CMPSCI 687 Homework 4 - Fall 2022

Due **November 15, 2022**, 11:55pm Eastern Time

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 LaTeX. 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.

Before starting this homework, please review this course's policies on plagiarism by reading Section 10 of the syllabus.

Part One: Written (35 Points Total)

1. **(6 points)** Consider an MDP, M, where $\gamma = 0.9$. Let $\mathcal{S} = \{s_1, s_2\}$ and assume that the agent is following some policy, π. The agent executed π three times and obtained the following trajectories, where (for simplicity) each trajectory is represented as a sequence of states and corresponding rewards:

   - **Trajectory 1:** $s_1, 8, s_1, 1.5, s_2, -1, s_1, 5, s_2, 4, s_2, 3, s_1, 0, s_1, 0, s_2, 0$.
   - **Trajectory 2:** $s_2, 4, s_1, 6, s_2, -3, s_1, 9, s_2, 3$.
   - **Trajectory 3:** $s_2, 4, s_1, 9, s_2, 14$.

   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.

   Solution 1.

   Second Visit Monte Carlo

   Trajectory 1: $s_1, 8, s_1, 1.5, s_2, -1, s_1, 5, s_2, 4, s_2, 3, s_1, 0, s_1, 0, s_2, 0$

   $$\begin{aligned} s_1 &= 1.5 + \gamma(-1) + \gamma^2(5) + \gamma^3(4) + \gamma^4(3) + \gamma^5(0) + \gamma^6(0) + \gamma^7(0) \\ &= 1.5 + 0.9(-1) + 0.9^2(5) + 0.9^3(4) + 0.9^4(3) + 0.9^5(0) + 0.9^6(0) + 0.9^7(0) \\ &= 9.5343 \\ s_2 &= (4) + \gamma(3) + \gamma^2(0) + \gamma^3(0) + \gamma^4(0) \\ &= (4) + 0.9(3) + 0.9^2(0) + 0.9^3(0) + 0.9^4(0) \\ &= 6.7 \end{aligned}$$

   Trajectory 2: $s_2, 4, s_1, 6, s_2, -3, s_1, 9, s_2, 3$

   $$\begin{aligned} s_1 &= 9 + \gamma 3 \\ &= 11.7 \\ s_2 &= -3 + \gamma 9 + \gamma^2 3 \\ &= 7.53 \end{aligned}$$

Trajectory 3: $s_2, 4, s_1, 9, s_2, 14$.

$$s_2 = 14$$

According to Second visit Monte Carlo, the state value function are as follows

$$\begin{aligned} v(s_1) &= (9.5343 + 11.7)/2 = 10.61715 \\ v(s_2) &= (6.7 + 7.53 + 14)/3 = 9.41 \end{aligned}$$

Every visit Monte Carlo

Trajectory 1: $s_1, 8, s_1, 1.5, s_2, -1, s_1, 5, s_2, 4, s_2, 3, s_1, 0, s_1, 0, s_2, 0$

$$\begin{aligned} s_1^1 &= 8 + \gamma 1.5 + \gamma^2(-1) + \gamma^3(5) + \gamma^4(4) + \gamma^5(3) + \gamma^6(0) + \gamma^7(0) + \gamma^8(0) \\ &= 8 + 0.91.5 + 0.9^2(-1) + 0.9^3(5) + 0.9^4(4) + 0.9^5(3) + 0.9^6(0) + 0.9^7(0) + 0.9^8(0) \\ &= 16.58 \\ s_1^2 &= 1.5 + \gamma(-1) + \gamma^2(5) + \gamma^3(4) + \gamma^4(3) + \gamma^5(0) + \gamma^6(0) + \gamma^7(0) \\ &= 1.5 + 0.9(-1) + 0.9^2(5) + 0.9^3(4) + 0.9^4(3) + 0.9^5(0) + 0.9^6(0) + 0.9^7(0) \\ &= 9.5343 \\ s_1^3 &= (5) + \gamma(4) + \gamma^2(3) + \gamma^3(0) + \gamma^4(0) + \gamma^5(0) \\ &= (5) + 0.9(4) + 0.9^2(3) + 0.9^3(0) + 0.9^4(0) + 0.9^5(0) \\ &= 11.03 \\ s_1^4 &= (0) + 0.9(0) + 0.9^2(0) \\ &= 0 \\ s_1^5 &= (0) + 0.9(0) \\ &= 0 \\ s_2^1 &= -1 + \gamma(5) + \gamma^2(4) + \gamma^3(3) + \gamma^4(0) + \gamma^5(0) + \gamma^6(0) \\ &= -1 + 0.9(5) + 0.9^2(4) + 0.9^3(3) + 0.9^4(0) + 0.9^5(0) + 0.9^6(0) \\ &= 8.927 \\ s_2^2 &= (4) + \gamma(3) + \gamma^2(0) + \gamma^3(0) + \gamma^4(0) \\ &= (4) + 0.9(3) + 0.9^2(0) + 0.9^3(0) + 0.9^4(0) \\ &= 6.7 \\ s_2^3 &= 3 + \gamma 0 + \gamma^2 0 + \gamma^3 0 \\ &= 3 + 0.9 * 0 + 0.9^2 0 + 0.9^3 0 \\ &= 3 \\ s_2^4 &= 0 \end{aligned}$$

Trajectory 2: $s_2, 4, s_1, 6, s_2, -3, s_1, 9, s_2, 3$

$$\begin{aligned}
s_1^1 &= 6 + \gamma - 3 + \gamma^2 9 + \gamma^3 3 \\
&= 6 + 0.9 * -3 + 0.9^2 * 9 + 0.9^3 * 3 \\
&= 12.777 \\
s_1^2 &= 9 + \gamma 3 \\
&= 11.7 \\
s_2^1 &= 4 + \gamma 6 + \gamma^2 - 3 + \gamma^3 9 + \gamma^4 3 \\
&= 4 + 0.9 * 6 + 0.9^2 * -3 + 0.9^3 * 9 + 0.9^4 * 3 \\
&= 15.4993 \\
s_2^2 &= -3 + \gamma 9 + \gamma^2 3 \\
&= 7.53 \\
s_2^3 &= 3
\end{aligned}$$

Trajectory 3: $s_2, 4, s_1, 9, s_2, 14$.

$$\begin{aligned}
s_1 &= 9 + \gamma 14 \\
&= 9 + 0.9 * 14 \\
&= 21.6 \\
s_2^1 &= 4 + \gamma 9 + \gamma^2 14 \\
&= 4 + 0.99 + 0.9^2 14 \\
&= 23.44 \\
s_2^2 &= 14
\end{aligned}$$

According to Every visit Monte Carlo, the state value function are as follows

$$\begin{aligned}
v(s_1) &= (16.58 + 9.5343 + 11.03 + 0 + 0 + 12.777 + 11.7 + 21.6)/8 = 10.40 \\
v(s_2) &= (8.927 + 6.7 + 3 + 0 + 15.4993 + 7.53 + 3 + 23.44 + 14)/9 = 9.12
\end{aligned}$$

2. **(14 Points)** When introducing First-Visit Monte Carlo, we showed that it is possible to construct an unbiased estimator of $v^\pi(s)$ by computing the discounted return starting from the first occurrence of s within a given trajectory generated by π. Similarly, it is possible to construct an unbiased estimator of $v^\pi(s)$ by computing the discounted return starting from the *second* occurrence of s within a given trajectory constructed by running π. Consider, now, a new Monte Carlo algorithm that constructs an estimator of $v^\pi(s)$ by computing the discounted return starting from the *second-to-last* occurrence of s within a given trajectory constructed by running π. Consider the MDP shown in Figure 1. In this MDP, the agent may transition from s back to s, say, N times, and then transition to s_∞. The estimator we are proposing would take averages of the returns starting from the second-to-last (penultimate) visit to s. Prove that the *Second-to-Last* Monte Carlo estimator is unbiased (just like the *Second-Visit* Monte Carlo estimator is unbiased) or argue formally, using the MDP introduced below to support your argument, that Second-to-Last Monte Carlo is not unbiased; that is, that the expected valor of the estimates it constructs is not equal to $v^\pi(s)$.

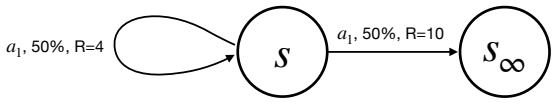


Figure 1: An MDP.

Solution 2. The Expected value of state s can be calculated using the bellman eqn as follows,

$$
\begin{aligned}
v^{\pi}(s) &= \sum_{a \in a_1} \pi(s,a) \sum_{s'} p(s,a,s') \left(R(s,a) + \gamma v^{\pi}(s') \right) \\
&= 1 * (0.5 * (4 + \gamma * v^{\pi}(s)) + 0.5 * (4 + \gamma * v^{\pi}(s_{\infty}))) \\
v^{\pi}(s) &= 1 * (0.5 * (4 + \gamma * v^{\pi}(s) + 0.5 * (10 + \gamma * 0))
\end{aligned}
$$

Solving for $v^{\pi}(s)$ gives,

$$v^{\pi}(s) = \frac{7}{1 - 0.5 * \gamma}$$

And using the "second to last visit" Monte Carlo we can estimate $v^{\pi}(s)$ as follows,

$$v^{\pi}(s) = 4 + \gamma * (10)$$

Now the expected and the estimated values of $v^{\pi}(s)$ will be equal only in the following case,

$$\frac{7}{1 - 0.5 * \gamma} = 4 + \gamma * (10)$$

$$5\gamma^2 - 8\gamma + 3 = 0$$

$$\gamma = 1, \gamma = \frac{3}{5}$$

We see that the "Second to last visit" Monte Carlo would give the value of the state equal to the expected value only in certain cases, and not for all value of $\gamma < 1$. It is unbiased only for certain γ. Therefore it is a **biased estimator**.

3. **(15 Points)** The First-Visit Monte Carlo algorithm uses an estimator, $\hat{v}(s)$, of the value of some state s, constructed by taking the average of all returns G_i^s, where G_i^s is the return observed from the first occurrence of state s in the i-th trajectory. In class we showed that First-Visit Monte Carlo is unbiased. However, when comparing Monte Carlo policy evaluation methods and TD methods, we argued that Monte Carlo methods have higher variance. Suppose, for instance, that we collect n trajectories and compute the First-Visit Monte Carlo estimator, $\hat{v}_n^1(s)$, of $v^{\pi}(s)$. If we repeat this process and collect a new set of n trajectories, and compute a new estimator, $\hat{v}_n^2(s)$, based on the new trajectories, it might be the case that $\hat{v}_n^2(s)$ is very different from $\hat{v}_n^1(s)$, even though both are unbiased. In other words, Monte Carlo estimates may differ significantly every time we run the algorithm, which means that the First-Visit Monte Carlo estimator, $\hat{v}_n$ (i.e., the First-Visit Monte Carlo estimator computed based on n trajectories) has high variance. Write down an equation for the variance of this estimator. How does the variance depend on n? Given that this estimator is unbiased, what does the expression for its variance tell you about the "quality" of First-Visit Monte Carlo estimators as n grows?

Solution 3.

$$\hat{v}_n(s) = \frac{1}{N} \sum_{n=1}^{N} G_n$$

Variance of the above equation can be calculated as follows,

$$
\begin{aligned}
Var(\hat{v}_n(s)) &= Var(\frac{1}{N} \sum_{n=1}^{N} G_n) \\
&= \frac{1}{N^2} Var(\sum_{n=1}^{N} G_n) \\
&= \frac{1}{N^2} \big(\sum_{n=1}^{N} Var(G_n) + 2 \sum_{m=1}^{N} \sum_{k=1}^{N} Cov(G_m, G_k) \big)
\end{aligned}
$$

Variance is inversely proportional to square of N. And we know that,

$$\begin{aligned} Cov(X,Y) &= E[XY] - E[X]E[Y] \\ Var(X) &= E[X^2] - E[X]^2 \end{aligned}$$

Therefore we can say that both $Var(G_n)$ is bounded since G_i are bounded since the returns are bounded and assuming $\gamma < 1$ and $Cov(G_m, G_k)$ is zero since G_i are independent.

So if $N \to \infty$,

$$Var(\hat{v}_n(s)) = lim_{N\to\infty} \frac{(BoundedValue)}{N^2} \to 0$$

As N grows, the variance of the estimator converges to zero, and that is a quality which we would like to have in an estimator. Zero variance means, that the estimator majorly gives a non-fluctuating return. The returns will not vary much.

Part Two: Programming (65 Points Total)

In this question, you will implement the First-Visit and Every-Visit Monte Carlo algorithms to evaluate policies for the 687-Gridworld, and the Monte Carlo with ϵ-soft policies algorithm to find near-optimal policies for the 687-GridWorld 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 687-GridWorld domain can be found in Homework 3.

2. 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 2.
- The "standard" implementation of Monte Carlo with ϵ-policies updates the policy as follows:

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

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:

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

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

1. **(25 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 687-Gridworld. **Remember that both the optimal value function and optimal policy for the 687-Gridworld domain were identified in the previous homework by using the Value Iteration algorithm. For reference, both of them are shown in Figure 2.** 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

Value Function

4.0187	4.5548	5.1575	5.8336	6.4553
4.3716	5.0324	5.8013	6.6473	7.3907
3.8672	4.3900	0.0000	7.5769	8.4637
3.4182	3.8319	0.0000	8.5738	9.6946
2.9977	2.9309	6.0733	9.6946	0.0000

Policy

→	→	→	↓	↓
→	→	→	↓	↓
↑	↑		↓	↓
↑	↑		↓	↓
↑	↑	→	→	G

Figure 2: Optimal value function and optimal policy for the 687-GridWorld domain.

not let the agent be initialized in an Obstacle 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 2.

(Question 1a. 10 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^*}.

Solution 1a. Taking multiple trajectories ensures that the agent visits each state. The selection of the start state is uniform random, and therefore over multiple trajectories the agent would start from almost all the states.

There are 2 sources of randomness in First Visit Monte Carlo

- Initial state
- Transition function

That is why we run the algorithm multiple times to calculate the number of trajectories/iterations/episodes it takes for the value function to converge.

Fig 3 shows 3 different runs of First Visit Monte Carlo. Value 3a is obtained after 1010 iterations, value 3b is obtained after 2748 iterations and value 3c is obtained after 3990 iterations. Max norm for all three values is < 0.1.

On an average over 20 runs for First Visit Monte Carlo, it took **8300** iterations.

(Question 1b. 10 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^*}.

Solution 1b. Fig 4 shows the value function for Every visit Monte Carlo for 3 different runs of the algorithm.

Fig 4a shows the value function which took 4158 iterations to converge, Fig 4b shows value function which took 4078 iterations and Fig 4c took 8760 iterations to converge.

On an average over 20 runs of the Every Visit Monte Carlo, it took **6696** trajectories to converge.

(Question 1c. 5 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?

Solution 1c. Looking at the average number of iterations to converge, we can see that Every visit Monte Carlo takes less number of iterations every time to converge to the optimal value function. This confirms the theory that every visit Monte Carlo requires lesser number of trajectories to converge than first visit. Therefore sample complexity of Every visit is lesser.

Both First Visit and Every Visit Monte Carlo are consistent in limit when the amount of data converges to infinity, but since the convergence criteria that we are using ($MaxNorm < 0.1$) is a very strict criteria in which even if

```
[[4.10839522 4.59191401 5.20983173 5.85938212 6.37830684]
 [4.41144588 4.97746846 5.7428519  6.60923978 7.32048942]
 [3.88737623 4.34438561 0.         7.52500021 8.45749812]
 [3.46566796 3.74078526 0.         8.54063143 9.71674051]
 [3.0035511  3.02903756 6.1408     9.67654551 0.        ]]
```

(a) First run: 1010 iterations

```
[[4.02557745 4.57896311 5.15595381 5.85467521 6.44306198]
 [4.38989131 5.04642673 5.83913614 6.69755137 7.39647147]
 [3.91431494 4.39373018 0.         7.60477597 8.49243548]
 [3.42428748 3.81688888 0.         8.59204828 9.68554762]
 [3.00080188 2.84103529 5.97781032 9.70303391 0.        ]]
```

(b) Second run: 2748 iterations

```
[[4.09190103 4.58853882 5.17697666 5.82962656 6.44319732]
 [4.3319186  5.00565118 5.79089897 6.63213843 7.3566019 ]
 [3.82588102 4.37540089 0.         7.57594366 8.45592973]
 [3.41560126 3.82946483 0.         8.58743454 9.70804921]
 [2.98189326 2.84213107 6.09950332 9.70197705 0.        ]]
```

(c) Third run: 3990 iterations

Figure 3: Value function for different runs of First visit Monte Carlo

```
[[4.05213523 4.56273136 5.16653749 5.8479979  6.49820684]
 [4.29900493 5.00701902 5.76321625 6.63196168 7.40220115]
 [3.80927384 4.4139474  0.         7.58672462 8.49053861]
 [3.38651002 3.83945787 0.         8.57739144 9.70397401]
 [3.0161413  2.89668518 6.08178535 9.69536984 0.        ]]
```

(a) First run: 4158 iterations

```
[[3.98270729 4.56011706 5.20144455 5.84963519 6.53385251]
 [4.37407081 5.0227797  5.79341013 6.66931951 7.41799958]
 [3.88178643 4.4013315  0.         7.60560556 8.4713624 ]
 [3.41384609 3.84556291 0.         8.58530542 9.70158755]
 [3.0471017  3.03024519 6.15048073 9.69568275 0.        ]]
```

(b) Second run: 4078 iterations

```
[[4.0196278  4.55373154 5.14137489 5.82437987 6.45826421]
 [4.36010694 5.01655374 5.79490957 6.63097263 7.3960345 ]
 [3.86799111 4.35819831 0.         7.56969133 8.44607045]
 [3.4040383  3.83294614 0.         8.57055598 9.68925604]
 [3.0421922  2.98442599 6.15600059 9.69223504 0.        ]]
```

(c) Third run: 8760 iterations

Figure 4: Value function for different runs of Every visit Monte Carlo

```
[[2.9221339  3.50633348 4.16664742 4.92983318 5.54667928]
 [3.41818265 4.07113144 4.90321955 5.87517324 6.57862583]
 [2.89091438 3.32641857 0.         6.91850545 7.8878964 ]
 [2.49605472 2.430839   0.         7.98005982 9.34428087]
 [2.01803409 1.20830475 3.43739282 8.48930415 0.        ]]
```

(a) $\epsilon = 0.2$

```
[[3.63380893 4.11573343 4.79158388 5.44776255 6.00181018]
 [3.92673161 4.56717747 5.35246759 6.24352669 6.98705376]
 [3.4422367  3.91212404 0.         7.25079422 8.16558917]
 [2.95020066 3.35057489 0.         8.28075696 9.51896344]
 [2.51350857 2.10741705 5.05867576 9.18043811 0.        ]]
```

(b) $\epsilon = 0.1$

```
[[3.77493641 4.32448017 4.96117608 5.63481384 6.31734918]
 [4.15498737 4.80018627 5.56653185 6.43799839 7.21999311]
 [3.64527596 4.13292145 0.         7.39212994 8.3414079 ]
 [3.16601345 3.58052031 0.         8.39904199 9.61981917]
 [2.78246436 2.66729271 5.43618037 9.46997107 0.        ]]
```

(c) $\epsilon = 0.05$

Figure 5: Value function for different epsilon values

during a certain run if the value reaches below 0.01, the algorithm is said to have converged. And due to the presence of high level of stochasticity, identifying which algorithm is more complex is not possible. Therefore we ran both the algorithms multiple times to get an average number of trajectories required in both the cases, and empirically we see that Every Visit Monte Carlo took almost 1500 trajectories less than First Visit Monte Carlo.

2. **(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 an Obstacle 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.

   Solution 2a. I have used the bellman equation to compute the value function using $\hat{q}$ and $\hat{\pi}$.

$$v^{\pi}(s) = \sum_{a} \pi(s,a)\, q^{\pi}(s,a)$$

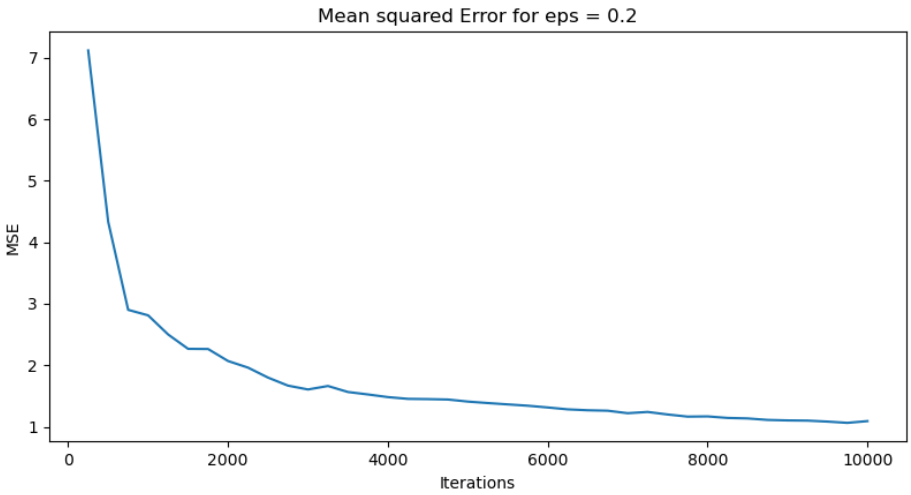
   The value function obtained for different values of epsilon can be seen in Fig 5.

   Table 1 shows the Max norm between the $\hat{v}$ and v^{π^*}

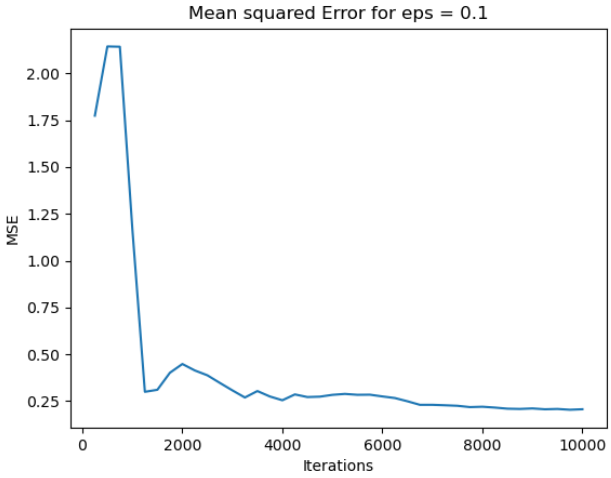
   (Question 2b. 8 Points) Construct learning curves for the experiment above. In particular, for each value of ϵ, construct a graph where the x axis shows the number of iterations, and where the y axis shows the mean squared error between your current estimate of the value function, $\hat{v}$, and the true optimal value function, v^{π^*}. The mean square error between two value functions, v_1 and v_2, can be computed as $\frac{1}{|\mathcal{S}|}\sum_s (v_1(s) - v_2(s))^2$. You should plot the mean square error every 250 iterations.

   Solution 2b. The corresponding learning curves for 3 different values of ϵ are shown in Fig 6. Table 1 shows the MSE between the $\hat{v}$ and v^{π^*}

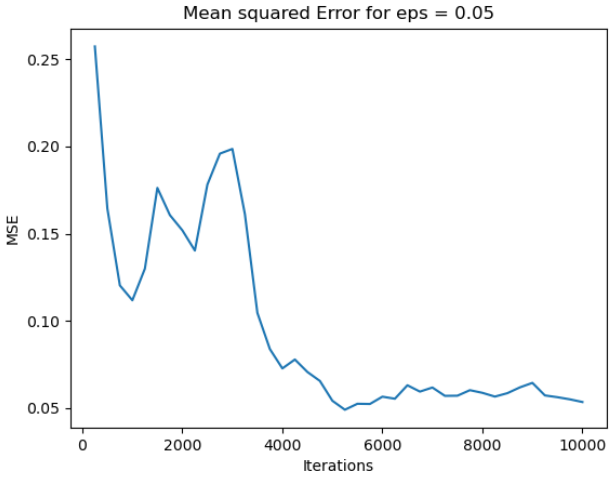
   (Question 2c. 12 Points) Repeat the experiments proposed in questions *(2a)* and *(2b)* but now using a *decay schedule* for ϵ. In particular, you should initialize $\epsilon = 1$ and decay it down to 0.05, as the number of iterations goes from 0 to $10,000$. You could try, for example, to decay ϵ by 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.



(a) $\epsilon = 0.2$



(b) $\epsilon = 0.1$

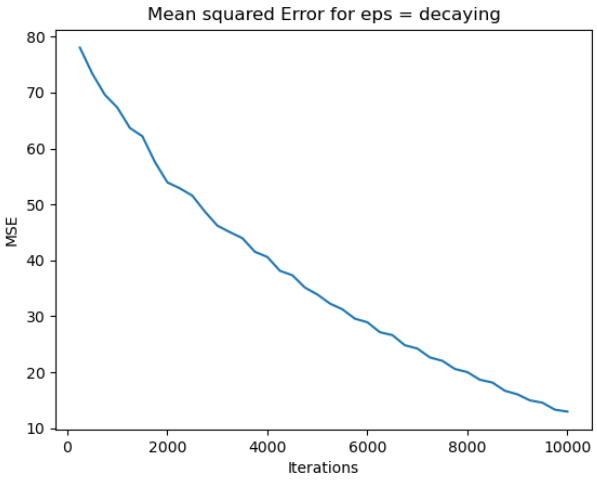


(c) $\epsilon = 0.05$

Figure 6: Learning Curves for different epsilon values

```
[[ 1.37818321  1.77473661  2.28250019  2.84004227  3.52981151]
 [ 1.05516152  1.67445813  2.4763729   3.43012267  4.86735824]
 [ 0.70160417  0.8194761   0.          4.07991521  6.51487286]
 [ 0.15489329 -0.26454583  0.          5.52414335  9.1693901 ]
 [-0.74917829 -2.44780319 -4.57909682  8.38204475  0.        ]]
```

(a) Value function for ϵ decay



(b) Learning Curve for ϵ decay

Figure 7: Value function and Learning Curve for epsilon decay (1 to 0.05)

Solution 2c. The value function and the learning for epislon decay can be seen in the Fig 7

Table 1 shows the Max norm and MSE between the $\hat{v}$ and v^{π^*}

(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.

Solution 2d. From the values shown in Table 1 we can see that the epsilon with the lowest value $\epsilon = 0.05$ gives a value function closest to the v^{π^*}, i.e. the lowest Max-norm and the lowest mean squared error. Exploration in the beginning is good, since the policy is not optimal yet, but with every iteration the policy gets updated with the actions that have the highest q value. The agent should explore less towards the end. In higher epsilon values, the agent keeps exploring with a higher probability even when the policy has been updated with the best actions. Smaller values of epsilon make sure that the agent explores less, and this less level of exploration is sufficient in the start, and as the policy gets updated with the best actions, the agent still explores less as compared to higher epsilon case.

For epsilon decay case, since we are starting with a very high value of epsilon, and ending the iterations when epsilon reaches 0.05, the overall convergence to the optimal value function is poor and the Max norm and MSE remain high. It is worse than $\epsilon = 0.2$ case, since most of the epsilon values that are being used are higher than 0.2 and thus the agent explores more even after obtaining best actions for each state. Epsilon values are lower than 0.2 only in the last 2000 iterations, which is only 20% of the total iterations.

ϵ	0.2	0.1	0.05	decay
Max Norm	2.64	1.01	0.64	10.65
MSE	1.09	0.21	0.05	12.98

Table 1: Max Norm and MSE after 10k iterations