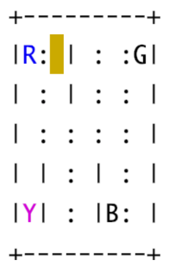
Markov Decision Processes

The goal of this assignment is to explore and compare how different algorithms perform for solving a markov decision process. For this assignment we used two environments taxi and forest management for comparison.

Taxi-v3 environment:
The problem that we are solving is the optimization of a self-driving taxi using reinforcement learning algorithms. The objective of this problem is to pick up and drop off passengers efficiently and safely in a simulated environment. For this, we used the Taxi-v3 environment provided in the OpenAI Gym.

Environment Description:
There are four designated locations in the grid world indicated by R(ed), G(reen), Y(ellow), and B(lue). In each episode, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger which marks the end of episode. The size of the grid is 5x5 therefore there are 500 discrete states in the environment since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations



```
+---------+
|R:█| : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

The filled square represents the taxi, which is yellow without a passenger and green with a passenger. The pipe ("|") represents a wall which the taxi cannot cross. R, G, Y, B are the possible pickup and destination locations. The blue letter represents the current passenger pick-up location, and the purple letter is the current destination

The action space of the environment consists of 6 actions 0: move south, 1: move north, 2: move east ,3: move west, 4: pick up passenger, 5: drop off passenger.

Rewards:
There is a default per-step reward of -1, except for delivering the passenger, which is +20, or executing "pickup" and "drop-off" actions illegally, which is -10.

This environment introduces stochasticity in the pick-up and drop-off position of passengers as well as taxi location on the grid thus it simulates simpler version the taxi dispatch problem. Finding optimal solution to this problem could help in optimization of fleet for cab aggregators, route optimization for delivery companies. Thus, this environment embarks a challenging problem to solve.

Policy Iteration:
In policy iteration, we evaluate a given policy π using Bellman Expectation Equation. The Bellman Expectation equation tells how much reward our agent is going to get following a policy π and is defined as:

$$v_{k+1}(s) = \sum_{a \epsilon A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a \, v_k(s') \right)$$

The policy iteration consists of two steps policy evaluation and policy improvement. First, we evaluate a policy say π using Bellman Expectation Equation as described above and then we improve the policy by acting greedily to the evaluated value function to find us an Optimal Policy. We are going to iterate this process until we get our true value function. The two steps of policy iteration algorithm i.e. policy evaluation and policy improvement is given below:

Policy Evaluation

$$v_{\pi(s)} = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

Policy improvement:
Act greedily to the evaluated value function which gives a policy better than the previous one

$$\pi' = greedy(v_\pi)$$

We iterate these two processes until it **converges to Optimal Value Function** and **Optimal Policy.**

Value Iteration:
For value Iteration, we used Bellman Optimal Equation which is given by

$$v_*(s) \leftarrow \max_{a \epsilon A} R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_*(s')$$

We start off with some random Value Function (say zero) and then we iterate this process using Bellman Optimal Equation. That is, we compute the new value function of a state by using the values of the previous iteration in the Bellman Optimal Equation. We iterate this process until it converges to Optimal Value Function. We then use compute the policy from optimal value function.

Fig 1: Tuning gamma for policy iteration

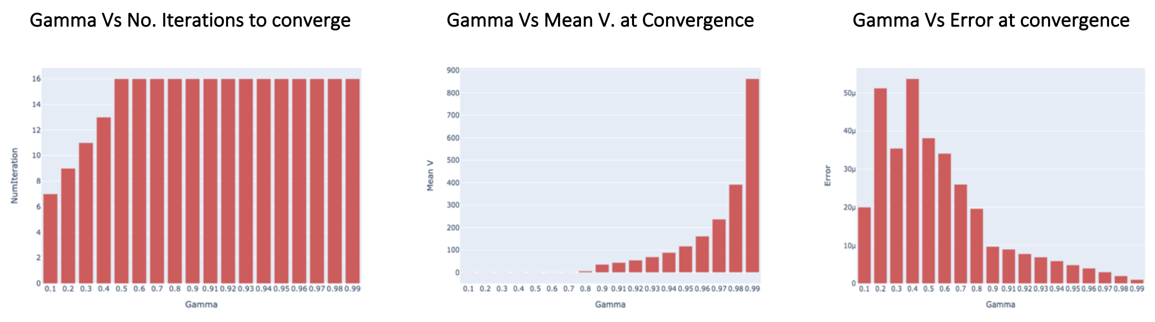
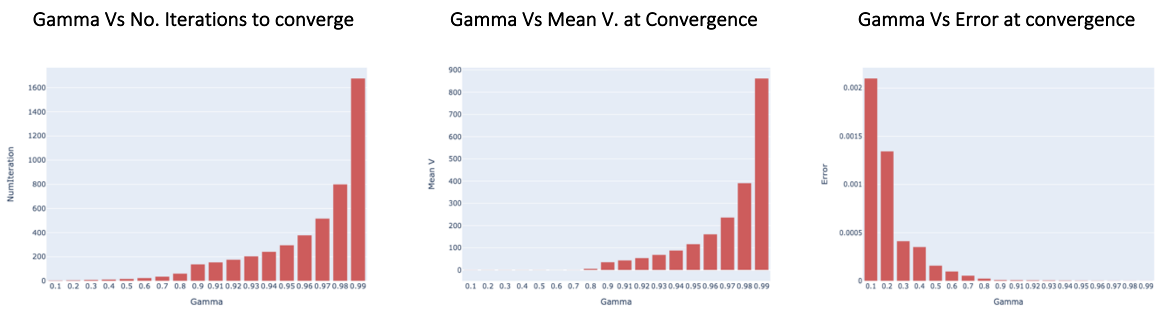


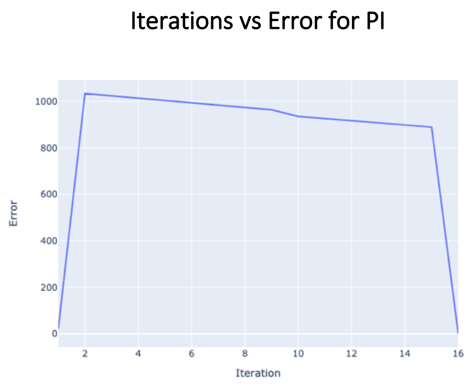
Fig 2: Tuning gamma for value iteration



The optimal discount factor for policy iteration and value iteration was found to be 0.99 with mean value function around 860. Both the algorithms didn't converge when gamma was 1. The error is defined by maximum of absolute difference between value function of current iteration vs previous iteration. The policy iteration 16 iterations only to converge whereas value iteration took 693 iteration to converge keeping converge criteria of 0.01. Table 1 summarises the comparison between policy and value iteration. It was observed that both algorithms converge to the same policy.

	Gamma	Mean V	No. of Iterations	Execution Time
Policy Iteration	0.99	860	16	0.522
Value Iteration	9.99	860	693	0.528

Fig. 2. Error plateaued shows convergence of PI and VI as iteration increases



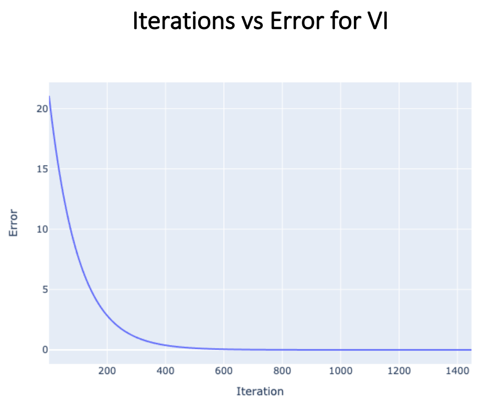
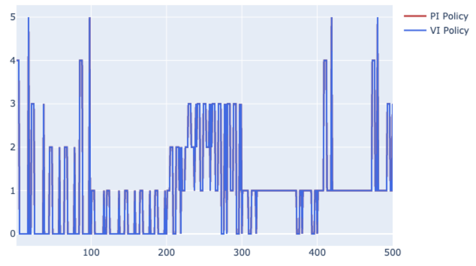
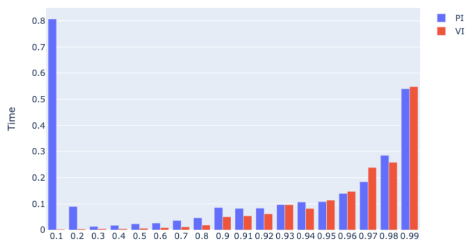


Figure 3.

Policies from both policy iteration and value iteration converges to the same values as indicated in this figure



Time Taken by PI and VI for different values of Gamma



Q-learning

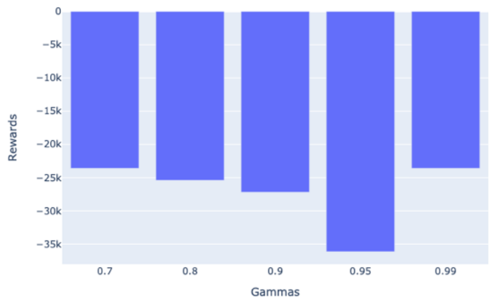
Unlike policy iteration and value iteration, Q-learning is an off-policy reinforcement learning algorithm which learns about the environment by trials and hence it doesn't need state transition table. Q-learning updates the value function based on an Bellman equation. The Q-learning algorithm is given by below algorithm.

```
Algorithm 1: Epsilon-Greedy Q-Learning Algorithm
Data: $\alpha$: learning rate, $\gamma$: discount factor, $\epsilon$: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated
        optimal policy $\pi*$
/* Initialization                                              */
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) $\leftarrow$ 0;
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                              */
for each episode do
    /* Initialize state S, usually by resetting the
       environment                                             */
    Initialize state S;
    for each step in episode do
        do
            /* Choose action A from S using epsilon-greedy
               policy derived from Q                           */
            A $\leftarrow$ SELECT-ACTION(Q, S, $\epsilon$);
            Take action A, then observe reward R and next state S';
            Q(S, A) $\leftarrow$ Q(S, A) + $\alpha$ [ R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
            S $\leftarrow$ S';
        while S is not terminal;
    end
end
```

We applied Q-learning with ϵ-greedy strategy. We started with the ϵ value of 1.0 which decreased value after each episode by factor of 0.99 keeping minimum to 0.1. This strategy enabled the agent to maximize exploration in the initial phrases and exploit the learning in the later iterations. We also used this method for the learning rate where we started with 0.1, decreased learning rate by 0.99 after every episode till it reached 0.001. We tried tuning discount factor with different values [0.7, 0.8, 0.9, 0.95, 0.,99] and compare performance of agents on 100 episodes. We define penalty of value 1 when taxi reached incorrect location to pick up passenger or drop-off at incorrect location. We then plotted sum of rewards and penalties for different values of gamma. Based on this, we decided to use gamma = 0.99. We also compared the performance of value iteration and policy iteration, both the algorithms performed better than Q-learning with cumulative rewards of -20000 and penalties of 0. This may be due to the fact that policy iteration and value iteration both are model based algorithms where they know have knowledge of transition probabilities of states of the environment whereas Q-learning algorithm being a model free algorithm tries to learn about environment by trials.

Fig 4: Q-learning rewards and penalties for different values of gamma

Gammas Vs Rewards



Gammas vs Penalties

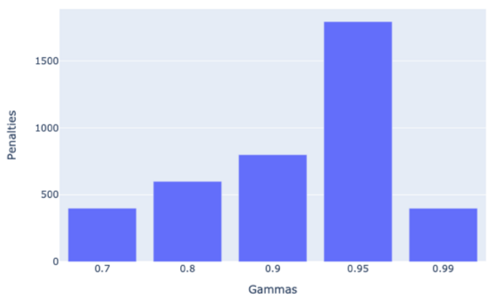
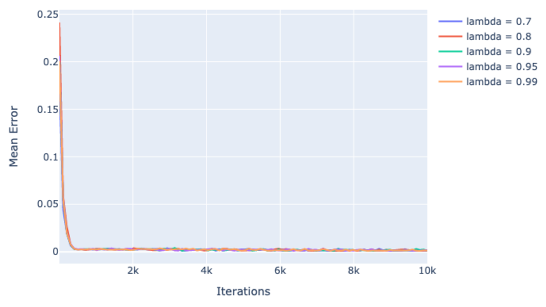
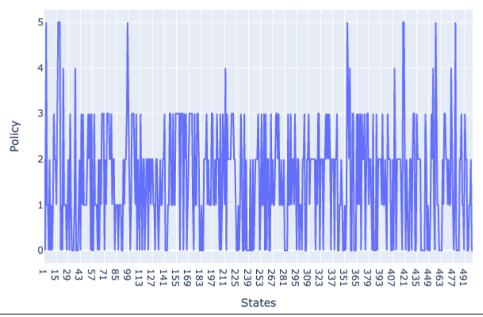


Fig 5

Mean Error on 100 consequtive episodes vs Episodes	Learned Policy by Q-learning agent on Taxi v3 env
	

Forest Management Problem:

A forest is managed by two actions 'Wait' and 'Cut'. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest.

Here is how the problem is modelled. Let {0, 1 . . . S-1 } be the states of the forest with S-1 being the oldest. Let 'Wait' be action 0 and 'Cut' be action 1. After a fire, the forest is in the youngest state, that is state 0. The transition matrix P and Reward matrix R of the problem can then be defined as follows::

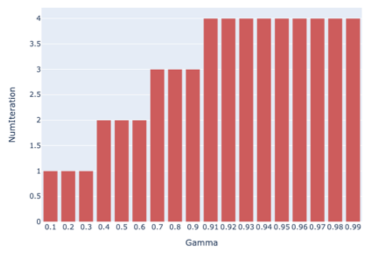
$$
P[0, :, :] = \left|\begin{array}{ccccccc}
p & 1-p & 0 & . & . & . & 0 \\
. & 0 & 1-p & & & & . \\
. & . & 0 & . & & & . \\
. & . & . & & . & & . \\
. & . & . & & & . & . \\
. & . & . & & & & 1-p \\
p & 0 & 0 & 0 & . & 0 & 1-p
\end{array}\right|
\qquad
R[:, 0] = \left|\begin{array}{c}
0 \\ . \\ . \\ . \\ . \\ 0 \\ R1
\end{array}\right|
$$

$$
P[1, :, :] = \left|\begin{array}{ccccccc}
1 & 0 & . & . & . & . & 0 \\
. & . & . & & & & . \\
. & . & . & . & & & . \\
. & . & . & & . & & . \\
. & . & . & & & . & . \\
. & . & . & & & & . \\
1 & 0 & . & . & . & . & 0
\end{array}\right|
\qquad
R[:, 1] = \left|\begin{array}{c}
0 \\ . \\ . \\ . \\ R2
\end{array}\right|
$$

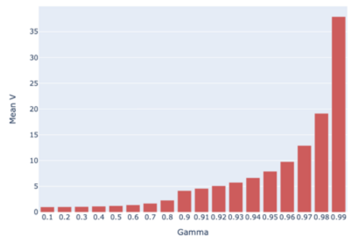
We applied policy iteration and value iteration on the forest problem with 1000 states with varying discount factor. As expected, policy iteration took less number of iteration to converge as compared to value iteration. The discount factor of 0.99 was found to be optimal value for both value iteration and policy iteration. Also, both the algorithms came up with similar policy however, there is significant difference between the value functions generated by both the algorithms. The policy iteration algorithm projected higher values of value function as compared to value iteration which can be visualized in fig 9.

Fig 6: Tuning gamma for policy iteration for Forest management problem

Gamma Vs No. Iterations to converge



Gamma Vs Mean V. at Convergence



Gamma Vs Error at convergence

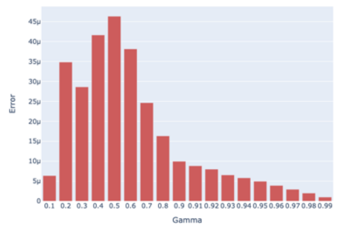
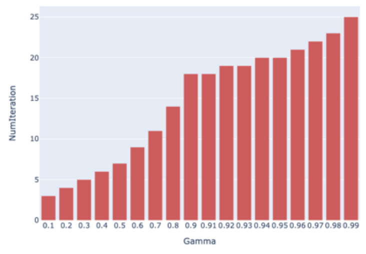
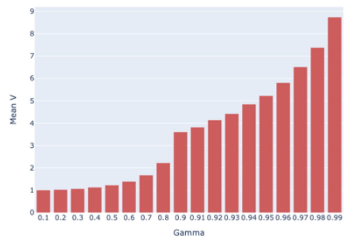


Fig 7: Tuning gamma for value iteration for Forest management problem

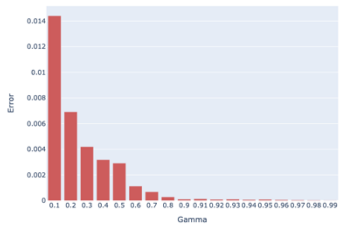
Gamma Vs No. Iterations to converge



Gamma Vs Mean V. at Convergence



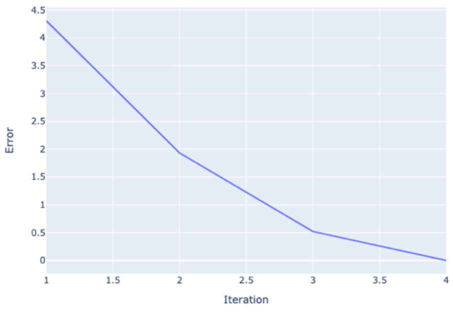
Gamma Vs Error at convergence



It is evident from fig 8. that both algorithms converged as iterations progressed as error approached 0.

Fig 8: Error plateaued in below figure indicate convergence of PI and VI as iteration progressed

Iterations vs Error for PI



Iterations vs Error for VI

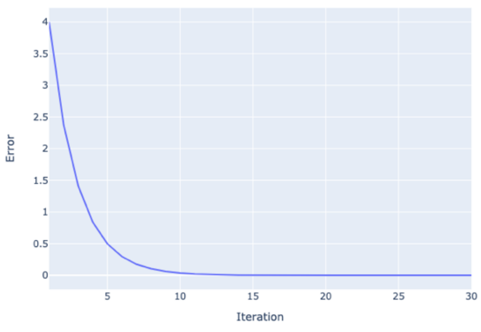
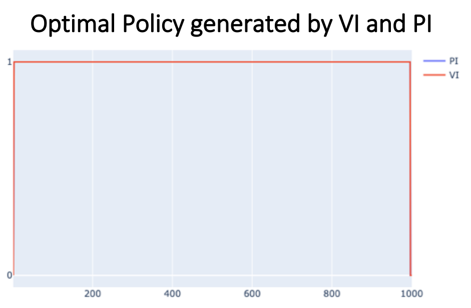
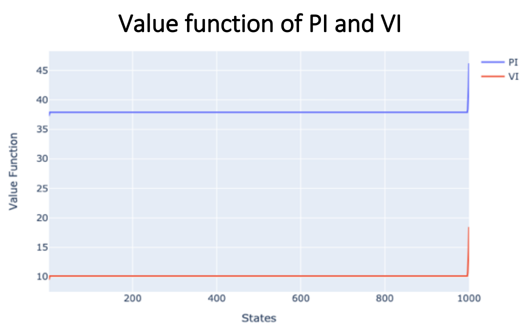


Fig 9: Optimal Policy and Value function generated by VI and PI

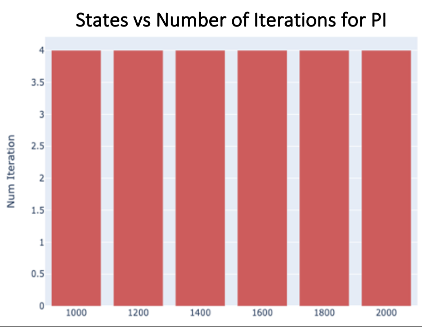




Effect of number of states:

The number of states didn't have any effect on the no. of iteration the policy iteration and value iteration algorithms took to converge. One interesting observation was that the agent followed a constant policy where agents policy was 0 in the first and last states whereas it is 1 in the rest of the states. Owing to constant behaviour it didn't matter to the agent how many states were there in the environment since optimal policy followed a constant policy.

Fig 10: Number of iterations to converge PI and VI for different values of states



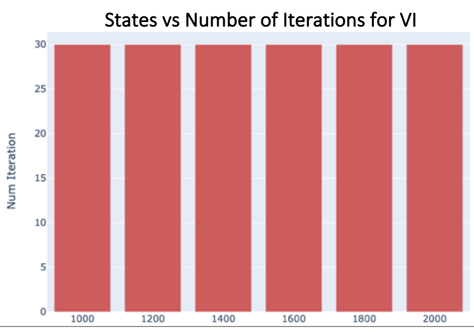
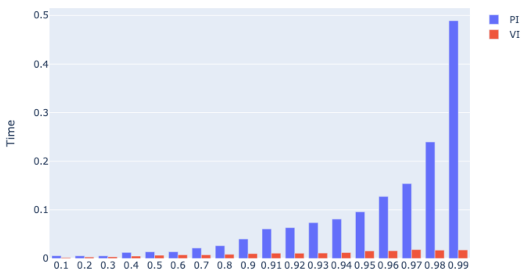


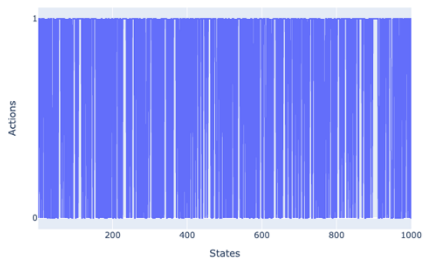
Fig 11: Time Taken by VI and PI for different values of Gamma



Q-learning on Forest Management Problem:

We applied Q-learning on forest management problem with epsilon greedy policy, we started with epsilon=0.5, and ends at epsilon_min=0.1 with decay factor of epsilon_decay=0.99. We also tried to tune α starting with alpha=0.1 and gradually decaying alpha with a factor of alpha_decay=0.99 till alpha_min=0.001. We compare the performance of policy obtained from Q-learning to the policy obtained using policy iteration. We defined policy obtained from value iteration as optimal policy. We also defined penalty as sum of actions where optimal policy and given policy doesn't match. We tried to find optimal value of discount factor by comparing penalties obtained by comparing policy obtained from Q-learning algorithm with optimal policy. We found that Q-learning works horribly of forest management data when trained for 10000 episode where penalty was minimum 95% irrespective of discount factor. However as we increased episodes size, penalty decreased. For 5000000 penalty was 347. We believe that if we run Q-learning with more episodes, penalty would have been further decreased. It implied that as number of states increased, the no. of episodes it required for Q-learning also increased. Fig. 12 represents the policy generated by Q-learning algorithm for discount factor 0.99 where penalty was least.

Fig 12: Policy generated by Q-learning for Forest management problem



Conclusion:

- Policy generated by Policy Iteration and Value Iteration are identical in both of the MDPs that we have used.
- The policy iteration algorithm takes less number of iterations to converge.
- As the no. of states increase the no. of episodes it required to converge Q-learning also increased.
- In both the MDPs, value iteration was faster than policy iteration.