculate the Optimal Value Function $V^*(s)$ for all $s \in \mathcal{S}$. Given $V^*(s)$, what is the optimal action, a^*, that maximizes the optimal value function?

Part (B): Optimal Policy

Calculate an Optimal Deterministic Policy $\pi^*(s)$ for all $s \in \mathcal{S}$.

Part (C): Changing the Action Space

Let's assume that we modify the action space such that instead of $a \in [0, 1]$ for all states, we restrict the action space to $a \in \left[0, \frac{1}{s}\right]$ for state s. This means that higher states have more restricted action spaces. How does this constraint affect:

- The form of the Bellman optimality equation?
- The optimal value function, $V^*(s)$?
- The structure of the optimal policy, $\pi^*(s)$?

Part (A) Answer

The optimal value function $V^*(s)$ is as follows.

$$V^* = \max_{a \in [0,1]} 0.5\mathbb{E}_a \left[\pi(s,a) \left(a[(1-a) + V^*] + (1-a)[(1+a) + V^*] \right) \right] \tag{2}$$

Since only the change in state and not the realization of the state affects the reward function, we can drop the dependency on s in our value function.

Maximizing $a[(1-a) + V^*(s+1)] + (1-a)[(1+a) + V^*(s)]$ yields $a = \frac{1}{4}$.

Part (B) Answer

The deterministic policy is $\pi^*(s) = \pi(s, \frac{1}{4}) = 1$.

Part (C) Answer

Bellman Optimality Equation Change:

The agent's Bellman Optimality equation is not changed much, but we can no longer drop the dependence on the state in V^* as before.

$$0.5\mathbb{E}_a \left[\pi(s,a) \left(a[(1-a) + V^*(s+1)] + (1-a)[(1+a) + V^*(s)] \right) \right] \tag{3}$$

Optimal Value Function Change:

The optimal value function $V^*(s)$ is now as follows, with the key difference in the restriction on the value of a in the max operator.

$$V^*(s) = \max_{a \in [0,\frac{1}{s}]} 0.5\mathbb{E}_a \left[\pi(s,a) \left(a[(1-a) + V^*(s+1)] + (1-a)[(1+a) + V^*(s)] \right) \right] \quad (4$$

Optimal Policy Change:

The structure of the optimal policy has changed as it is now to choose $\max\{\frac{1}{4}, \frac{1}{s}\}$. We know this to be the case since $(a[(1-a) + V^*(s+1)] + (1-a)[(1+a) + V^*(s)])$ is strictly increasing for $a < \frac{1}{4}$, so the agent will choose the highest possible a if restricted to an action lower than $\frac{1}{4}$.

Question 3: Frog in a Pond (Led by ______)

Consider an array of $n+1$ lilypads on a pond, numbered 0 to n. A frog sits on a lilypad other than the lilypads numbered 0 or n. When on lilypad i ($1 \leq i \leq n-1$), the frog can croak one of two sounds: **A** or **B**.

- If it croaks **A** when on lilypad i ($1 \leq i \leq n-1$):
  - It is thrown to lilypad $i-1$ with probability $\frac{i}{n}$.
  - It is thrown to lilypad $i+1$ with probability $\frac{n-i}{n}$.
- If it croaks **B** when on lilypad i ($1 \leq i \leq n-1$):
  - It is thrown to one of the lilypads $0, \ldots, i-1, i+1, \ldots, n$ with uniform probability $\frac{1}{n}$.

A snake, perched on lilypad 0, will eat the frog if it lands on lilypad 0. The frog can escape the pond (and hence, escape the snake!) if it lands on lilypad n.

Problem Statement

What should the frog croak when on each of the lilypads $1, 2, \ldots, n-1$, in order to maximize the probability of escaping the pond (i.e., reaching lilypad n before reaching lilypad 0)?

Although there are multiple ways to solve this problem, we aim to solve it by modeling it as a **Markov Decision Process (MDP)** and identifying the **Optimal Policy**.

Subquestions

Part (A): MDP Modeling

Express the frog-escape problem as an MDP using clear mathematical notation by defining the following components:

- **State Space**: Define the possible states of the MDP.
- **Action Space**: Specify the actions available to the frog at each state.
- **Transition Function**: Describe the probabilities of transitioning between states for each action.
- **Reward Function**: Specify the reward associated with the states and transitions.

Part (B): Python Implementation

There is starter code below to solve this problem programatically. Fill in each of the 6 `TODO` areas in the code. As a reference for the transition probabilities and rewards, you can make use of the example in slide 16/31 from the following slide deck: https://github.com/coverdrive/technical-documents/blob/master/finance/cme241/Tour-MP.pdf.

Write Python code that:

- Models this MDP.
- Solves the **Optimal Value Function** and the **Optimal Policy**.

Feel free to use/adapt code from the textbook. Note, there are other libraries that are needed to actually run this code, so running it will not do anything. Just fill in the code so that it could run assuming that the other libraries are present.

Part (C): Visualization and Analysis

After running the code, we observe the following graphs for $n = 3$, $n = 10$, and $n = 25$:

FrogGraphs

What patterns do you observe for the **Optimal Policy** as you vary n from 3 to 25? When the frog is on lilypad 13 (with 25 total), what action should the frog take? Is this action different than the action the frog should take if it is on lilypad 1?

Part (A) Answer

State Space:

The state space is $\mathcal{S} = \{i \in \mathbb{N} | 0 \leq i \leq n\}$. The set of terminal states is $\mathcal{T} = \{0, n\}$.

Action Space:

The frog's actions set is $\mathcal{A} = \{A, B\}$.

Transition Function:

The transition function $\pi \mathcal{S} \times \mathcal{A} \to [0, 1]$ can be defined as:

$$\pi(i, a, i') = \begin{cases} 0 & \text{if } i \in \mathcal{T} \\ \frac{i}{n} & \text{if } i \notin \mathcal{T} \text{ and } i' = i - 1 \text{ and a} = \text{A} \\ \frac{n-i}{n} & \text{if } i \notin \mathcal{T} \text{ and } i' = i + 1 \text{ and a} = \text{A} \\ \frac{1}{n} & \text{if } i \notin \mathcal{T} \text{ and a} = \text{B} \end{cases} \tag{5}$$

Reward Function:

The reward function $\mathcal{R} : \mathcal{S} \to \bar{\mathbb{R}}$ is defined

$$R(i) = \begin{cases} \infty & \text{if } i = n \\ -\infty & \text{if } i = 0 \end{cases} \quad (6)$$

Part (B) Answer

In [7]:

```
MDPRefined = dict
def get_lily_pads_mdp(n: int) -> MDPRefined:
    data = {
        i: {
            'A': {
                i - 1: i/n, # TODO: fill in with the correct transition probabiliti
                i + 1: (n-i)/n, # TODO: fill in with the correct transition probabi
            },
            'B': {
                i: 1/n, # TODO: fill in with the correct transition probabilities
            }
        } for i in range(1, n)
    }
    data[0] = 0 # TODO: this is the initial state, so what would be the correct tra
    data[n] = 0 # TODO: similarly, this is the terminal state, so what would be the
    return MDPRefined(data)

# This will give us our optimal value function by taking the fixed point of the Bel
Mapping = dict
def direct_bellman(n: int) -> Mapping[int, float]:
    probs = get_lily_pads_mdp(n)
    vf = [0.5] * (n + 1)
    vf[0] = -1000000000
    vf[n] = 1000000000
    tol = 1e-8
    epsilon = tol * 1e4
    while epsilon >= tol:
        old_vf = [v for v in vf]
        for i in range(1, n):
            vf_A = probs[i]["A"][i-1]*old_vf[i-1] + probs[i]["A"][i+1]*old_vf[i+1]
            vf_B = 0
            for b in range(0,n):
                vf_B = vf_B + probs[i]["B"][i]*old_vf[b]
            vf[i] = 0.5*(max(vf_A,vf_B)) # TODO: fill in with the Bellman update
        epsilon = max(abs(old_vf[i] - v) for i, v in enumerate(vf))
    return {v: f for v, f in enumerate(vf)}

# We can recover the optimal policy as follows (for n = 25)
n = 25
optimal_policy = np.repeat("C", n-1)
vStar = direct_bellman(n)
probs = get_lily_pads_mdp(n)
for i in range(1, n):
    vf_A = probs[i]["A"][i-1]*vStar[i-1] + probs[i]["A"][i+1]*vStar[i+1]
    vf_B = 0
    for b in range(0,n):
        vf_B = vf_B + probs[i]["B"][i]*vStar[b]
    if vf_A >= vf_B:
        optimal_policy[i-1] = "A"
    else:
        optimal_policy[i-1] = "B"
print(optimal_policy)
```

```
['B' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'
 'A' 'A' 'A' 'A' 'A' 'A']
```

Part (C) Answer

The optimal policy is represented by the action with a higher value for Q^*. In all 3 parameterizations, the optimal policy for the frog is to choose A for every lilypad > 1. Therefore the frog's optimal action on lilypad 13 is A, but when the frog is on lilypad 1 the frog should choose B.

Question 5: Manual Value Iteration (Led by Elliot Porter)

Consider a simple MDP with $\mathcal{S} = \{s_1, s_2, s_3\}, \mathcal{T} = \{s_3\}, \mathcal{A} = \{a_1, a_2\}$. The State Transition Probability function

$$\mathcal{P} : \mathcal{N} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$$

is defined as:

$$\mathcal{P}(s_1, a_1, s_1) = 0.25, \mathcal{P}(s_1, a_1, s_2) = 0.65, \mathcal{P}(s_1, a_1, s_3) = 0.1$$

$$\mathcal{P}(s_1, a_2, s_1) = 0.1, \mathcal{P}(s_1, a_2, s_2) = 0.4, \mathcal{P}(s_1, a_2, s_3) = 0.5$$

$$\mathcal{P}(s_2, a_1, s_1) = 0.3, \mathcal{P}(s_2, a_1, s_2) = 0.15, \mathcal{P}(s_2, a_1, s_3) = 0.55$$

$$\mathcal{P}(s_2, a_2, s_1) = 0.25, \mathcal{P}(s_2, a_2, s_2) = 0.55, \mathcal{P}(s_2, a_2, s_3) = 0.2$$

The Reward Function

$$\mathcal{R} : \mathcal{N} \times \mathcal{A} \rightarrow \mathbb{R}$$

is defined as:

$$\mathcal{R}(s_1, a_1) = 8.0, \mathcal{R}(s_1, a_2) = 10.0$$

$$\mathcal{R}(s_2, a_1) = 1.0, \mathcal{R}(s_2, a_2) = -1.0$$

Assume a discount factor of $\gamma = 1$.

Problem Statement

Your task is to determine an Optimal Deterministic Policy **by manually working out** (not with code) the first two iterations of the Value Iteration algorithm.

Subquestions

Part (A): 2 Iterations

1. Initialize the Value Function for each state to be its $\max$ (over actions) reward, i.e., we initialize the Value Function to be $v_0(s_1) = 10.0, v_0(s_2) = 1.0, v_0(s_3) = 0.0$. Then manually calculate $q_k(\cdot, \cdot)$ and $v_k(\cdot)$ from $v_{k-1}(\cdot)$ using the Value Iteration update, and then calculate the greedy policy $\pi_k(\cdot)$ from $q_k(\cdot, \cdot)$ for $k = 1$ and $k = 2$ (hence, 2 iterations).

Part (B): Argument

1. Now argue that $\pi_k(\cdot)$ for $k > 2$ will be the same as $\pi_2(\cdot)$. *Hint*: You can make the argument by examining the structure of how you get $q_k(\cdot, \cdot)$ from $v_{k-1}(\cdot)$. With this argument, there is no need to go beyond the two iterations you performed above, and so you can establish $\pi_2(\cdot)$ as an Optimal Deterministic Policy for this MDP.

Part (C): Policy Evaluation

1. Using the policy $\pi_2(\cdot)$, compute the exact value function $V^{\pi_2}(s)$ for all $s \in S$.

Part (D): Sensitivity Analysis

Assume the reward for $R(s_1, a_2)$ is modified to 11.0 instead of 10.0.

1. Perform one iteration of Value Iteration starting from the initialized value function $v_0(s)$, where $v_0(s)$ remains the same as in the original problem.
2. Determine whether this change impacts the Optimal Deterministic Policy $\pi(\cdot)$. If it does, explain why.

Part (A) Answer

Our starting guess for the value function is:

$$v_0(s_1) = 10.0 \tag{7}$$
$$v_0(s_2) = 1.0 \tag{8}$$
$$v_0(s_3) = 0.0 \tag{9}$$

Our Bellman Operator $\mathcal{B}$ is:

$$v_k = \mathcal{B}(v_{k-1}) = r(s,a) + \sum_{s'} \mathcal{P}(s,a,s')v_{k-1}(s') \tag{10}$$

Therefore the action value functions are as follows.

$$q_1(s_1,a_1) = 8.0 + 0.25 * 10 + 0.65 * 1 + 0.1 * 0 = 11.5 \tag{11}$$
$$q_1(s_2,a_1) = 1.0 + 0.1 * 10 + 0.4 * 1 + 0.5 * 0 = 2.4 \tag{12}$$
$$q_1(s_1,a_2) = 10.0 + 0.3 * 10 + 0.15 * 1 + 0.55 * 0 = 13.15 \tag{13}$$
$$q_1(s_2,a_2) = -1.0 + 0.25 * 10 + 0.55 * 1 + 0.2 * 0 = 2.05 \tag{14}$$

Applying a greedy policy yields the following state value functions.

$$v_1(s_2) = 13.5 \tag{15}$$
$$v_1(s_2) = 2.4 \tag{16}$$
$$v_1(s_3) = 0.0 \tag{17}$$

Repeat one more round. Action value functions.

$$q_2(s_1,a_1) = 8.0 + 0.25 * 13.5 + 0.65 * 2.4 + 0.1 * 0 = 12.935 \tag{18}$$
$$q_2(s_2,a_1) = 1.0 + 0.1 * 13.5 + 0.4 * 2.4 + 0.5 * 0 = 3.31 \tag{19}$$
$$q_2(s_1,a_2) = 10.0 + 0.3 * 13.5 + 0.15 * 2.4 + 0.55 * 0 = 14.41 \tag{20}$$
$$q_2(s_2,a_2) = -1.0 + 0.25 * 13.5 + 0.55 * 2.4 + 0.2 * 0 = 3.695 \tag{21}$$

Applying a greedy policy yields the following state value functions.

$$v_2(s_2) = 14.41 \tag{22}$$
$$v_2(s_2) = 3.695 \tag{23}$$
$$v_2(s_3) = 0.0 \tag{24}$$

Part (B) Answer:

In iteration 2 of our fixed point algorithm - the optimal deterministic policy is to choose action a_2 in both states s_1 and s_2. This is because the action value function is higher in both states under action 2. This will be the optimal policy for any iteration k because the only way that a_1 will be preferred is if the value function in state 1 or 2 decreases, or if the value in the terminal state somehow becomes nonzero. However, the weighted average of the values in state 1 and 2 must be weakly increasing over iterations of our algorithm, meaning that action 2 will be optimal in perpetuity.

Part (C) Answer:

Given that choosing a_2 is always optimal - we can solve for our optimal values as follows.

$$V^{\pi^2}(s_1) = 10.0 + 0.3 * V^{\pi^2}(s_1) + 0.15 * V^{\pi^2}(s_2) \tag{25}$$
$$V^{\pi^2}(s_2) = -1.0 + 0.25 * V^{\pi^2}(s_1) + 0.55 * V^{\pi^2}(s_2) \tag{26}$$
$$V^{\pi^2}(s_3) = 0 \tag{27}$$

Solving this system yields

$$V^{\pi^2}(s_1) = 15.68 \tag{28}$$
$$V^{\pi^2}(s_2) = 6.49 \tag{29}$$
$$V^{\pi^2}(s_3) = 0 \tag{30}$$

Part (D) Answer

Value Iteration:

Our starting guess for the value function is now

$$v_0(s_1) = 11.0 \tag{31}$$
$$v_0(s_2) = 1.0 \tag{32}$$
$$v_0(s_3) = 0.0 \tag{33}$$

Our Bellman Operator $\mathcal{B}$ is:

$$v_k = \mathcal{B}(v_{k-1}) = r(s,a) + \sum_{s'} \mathcal{P}(s,a,s') v_{k-1}(s') \tag{34}$$

Therefore the action value functions are as follows.

$$q_1(s_1,a_1) = 8.0 + 0.25 * 11 + 0.65 * 1 + 0.1 * 0 = 11.75 \tag{35}$$
$$q_1(s_2,a_1) = 1.0 + 0.1 * 11 + 0.4 * 1 + 0.5 * 0 = 2.5 \tag{36}$$
$$q_1(s_1,a_2) = 11 + 0.3 * 11 + 0.15 * 1 + 0.55 * 0 = 14.45 \tag{37}$$
$$q_1(s_2,a_2) = -1.0 + 0.25 * 11 + 0.55 * 1 + 0.2 * 0 = 2.3 \tag{38}$$

Applying a greedy policy yields the following state value functions.

$$v_1(s_2) = 13.5 \tag{39}$$
$$v_1(s_2) = 2.4 \tag{40}$$
$$v_1(s_3) = 0.0 \tag{41}$$

Optimal Deterministic Policy:

The optimal deterministic policy remains unchanged - always pick a_2. The gap between $q_1(s_2, a_1)$ and $q_1(s_2, a_2)$ is even smaller than in the previous parameter setup so in the second iteration of our algorithm we will have the same result emerge.

Question 6: Fixed-Point and Policy Evaluation True/False Questions (Led by Andrew Sung)

Recall Section: Key Formulas and Definitions

Bellman Optimality Equation

The Bellman Optimality Equation for state-value functions is:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s') \right].$$

For action-value functions:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} Q^*(s',a').$$

Contraction Property

The Bellman Policy Operator B^π is a contraction under the L^∞-norm:

$$\|B^\pi(X) - B^\pi(Y)\|_\infty \leq \gamma \|X - Y\|_\infty.$$

This guarantees convergence to a unique fixed point.

Policy Iteration

Policy Iteration alternates between:

1. **Policy Evaluation**: Compute V^π for the current policy π.
2. **Policy Improvement**: Generate a new policy π' by setting:

$$\pi'(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} P(s,a,s') V^\pi(s') \right].$$

Discounted Return

The discounted return from time step t is:

$$G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i,$$

where $\gamma \in [0,1)$ is the discount factor.

True/False Questions (Provide Justification)

1. **True/False**: If $Q^\pi(s,a) = 5$, $P(s,a,s') = 0.5$ for $s' \in \{s_1, s_2\}$, and the immediate reward $R(s,a)$ increases by 2, the updated action-value function $Q^\pi(s,a)$ also increases by 2.

---

2. **True/False**: For a discount factor $\gamma = 0.9$, the discounted return for rewards $R_1 = 5, R_2 = 3, R_3 = 1$ is greater than 6.

3. **True/False**: The Bellman Policy Operator $B^\pi(V) = R^\pi + \gamma P^\pi \cdot V$ satisfies the contraction property for all $\gamma \in [0, 1)$, ensuring a unique fixed point.

---

4. **True/False**: In Policy Iteration, the Policy Improvement step guarantees that the updated policy π' will always perform strictly better than the previous policy π.

---

5. **True/False**: If $Q^\pi(s, a) = 10$ for all actions a in a state s, then the corresponding state-value function $V^\pi(s) = 10$, regardless of the policy π.

---

6. **True/False**: The discounted return $G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i$ converges to a finite value for any sequence of bounded rewards if $\gamma < 1$.

Answers (Provide justification, brief explanations are fine)

Question 1:

Answer: TRUE

Justification:
By definition of the action-value function under a policy π:

$$Q^\pi(s,a) \;=\; R(s,a) \;+\; \gamma \sum_{s'} P(s,a,s')\, V^\pi(s').$$

Increasing $R(s, a)$ by 2 adds exactly $+2$ to $Q^\pi(s, a)$ because the immediate reward term appears *without* a discount factor (i.e., multiplied by $\gamma^0 = 1$). Therefore, $Q^\pi(s, a)$ also increases by 2.

However, we note that if $R(s, a)$ is effected for future rewards as well - then the action value may increase more than 2 rendering this statement false.

Question 2:

Answer: TRUE

Justification:
The discounted sum is

$$5 + 0.9 \times 3 + 0.9^2 \times 1 \;=\; 5 + 2.7 + 0.81 \;=\; 8.51,$$

which is clearly more than 6.

Question 3:

Answer: TRUE

Justification:
It is a standard result that B^π is a γ-contraction in the L^∞ norm as long as $\gamma < 1$. By the Banach Fixed-Point Theorem, a γ-contraction has exactly one fixed point, which is V^π.

Question 4:

Answer: FALSE

Justification:
The Policy Improvement Theorem states that π' is *at least as good as* π. It will be strictly better unless π was already optimal. If π is optimal, then $\pi' = \pi$, so there is no strict improvement.

Question 5:

Answer: TRUE

Justification:
Recall that

$$V^\pi(s) \;=\; \sum_a \pi(a \mid s)\, Q^\pi(s,a).$$

If $Q^\pi(s,a) = 10$ *for every action* a, then any weighted average of 10 is still 10, so $V^\pi(s) = 10$ regardless of how π distributes probability over a.

Question 6:

Answer: TRUE

Justification:
If $|R_i| \le M$ for all i, then each term satisfies

$$|\gamma^{\,i-(t+1)} R_i| \;\le\; M\,\gamma^{\,i-(t+1)}.$$

This is bounded by the convergent geometric series $\sum_{i=t+1}^{\infty} M\,\gamma^{\,i-(t+1)}$. Therefore, for $\gamma < 1$, G_t converges to a finite value.