CS780: Deep Reinforcement Learning

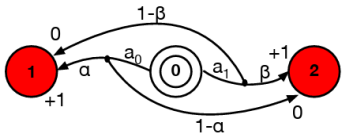
Assignment #1

Name: Rajarshi Dutta
Roll No.: 200762

All the experiments are done with **global seed** set to **123**

Solution to Problem 1: Multi-armed Bandits

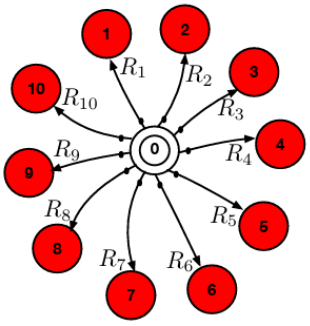
The given problem lies in implementing **2-armed Bernoulli Bandit** and **10-armed Gaussian** Bandit. The **Bernoulli Bandit** essentially has 3 states with 2 terminal states named as 1 and 2. The actual rewards are essential sampled from bernoulli distributions with parameters as α and β.



(a) Two-armed Bernoulli Bandit

$$R_0 \sim Bernoulli(\alpha) \tag{1}$$
$$R_1 \sim Bernoulli(\beta) \tag{2}$$

The Gaussian Distribution is a 10-armed bandit where for each arm k, the action value $q_(k)$ is sampled from a standard Gaussian distribution and reward for the corresponding arm is sample from a gaussian distribution which has mean $q_(k)$ and unit standard deviation:

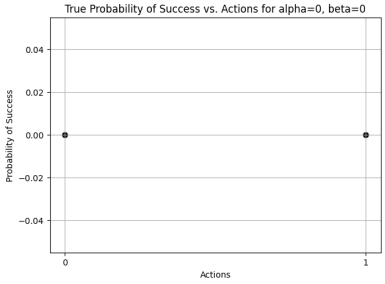


(b) Ten-armed Gaussian Bandit

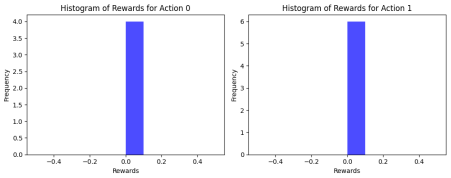
$$q^*(k) \sim \mathcal{N}(\mu = 0, \sigma^2 = 1) \tag{3}$$
$$R_k \sim \mathcal{N}(\mu = q^*(k), \sigma^2 = 1) \tag{4}$$

1. For the first task, the environment **Bernoulli Bandit** is created and simulated for 5 different combinations of α and β namely from ((0,0), (1,0), (0,1), (0.5,0.5), (1,1)) and the plots for each of settings are illustrated below highlighting the true rewards and the actual rewards obtained for the setting. For the first setting (α, β) are both 0 thus the actual rewards are opposite to the true rewards since the agent slips on taking

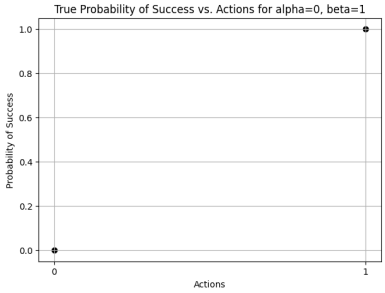
every action and for the case of (α, β) both 1, the actual rewards obtained are equal to true rewards. The case for (α, β) being both 0.5 gives a combinations of rewards of 0 and 1 since the chances of slipping and going to the correct state is **equally likely**



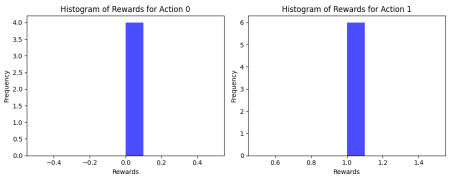
(a) Actions for $\alpha = 0$ and $\beta = 0$



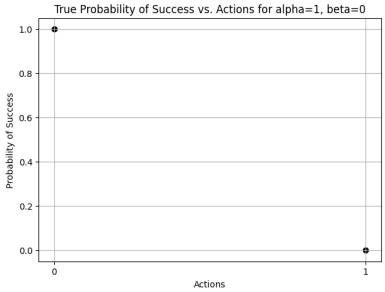
(b) Frequency distribution for rewards for $\alpha = 0$ and $\beta = 0$



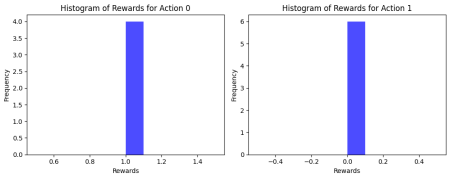
(a) Actions for $\alpha = 0$ and $\beta = 1$



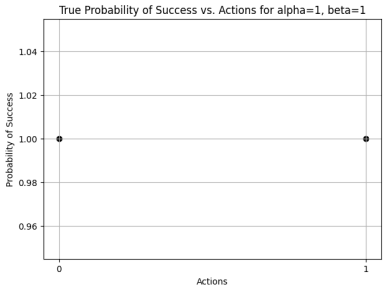
(b) Frequency distribution for rewards for $\alpha = 0$ and $\beta = 1$



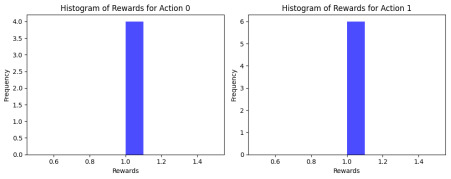
(a) Actions for $\alpha = 1$ and $\beta = 0$



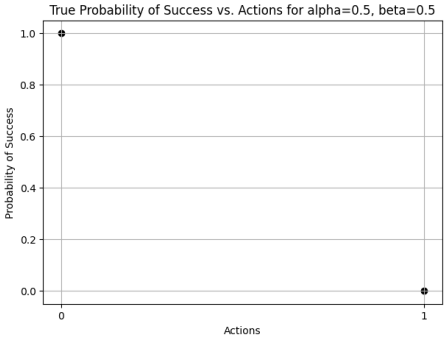
(b) Frequency distribution for rewards for $\alpha = 1$ and $\beta = 0$



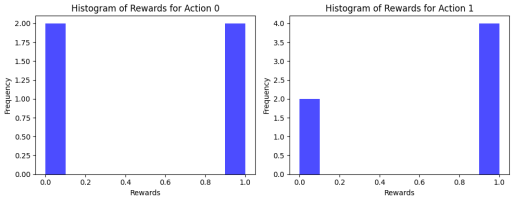
(a) Actions for $\alpha = 1$ and $\beta = 1$



(b) Frequency distribution for rewards for $\alpha = 1$ and $\beta = 1$

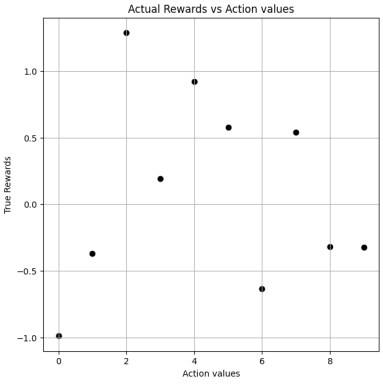


(a) Actions for $\alpha = 0.5$ and $\beta = 0.5$

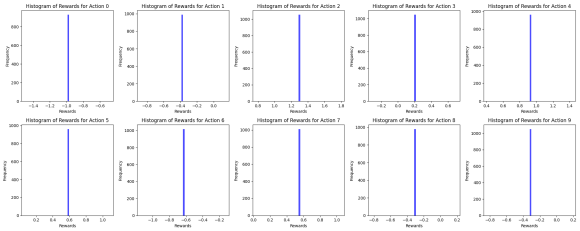


(b) Frequency distribution for rewards for $\alpha = 0.5$ and $\beta = 0.5$

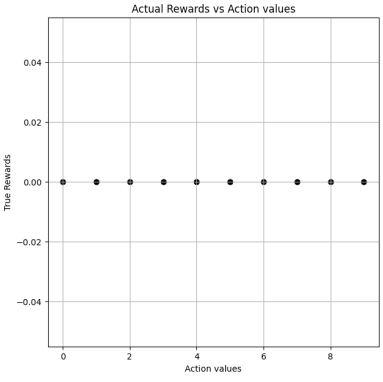
2. Similary, the **Gaussian Bandit** env is created and the plot for actions and distribution of true rewards are plotted for each action in the action space. From the plots, we see that for σ^2 set to 0, the frequency distribution of true rewards for all actions follows a normal distribution. The initial sampling of action values $q_{(k)}$ from a standard Gaussian distribution means that, on average, some arms will inherently be better (i.e., offer higher rewards) than others. The spread of the initial $q_{(k)}$ values influences the difficulty of the bandit problem—wider spreads might make it easier to distinguish between the best and worst arms, whereas narrower spreads make this differentiation harderThe global seed is set to **123**. The diagrams are illustrated below:



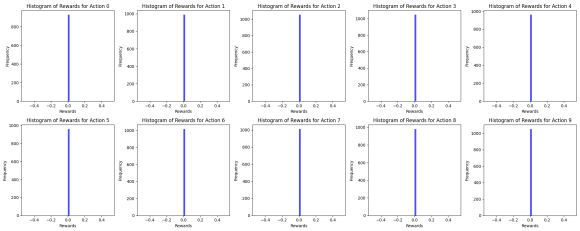
(a) Actions for $\sigma_1^2 = 1$ and $\sigma_2^2 = 0$



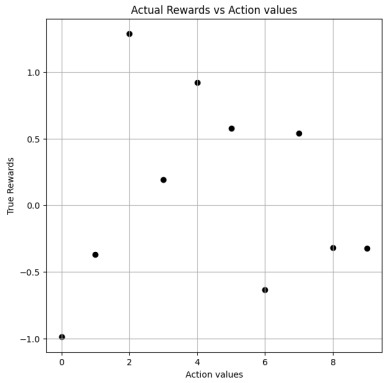
(b) Frequency distribution for rewards for $\sigma_1^2 = 1$ and $\sigma_2^2 = 0$



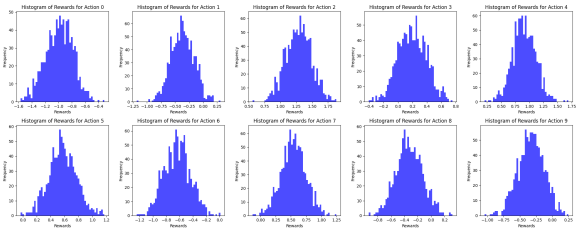
(a) Actions for $\sigma_1^2 = 0$ and $\sigma_2^2 = 0$



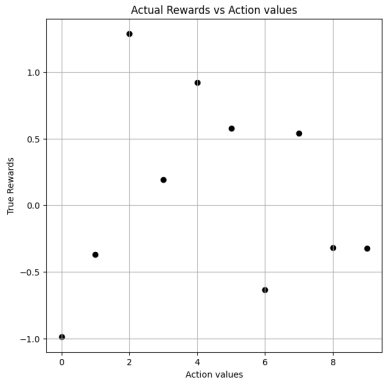
(b) Frequency distribution for rewards for $\sigma_1^2 = 0$ and $\sigma_2^2 = 0$



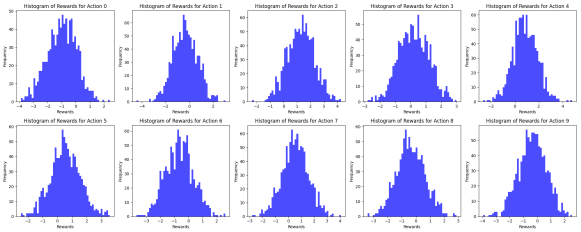
(a) Actions for $\sigma_1^2 = 1$ and $\sigma_2^2 = 0.3$



(b) Frequency distribution for rewards for $\sigma_1^2 = 1$ and $\sigma_2^2 = 0.3$



(a) Actions for $\sigma_1^2 = 1$ and $\sigma_2^2 = 1$



(b) Frequency distribution for rewards for $\sigma_1^2 = 1$ and $\sigma_2^2 = 1$

3. The next section discusses about the different algorithms for simulating the agent in the environment which mostly balances exploration and exploitation.

   - **Pure Exploitation**: This strategy purely exploits on the **Q_values** of the environment. This might lead to suboptimal outcomes if the initial estimates are inaccurate or there is a drastic change in the dynamics of the environment. The **Q** array stores the current estimates of each action. It is updated after every action based on the received reward. The array **Q_est** array essentially stores the averaged **Q_values** for each action over all episodes which helps to understand how the actions evolve.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	0	0	0.0000	0.0000
2	0	0	0.0000	0.0000
3	0	0	0.0000	0.0000
4	0	1	0.2500	0.0000
5	0	1	0.4000	0.0000
6	0	0	0.3333	0.0000
7	0	0	0.2857	0.0000
8	0	0	0.2500	0.0000
9	0	0	0.2222	0.0000
10	0	1	0.3000	0.0000

Table 1: Action, reward, and estimated Q-values per episode for Bernoulli Bandit

   - **Pure Exploration**: This strategy focuses more on exploration of the environment where the agent essentially randomly samples from the action space on every step in the episode. The main motive of this strategy is to gather as much information as possible about all the available actions to identify the best action or to estimate the value of each action as accurately as possible. This is particularly

useful in scenarios where the cost of exploration is low, or when the goal is to map out the value landscape of actions rather than to immediately maximize rewards.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	0	1	1.0000	0.0000
2	1	0	1.0000	0.0000
3	0	0	0.5000	0.0000
4	0	0	0.3333	0.0000
5	1	0	0.3333	0.0000
6	1	0	0.3333	0.0000
7	1	0	0.3333	0.0000
8	1	0	0.3333	0.0000
9	0	0	0.2500	0.0000
10	1	0	0.2500	0.0000

Table 2: Action, reward, and estimated Q-values per episode with Bernoulli Bandit

- **Epsilon Greedy Exploration**: The agent balances between exploration and exploitation. For every action, it selects a random number from a uniform distribution, if the random number is less than exploration, the agent focuses on exploitation where it chooses the action corresponding to the best Q value otherwise it goes for exploration where it randomly samples from the action space.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	0	1	1.0000	0.0000
2	0	0	0.5000	0.0000
3	0	0	0.3333	0.0000
4	0	1	0.5000	0.0000
5	1	0	0.5000	0.0000
6	0	0	0.4000	0.0000
7	0	0	0.3333	0.0000
8	0	1	0.4286	0.0000
9	0	0	0.3750	0.0000
10	0	0	0.3333	0.0000

Table 3: Action, reward, and estimated Q-values per episode for Bernoulli Bandit

- **Decaying Epsilon Greedy Exploration**: The above scheme is same as the previous one but the value of epsilon is gradually decayed over the course of episodes so that the agent first manage to explore all possible actions in the environment . The agent is updating its estimates over a wide variety of possible experiences and towards the end the agent begins to exploit on the available choices. The given tables shows the rewards for linearly and exponentially varying ϵ from 1 to 0.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	1	0	0.0000	0.0000
2	1	0	0.0000	0.0000
3	0	0	0.0000	0.0000
4	0	1	0.5000	0.0000
5	1	0	0.5000	0.0000
6	0	0	0.3333	0.0000
7	0	0	0.2500	0.0000
8	0	1	0.4000	0.0000
9	0	0	0.3333	0.0000
10	0	0	0.2857	0.0000

Table 4: Action, reward, and estimated Q-values per episode for Bernoulli Bandit (Linear decaying of ϵ)

- **Softmax Exploration**: The agent calculates the softmax of probabilities while taking an action in every episode. We see that its algorithm provides sufficient exploration amidst exploitation. The

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	1	0	0.0000	0.0000
2	0	0	0.0000	0.0000
3	0	0	0.0000	0.0000
4	0	1	0.3333	0.0000
5	1	0	0.3333	0.0000
6	0	0	0.2500	0.0000
7	0	0	0.2000	0.0000
8	0	1	0.3333	0.0000
9	0	0	0.2857	0.0000
10	0	0	0.2500	0.0000

Table 5: Action, reward, and estimated Q-values per episode for Bernoulli Bandit (Exponential decaying of ϵ)

main problem with greedy exploration is when it explores it equally chooses among all actions. In this algorithm, the greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates. These are called softmax action selection rules. High temperatures cause the actions to be nearly equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. The given plot has an initial temperature of 100 which is decayed exponentially to 0.01 over 10 episodes.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	1	0	0.0000	0.0000
2	1	0	0.0000	0.0000
3	1	0	0.0000	0.0000
4	0	0	0.0000	0.0000
5	0	0	0.0000	0.0000
6	1	0	0.0000	0.0000
7	0	0	0.0000	0.0000
8	0	1	0.2500	0.0000
9	0	0	0.2000	0.0000
10	0	0	0.1667	0.0000

Table 6: Action, reward, and estimated Q-values per episode for Bernoulli Bandit (Exponentially varying temperature τ

- **Upper Confidence Bound**: This action selection uses uncertainty in the action-value estimates for balancing exploration and exploitation. Since there is inherent uncertainty in the accuracy of the action-value estimates when we use a sampled set of rewards thus UCB uses uncertainty in the estimates to drive exploration. The Upper Confidence Bound follows the principle of optimism in the face of uncertainty which implies that if we are uncertain about an action, we should optimistically assume that it is the correct action.

Episode	Action	Reward	Q-value (Action 0)	Q-value (Action 1)
1	0	0	0.0000	0.0000
2	1	0	0.0000	0.0000
3	0	0	0.0000	0.0000
4	1	0	0.0000	0.0000
5	0	1	0.3333	0.0000
6	0	0	0.2500	0.0000
7	0	0	0.2000	0.0000
8	0	0	0.1667	0.0000
9	0	0	0.1429	0.0000
10	0	1	0.2500	0.0000

Table 7: Action, reward, and estimated Q-values per episode with interspersed actions

4. The next plot shows the variation of average rewards vs episodes for all 6 different exploration strategies over 1000 episodes for 50 different instances of the **Bernoulli Bandit** environment. The first plot has the following hyperparameters

Hyperparameter	Value
Initial Value (ϵ)	1
Final Value (ϵ)	0.1
Initial Value (τ)	10
Final Value (τ)	0.1
c	0.8
Decay type	Exponential

Figure 10: Hyperparameters for different strategies

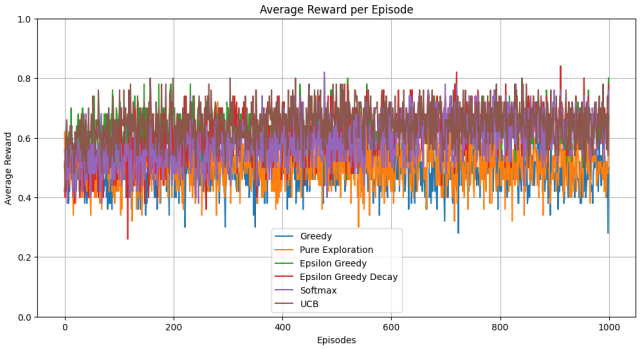


Figure 11: Average Rewards vs Episodes for 6 different Strategies (Bernoulli Bandit)

Hyperparameter	Value
Initial Value (ϵ)	1
Final Value (ϵ)	0.1
Initial Value (τ)	100
Final Value (τ)	0.1
c	0.5
Decay type	Exponential

Figure 12: Hyperparameters for different strategies

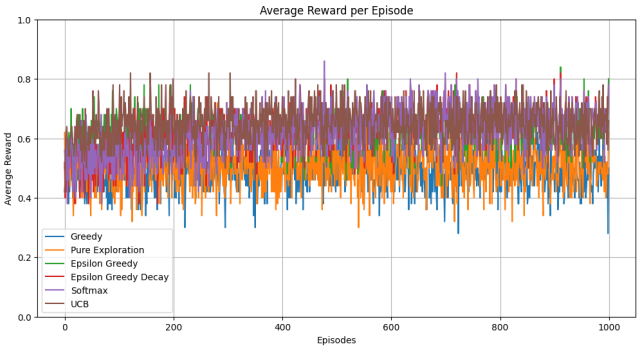
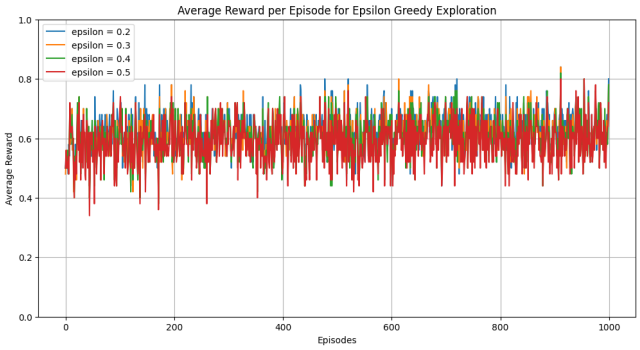


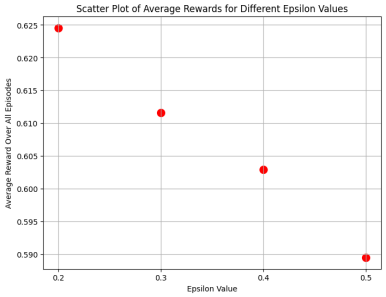
Figure 13: Average Rewards vs Episodes for 6 different Strategies (Bernoulli Bandit)

From the plots, we can see that the best performing algorithm in terms of the average accumulated rewards over all episodes is the **Upper Confidence Bound** followed by **Softmax**, **Epsilon Greedy** with the last being the **Pure Exploitation** approach. Thus the agent needs to maintain an optimal balance between exploration and exploitation to acheive more rewards in the environment.

The next plots show the variation of average rewards vs episodes for different values of ϵ in **Epsilon Greedy Exploration**. We find that the overall reward obtained for the case of $\epsilon = 0.2$ is the largest as compared to that for other values of ϵ.



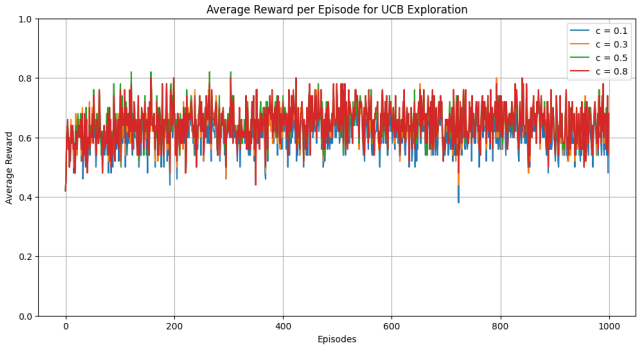
(a) Average Rewards vs Episodes for different ϵ values in Bernoulli Bandit



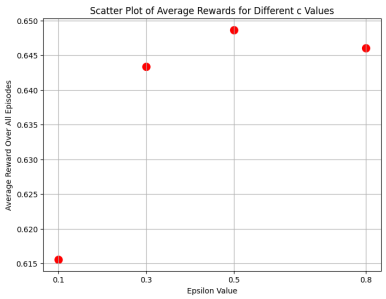
(b) Comparison for different ϵ values

The next plot shows the variation of average rewards vs episodes for different values of c in **UCB**. We find that the overall average reward obtained for the case of c=0.5 is the largest with that of c=0.1 being

the lowest.



(a) Average Rewards vs Episodes for different c values in Bernoulli Bandit



(b) Comparison for different c values

5. The next plot shows the variation of average rewards vs episodes for all 6 different exploration strategies over 1000 episodes for 50 different instances of the **Gaussian** environment. The first plot has the following hyperparameters

Hyperparameter	Value
Initial Value (ϵ)	1
Final Value (ϵ)	0.1
Initial Value (τ)	10
Final Value (τ)	0.1
c	0.8
Decay type	Exponential
variance	1

Figure 16: Hyperparameters for different strategies

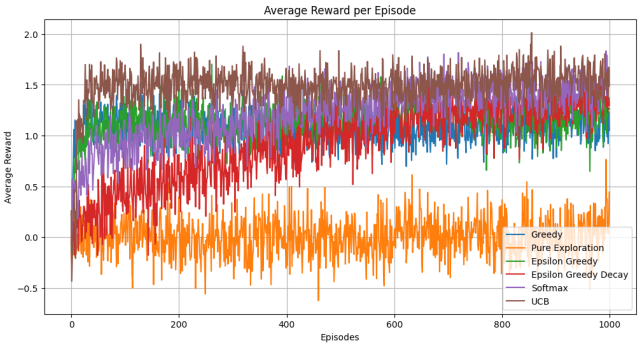


Figure 17: Average Rewards vs Episodes for 6 different Strategies (10 - armed Gaussian Bandit)

Hyperparameter	Value
Initial Value (ϵ)	1
Final Value (ϵ)	0.1
Initial Value (τ)	100
Final Value (τ)	0.1
c	0.3
Decay type	Exponential
variance	1

Figure 18: Hyperparameters for different strategies

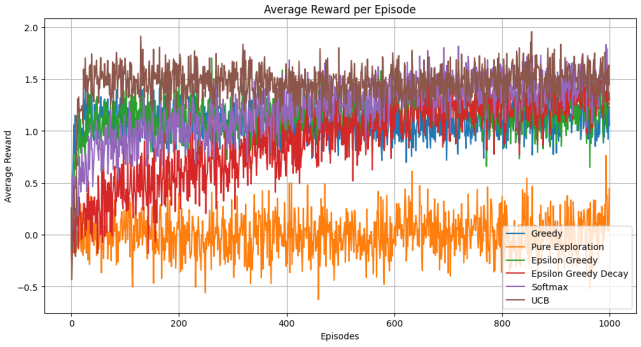
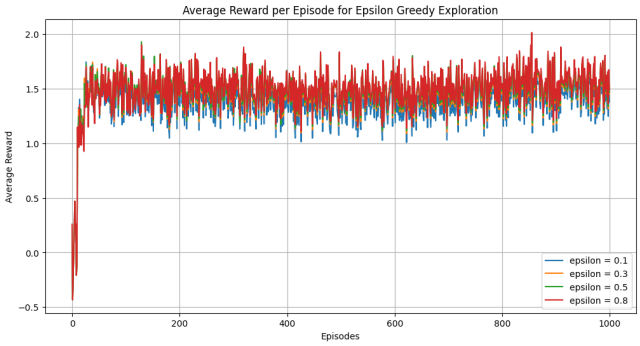


Figure 19: Average Rewards vs Episodes for 6 different Strategies (10 - armed Gaussian Bandit)

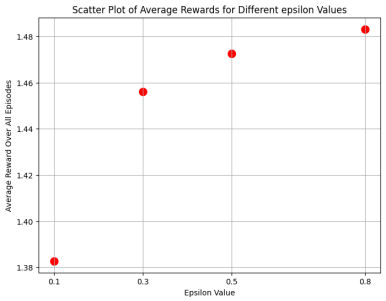
From the first plot, we see that the average reward for the case of **Upper Confidence Bound** is the largest. In the begining of the exploration process, the **epsilon greedy** algorithm generated more average reward than that of **Softmax** but falls below it after around 300 epsiodes. We also see that **Pure Exploitation** generated more average reward than **Pure Exploration** overall with the reward for **Pure Exploration** being the lowest. Average reward generated by the **Epsilon greedy decay** shows a steady increase from the initial episodes itself.

The next plots show the variation of average rewards vs episodes for different values of ϵ in **Epsilon Greedy Exploration**. We find that the overall reward obtained for the case of $\epsilon = 0.8$ is the largest as

compared to that for other values of ϵ, with smallest being that for $\epsilon = 0.1$

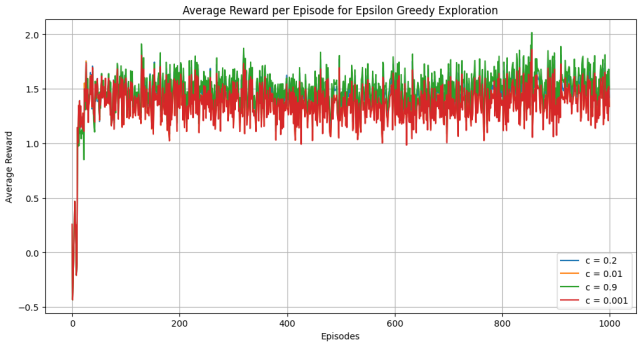


(a) Average Rewards vs Episodes for different ϵ values in 10 - armed Gaussian Bandit

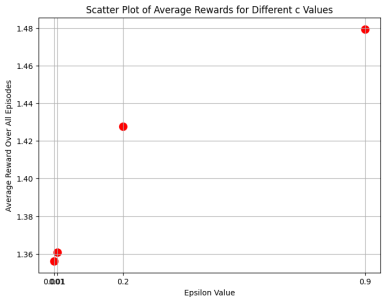


(b) Comparison for different ϵ values

The next plot shows the variation of average rewards vs episodes for different values of c in **UCB**. We find that the overall average reward obtained for the case of c=0.9 is the largest with that of c=0.01 being the lowest.

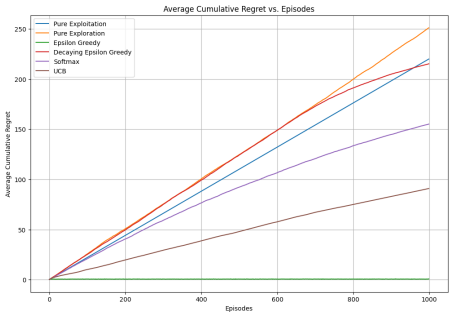


(a) Average Rewards vs Episodes for different c values in 10 - armed Gaussian Bandit

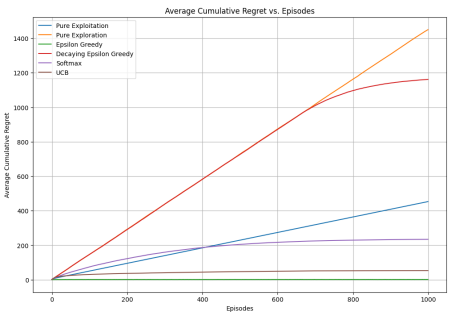


(b) Comparison for different c values

6. The set of plots shows the **Cumulated Regret** vs Episodes for the case of **Bernoulli Bandit** and **10 - armed Gaussian Bandit** setting over 1000 epsiodes and 50 instances of the environment. We find from the plots that the **Pure Exploration** strategy always has the maximum cumulated regret over episodes and the regret shows almost a linear trend whereas the **Epsilon Greedy** strategy always has the lowest regret. For the case of **Gaussian Bandit** the second best performing algorithm is **UCB** and the **Decaying Epsilon Greedy** seems to merge with **Pure Exploration** in the initial Episodes. In the case of **Bernoulli Bandit**, we see that **UCB** shows a slight increasing trend as compared to that of **Gaussian Bandit**



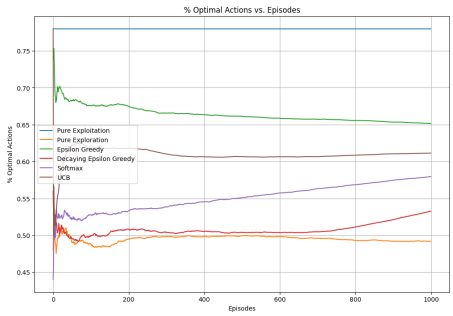
(a) Cumulative Regret vs Episodes for different strategies in Bernoulli Bandit



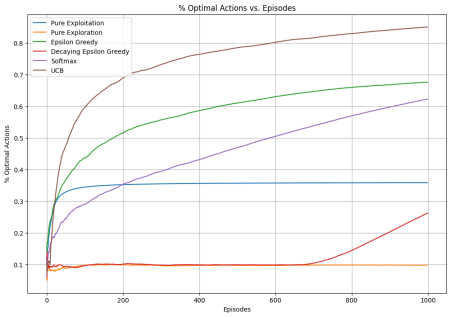
(b) Cumulative Regret vs Episodes for different strategies in Gaussian Bandit

7. The set of plots shows the **Cumulated Regret** vs Episodes for the case of **Bernoulli Bandit** setting over 1000 epsiodes and 50 instances of the environment. For the case of **Bernoulli Bandit**, we see that

Pure Exploitation has the maximum percentage of optimal actions with **Pure Exploration** being the lowest. **UCB** has somewhat greater than 60% of optimal actions. From the plot for **Gaussian Bandit**, we see that **UCB** has the maximum percentage of optimal actions, followed by **Epsilon Greedy** and **Pure Exploitation** becomes stagnant after isome initial episodes.



(a) % Optimal Actions vs Episodes for different strategies in Bernoulli Bandit



(b) % Optimal Actions vs Episodes for different strategies in Gaussian Bandit

Solution to Problem 2: MC Estimates and TD Learning

The given problem is about implementing **Monte Carlo prediction** and **Temporal difference learning** for solving a **Random Walk environment** consisting of 5 non-terminal states and 2 terminal states. The agent receives a reward of +1 upon reaching the terminal state 6. The environment is completely random and slipperiness is introduced with two subsequent actions namely **left** and **right**. The discount factor is $\gamma = 0.99$ and the policy is assigned to as **move left**. The entire implementation can be found inside the colab notebook.

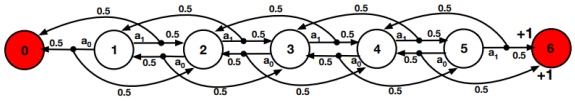
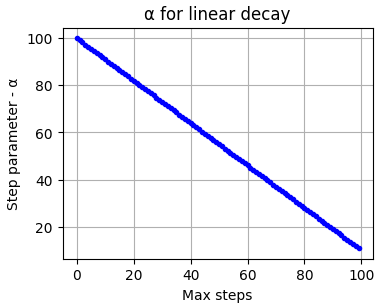


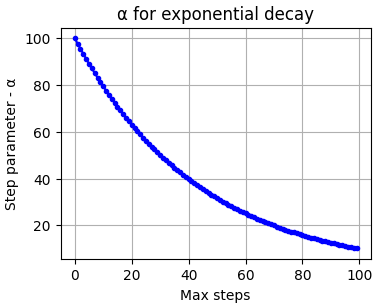
Figure 3: Random Walk Environment

1. The environment resets the agent to the mid point of the all the nodes every time a new episode is initiated. The optimal state-value functions obtained though is independent of the starting point of the agent. The **generateTrajectory()** function generates a new episode containing the experience tuple (state, actions and indicators whether epsiode has terminated or not) and MC and TD plots are obtained y varying the step parameter implemented in the function **decayAlpha()**

2. The plots for linear and exponential decay of the step parameter is shown below. The **initial value** is set to and **final value** is set to , **max steps** is set to **100**]



(a) Linear Decay



(b) Exponential Decay

3. The **MonteCarloPrediction()** implements the MC algorithm for calculating the optimal state-values for each state. This is an **offline** learning method since the values are averaged after each episode and required completion of the current episode. Both the variants are incorporated in the function namely

FVMC which denotes every first visit to a state is considered in average within an episode and **EVMC** which includes all the visits to a state into average within an episode. The function returns the value functions for all the states corresponding to the all episodes in a 2D list. The seed is set to **123**

Table 8: Optimal State values for EVMC with a seed = 123

Episodes	Initial Value	Final Value	S0	S1	S2	S3	S4	S5	S6
100	0.5	0.01	0.0	0.293	0.340	0.45	0.653	0.8221	0.0
200	0.3	0.02	0.0	0.104	0.276	0.503	0.632	0.8223	0.0
300	0.3	0.02	0.0	0.158	0.258	0.457	0.695	0.893	0.0
300	0.1	0.01	0.0	0.152	0.304	0.474	0.698	0.877	0.0
500	0.1	0.01	0.0	0.118	0.243	0.368	0.570	0.864	0.0

4. The **TemporalDifferencePrediction()** is an online learning method since it doesn't depend on episode completion. It works by bootstrapping from the current estimate of the value function. The environment is reset in the beginning of every episode and the reward is calculated by considering the value estimates of the next state. The agent learns by minimizing the td error. The seed is set to **123**

Table 9: Optimal State values for TD with a seed = 123

Episodes	Initial Value	Final Value	S0	S1	S2	S3	S4	S5	S6
100	0.5	0.01	0.0	0.120	0.287	0.505	0.684	0.862	0.0
200	0.3	0.02	0.0	0.107	0.249	0.502	0.682	0.842	0.0
300	0.3	0.02	0.0	0.124	0.273	0.459	0.643	0.839	0.0
300	0.1	0.01	0.0	0.108	0.254	0.434	0.637	0.831	0.0
500	0.1	0.01	0.0	0.133	0.269	0.422	0.603	0.823	0.0

5. The first set of plot depicts the **FVMC** algorithm with a constant value of α which is set to **0.5** and another with exponential decaying α for **500** episodes. The values of every state with successive episodes is plotted. While there is considerable fluctuation, it does not appear that the estimates are converging to the true values within the 500 episodes shown. This could suggest that the algorithm may need more episodes to converge, or it could indicate issues with the chosen parameters for learning (such as the learning rate).

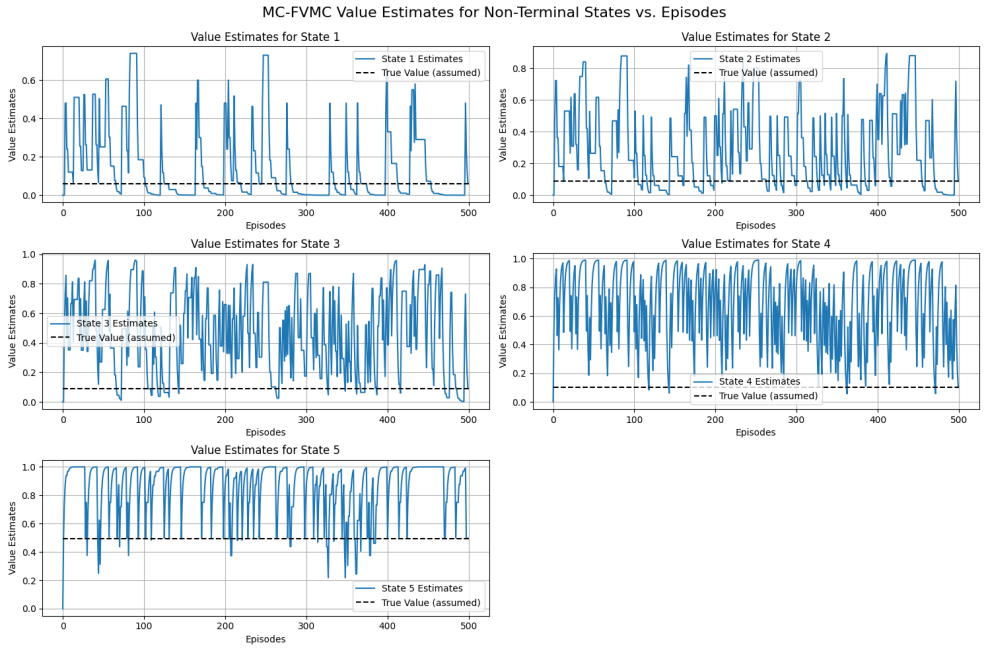


Figure 25: FVMC plots for constant $\alpha = 0.5$

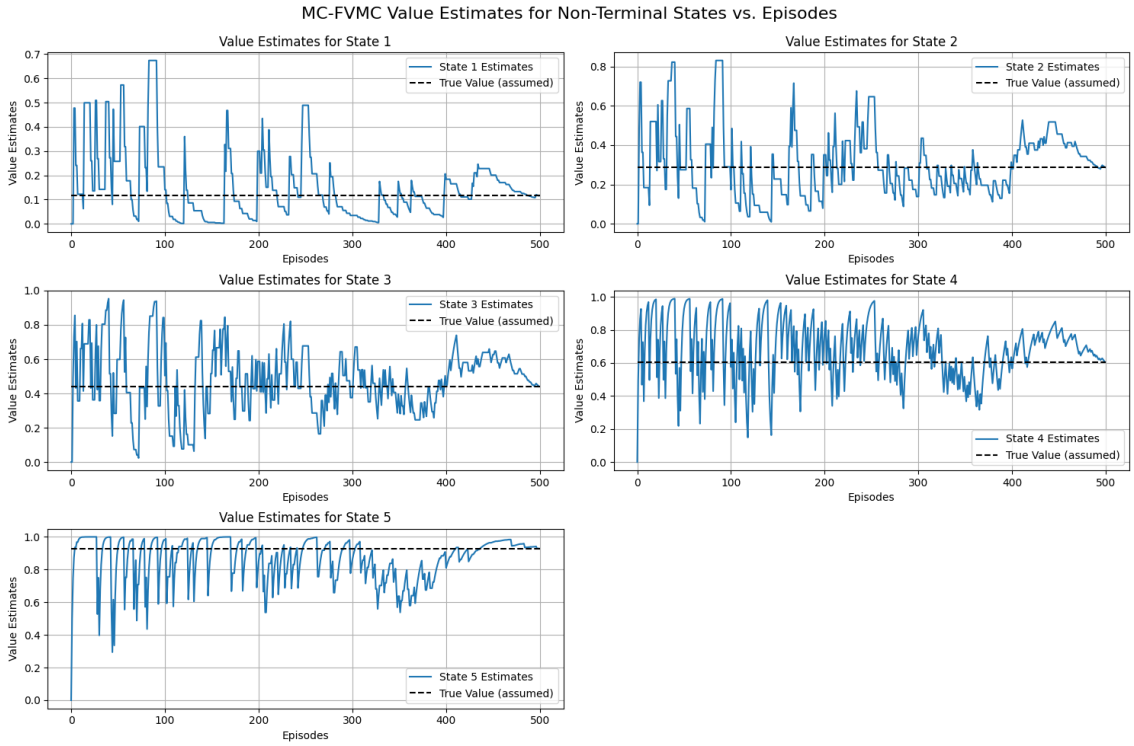


Figure 26: FVMC plots for decaying $\alpha = 0.5$ to 0.01

6. The first set of plot depicts the **EVMC** algorithm with a constant value of α which is set to **0.5** and another with exponential decaying α for **500** episodes. The values of every state with successive episodes is plotted. Almost all the states have a high variance suggesting larger number of episodes to converge.

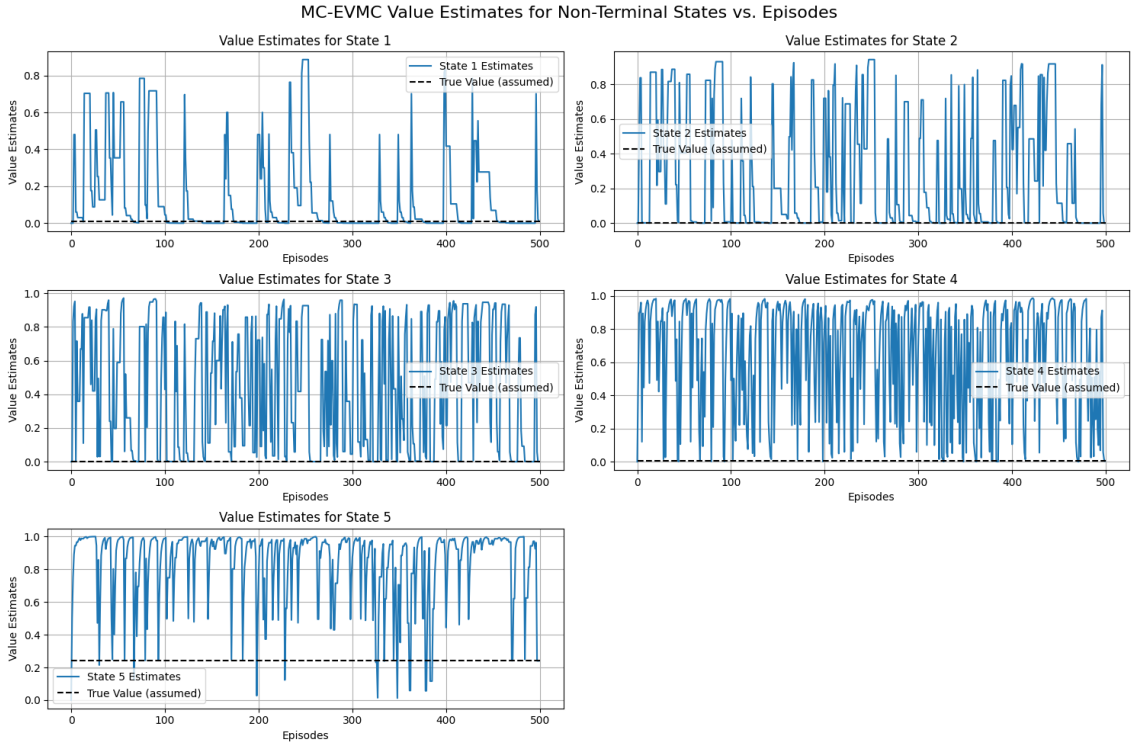


Figure 27: EVMC plots for constant $\alpha = 0.5$

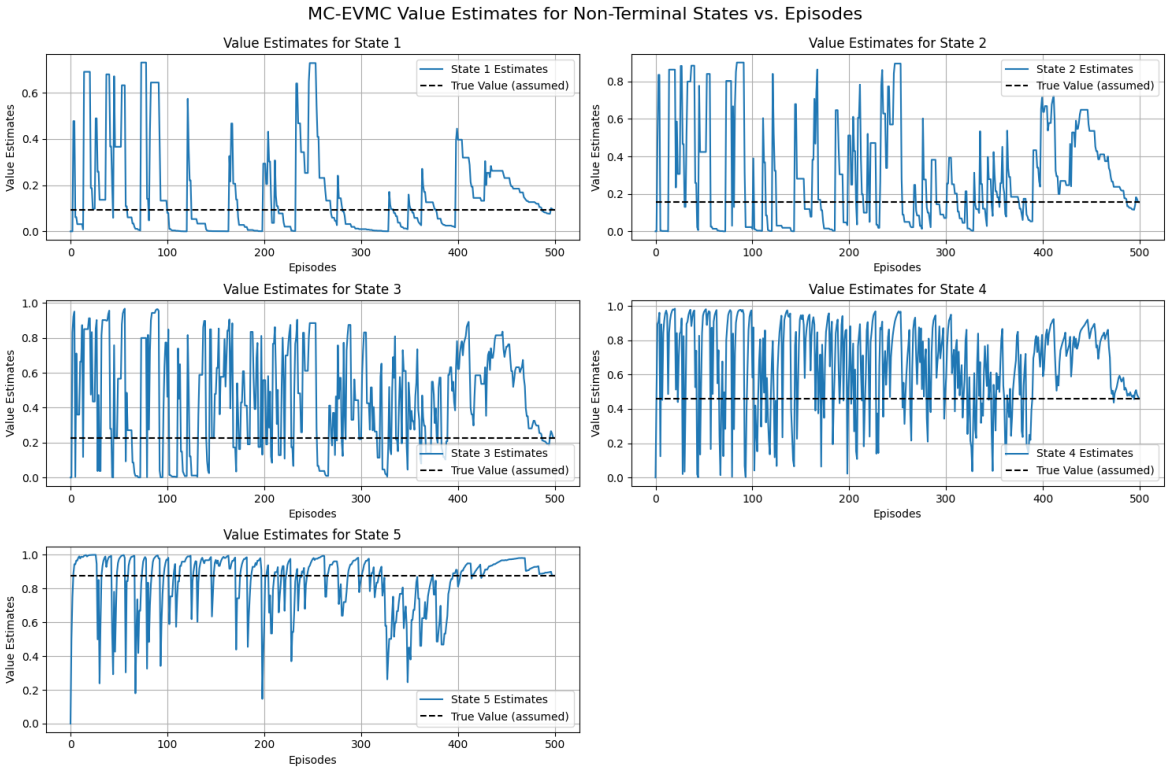


Figure 28: EVMC plots for decaying $\alpha = 0.5$ to 0.01

7. The next set of plots depicts the **TD** value estimates for each of the states over the course of **500** episodes. The value estimates for all **States** seem to converge properly for decaying α to the true estimate as compared to a bit of random fluctuations for the case of constant $\alpha = 0.5$ which makes sense since initially the agent tries exploring the sample space and as the value estimates gets closer to true values, it starts refining estimates with a greater precision, thus maintaining a balance between **exploration** and **exploitation**.

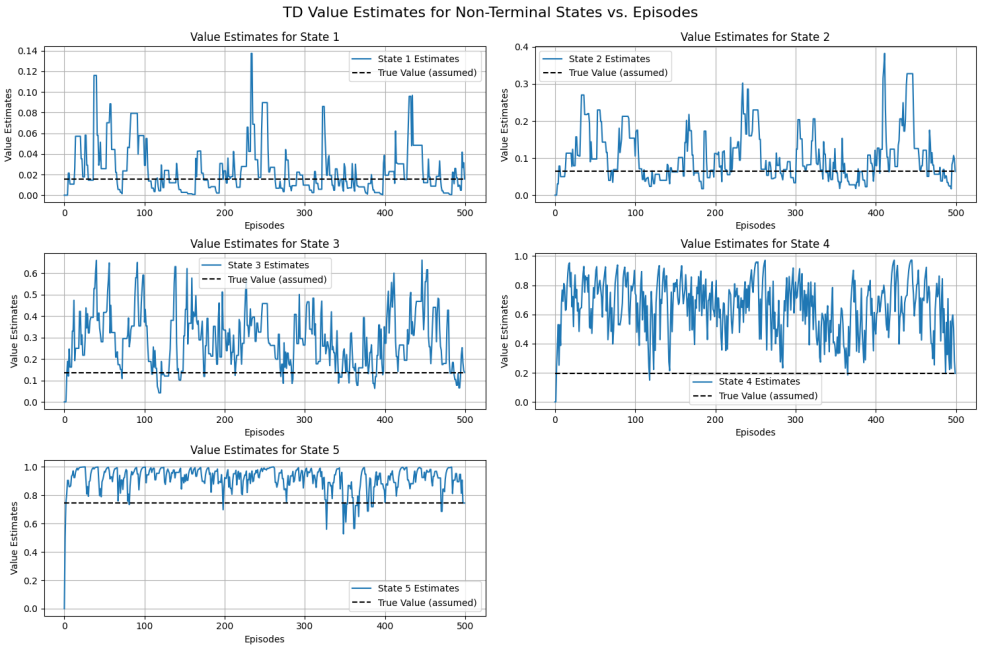


Figure 29: TD plots for constant $\alpha = 0.5$

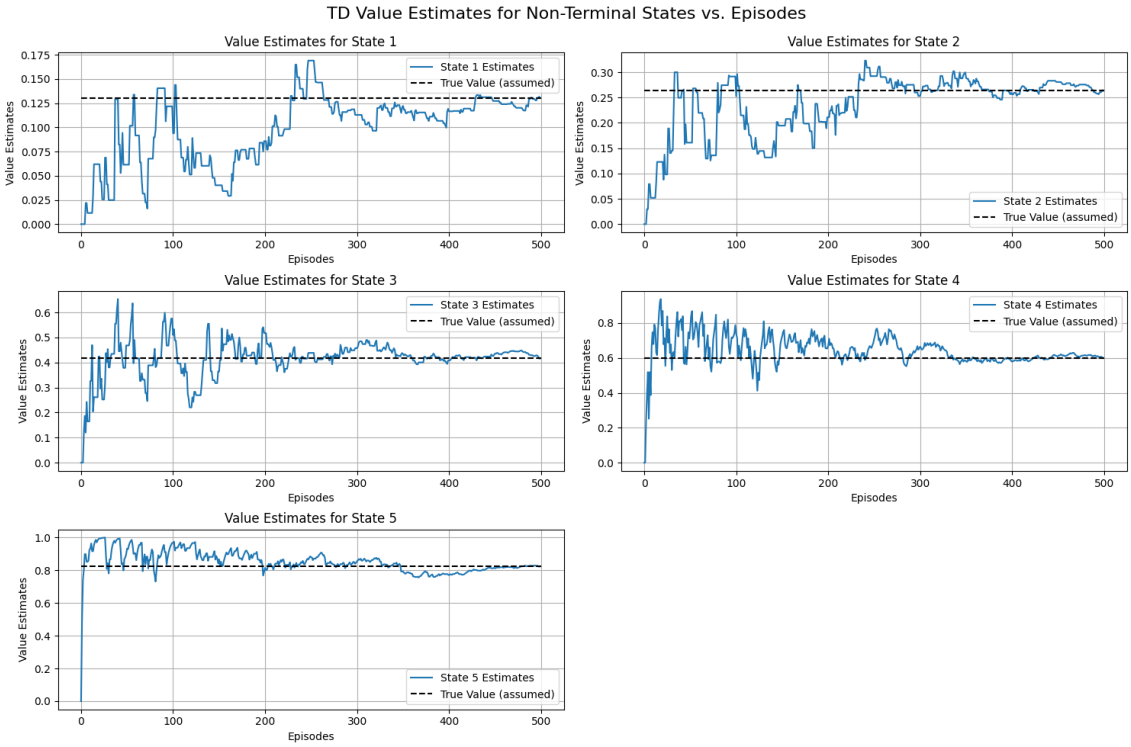
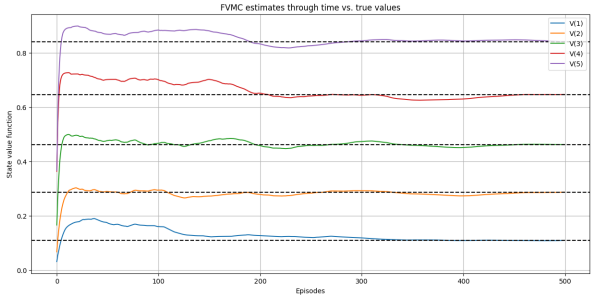


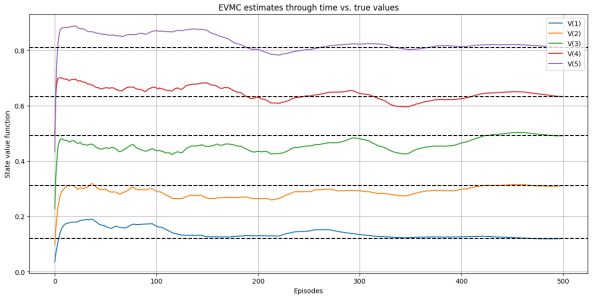
Figure 30: TD plots for decaying $\alpha = 0.5$ to 0.01

8. In the above plots, **Monte Carlo** does not seem to converge to the true value within the **500** episodes, as the estimates do not stabilize around the dashed line representing the true value. Some state estimates in both methods seem to have less variance as the number of episodes increases (e.g., **State 4**), while others maintain high variance throughout (e.g., **State 3**). The plots also depicts that **State 6** has a high variance so it must be challenging for the agent to estimate correctly according to the above mentioned policy. Whereas, the **TD** estimate are less fluctuating indicating a comparatively stable learning process and quicker convergence around the true estimate especially for the case of decaying α. The values quickly converge with less variance to the true estimates indicating a stable learning process and appropriate discount factor γ. The exponentially varying alpha suggest that there is a good balance between agent adaptation to problem space and exploiting later.

9. The next set of plots describe the averaged out value estimates for all the **seeds ranging from 100 to 300**. The other set of plots kind of zooms into the initial stages of the learning process via taking x axis in the log scale. The first three plots are for the **FVMC, EVMC and TD** averaged estimates over **500** episodes with α decaying from **0.5** to **0.01**. These plot eliminates the noise and its clear that TD plots converge way more quickly with less variance than Monte Carlo estimates.



(a) FVMC Averaged out value estimates over 500 episodes



(b) EVMC Averaged out value estimates over 500 episodes

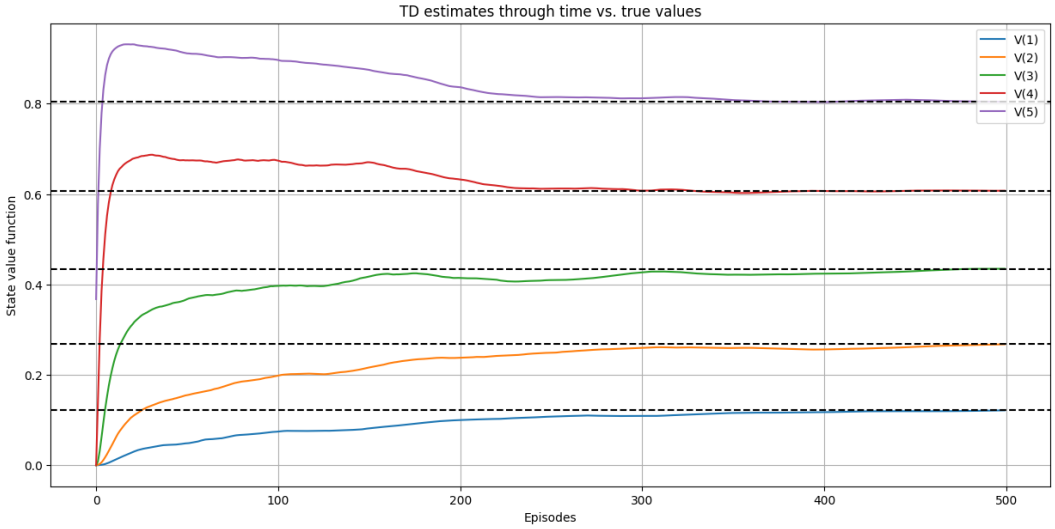


Figure 32: TD Averaged out value estimates over 500 episodes

10. The first set of plot depicts the **FVMC** algorithm with a constant value of α which is set to **0.5** and another with exponential decaying α for **500** episodes in the log scale. The values of every state with successive episodes is plotted. The log scale helps to focus attention towards the early stage of learning. Thus the exploratory behaviour of the agent is captured precisely in the log scale. The decaying α plots are way smoother than constant α plots. While both sets show the characteristic high variance of Monte Carlo methods, the second set appears to exhibit slightly more stability and closer alignment with the true value in certain states.

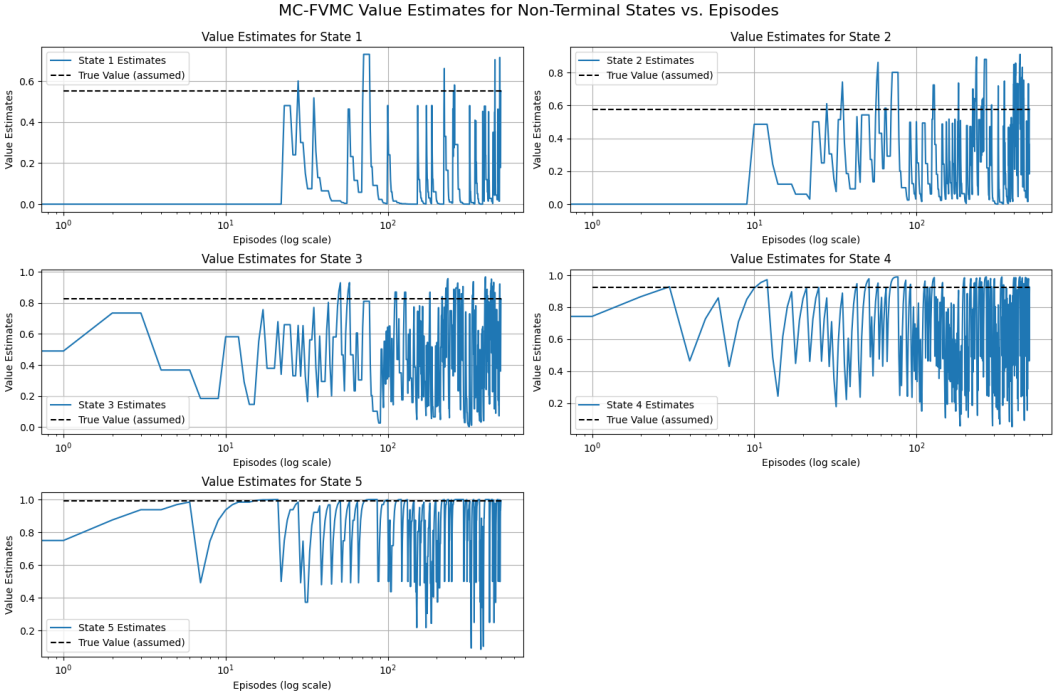


Figure 33: MC-FVMC plots for constant $\alpha = 0.5$

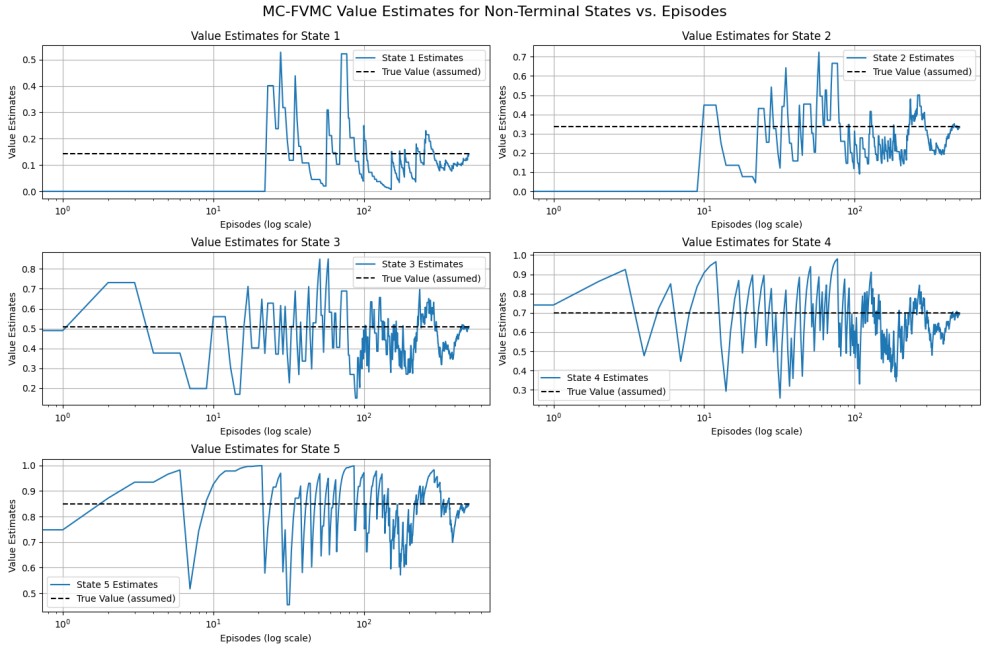


Figure 34: MC-FVMC for decaying $\alpha = 0.5$ to 0.01

11. The first set of plot depicts the **EVMC** algorithm with a constant value of α which is set to **0.5** and another with exponential decaying α for **500** episodes. The values of every state with successive episodes is plotted. The log scale helps to focus attention towards the early stage of learning. Thus the exploratory behaviour of the agent is captured precisely in the log scale. From the constant α plots, for **States 1 and 2**, the value estimates start with large swings and gradually show less variance as the number of episodes increases. However, they do not seem to converge to the true value within the range shown, **State 3** has a huge fluctuation throughout and doesn't seem to converge at all. **State 4** shows initial convergence towards the true value but then diverges and continues with significant variance. **State 5** shows a gradual decrease in variance and seems to be converging towards the true value, which is a positive sign. There is significantly low variance for the decaying α plots suggesting a stable learning process for all the states. The **EVMC** estimates seem to be noisier than that of **FVMC** estimates for almost all states.

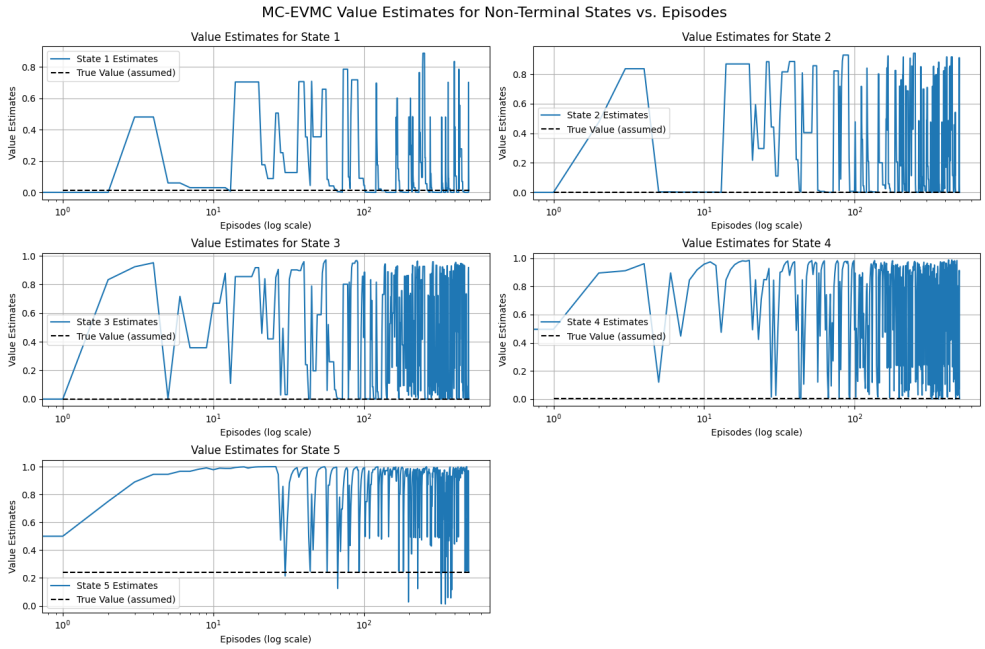


Figure 35: MC-EVMC plots for constant $\alpha = 0.5$

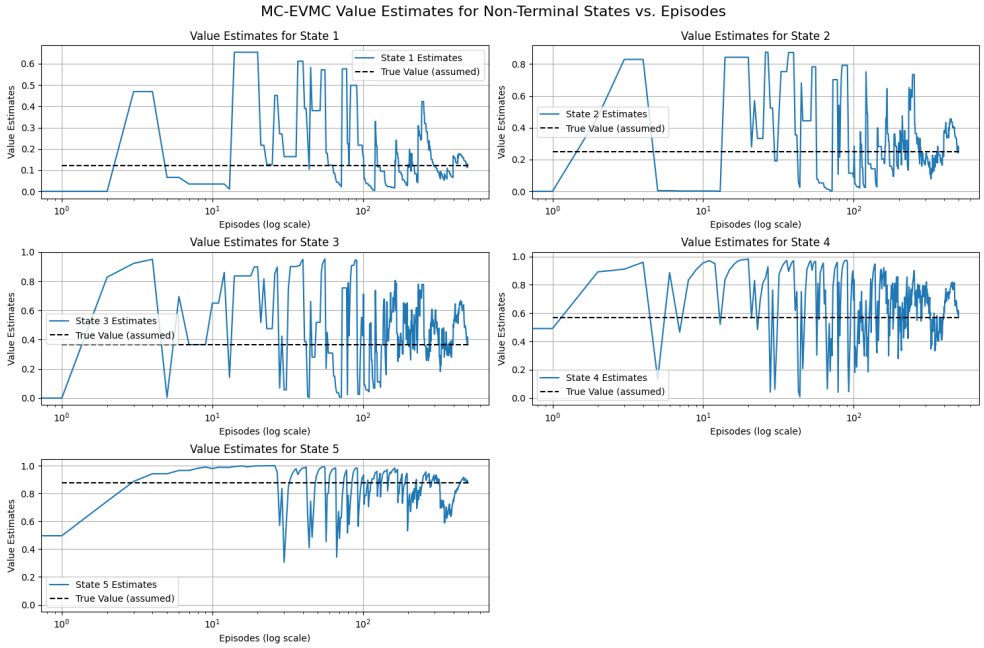


Figure 36: MC-EVMC for decaying $\alpha = 0.5$ to 0.01

12. These plots depict the **TD estimates** with episodes in log scale, focusing more on the initial episodes of exploration. The TD learning method appears to produce relatively stable estimates compared to the previous Monte Carlo methods. This is indicated by the lower amplitude of fluctuations in the value estimates. Almost all of the agents shows higher variance initially but depicts a slight degree of convergence towards the true value with decreasing fluctuations in the later stages. The estimates oscillate more frequently in the later stages for the case of decaying α as compared to that of constant α suggesting a stable learning rate overall. Despite the general trend toward stability, there's noticeable noise in the learning process which is eliminated by averaging across different sets. Overall, the **TD** plots are better than **MC** plots in terms of degree of fluctuations.

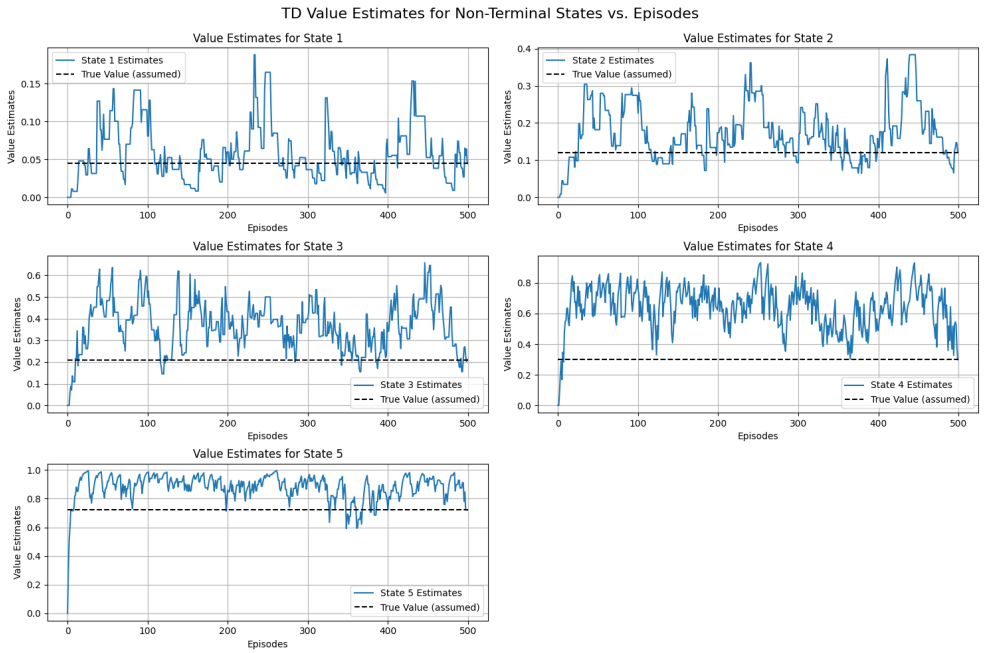


Figure 37: TD plots for constant $\alpha = 0.5$

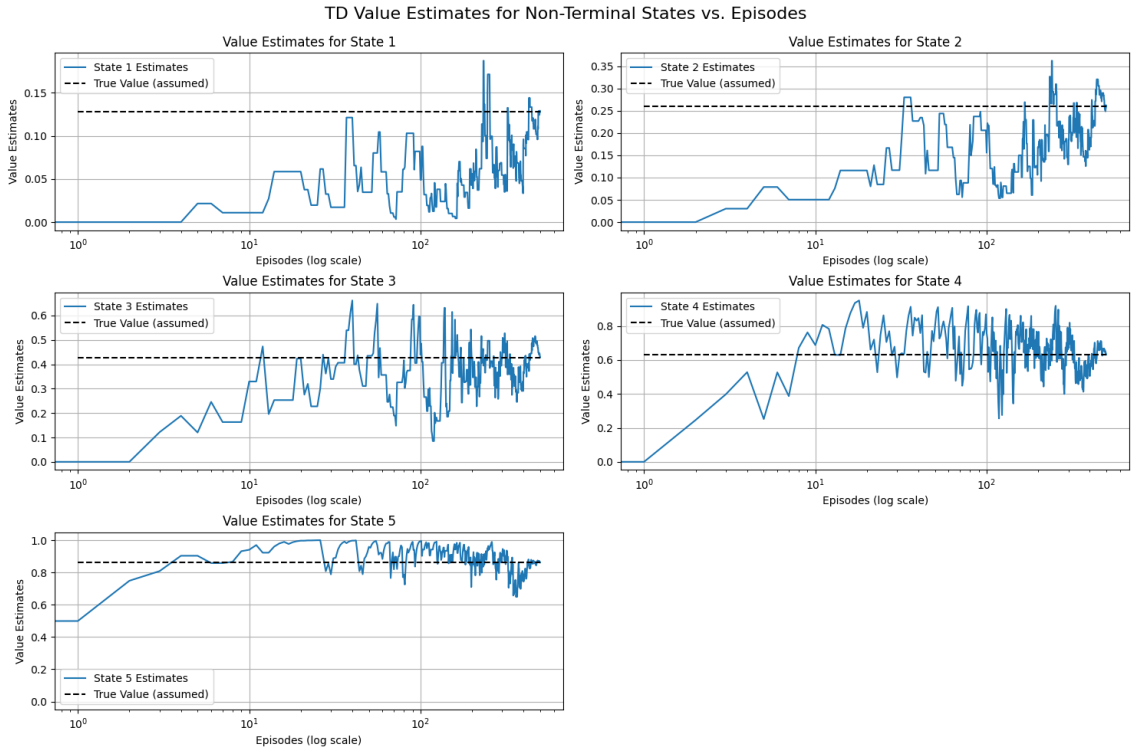
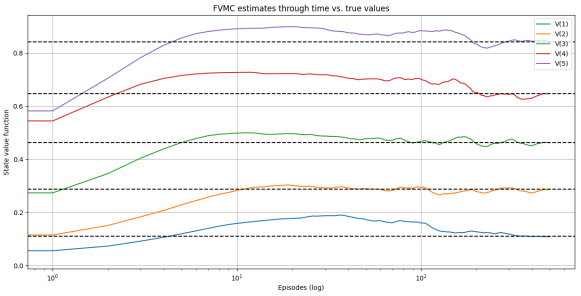
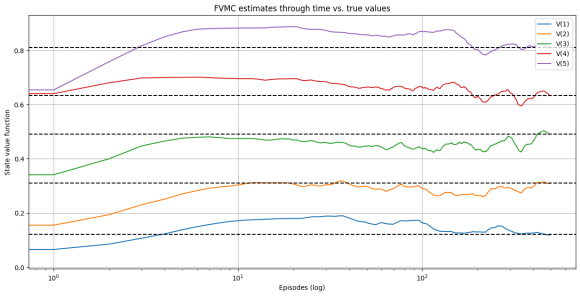


Figure 38: TD plots for decaying $\alpha = 0.5$ to 0.01

13. The next set of plots describe the averaged out value estimates for all the **seeds ranging from 100 to 300**. The other set of plots kind of zooms into the initial stages of the learning process via taking x axis in the log scale. The first three plots are for the **FVMC, EVMC and TD** averaged estimates over **500** episodes with α decaying from **0.5** to **0.01**. These plot eliminates the noise. The **TD** plot (third image) also shows a trend toward convergence, with the value estimates progressively approaching the true values because of bootstrapping. The **FVMC** plots display greater variance in the initial episodes, especially visible on a logarithmic scale. The variance seems to decrease as the number of episodes increases. The TD estimates appear smoother and less variable from the start. All methods seem to have difficulty with certain states. For example, **State 2** in the **FVMC** plots and **State 4** in the **TD** plot show more deviation from the true value compared to other states. Thus overall, the **TD averaged** estimates indicates consistent and more stable learning from early episodes itself because of its incremental approach.



(a) FVMC Averaged out value estimates over 500 episodes



(b) EVMC Averaged out value estimates over 500 episodes

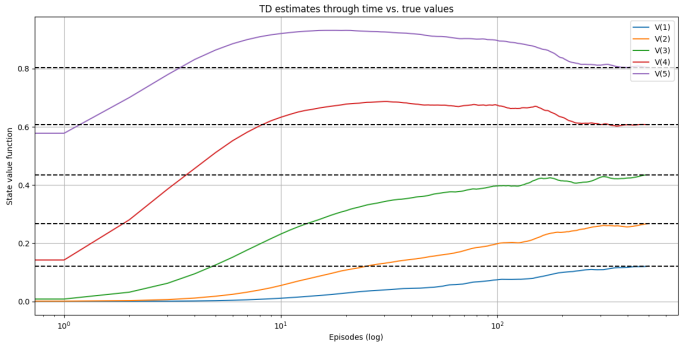
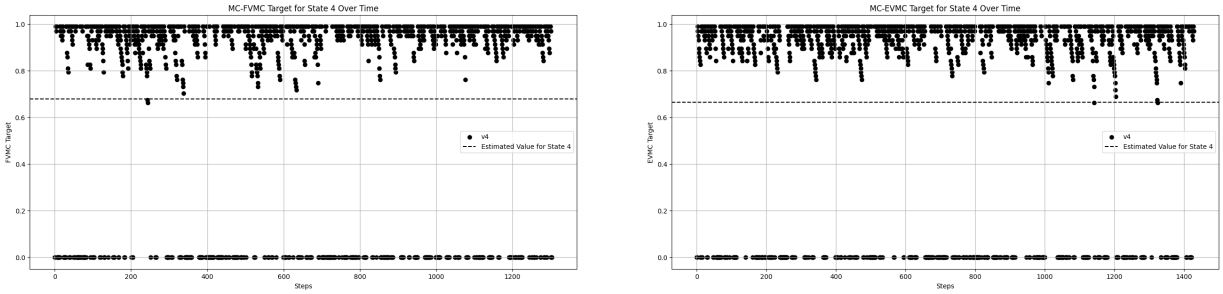


Figure 40: TD Averaged out value estimates over 500 episodes

14. The **FVMC-target** and **EVMC-target** plots for state 4 is shown below. The plot depicts a higher variance in the beginning of episodes and the values finally becomes stationary which indicates that the learning is almost complete. The values don't show an upward or downward trend which denotes that the algorithm is stable and not diverging and there are no sudden explorations in later episodes. Since, **Monte Carlo estimation** targets involves calculations of returns after an entire episode, the plot also denotes sufficient exploration of the environment by the agent. Since the convergence to the estimated value appears consistent, one might infer that the learning process could potentially be stopped early, as additional episodes may not provide significant new information regarding the value of State 4



(a) FVMC-Targets for state 4

(b) EVMC Targets for state 4

15. The **TD-target** plot for state 4 is shown below. The target plot depicts that the the **TD-targets** initially have a high variance which denotes that the agent is beginning to explore and dynamics. The agent is updating its estimates over a wide variety of possible experiences. The faster convergence also denotes that the step parameter is comparatively good otherwise the values might have converged slower. Since, the target values come closer to each other, this indicates that the learning is almost complete and the targets have converged to a final value. The gap between the **true** value and **targets** converged indicates that the policy is not that optimal for the environment since the plot has a steady state error in the later episodes. The plot also suggest that the balance between exploration and exploitation in the policy is managing to provide a good sampling of experiences from which to learn the value of State 4.

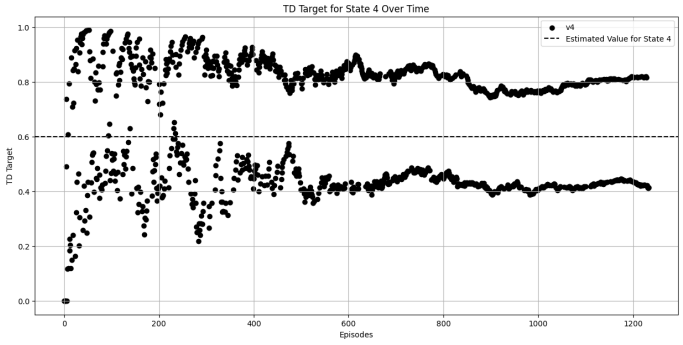


Figure 42: TD Targets for state 4