

COMPSCI 687 Homework 4 - Fall 2024

Due **November 15**, 11:55pm Eastern Time

SHREYA BIRTHARE

1 Instructions

This homework assignment consists of a written portion and a programming portion. While you may discuss problems with your peers (e.g., to discuss high-level approaches), you must answer the questions on your own. In your submission, do explicitly list all students with whom you discussed this assignment. Submissions must be typed (handwritten and scanned submissions will not be accepted). You must use L^AT_EX. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code. You **must** use Python 3 for your homework code. You may not use any reinforcement learning or machine learning specific libraries in your code, e.g., TensorFlow, PyTorch, or scikit-learn. You *may* use libraries like numpy and matplotlib, though. The automated system will not accept assignments after 11:55pm on November 15. The tex file for this homework can be found [here](#).

Part One: Written (40 Points Total)

In all questions below, unless stated otherwise:

- Always show your work step by step, explaining your reasoning in detail and how it supports your answer or argument.
- Simplify all expressions where possible.
- If any part of a question is unclear (e.g., what is being asked, or the type of discussion required), please consult the TAs.
- For all questions—especially open-ended ones—clearly state your assumptions (e.g., “I interpret this question as X or Y. I find X more reasonable due to reasons A, B, and C, so I will proceed with that assumption”).

1. (**10 Points**) Consider an MDP, M , where $\gamma = 0.75$. Let $\mathcal{S} = \{s_1, s_2, s_3\}$ and assume that the agent is following some policy, π . The agent executed π four times and obtained the following trajectories, where (for simplicity) each trajectory is represented as a sequence of states and corresponding rewards:

- **Trajectory 1:** $s_1, 3, s_2, -2, s_3, 7, s_1, 5, s_2, 4, s_3, -1, s_2, 0$.
- **Trajectory 2:** $s_2, -3, s_1, 6, s_1, 2, s_3, -4, s_1, 8, s_3, 10$.
- **Trajectory 3:** $s_3, 4, s_1, -1, s_2, 6, s_3, 2, s_2, 7, s_1, -1, s_3, 3, s_1, 3$.
- **Trajectory 4:** $s_1, -5, s_2, 1, s_1, 4, s_3, 6, s_2, -3, s_3, 4, s_1, 9$.

Estimate the state value function using *Second-Visit Monte Carlo* and also using *Every-Visit Monte Carlo*. Show all returns used in your computation of these estimates.

Second Visit MC:

Trajectory 1:

$$\begin{aligned}
s_1 &= 5 + \gamma * 4 + \gamma^2 * (-1) + \gamma^3 * 0 \\
&= 5 + 0.75 * 4 + 0.75^2 * (-1) + 0.75^3 * 0 \\
&= 5 = 3 - 0.5625 \\
&= 7.4375
\end{aligned} \tag{1.1}$$

$$\begin{aligned}
s_2 &= 4 + \gamma * (-1) + \gamma^2 * 0 \\
&= 4 + 0.75 * (-1) + 0.75^2 * 0 \\
&= 4 - 0.75 \\
&= 3.25
\end{aligned} \tag{1.2}$$

$$\begin{aligned}
s_3 &= -1 + \gamma * 0 \\
&= -1 + 0.75 * 0 \\
&= -1
\end{aligned} \tag{1.3}$$

Trajectory 2:

$$\begin{aligned}
s_1 &= 2 + \gamma * (-4) + \gamma^2 * 8 + \gamma^3 * 10 \\
&= 2 + 0.75 * (-4) + 0.75^2 * 8 + 0.75^3 * 10 \\
&= 2 - 3 + 4.5 + 4.21875 \\
&= 7.71875
\end{aligned} \tag{2.1}$$

$$s_2 = 0 \tag{2.2}$$

(as there is no second s_2 in Trajectory 2)

$$s_3 = 10 \tag{2.3}$$

Trajectory 3:

$$\begin{aligned}
s_1 &= -1 + \gamma * 3 + \gamma^2 * 3 \\
&= -1 + 0.75 * 3 + 0.75^2 * 3 \\
&= -1 + 2.25 + 1.6875 \\
&= 2.9375
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
s_2 &= 7 + \gamma * (-1) + \gamma^2 * 3 + \gamma^3 * 3 \\
&= 7 + 0.75 * (-1) + 0.75^2 * 3 + 0.75^3 * 3 \\
&= 7 - 0.75 + 1.6875 + 1.265625 \\
&= 9.203125
\end{aligned} \tag{3.2}$$

$$\begin{aligned}
s_3 &= 2 + \gamma * 7 + \gamma^2 * (-1) + \gamma^3 * 3 + \gamma^4 * 3 \\
&= 2 + 0.75 * 7 + 0.75^2 * (-1) + 0.75^3 * 3 + 0.75^4 * 3 \\
&= 2 + 5.25 - 0.5625 + 1.265625 + 0.9492188 \\
&= 8.9023438
\end{aligned} \tag{3.3}$$

Trajectory 4:

$$\begin{aligned}
s_1 &= 4 + \gamma * 6 + \gamma^2 * (-3) + \gamma^3 * 4 + \gamma^4 * 9 \\
&= 4 + 0.75 * 6 + 0.75^2 * (-3) + 0.75^3 * 4 + 0.75^4 * 9 \\
&= 4 + 4.5 - 1.6875 + 1.6875 + 2.8476562 \\
&= 11.347656
\end{aligned} \tag{4.1}$$

$$\begin{aligned}
s_2 &= -3 + \gamma * 4 + \gamma^2 * 9 \\
&= -3 + 0.75 * 4 + 0.75^2 * 9 \\
&= -3 + 3 + 5.0625 \\
&= 5.0625
\end{aligned} \tag{4.2}$$

$$\begin{aligned}
s_3 &= 4 + \gamma * 9 \\
&= 4 + 0.75 * 9 \\
&= 10.75
\end{aligned} \tag{4.3}$$

From equations 1.1, 2.1, 3.1, 4.1:

$$\begin{aligned}
s_1 &= \frac{(7.4375 + 7.71875 + 2.9375 + 11.347656)}{4} \\
&= \frac{29.441406}{4} \\
&= 7.3603515
\end{aligned}$$

From equations 1.2, 3.2, 4.2: *(Note equation 2.2 won't be considered for calculation as we do not see second occurrence of s_2 in trajectory 2)*

$$\begin{aligned}
s_2 &= \frac{3.25 + 9.203125 + 5.0625}{3} \\
&= 5.8385417
\end{aligned}$$

From equations 1.3, 2.3, 3.3, 4.3:

$$\begin{aligned}
s_3 &= \frac{-1 + 10 + 8.9023438 + 10.75}{4} \\
&= 7.163086
\end{aligned}$$

Every Visit MC:

Trajectory 1:

$$\begin{aligned}
s_1 &= 3 + \gamma * (-2) + \gamma^2 * 7 + \gamma^3 * 5 + \gamma^4 * 4 + \gamma^5 * (-1) + \gamma^6 * 0 \\
&= 3 + 0.75 * (-2) + 0.75^2 * 7 + 0.75^3 * 5 + 0.75^4 * 4 + 0.75^5 * (-1) + 0.75^6 * 0 \\
&= 8.5751953125 \\
s_2 &= -2 + \gamma * 7 + \gamma^2 * 5 + \gamma^3 * 4 + \gamma^4 * (-1) + \gamma^5 * 0 \\
&= -2 + 0.75 * 7 + 0.75^2 * 5 + 0.75^3 * 4 + 0.75^4 * (-1) + 0.75^5 * 0 \\
&= 7.43359375 \\
s_3 &= 7 + \gamma * 5 + \gamma^2 * 4 + \gamma^3 * (-1) + \gamma^4 * 0 \\
&= 7 + 0.75 * 5 + 0.75^2 * 4 + 0.75^3 * (-1) + 0.75^4 * 0 \\
&= 12.578125 \\
s_1 &= 5 + \gamma * 4 + \gamma^2 * (-1) + \gamma^3 * 0 \\
&= 5 + 0.75 * 4 + 0.75^2 * (-1) + 0.75^3 * 0 \\
&= 7.4375 \\
s_2 &= 4 + \gamma * (-1) + \gamma^2 * 0 \\
&= 4 + 0.75 * (-1) + 0.75^2 * 0 \\
&= 3.25 \\
s_3 &= -1 + \gamma * 0 \\
&= -1 + 0.75 * 0 \\
&= -1 \\
s_2 &= 0
\end{aligned}$$

So for Trajectory 1:

$$s_1 = [8.5751953125, 7.4375] \quad (5.1)$$

$$s_2 = [7.43359375, 3.25, 0] \quad (5.2)$$

$$s_3 = [12.578125, -1] \quad (5.3)$$

Trajectory 2:

$$\begin{aligned}
s_2 &= -3 + \gamma * 6 + \gamma^2 * 2 + \gamma^3 * (-4) + \gamma^4 * 8 + \gamma^5 * 10 \\
&= -3 + 0.75 * 6 + 0.75^2 * 2 + 0.75^3 * (-4) + 0.75^4 * 8 + 0.75^5 * 10 \\
&= 5.841796875 \\
s_1 &= 6 + \gamma * 2 + \gamma^2 * (-4) + \gamma^3 * 8 + \gamma^4 * 10 \\
&= 6 + 0.75 * 2 + 0.75^2 * (-4) + 0.75^3 * 8 + 0.75^4 * 10 \\
&= 11.7890625 \\
s_1 &= 2 + \gamma * (-4) + \gamma^2 * 8 + \gamma^3 * 10 \\
&= 2 + 0.75 * (-4) + 0.75^2 * 8 + 0.75^3 * 10 \\
&= 7.71875 \\
s_3 &= -4 + \gamma * 8 + \gamma^2 * 10 \\
&= -4 + 0.75 * 8 + 0.75^2 * 10 \\
&= 7.625 \\
s_1 &= 8 + \gamma * 10 \\
&= 8 + 0.75 * 10 \\
&= 15.5 \\
s_3 &= 10
\end{aligned}$$

So for Trajectory 2:

$$s_1 = [11.7890625, 7.71875, 15.5] \quad (6.1)$$

$$s_2 = [5.841796875] \quad (6.2)$$

$$s_3 = [7.625, 10] \quad (6.3)$$

Trajectory 3:

$$\begin{aligned} s_3 &= 4 + \gamma * (-1) + \gamma^2 * 6 + \gamma^3 * 2 + \gamma^4 * 7 + \gamma^5 * (-1) + \gamma^6 * 3 + \gamma^7 * 3 \\ &= 4 + 0.75 * (-1) + 0.75^2 * 6 + 0.75^3 * 2 + 0.75^4 * 7 + 0.75^5 * (-1) + 0.75^6 * 3 + 0.75^7 * 3 \\ &= 10.3806762695 \end{aligned}$$

$$\begin{aligned} s_1 &= -1 + \gamma * 6 + \gamma^2 * 2 + \gamma^3 * 7 + \gamma^4 * (-1) + \gamma^5 * 3 + \gamma^6 * 3 \\ &= -1 + 0.75 * 6 + 0.75^2 * 2 + 0.75^3 * 7 + 0.75^4 * (-1) + 0.75^5 * 3 + 0.75^6 * 3 \\ &= 8.50756835938 \end{aligned}$$

$$\begin{aligned} s_2 &= 6 + \gamma * 2 + \gamma^2 * 7 + \gamma^3 * (-1) + \gamma^4 * 3 + \gamma^5 * 3 \\ &= 6 + 0.75 * 2 + 0.75^2 * 7 + 0.75^3 * (-1) + 0.75^4 * 3 + 0.75^5 * 3 \\ &= 12.6767578125 \end{aligned}$$

$$\begin{aligned} s_3 &= 2 + \gamma * 7 + \gamma^2 * (-1) + \gamma^3 * 3 + \gamma^4 * 3 \\ &= 2 + 0.75 * 7 + 0.75^2 * (-1) + 0.75^3 * 3 + 0.75^4 * 3 \\ &= 8.90234375 \end{aligned}$$

$$\begin{aligned} s_2 &= 7 + \gamma * (-1) + \gamma^2 * 3 + \gamma^3 * 3 \\ &= 7 + 0.75 * (-1) + 0.75^2 * 3 + 0.75^3 * 3 \\ &= 9.203125 \end{aligned}$$

$$\begin{aligned} s_1 &= -1 + \gamma * 3 + \gamma^2 * 3 \\ &= -1 + 0.75 * 3 + 0.75^2 * 3 \\ &= 2.9375 \end{aligned}$$

$$\begin{aligned} s_3 &= 3 + \gamma * 3 \\ &= 3 + 0.75 * 3 \\ &= 5.25 \end{aligned}$$

$$s_1 = 3$$

So for Trajectory 3:

$$s_1 = [8.50756835938, 2.9375, 3] \quad (7.1)$$

$$s_2 = [12.6767578125, 9.203125] \quad (7.2)$$

$$s_3 = [10.3806762695, 8.90234375, 5.25] \quad (7.3)$$

Trajectory 4:

$$\begin{aligned}
s_1 &= -5 + \gamma * 1 + \gamma^2 * 4 + \gamma^3 * 6 + \gamma^4 * (-3) + \gamma^5 * 4 + \gamma^6 * 9 \\
&= -5 + 0.75 * 1 + 0.75^2 * 4 + 0.75^3 * 6 + 0.75^4 * (-3) + 0.75^5 * 4 + 0.75^6 * 9 \\
&= 2.13305664063 \\
s_2 &= 1 + \gamma * 4 + \gamma^2 * 6 + \gamma^3 * (-3) + \gamma^4 * 4 + \gamma^5 * 9 \\
&= 1 + 0.75 * 4 + 0.75^2 * 6 + 0.75^3 * (-3) + 0.75^4 * 4 + 0.75^5 * 9 \\
&= 9.5107421875 \\
s_3 &= 4 + \gamma * 6 + \gamma^2 * (-3) + \gamma^3 * 4 + \gamma^4 * 9 \\
&= 4 + 0.75 * 6 + 0.75^2 * (-3) + 0.75^3 * 4 + 0.75^4 * 9 \\
&= 11.34765625 \\
s_3 &= 6 + \gamma * (-3) + \gamma^2 * 4 + \gamma^3 * 9 \\
&= 6 + 0.75 * (-3) + 0.75^2 * 4 + 0.75^3 * 9 \\
&= 9.796875 \\
s_2 &= -3 + \gamma * 4 + \gamma^2 * 9 \\
&= -3 + 0.75 * 4 + 0.75^2 * 9 \\
&= 5.0625 \\
s_3 &= 4 + \gamma * 9 \\
&= 4 + 0.75 * 9 \\
&= 10.75 \\
s_1 &= 9
\end{aligned}$$

So for Trajectory 4,

$$s_1 = [2.13305664063, 11.34765625, 9] \quad (8.1)$$

$$s_2 = [9.5107421875, 5.0625] \quad (8.2)$$

$$s_3 = [9.796875, 10.75] \quad (8.3)$$

Using equations 5.1, 6.1, 7.1, 8.1:

$$\begin{aligned}
s_1 &= \frac{8.5751953125 + 7.4375 + 11.7890625 + 7.71875 + 15.5 + 8.50756835938 + 2.9375 + 3 + 2.13305664063 + 11.34765625 + 9}{11} \\
&= 7.9951171875
\end{aligned}$$

Using equations 5.2, 6.2, 7.2, 8.2:

$$\begin{aligned}
s_2 &= \frac{7.43359375 + 3.25 + 0 + 5.841796875 + 12.6767578125 + 9.203125 + 9.5107421875 + 5.0625}{8} \\
&= 6.62231445313
\end{aligned}$$

Using equations 5.3, 6.3, 7.3, 8.3:

$$\begin{aligned}
s_3 &= \frac{12.578125 - 1 + 7.625 + 10 + 10.3806762695 + 8.90234375 + 5.25 + 9.796875 + 10.75}{9} \\
&= 8.25366889106
\end{aligned}$$

Final Answers:

Second Visit MC: $s_1 = 7.3603515$, $s_2 = 5.8385417$, $s_3 = 7.163086$

Every Visit MC: $s_1 = 7.9951171875$, $s_2 = 6.62231445313$, $s_3 = 8.25366889106$

2. **(8 Points)** In class, when discussing TD learning algorithms, we defined the temporal difference error (TD error), δ_t , as $\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t)$ where v could be, for instance, the current estimate of the value function. What is $\mathbb{E}[\delta_t | S_t = s]$ if δ_t uses the true state-value function, v^π ? Show all the steps of your derivation.

$$\begin{aligned}
\mathbb{E}[\delta_t | S_t = s] &= \mathbb{E}[R_t + \gamma v(S_{t+1}) - v(S_t) | S_t = s] \\
&= \mathbb{E}[R_t + \gamma v(S_{t+1}) | S_t = s] - \mathbb{E}[v(S_t) | S_t = s] \\
&= \mathbb{E}[R_t | S_t = s] + \gamma \mathbb{E}[v(S_{t+1}) | S_t = s] - v(S_t) \\
&\quad (\mathbb{E}[v(S_t) | S_t = s] = v(S_t) \text{ as expected value of a state at time } t \text{ given that at time } t \text{ it is in state } S \\
&\quad \text{is the same as its value function at time } t \text{ and given it uses the true state-value function.}) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \mathbb{E}[R_t | S_t = s, A_t = a] + \gamma \sum_{a \in \mathcal{A}} \pi(s, a) \mathbb{E}[v(S_{t+1}) | S_t = s, A_t = a] - v(S_t) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) R(s, a) + \gamma \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') \mathbb{E}[v(S_{t+1}) | S_t = s, A_t = a, S_{t+1} = s'] - v(S_t) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) R(s, a) + \gamma \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') v(S_{t+1}) - v(S_t) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') R(s, a) + \gamma \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') v(S_{t+1}) - v(S_t) \\
&\quad (\text{As } \sum_{s' \in \mathcal{S}} p(s, a, s') = 1 \text{ and multiplying this with } R(s, a) \text{ does not affect that term}) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') \left[R(s, a) + \gamma v(S_{t+1}) \right] - v(S_t) \\
&\quad (\text{Given that true state value is used}) \\
&= v(S_t) - v(S_t) \\
&= 0
\end{aligned}$$

3. **(22 Points Total)** Assume you have a fixed batch of training data consisting of many possible transitions/experiences generated by running a given policy, π . In particular, assume that the training data consists of n sample experiences that the agent collected while interacting with its environment, where each experience consists of a tuple of state, action, reward, next state. The training dataset, then, is $D = (s, a, r, s')_{i=1}^n$. Given D , one constructs an estimate of v^π in two ways: (1) by running TD(0) repeatedly over the experiences in D , until the resulting estimator of the value function, \hat{v}_{TD} , converges; or (2) by using the data in D to construct a maximum likelihood estimate of p and R , assuming that these models are correct, and then using Dynamic Programming to compute an estimate, \hat{v}_{DP} , of v^π . In class, we discussed that both of these approaches lead to the same value function estimate: $\hat{v}_{\text{TD}} = \hat{v}_{\text{DP}}$. In this question, you will show empirically that this claim is correct. For simplicity, we assume that the agent is following a deterministic policy—the agent always takes the same action in each state. Under this assumption, we will represent each experience as a tuple of state, reward, next state: (s, r, s') . Suppose the agent ran some policy π and constructed a training dataset, D , which contains the following transitions/experiences:

- $(s_2, 2, s_1)$
- $(s_1, -1, s_2)$
- $(s_2, 3, s_\infty)$
- $(s_2, 2, s_1)$

- $(s_1, -1, s_2)$
- $(s_1, 5, s_\infty)$
- $(s_2, 2, s_1)$
- $(s_1, -1, s_2)$
- $(s_2, 3, s_\infty)$
- $(s_1, -1, s_2)$
- $(s_2, 3, s_\infty)$

(Question 3a. 4 Points) Use the samples above to approximate p and R , and then show a diagram depicting the maximum likelihood MDP estimate that you constructed based on them. When presenting your MDP estimate, states should be depicted as nodes and edges between states should indicate both the transition probability between those states, as well as the reward associated with the transition.

$$\begin{aligned}
 p(s_2, 2, s_1) &= \frac{\text{number of times } p(s_2, 2, s_1) \text{ seen}}{\text{number of samples with } s_2 \text{ as initial state}} \\
 &= \frac{3}{6} \\
 &= 0.5
 \end{aligned}$$

$$\begin{aligned}
 p(s_1, -1, s_2) &= \frac{\text{number of times } p(s_1, -1, s_2) \text{ seen}}{\text{number of samples with } s_1 \text{ as initial state}} \\
 &= \frac{4}{5} \\
 &= 0.8
 \end{aligned}$$

$$\begin{aligned}
 p(s_2, 3, s_\infty) &= \frac{\text{number of times } p(s_2, 3, s_\infty) \text{ seen}}{\text{number of samples with } s_2 \text{ as initial state}} \\
 &= \frac{3}{6} \\
 &= 0.5
 \end{aligned}$$

$$\begin{aligned}
 p(s_1, 5, s_\infty) &= \frac{\text{number of times } p(s_1, 5, s_\infty) \text{ seen}}{\text{number of samples with } s_1 \text{ as initial state}} \\
 &= \frac{1}{5} \\
 &= 0.2
 \end{aligned}$$

$$R(s_1, a, s_2) = -1$$

$$R(s_2, a, s_1) = 2$$

$$R(s_2, a, s_\infty) = 3$$

$$R(s_1, a, s_\infty) = 5$$

(As policy is deterministic and everytime in the episode we see that when it transitions from one state to another no matter how many times, what the action is the reward is the same only. Hence the reward function $R(s, a, s')$ will be the same as 'r' in (s, r, s')).

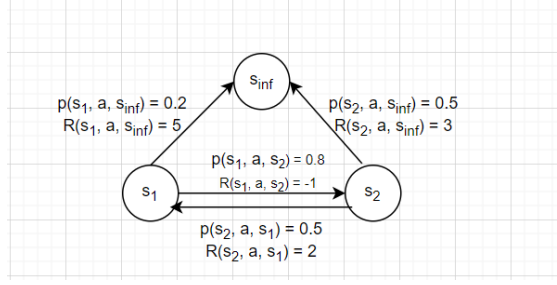


Figure 1: MDP Diagram

(Question 3b. 8 Points) Use Dynamic Programming to determine the value of states s_1 and s_2 , according to the estimated MDP. Report your findings.

By Bellman Equation, we can write, $v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s, a, s') \left(R(s, a, s') + \gamma v^\pi(s') \right)$

Note that we have already proved this in Q2 of HW2.

For a deterministic policy, we can rewrite the above as: $v^\pi(s) = \sum_{s' \in \mathcal{S}} p(s, a, s') \left(R(s, a, s') + \gamma v^\pi(s') \right)$

Considering $\gamma = 1$ and we know that $v^\pi(s_\infty) = 0$,

$$\begin{aligned}
 v^\pi(s_1) &= \sum_{s' \in \mathcal{S}} p(s_1, a, s') \left(R(s_1, a, s') + v^\pi(s') \right) \\
 &= p(s_1, a, s_2) \left(R(s_1, a, s_2) + v^\pi(s_2) \right) + p(s_1, a, s_\infty) \left(R(s_1, a, s_\infty) + v^\pi(s_\infty) \right) \\
 &= 0.8 * (-1 + v^\pi(s_2)) + 0.2 * (5 + 0) \\
 &= 0.8 * v^\pi(s_2) + 0.2
 \end{aligned} \tag{3b.1}$$

$$\begin{aligned}
 v^\pi(s_2) &= \sum_{s' \in \mathcal{S}} p(s_2, a, s') \left(R(s_2, a, s') + v^\pi(s') \right) \\
 &= p(s_2, a, s_1) \left(R(s_2, a, s_1) + v^\pi(s_1) \right) + p(s_2, a, s_\infty) \left(R(s_2, a, s_\infty) + v^\pi(s_\infty) \right) \\
 &= 0.5 * (2 + v^\pi(s_1)) + 0.5(3 + 0) \\
 &= 0.5 * v^\pi(s_1) + 2.5
 \end{aligned} \tag{3b.2}$$

Solving equations 3b.1 and 3b.2 gives:

$$\begin{aligned}
 v^\pi(s_1) &= 3.66666666667 \\
 v^\pi(s_2) &= 4.33333333333
 \end{aligned}$$

(Question 3c. 10 Points) Now, implement TD(0) and apply its update rule over the samples/experiences in D to construct a TD-based estimator of the value function. In particular, you should use $\alpha = 0.005$, set $\gamma = 1$, initialize the value estimate of all states to 0, and run TD(0) 10,000 times over all experiences in D . That is, you will apply the TD(0) update rule over the first experience in D ; then over the second experience in D ; and so on. Do this until you have processed all 11 experiences in D . This corresponds

to running TD(0) once over the dataset. You should run TD(0) a total of 10,000 times over the dataset D . Report the final TD estimates of $v(s_1)$ and $v(s_2)$. Do they seem to be converging to the value estimates computed via Dynamic Programming over the maximum likelihood MDP? After executing 10,000 runs of TD(0) over D , do the estimates constructed by TD(0) and DP match exactly? If not, why do you think that is the case, and what would be necessary to make them coincide (as dictated by the theory)?

```

1 alpha = 0.005
2 gamma = 1
3 N = 10000
4
5 experiences = [(('s1',2,'s2'), ('s1',3,'s1'), ('s1',2,'s2'), ('s1',3,'s1'), ('s1',3,'s1'), ('s1',3,'s1'), ('s1',3,'s1'), ('s1',3,'s1'), ('s1',3,'s1'), ('s1',3,'s1'))]
6
7 v_s1 = 0
8 v_s2 = 0
9
10 for i in range(N):
11     for start_state, reward, end_state in experiences:
12         if start_state == 's1':
13             v_s1 = gamma + alpha*(reward + gamma*v_s2 if end_state == 's2' else 0) - v_s1
14         elif start_state == 's2':
15             v_s2 = gamma + alpha*(reward + gamma*v_s1 if end_state == 's1' else 0) - v_s2
16
17     print("v(s1): ", v_s1)
18     print("v(s2): ", v_s2)
19
20 if __name__ == '__main__':
21     v(s1) = 3.670751236399101
22     v(s2) = 4.3300654794667945

```

Figure 2: Value functions using TD(0) for 10000 iterations

Using Dynamic Programming:

$$v^{\pi}(s_1) = 3.666666666667$$

$$v^{\pi}(s_2) = 4.333333333333$$

Using TD(0):

$$v^{\pi}(s_1) = 3.670751236399101$$

$$v^{\pi}(s_2) = 4.3300654794667945$$

The estimates using TD(0) do seem to be converging closely to the values obtained through Dynamic Programming over the maximum likelihood MDP.

However, we can see that the values don't exactly match.

This could be due to: alpha, which controls converging rate. Different values of alpha (non-zero alphas) affect the result differently and can induce some approximation error. Order of updates can also affect the calculation as computation is done sequentially, meaning the update order of experiences can slightly influence the intermediate value estimates. Number of iterations could also be a potential reason for eg instead of 10000 iterations, if we run it for 10100 iterations, we might get the exact results.

The necessary changes to make them coincide will be to decrease alpha, instead of using a fixed alpha we can use a decreasing alpha (alpha decreases as number of iterations increase) to avoid step size/approximation error and increase the number of iterations so that it converges slowly and more accurately.

Part Two: Programming (60 Points Total)

In this question, you will implement the First-Visit and Every-Visit Monte Carlo algorithms to evaluate policies for the Cat-vs-Monsters domain, and the Monte Carlo with ϵ -soft policies algorithm to find near-optimal policies for the Cat-vs-Monsters domain. **Notice that you may not use existing RL code for this problem—you must implement the learning algorithm and the environment entirely on your own and from scratch.** The complete definition of the Cat-vs-Monsters domain can be found in Homework 3.






	State 2	State 3		State 5
State 6	State 7	State 8		State 10
State 11	Forbidden Furniture	Forbidden Furniture	Forbidden Furniture	State 12
State 13	State 14	Forbidden Furniture	State 15	State 16
State 17		State 19	State 20	

Figure 3: The Cat-vs-Monsters domain.

Questions (Programming)

General instructions:

- You should initialize the value estimate of all states to zero.
- Whenever displaying values (e.g., the value of a state), use 4 decimal places; e.g., $v(s) = 9.4312$.
- Whenever showing the value function and policy learned by your algorithm, present them in a format resembling that depicted in Figure 4.
- The “standard” implementation of Monte Carlo with ϵ -policies updates the policy as follows:

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a = a^* \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise,} \end{cases} \quad (1)$$

where $a^* = \arg \max_{a \in \mathcal{A}} q(s, a)$. In the more general case, though, when more than one action may be optimal with respect to $q(s, a)$, the equation for action probabilities is given by:

$$\pi(s, a) = \begin{cases} \frac{1-\epsilon}{|\mathcal{A}^*|} + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a \in \mathcal{A}^* \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise,} \end{cases} \quad (2)$$

where $\mathcal{A}^* = \arg \max_{a \in \mathcal{A}} q(s, a)$. **You should implement this latter form.**

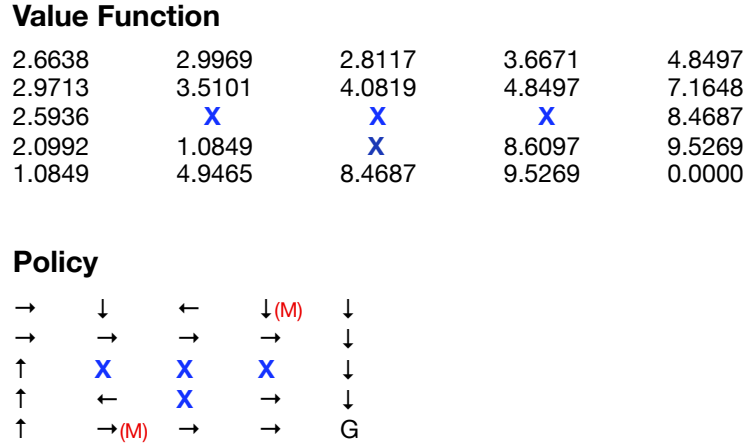


Figure 4: Optimal value function and optimal policy for the Cat-vs-Monsters domain.

1. **(20 Points)** First, implement First-Visit Monte Carlo and use it to construct an estimate, \hat{v} , of the value function of the optimal policy, π^* , for the Cat-vs-Monsters domain. **Remember that both the optimal value function and optimal policy for the Cat-vs-Monsters domain were identified in the previous homework by using the Value Iteration algorithm. For reference, both of them are shown in Figure 4.** When sampling trajectories, set d_0 to be a uniform distribution over \mathcal{S} to ensure that you are collecting returns from all states. Do not let the agent be initialized in a Forbidden Furniture state. Notice that using this alternative definition of d_0 (instead of the original one) is important because we want to generate trajectories from all states of the MDP, not just states that are visited under the optimal deterministic policy shown in Figure 4.

(Question 1a. 8 Points) Report the value function estimate computed by First-Visit Monte Carlo. Run your algorithm until you get a good estimate, \hat{v} , of v^{π^*} ; for instance, an estimate \hat{v} such that the Max-Norm between \hat{v} and v^{π^*} is at most 0.1. Report, also, how many iterations (trajectories) were necessary for your algorithm to identify such an accurate estimate of v^{π^*} .

```
Estimated Value Function using First-Visit Monte Carlo:
[[2.64696838 2.98428491 2.75836578 3.56853442 4.85895907]
 [2.95802445 3.48637388 4.07076583 4.80205315 7.13297904]
 [2.58369027 0. 0. 0. 8.46087393]
 [2.05684464 1.17865945 0. 8.6272001 9.5266583 ]
 [1.1129969 4.95736015 8.48280033 9.54740927 0. ]]

Total trajectories needed for convergence (max-norm < 0.1): 15313
```

Figure 5: Value Function and number of iterations using first visit MC

(Question 1b. 8 Points) Repeat the experiment above, but now using Every-Visit Monte Carlo. Report the value function estimate constructed by this algorithm and report how many iterations (trajectories) were necessary for it to identify an accurate estimate of v^{π^*} .

```
Estimated Value Function using Every-Visit Monte Carlo:
[[2.56451236 2.97033026 2.90205953 3.75829544 4.81889039]
 [3.01940378 3.51283498 4.09143634 4.87048023 7.15409821]
 [2.63731905 0. 0. 0. 8.46279127]
 [2.11932044 1.1074043 0. 8.60347848 9.52270901]
 [1.03631565 4.88847717 8.50112632 9.53591615 0. ]]

Total trajectories needed for convergence (max-norm < 0.1): 14852
```

Figure 6: Value Function and number of iterations using every visit MC

(Question 1c. 4 Points) How would you compare the empirical sample complexity of First-Visit Monte Carlo and Every-Visit Monte Carlo, in terms of the number of trajectories needed to estimate the value function?

First-Visit MC took more trajectories to converge than Every-Visit MC as seen in the figures above. On running the code multiple times, on an average it was observed that first visit MC took more trajectories to converge in general.

In Every-Visit MC, the value of a state is updated every time it is visited during an episode. This leads to more updates per state within a single trajectory, meaning each trajectory provides more information to update the value function. Because Every-Visit MC performs multiple updates per state in a single episode, it can make faster progress towards the optimal value function with fewer episodes/trajectories compared to First-Visit MC. First-Visit MC, on the other hand, requires more trajectories to converge because it only updates each state the first time it's visited in an episode. Therefore, more episodes are needed to gather enough information to converge to the optimal value function.

Both methods produce similar value functions, with differences in individual entries being relatively small. The closeness of the values indicates that both methods are effective in estimating the value function, but Every-Visit MC achieved this with slightly fewer trajectories.

- (40 Points) Implement the Monte Carlo with ϵ -policies algorithm to (1) construct \hat{q} , an estimate of the optimal **action-value function**, q^{π^*} ; and (2) construct an estimate, $\hat{\pi}$, of the optimal policy, π^* . Initialize \hat{q} with zeros for all states and actions, and initialize $\hat{\pi}$ as a uniform random distribution over actions, for all states. As before, when sampling trajectories, set d_0 to be a uniform distribution over \mathcal{S} to ensure that you are collecting returns from all states. Do not let the agent be initialized in a Forbidden Furniture state.

(Question 2a. 15 Points) Report the **value function** estimate, \hat{v} , computed by your algorithm when using three different values of ϵ (e.g., 0.2, 0.1, 0.05). Remember that the value function can be computed given the two quantities that you are estimating: \hat{q} and $\hat{\pi}$. You should run your algorithm for 10,000 iterations; each iteration corresponds to the process of sampling a trajectory, updating the estimate \hat{q} , and updating the policy.

```
*****Results for epsilon = 0.2:*****
Value Function (v_hat):
1.2936  1.4464  0.9989  1.5667  3.0586
1.3828  1.8299  2.4413  3.2016  6.0869
1.1211  0.0000  0.0000  0.0000  7.9218
0.4770  -0.5557  0.0000  8.2039  9.2172
-0.9805  2.4569  6.5681  9.0979  0.0000

*****Results for epsilon = 0.1:*****
Value Function (v_hat):
1.1668  1.2696  1.3414  2.4791  3.9574
1.8050  2.3131  3.0547  3.9573  6.6517
1.0715  0.0000  0.0000  0.0000  8.2200
0.8541  -0.5122  0.0000  8.4175  9.3818
-0.5131  3.8636  7.8719  9.3295  0.0000

*****Results for epsilon = 0.05:*****
Value Function (v_hat):
1.8136  2.2193  2.0334  2.6690  4.0401
2.1914  2.6471  3.2445  4.3439  6.8364
1.8681  0.0000  0.0000  0.0000  8.3394
1.3661  0.3226  0.0000  8.5016  9.4511
0.3132  4.1426  8.1278  9.4319  0.0000
```

Figure 7: Value Function for different ϵ

(Question 2b. 8 Points) Construct learning curves for the experiment above. In particular, for each value of ϵ , construct a graph where the x axis shows the number of iterations, and where the y axis

shows the mean squared error between your current estimate of the value function, \hat{v} , and the true optimal value function, v^{π^*} . The mean square error between two value functions, v_1 and v_2 , can be computed as $\frac{1}{|S|} \sum_s (v_1(s) - v_2(s))^2$. You should plot the mean square error every 250 iterations.

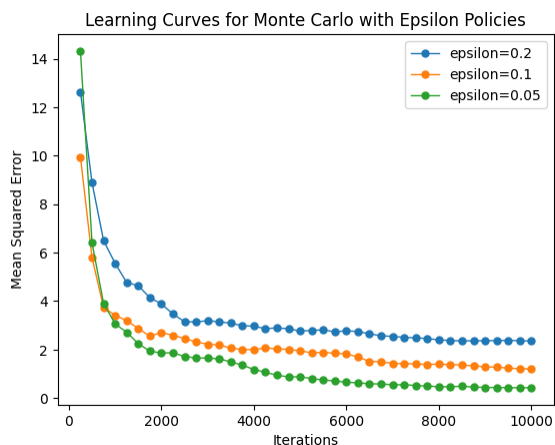


Figure 8: Learning Curves for different ϵ . MSE v/s iterations

(Question 2c. 12 Points) Repeat the experiments proposed in questions (2a) and (2b) but now using a *decay schedule* for ϵ . In particular, you should initialize $\epsilon = 1$ and decay it down to 0.05, as the number of iterations goes from 0 to 10,000. You could try, for example, to decay ϵ by 0.05 after every 500 iterations. As in the previous questions, you should report the value function estimate, \hat{v} , computed by your algorithm, and a learning curve showing mean squared error as a function of the number of iterations.

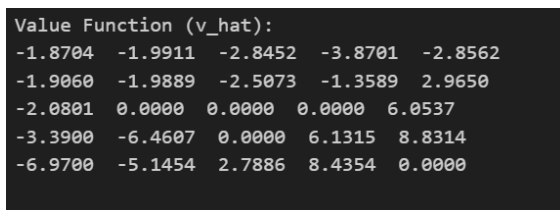


Figure 9: Value Function with decay schedule for ϵ

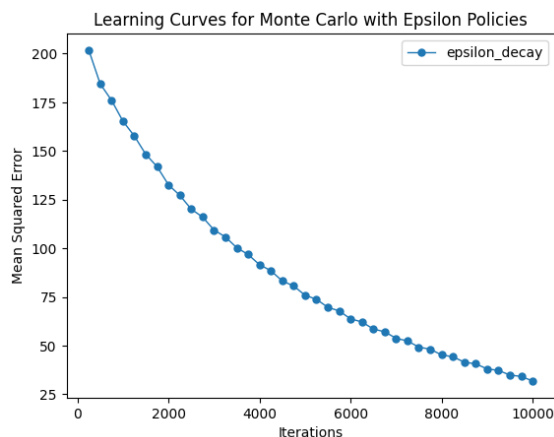


Figure 10: MSE v/s iterations curve with decay schedule for ϵ

(Question 2d. 5 Points) Which of the variants of the Monte Carlo with ϵ -policies algorithm, evaluated above, worked better in terms of identifying an accurate estimate of v^{π^*} ? Was it a variant that used a fixed value of ϵ (and, if so, what was the value of ϵ ?), or was it the variant that used a decay schedule for ϵ ? Discuss and interpret your findings.

From the above value functions and learning graphs, the best or the closest results to the true value function is observed when $\epsilon = 0.05$. We see that with different epsilons and with decay schedule too the graphs are decreasing i.e. MSE decreases as the number of iterations increase. However, the best result is seen when $\epsilon = 0.05$ i.e. it gives the least MSE, converging the most when compared to all the other variants. Even the value function for it was the closest to the true value function when compared to rest of the variants. This intuitively also makes sense as fixed small epsilon leads to faster and more precise convergence since the policy exploration is limited, enabling the algorithm to focus on exploitation of known rewards. When $\epsilon = 0.1$ we see worse result than when its 0.05 but better than what we see when $\epsilon = 0.2$ which is expected as we increase ϵ , we are increasing the exploration limit indicating there may be chances that the MSE would be high in comparison. With decay schedule, we see that the it didnt converge as quickly and after 10,000 iterations, MSE is still quite high when compared to fixed ϵ values even the calculated value function for it is not quite close to the true value function indicating it converges quite slowly than the fixed epsilons.