Assignment 4

Overview

In this assignment we have implemented Markov Decision making Process using the various iterative algorithms such as Policy Iteration, Value iteration and Q learning algorithms and have compared their performances. Convergence performance of each of these algorithms have been compared. All algorithms have been implemented on two MDP problems described in the subsequent sections of this document.

The frozen lake MDP problems

For the purpose of this assignment we have selected two forms of the frozen lake problem. One involves a 4x4 grid and another involves a 8x8 grid. The agent starts from one end of the grid and have to reach the other end which is the goal. Agent can only move in discrete steps in the grid. There are four types of position on the grid:

- Start: Location from where the agent start his/her journey
- Frozen surface: These are the allowable states where agent can move
- Hole: If agent moves to these states, they have to start all over again
- Goal: Agent finally have to reach this state to get a reward.

Agent has following 4 potential actions

- Left: 0
- Down: 1
- Right: 2
- Up: 3

The agent gets a reward of 0, if he/she falls into the hole and a reward of 1, if the agent make it to the goal. If the agent falls into the hole, the episode ends and then the agent has to start the process all over again. At each state agent can move into various locations (right, left, up and down). A schematic of a 4x4 grid is shown below

Start	Frozen	Frozen	Frozen
Frozen	Hole	Frozen	Hole
Frozen	Frozen	Frozen	Hole
Hole	Frozen	Frozen	Goal

Figure 1. Schematic of a 4x4 frozen lake problem

The second problem considered here is the scale up of the previous one. We intentionally chose the scaled up version of the frozen Lake problem so as to understand and trend the learning behavior of the agent under the same environment but with increased number of states.

Why Frozen Lake Problem?

We have chosen these problems to focus because we feel that this problem has significant practical importance in designing self-moving robots. The holes can be seen as obstacles and frozen surface can be seen as permissible positions where a robot can move to reach the goal. This type of learning can be applied to design simple robots that hops into the grid and delivers something at the goal location. Another application can be in designing games such as Snakes and Ladders. For implementation of the game, the grid will need little modification where there will be another state called normal state, a happy state which gets rewards. When the agent steps into hole, it gets a negative reward and can still continue in the game. The agent reaches the goal and collect rewards and penalties along the way. The path that results in maximum rewards can is the best path. In the game these paths will be basically dice moves that computer will make. Also since we can further grade the paths based on rewards and hence translate this into toughness level of the game for the user. For example: if the human player selects very tough level, the computer will start choosing paths with maximum rewards. If human player selects easiest, computer can choose paths with moderate and minimal rewards.

Solving using Value Iteration

Theoretical Background

In this section we implement the value iteration algorithm on both versions of the frozen lake problems. The value iteration involves evaluating the rewards for accomplishing a particular action. The future state that gives the maximum reward is chosen as the next state. This is performed for all the states. The value functions are stored in a vector and when agent moves to the next state the vector is updated with new value functions. The difference between two consecutive iterations is evaluated and if this difference is less than a chosen threshold value, the algorithm is considered to have reached convergence.

In the context of the frozen lake problem, the value iteration will update the value function for iteration# k+1 using the Bellman equation below.

$$v_{k+1}(s) = max\Sigma p(s', r|s, a)[r + \gamma v_k(s')$$

The frozen lake environment is simulated using the gym library in python. The probabilities of moving to s' are in built in the environment and are not manipulated here. The γ represents the discounted reward. It discounts the reward received in future.

Results

Frozen lake (4x4 grid)

Convergence for threshold: epsilon=exp(-20)

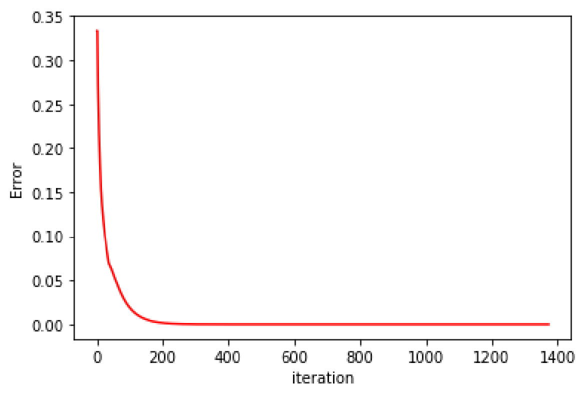

Figure 1. Convergence of the Value Iteration algorithm

The algorithm took a total of 1373 iterations to converge. The frozen lake grid space is shown below:

Optimal Solution from Value Iterations

Start	Frozen	Frozen	Frozen
Frozen	Hole	Frozen	Hole
Frozen	Frozen	Frozen	Hole
Hole	Frozen	Frozen	Goal

Down	Right	Down	Left
Down	Left	Down	Left
Right	Down	Down	Left
Left	Right	Right	Left

Figure 2. States of the frozen Grid Environment are shown on left and optimized policy obtained via value iterations is shown on the right.

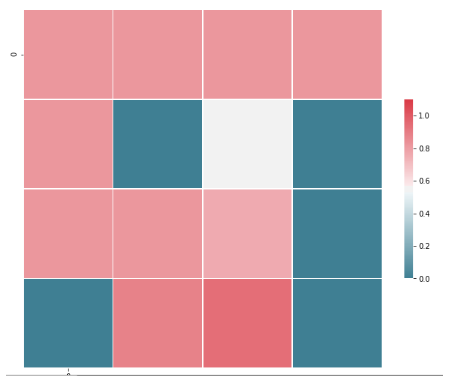

Figure 3. The heat map of the optimized policy

Figure 2 shows the optimized policy. Actions to be taken while being in each state are indicated at each step of the grid. Figure 3 shows the probability map that agent will be moved to a particular block.

Frozen Lake (8x8 grid)

Convergence for threshold: epsilon=exp(-20)

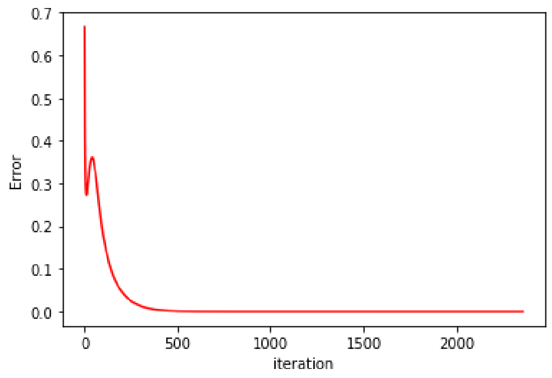

Figure 4. Convergence of the Value Iteration algorithm for 8x8 Frozen Lake problem

Start	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Frozen	Frozen	Hole	Frozen	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Frozen
Frozen	Hole	Hole	Frozen	Frozen	Frozen	Hole	Frozen
Frozen	Hole	Frozen	Frozen	Hole	Frozen	Hole	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Goal

Down	Down	Down	Down	Down	Down	Down	Down
Down	Down	Down	Right	Down	Down	Down	Down
Down	Down	Down	Left	Down	Right	Down	Down
Right	Right	Right	Right	Down	Left	Down	Down
Up	Up	Up	Left	Down	Down	Right	Down
Up	Left	Left	Right	Right	Down	Left	Down
Down	Left	Right	Up	Left	Down	Left	Down
Right	Right	Up	Left	Right	Right	Right	Left

Figure 5. Convergence of the Value Iteration algorithm for 8x8 Frozen Lake problem

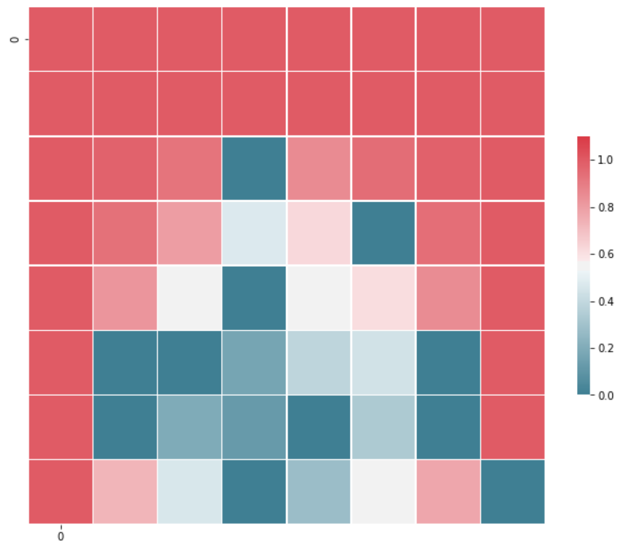

Figure 6. Convergence of the Value Iteration algorithm for 8x8 Frozen Lake problem

The Value iteration took approximately 2357 steps to converge. Figure 5 shows the policy obtained at the convergence of the value iteration. In figure 6 we see the probability map of choosing a particular state. It can be seen that steps near the holes have probabilities closer to zero, while steps farther from holes and frozen have probability ~1.

Solving using Policy Iteration

Theory

In policy iteration we start with a random policy and evaluate the value function for all states for this policy using the equation below

$$V(s) = \Sigma(p(s', r|s, \Pi(s))[r + \gamma V(s')]$$

We continue doing this until the difference between the consecutive value functions is less than a threshold. We then implement the policy improvement step using the equation below

$$\mathit{\Pi}(a) = argmax\Sigma(p(s', r|s, \Pi(s))[r + \gamma V(s')]$$

We then use this new policy to evaluate value function and then come back to improve the policy. This process continues until two consecutive policies are same. The parameter γ has been set to 0.9 for all the calculations.

Results

Frozen lake (4x4 grid)

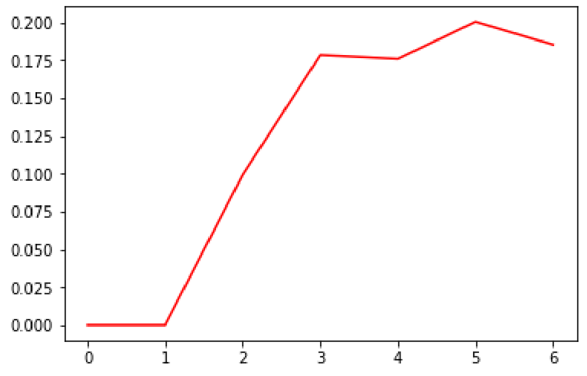
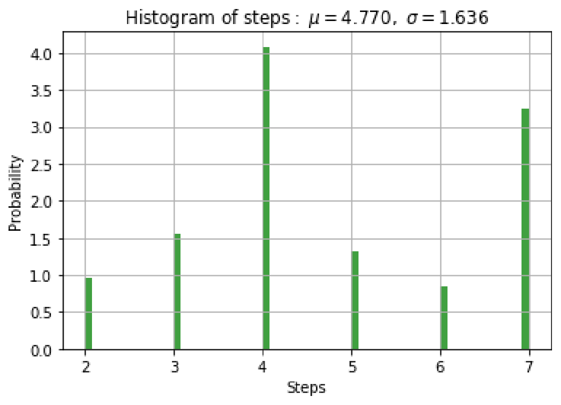

Figure 7. Mean Rewards at each policy iteration step (left) and histogram of number of steps taken by the policy iteration algorithm to converge.

The results show that the policy iteration converged after ~5 steps for a 4x4 frozen lake problem. Since the choice of initial policy is at random, we simulated this to generate a distribution (Figure 7, right). The policy converged to the same optimal policy as value iteration (Figure 2).

Frozen Lake (8x8 grid)

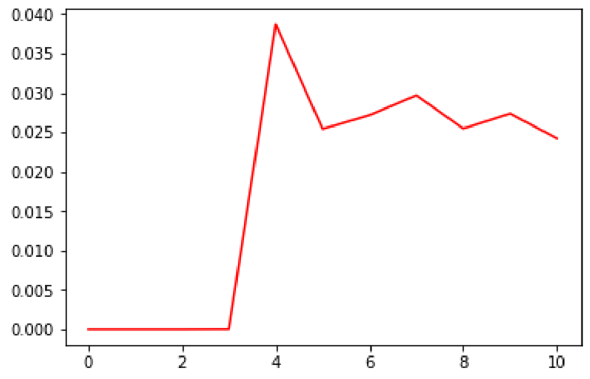
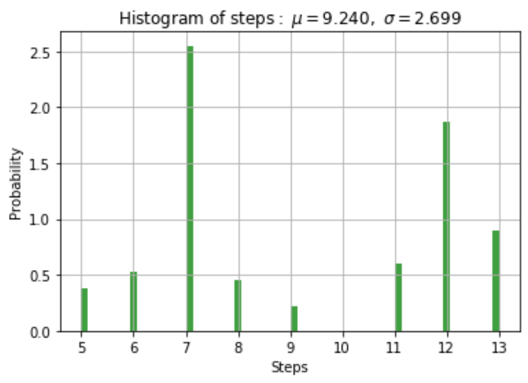

Figure 7. Mean Rewards at each policy iteration step (left) and histogram of number of steps taken by the policy iteration algorithm to converge.

The average number of steps required to arrive at an optimal policy for a 8x8 problem were 9. The final optimal policy was similar to that obtained at the convergence of Value Iteration. Please refer to Figure 5.

Policy Iteration and Value Iteration- A comparison

Looking at the algorithms for both Policy Iteration and Value iteration, they look pretty similar. The policy iteration algorithm differs from the Value iteration algorithm in the sense that in the latter.

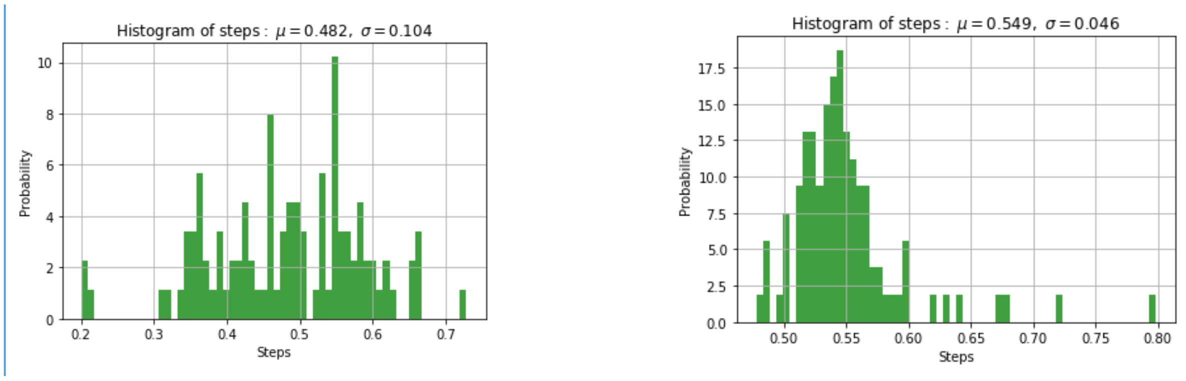

Figure 8. Histogram of time taken for convergence for a 4x4 grid for policy iteration algorithm (left) and value iteration algorithm (right).

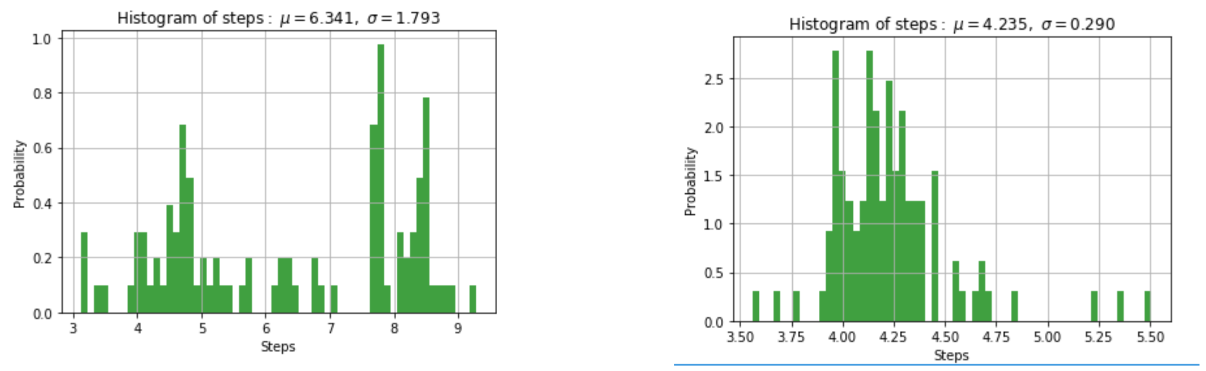

Figure 9. Histogram of time taken for convergence for a 8x8 grid for policy iteration algorithm (left) and value iteration algorithm (right).

Figure 8 shows the time distribution for convergence of the policy iteration algorithm (left) and value iteration (right) for a 4x4 frozen lake problem. The average time for convergence for the policy iteration for this case is ~0.48 seconds and for the value iteration this was 0.55 seconds. The time for convergence for value iteration was slightly greater than the policy iteration. Considering the standard deviation of the distribution we can say that these timings are similar. When we increased the number of states to 8x8 (Figure9), the time of convergence increased to ~6.3 seconds for policy iteration and ~4.2 seconds for value iteration. Again if we consider the standard deviations, we can say that the time of convergence for value iteration and policy iteration are similar. Theoretically, policy iteration will tend to converge faster as compared to value iteration because value iteration's convergence is based on a threshold value and difference between two consecutive value functions. However in the policy iterations we evaluate value function and then perform policy improvement. When two consecutive policies are same, the policy iteration terminates. It is possible that two different policies which are optimal have similar value functions and hence policy iteration can converge earlier. In value iteration it is bound to the threshold value that is specified for convergence. In the current scenario, reward is received only if the agent reaches the end goal. It makes sense then that the Value Iteration and the Policy iteration converges around the same time because no matter there are multiple optimal paths, value functions will be same during value iteration and policy iteration. Increasing number of states will have no impact either.

Applying Q learning algorithm

In this algorithm, the Q values are computed based on current action and state. These Q values include the reward based on the current state and the reward when certain action is taken. When a new action is taken, the Q values are updated using the equation below:

$$Q(state, action) < -(1 - \alpha)Q(state, action) + \alpha(reward + \gamma \max(Q(next\ state, all\ actions))$$

In the above equation α is the learning rate and γ is the discount factor for future rewards. We performed Q-learning algorithm on the both the 4x4 and 8x8 frozen lake problems. We explored using different learning rates (alphas) and plotted the mean reward and with number of iterations. The algorithm was ran and a Q table was generated for each problem.

Q table for 4x4 frozen lake problem

Start (S0)	Frozen (S1)	Frozen (S2)	Frozen (S3)
Frozen (S4)	Hole (S5)	Frozen (S6)	Hole (S7)
Frozen (S8)	Frozen (S9)	Frozen (S10)	Hole (S12)
Hole (S13)	Frozen (S14)	Frozen (S15)	Goal (S16)

States/Actions	Left	Down	Right	Up
S0	0.055	0.002	0.0019	0.0021
S1	0.001	0.00043	0.00018	0.057
S2	0.019	0.000718	0.000465	0.001
S3	0.00052	0.00037	0.000449	0.001
S4	0.105	0.00065	0.00199	0.0012
S5	0	0	0	0
S6	2.6e-5	4.36e-6	5.7e-3	4.19e-5
S7	0	0	0	0
S8	3.13e-03	1.29e-03	1.933e-03	2.55e-01
S9	3.66 e-03	0.176	0	0.0015
S10	2.16e-01	1.423 e-03	0	2.19 e-03
S11	0	0	0	0
S12	0	0	0	0
S13	0	0	0.21	0.00145
S14	0	0	0	0.63
S15	0	0	0	0

The Q values for each state for each action are show in the table. For some of the states and action combination the values doesn’t seem to be optimal. For example when agent is at S4, the Q value for going to right is slightly greater than the other values. To the right of S4 is the hole and hence the Q value should have been the minimum for this action as compared to other actions. We can further see that the

For exploration we did explore different learning rates (0.1, 0.01 and 0.001) but average reward was obtained to be 0 in all cases. Hence results are not shown here. We further analyzed the time of convergence for the algorithm by simulating the Q-learning 30 times and taking the average of time durations. Average duration was observed to be ~13 seconds.

We further ran this algorithm on the 8x8 grid and results are displayed in the table below:

Q table for 8x8 frozen lake problem

Start	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Frozen
Frozen	Frozen	Frozen	Frozen	Frozen	Hole	Frozen	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Frozen
Frozen	Hole	Hole	Frozen	Frozen	Frozen	Hole	Frozen
Frozen	Hole	Frozen	Frozen	Hole	Frozen	Hole	Frozen
Frozen	Frozen	Frozen	Hole	Frozen	Frozen	Frozen	Goal

State/Actions	Left	Down	Right	Up
S0	6.4 e-4	6.9 e-4	1.8e-3	7.8e-4
S1	8.0e-4	5.5e-4	1.9e-3	9.1e-4
S2	4.7e-4	2.71e-3	5.17e-4	7.7e-4
S3	5.5e-4	6.12e-4	7.24e-3	5.67e-4
S4	1.02e-3	1.06e-2	1.15e-3	1.49e-3
S5	5.25e-4	1.29e-3	1.30e-2	1.25e-3
S6	1e-3	1.65e-2	0	0
S7	2.3e-2	0	0	0
S8	6.8e-4	8.15e-4	6.29e-4	2.17e-3
S9	4.4e-4	3.2e-4	1.8e-3	7.2e-4
S10	8.18e-4	3.47e-4	3e-4	2.26e-3
S11	4.5e-4	4.6e-4	1.47e-4	4.07e-3
S12	6.98e-4	5.75e-4	5.34e-3	7.65e-4
S13	0	1.17e-3	0	2.6e-2
S14	6.2e-4	4.16e-4	1.8e-2	0
S15	0	2.48e-2	0	0
S16	4.96e-4	5.95e-4	2.09e-3	6.15e-4
S17	5.95e-4	3.3e-4	3.4e-4	2.3e-3
S18	3.2e-3	7.7e-5	4.83e-5	5.02e-5
S19	0	0	0	0
S20	2.5e-4	0	5.91e-3	2.19e-4
S21	6.36e-4	7.48e-4	2.6e-4	1.9e-2
S22				
S52	3.24e-5	4.04e-6	4.6e-6	9.5e-6
S53	0	0	0	0
S54	3.38e-5	2.98e-7	1.62e-1	1.18e-4

State/Actions	Left	Down	Right	Up
S55	0	0	0	0
S56	0	0	8.1e-1	0
S57	1.15e-4	8.68e-5	9.27e-5	1.14e-4
S58	9.8e-5	8.9e-5	5.9e-5	3.6e-5
S59	5.96e-5	6.58e-5	4.24e-5	5.06e-5
S60	0	0	0	0
S61	0	0	2.51e-2	7.83e-5
S62	2.19e-4	0	2.5e-4	5.49e-1
S63	0	4.8e-5	8.5e-5	2.95e-1
S64	0	0	0	0

The entire Q table has not been shown here due to space limitation, we have shown values for states (S0-S22 and S52-S64). Note that states are counted from Start as 0th state and then towards right (In a similar fashion as that of 4x4 grid). We can see that the Q values at the positions where there is a hole are zeros indicating that the agent has learned to not move to these locations. Further we evaluated time of convergence by simulating the Q learning algorithm 30 times. The average of these times was taken and was found to be approximately 23 seconds. For exploration purpose we varied the learning rates and tried to plot the Reward performance with iterations and we obtained a zero rewards hence the results have not been shown here. This is because the code is terminated as soon as the goal stage is reached and the reward is never calculated. We can see in the Q table for both 4x4 gird as well as 8x8 grid that the Q value of the goal state is 0.

Comparison of Q-learning with Value Iteration and Policy Iteration

The Q learning algorithm did not converge to the correct solution. In some cases we observed that the Q values were slightly misleading. The convergence time summary for the three learning algorithms are summarized in the table below:

Version of Frozen Lake problems	Value Iteration time (seconds)	Policy Iteration time (seconds)	Q-learning time (seconds)
4x4 grid problem	0.549	0.482	13
8x8 grid problem	4.2	6.3	23

We can see that the Q-learning algorithm took significant amount of time as compared to the policy iteration and value iteration algorithms. Moreover the Q table stores the Q values in memory. This further ads to the space complexity of the algorithm. Overall Policy iteration and Value iteration performed similar.

The theoretical concepts summarized in this assignment are based on reading from the following book. Update rules for the algorithms have also been taken from the book listed in the reference below:

References

Reinforcement Learning: An Introduction, 2nd Edition, *Richard S. Sutton and Andrew G Barto*