COL333 Assignment 3

Rahul Chhabra
2019CS11016

Prashant Mishra
2019CS50506

November 2021

1 Computing Policies

1.1 Formulation as Markov Decision Process

1.1.1 State Space

The state space consists of 650 states. Each state consists of position of taxi, position of passenger and a boolean representing whether the passenger is in taxi or not. So in total there are 25 possibilities for each position so 25 x 25 x 2 = 1250 possibilities but when the passenger is sitting in side the taxi, the taxi location and passenger location has to be same. So there are total of 25 x 25 (When Passenger is not sitting in taxi) + 25 (When passenger is sitting inside taxi) = 650 possible states.

Each state is represented by a tuple : (Taxi Location, Passenger location, is passenger sitting)

1.1.2 Action Space

The set of actions available to our taxi at any state are {North, South, East, West, pickup, dropdown}

1.1.3 Transition Model

Table 1: Transition Probabilities

Intended \ $Taken$	North	South	East	West	Pickup	DropDown
North	0.85	0.05	0.05	0.05	0.00	0.00
South	0.05	0.85	0.05	0.05	0.00	0.00
East	0.05	0.05	0.85	0.05	0.00	0.00
West	0.05	0.05	0.05	0.85	0.00	0.00
Pickup	0.00	0.00	0.00	0.00	1.00	0.00
PutDown	0.00	0.00	0.00	0.00	0.00	1.00

1.1.4 Reward Model

At each step if we perform putdown operation at drop location and passenger is sitting inside the taxi, then a reward of +20 is provided to the taxi. In case of a putdown operation or a pickup operation if passenger and taxi are not in same location, then a reward of -10 is given . In all other cases, -1 is rewarded for each step.

1.2 Simulation of taxi

We have simulated the taxi environment with given MDP, we have also added functionality of displaying the state and the actions taken which can be seen feom the figure below:

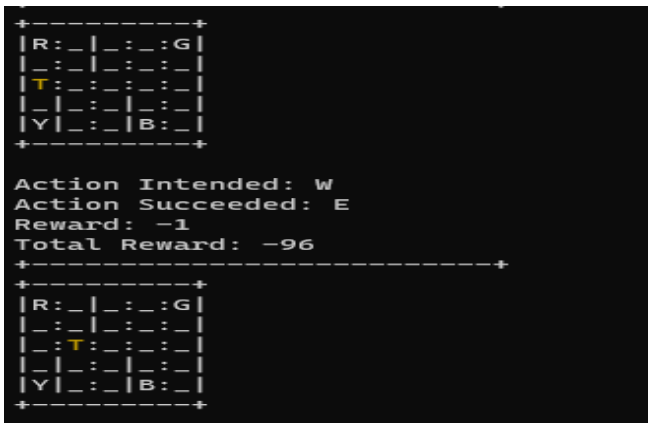

Figure 1: Simulation of taxi environment for 5 x 5 Grid

- Bellman equations characterize the optimal values:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- Value iteration computes them:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Figure 2: Bellman Equations for updates in Value Iteration

1.3 Value Iteration

We have implemented the Value Iteration Algorithm from scratch for finding the optimal values for all states. This has been done using the following Bellman Equations:

Let U be the true utility and let U^{π_i} be the estimate of the true utility at the iteration i. Note that U^{π_i}(s) can be calculated by executing the policy i starting from s. Then the policy loss for the policy π_i is the max-norm distance between U and U^{π_i}.
That is, policy loss for π_i = Norm(U − U^{π_i}).

The values of epsilon taken is : $1e^{-6}$

1.3.1 Effect of Discount Factor on convergence

1.3.2 Observations

1. The number of iterations required to converge the policy increases with increase in discount factor because with increase in discount factor, because more the discount factor, more is the weightage given to the previous observations, since initially the values are not enough close to the optimal values, their effect remain with the current values for longer time and it takes more iteration to remove it's effect.

2. The decrease in max-norm values is similiar the decreasing exponential curve and becomes more smooth with increase in value of discount factor. This is due to the persitent value effect of non-optimal values embeeded from initial iterations.

1.3.3 Plots

1.4 Execution of policy trained above for given instance with gamma = 0.1 and gamma = 0.99

The above trained policy was executed on the instance of problem with Initial passenger location as **Y**, inital taxi location as **R** and drop location as G. In case of 0.99, the taxi moves from the stating position, reached

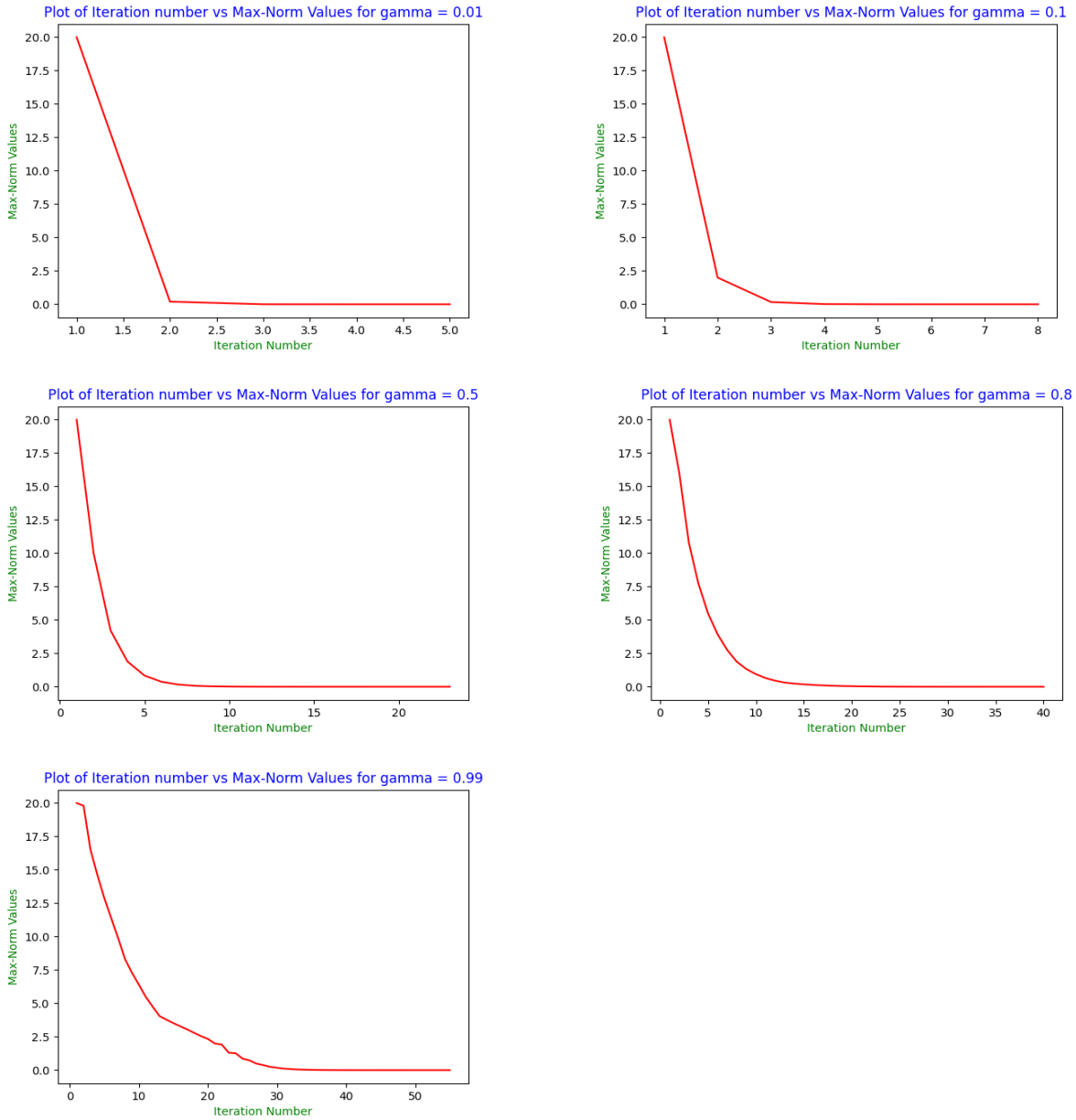

Figure 3: Plot of max norm with number of iterations

the pickup point, picks up the passenger and successfully . The trace of execution is as follows:
For GAMMA = 0.9

```
+---------+
|T:_|_:_:G|
|_:_|_:_:_|
|_:_:_:_:_|
|_|_:_|_:_|
|Y|_:_|B:_|
+---------+

((0, 0), (4, 0), 0)
Action Intended: S | Action Succeeded: S  |  Reward: -1 | Total Reward: -1
((1, 0), (4, 0), 0)
Action Intended: S | Action Succeeded: E  |  Reward: -1 | Total Reward: -2
((1, 1), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -3
((1, 0), (4, 0), 0)
Action Intended: S | Action Succeeded: S  |  Reward: -1 | Total Reward: -4
((2, 0), (4, 0), 0)
Action Intended: S | Action Succeeded: S  |  Reward: -1 | Total Reward: -5
((3, 0), (4, 0), 0)
Action Intended: S | Action Succeeded: S  |  Reward: -1 | Total Reward: -6
((4, 0), (4, 0), 0)
Action Intended: pickup | Action Succeeded: pickup  |  Reward: -1 | Total Reward: -7
((4, 0), (4, 0), 1)
Action Intended: N | Action Succeeded: N  |  Reward: -1 | Total Reward: -8
((3, 0), (3, 0), 1)
Action Intended: N | Action Succeeded: N  |  Reward: -1 | Total Reward: -9
((2, 0), (2, 0), 1)
Action Intended: E | Action Succeeded: E  |  Reward: -1 | Total Reward: -10
((2, 1), (2, 1), 1)
Action Intended: E | Action Succeeded: E  |  Reward: -1 | Total Reward: -11
((2, 2), (2, 2), 1)
Action Intended: N | Action Succeeded: N  |  Reward: -1 | Total Reward: -12
((1, 2), (1, 2), 1)
Action Intended: E | Action Succeeded: E  |  Reward: -1 | Total Reward: -13
((1, 3), (1, 3), 1)
Action Intended: N | Action Succeeded: N  |  Reward: -1 | Total Reward: -14
((0, 3), (0, 3), 1)
Action Intended: E | Action Succeeded: E  |  Reward: -1 | Total Reward: -15
((0, 4), (0, 4), 1)
Action Intended: putdown | Action Succeeded: putdown  |  Reward: 20 | Total Reward: 5
((0, 4), (0, 4), 0)
```

For GAMMA = 0.1

```
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -1
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -2
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: N  |  Reward: -1 | Total Reward: -3
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -4
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -5
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: E  |  Reward: -1 | Total Reward: -6
((0, 1), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -7
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -8
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -9
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: E  |  Reward: -1 | Total Reward: -10
((0, 1), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -11
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -12
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: N  |  Reward: -1 | Total Reward: -13
((0, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: S  |  Reward: -1 | Total Reward: -14
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -15
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -16
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -17
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -18
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: W  |  Reward: -1 | Total Reward: -19
((1, 0), (4, 0), 0)
Action Intended: W | Action Succeeded: N  |  Reward: -1 | Total Reward: -20
((0, 0), (4, 0), 0)
```

It can be clearly seen that for high value of discount factor, the policy trained was optimal and agent could reach and pick the passenger and drop it at the required position but in case of GAMMA = 0.1, the policy trained couldn't even let the taxi agent move from its initial position although from previous parts its clearly visible that at 0.1 the it takes lesser iterations to converge but to a non optimal policy which is visible from the following traces, This is due to the fact that at low values of dicount factor we give more weightage to the current rewards and don't care much about the optimal values we have attained till now.

1.4.1 Analysis of effect of Discount Reward on Policy Loss Values

1.4.2 Iterative method of evaluating policy loss

Observations:

- The number of iterations required to converge to an optimal policy is very less compared to value iteration at same discount factor. This is due to that fact that we are not learning optimal values, we are learning sub-optimal values and the policy starts to converge earlier than the values which is directly visible from the plots

- The number of iterations remains almost same with change in discount factor but there is some anomaly observed at discount factor of 0.1 where it takes more number of iterations, one of the reasons for this might be that at very low value of discount factor, the previous actions are not taken much seriously and the algorithm may converge to a policy which might not be close to optimal and take a very few iterations (can be visible from the plot of Gamma = 0.01) and at high value of gamma, since we always give high weightage to the optimal policy obtained till now, we may get to the optimal policy soon but not the optimal values. In case of gamma = 0.1 none of the factors seems to dominate, hence it shows high number of iterations.
- The policy iteration using linear method takes slightly more iterations as compared to the iterative method but time taken for each iteration is small since we ar enot using max-norm to converge, instead we just use simple matrix algebra which gives precise results faster.The plots obtained in linear method are better in terms of less variance and high precision which is visible from the clarity of curve.
- In case of Policy iteration with linear algebra the policy doesn't converge and starts oscilating and hence is stopped after 50 iterations. This can be reasoned from the above point where it seems that the oscillating policy is unable to decide the convergence due to the current reward it is getting is not corresponding to any of the suboptimal policies.
- The curevs look like that of decreasing exponentials and the also show weird behaviours at gamma = 0.9 which corresponds to a very very high bias towards the previously learnt policy values. In starting it decreases very slowly but after the goal state is explored by the the agent, it starts decreasing suddenly and shows an exponential decrease the reason being the newly obtained rewards outperforms the accumulated persistent rewards over time.
- We think that Policy iteration with linear method is better due to the forementioned reasons.

2 Incorporating Learning

The four learning algorithms were trained for this problem are:

1. Q Learning
2. Q Learning with decaying exploration rate
3. SARSA Learning
4. SARSA Learning with decaying exploration rate

To plot the graphs :

- Firstly the intervals taken are of size 100 episodes i.e. we first train the model for 100 episodes then run this on 100 instances of problem of get the average discounted rewards and plot it.
- Since the plots are **not** taken from the training itself, the show behaviour corresponding to that learnt policy and not the policy and rewards which we were obtaining during training since they have very high variance.

2.1 Plots of Q learning Graph

- In starting, when the number of episodes over which the q values are trained is low, the plot remains mostly on the negative side and shows a very high variance. This is due to the fact that for less number of episodes, the policy (from q values) is not the optimal one, so the agents usually shows some random kind of motion with very less affinity towards the goal state, this leads to a irregular behaviour of the curve for less number of episodes. Now when the number of episodes are increased, the policy starts coming towards optimal and hence the sum of discounted rewards starts increasing and it takes its optimal value, after taking the optimal value, the plot remains almost constant since we have attained the optimal policy and the reward corresponding to this policy remains almost same.
- In case of SARSA, the plot shows high variance even after convergence due to induced stochasticity from the one extra action taken. The one extra action and state taken adds a factor of stochasticity from the transition probabilities to our model which causes it to cause high variance and even deviate after convergence(state when the optimal reward value has been achieved)

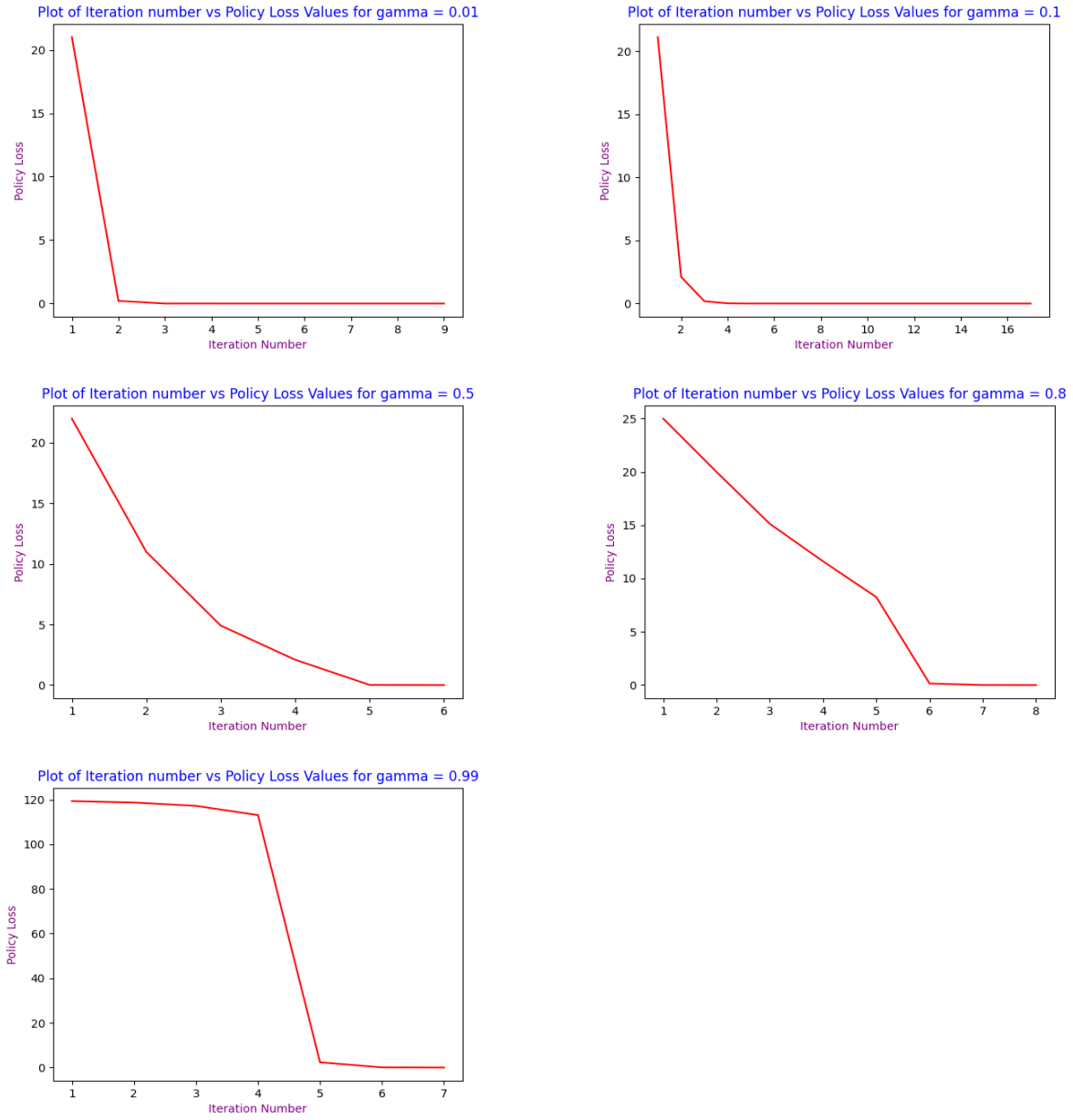

Figure 4: Plot of Policy Loss with number of iterations

- When decaying exploration is used in both the algorithms(Simple Q-Learning and SARSA), the algorithms seems to converge faster than usual since in case of decaying exploration rate, we have taken exploration rate inversely proportional to the iteration number and after large number of iteration, the need to explore random state actually decreases since after large number of explorations, the current values nearly correspond to the optimal policy and exploring random states just adds an unneccesary overhead.
- Although all algorithms converge to the same optimal value but simple Q learning with decaying exploration rate seems to converge faster and remains stable after the convergence, so we choose this as our best algorithm.
- The initial regret in this case is still on the higher side but due to its faster convergence, it will be better than others.

2.2 Execution of best learning algorithm on 5 instances of MDP

The best algorithm we have obtained from part 2 is Simple Q learning with decaying exploration rate. On executing it on 5 instances of model with different passenger and taxi initial location, it is observed that the agent is able to pick up and putdown the passenger at required location in most of the cases but the rewards .

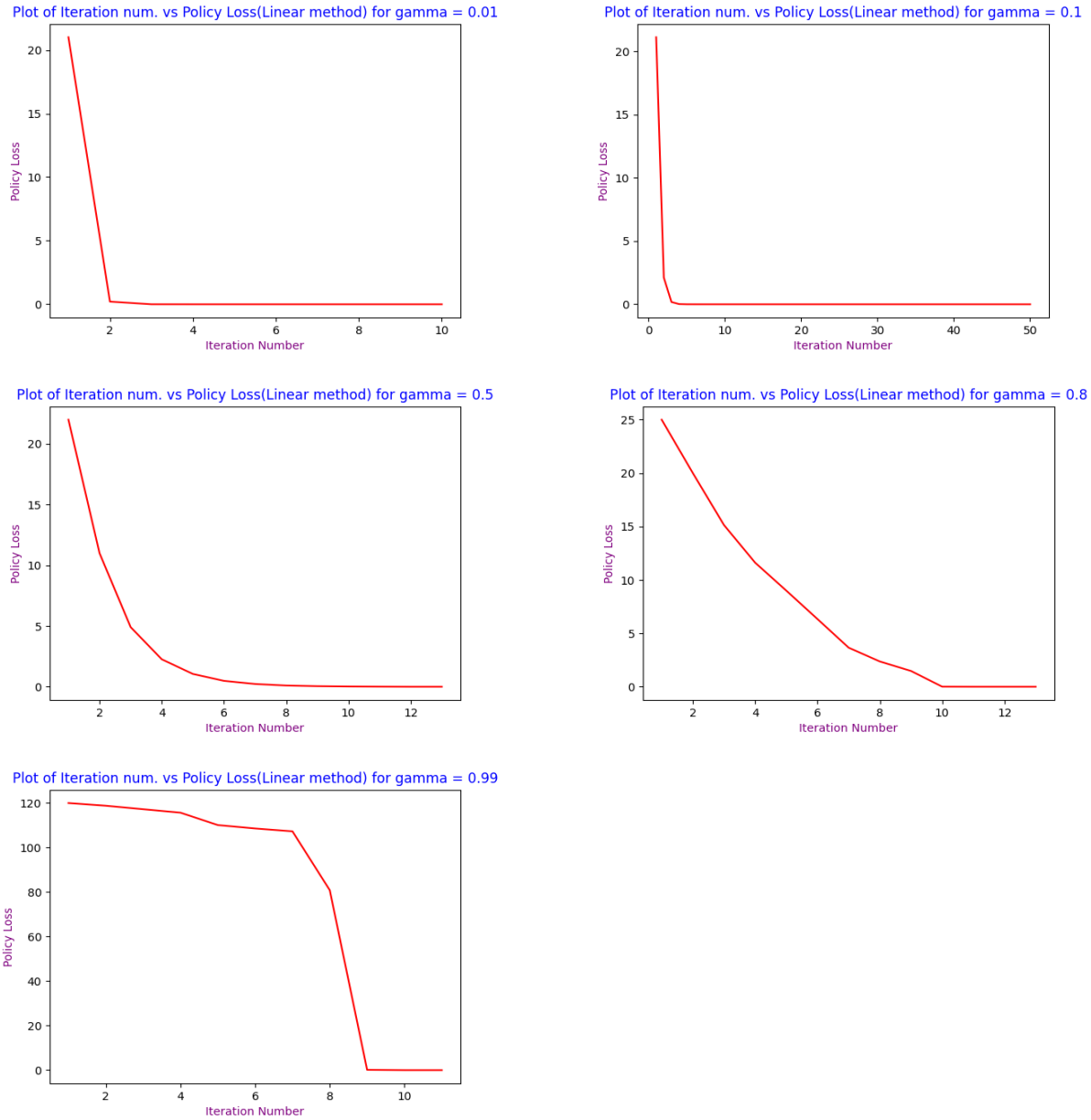

Figure 5: Plot of Policy Loss (Linear Algebra) with number of iterations

The rewards also come out to be sub-optimal according to the given instance of the problem.But still the offline algorithm works more better than this and provides the optimal policy whereas the online learning algorithm seems to have less optimal policy due to the restriction of number of episodes however on training the online learning algorithm on large episodes, the rewards were close to the optimal ones expected. The computation cost of doing such way is relatively higher than the offline one and this is due tot he fact that the agent doesn;t know about transitions and rewards and has to explore on its own .

2.3 Varying exploration rate with constant alpha (Learning Rate)

- As the equation for Q Learning update for online is given by :
  $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(reward + \gamma xargmax(Q(s'))))$
  where at step an action is taken randomly with probability ϵ and the greedy action (action corresponding to largest Q value) is taken woth probability 1 - ϵ

- With increase in value of epsilon, the variance in graph for lower iterations decreases. This is due to the fact that more epsilon means more exploration of random states and less bias towards the learnt values. Since initially learnt q values may not converge to good policy, so it is suitable to explore states randomly to get to the goal state, once we get to the goal state, the Q value for the states increases and hence after

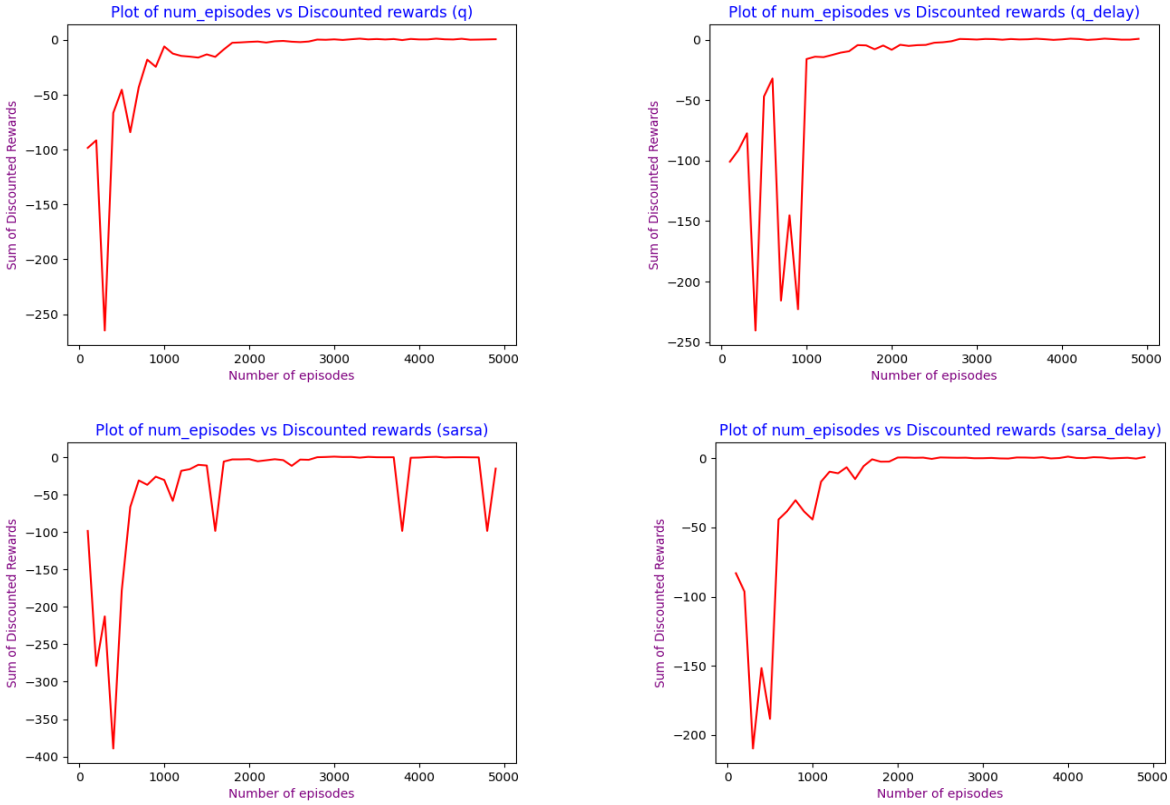

Figure 6: Plot of sum of discounted rewards (averaged over 100 runs) with number of episodes

around 2000 iterations, the plot starts becoming constant because the effect of exploring random states decreases due to very high q value compared to average expected reward for any state.

- At very high exploration rate, we can observe that the graph goes very smoothly towards convergence, this is because the agent is not biased towards the currently learnt policy (at any iteration) and always willing to explore new states, so it finds an approximate policy in the very starting of the exploration and genuinely keeps on improving it.
- At very low exploration rate, the graph obtained show zig-zag behaviour, each large drop in the graph shows us the regret at current policy (We have defined regret as the loss in sum of discounted reward obtained when the agent finds from random exploration that the current policy it has been working on it not the optimal i.e. it finds another action for any state that gives better rewards). From the graphs it is clearly visible that the number of regrets decreases with increase in exploration rate.

2.4 Varying learning rate (α) with constant exploration rate(ϵ)

- As the equation for Q Learning update for online is given by :
$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(reward + \gamma xargmax(Q(s'))))$
where at step an action is taken randomly with probability ϵ and the greedy action (action corresponding to largest Q value) is taken woth probability 1 - ϵ
- With increase in value of alpha, the algorithm starts to converge early as compared to low alpha because alpha defines the weightage of current observation, if alpha is high the current iteration is given more weightage and previous observations are given less weightage. Now since epsilon is 0.1 which is on the lower side, the random exploration will be less and the policy obtained will still have variance but will reach optimal policy faster.
- For lower learning rate, once the algorithm reaches the optimal value, it remains constant for other episodes whereas on case of high learning rate, the algorithm even after reaching the optimal value shows variance due to less affinity of model towards already learnt policy.

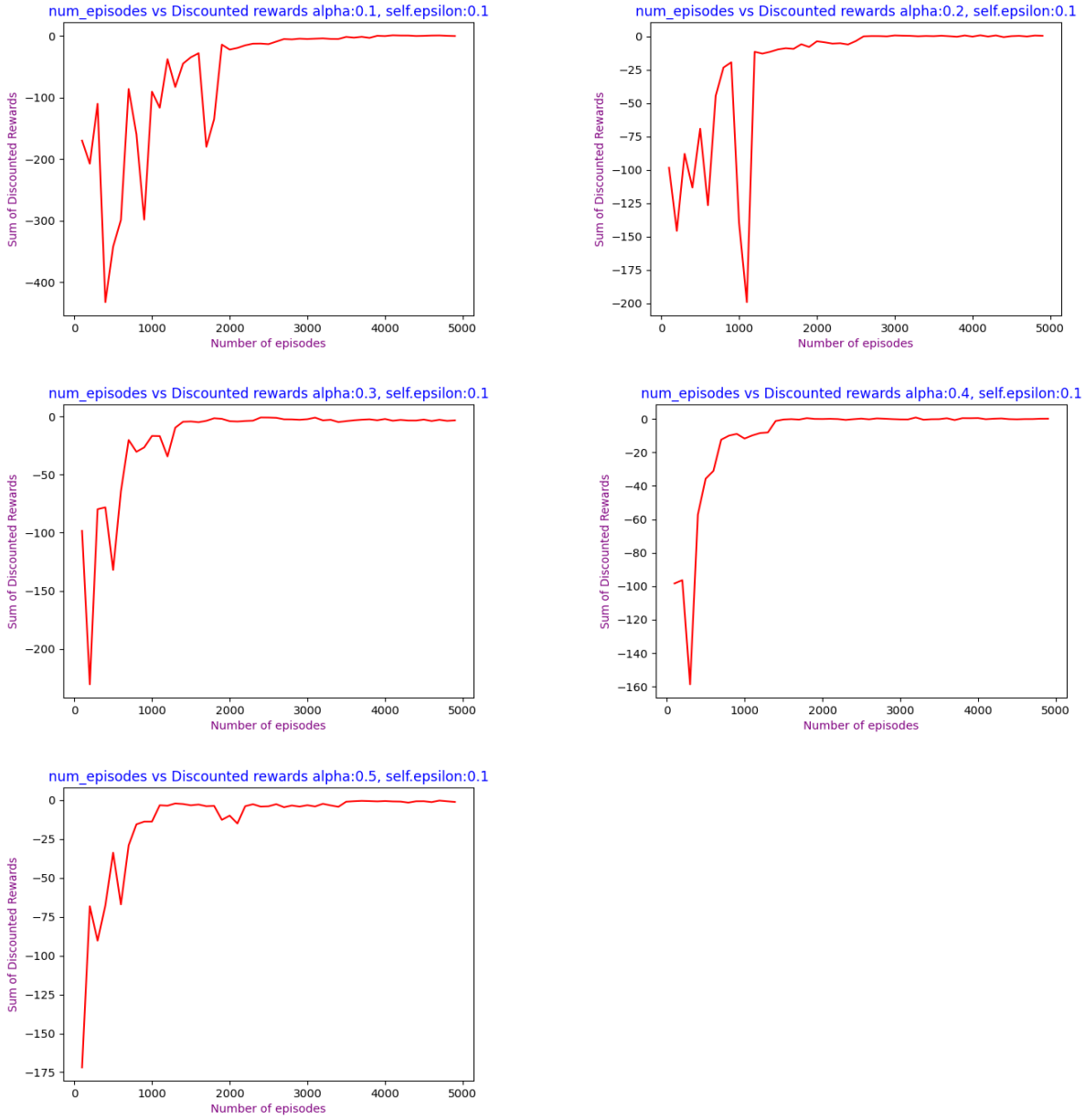

Figure 7: Plot of convergence with fixed exploration rate(epsilon) = 0.1 and varying alpha in [0.1,0.2,0.3,0.4,0.5] with number of episodes

2.5 10 x 10 Extension

The algorithm used to train model on 10 x 10 extension of the above problem is Q learning with decaying exploration rate with initial exploration rate = 0.1 ,learning rate = 0.25 and discount factor = 0.99.

The average Reward Obtained is : - 62.598

3 Running Code

python3 A3.py GAMMA ALPHA EPSILON

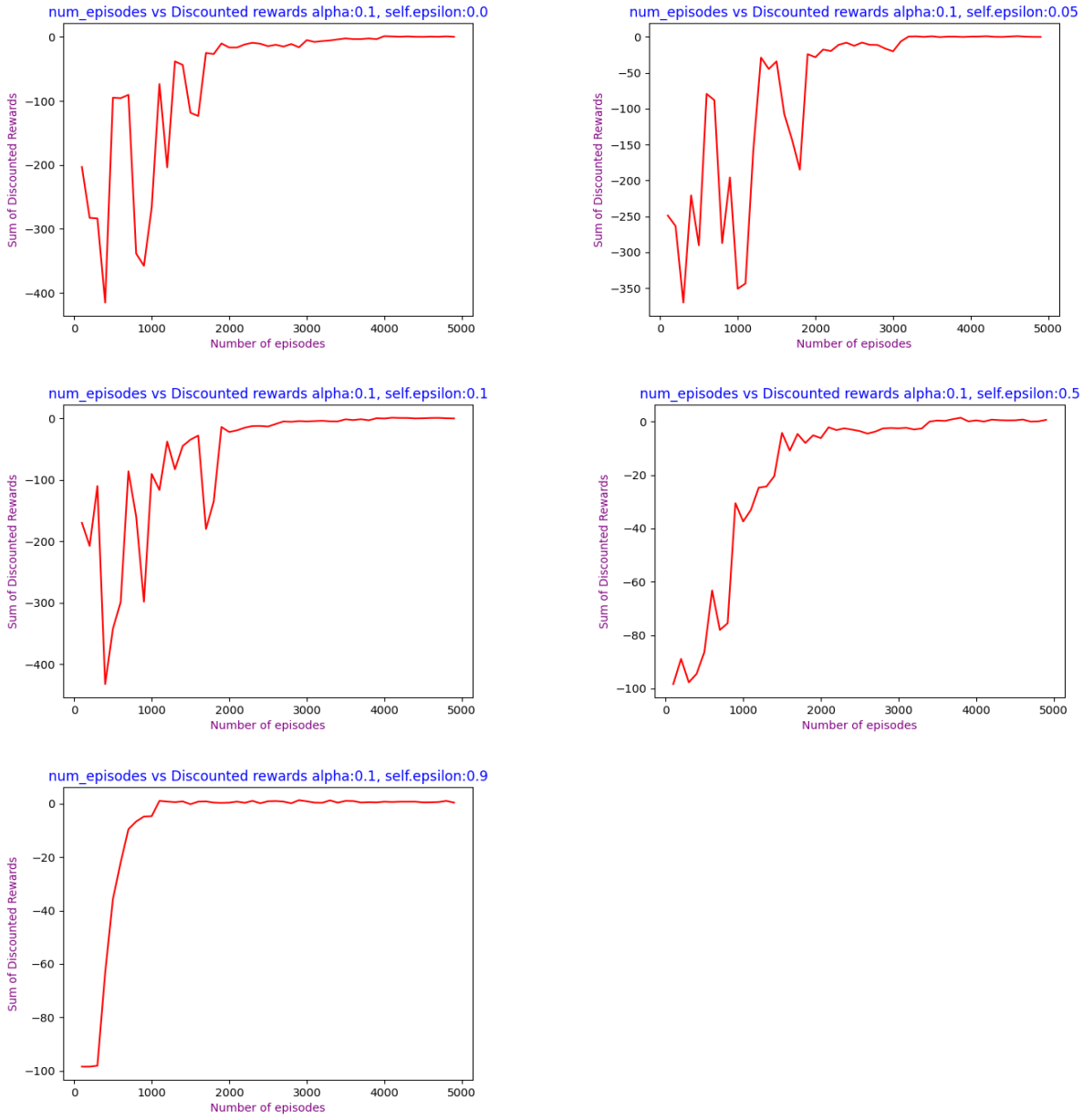

Figure 8: Plot of convergence with fixed exploration rate(epsilon) = 0.1 and varying alpha in [0.1,0.2,0.3,0.4,0.5] with number of episodes

Table 2: 10 x10 Extension results

S. No.	Taxi Location	Passenger Location	Drop Location	Reward (Discounted)
1	R	P	G	-62.762
2	R	B	C	-60.662
3	Y	P	R	-62.002
4	M	R	P	-62.922
5	R	P	G	-64.642