Planning Taxi Domain

# Part A: Computing Policies

## Taxi Domains

* **State space:** We have defined it as a list of 5-element tuple (x, y, status, a, b), in which (x,y) represent coordinate of taxi, (a,b) represent coordinate of passenger and status is True or False which represent whether taxi is occupied by passenger or not. **Goal state** is reached when taxi has dropped the passenger at destination which means (goal[0],goal[1],False,goal[0],goal[1]), where goal is the drop location. Also, when status is True then the states contain only those tuples for which (x,y)=(a,b).

Hence size of states space = 5\*5\*5\*5 (for status=False) + 5\*5 (for status=True) = 650

* **Action space:** There are 6 actions possible [‘N’,’S’,’E’,’W’,’Pick’,’Drop’]. At every possible states(other than the goal state, in which episode ends and no further action takes place) all six actions can be performed, whether it will result in something or not doesn’t matter.
* **Transition model:** T(s,a,s’) <- transition[state][action][result], it stores probability of getting into state s’(result) from s(state) over taking action a(action). As this table is very large so in code we have stored only those T(s,a,s’) values which are non zero.
* **Reward model:** R(s,a,s’) <- reward[state][action][result], it stores reward for a(action) when applied from state s(initial state) and reaches on state s’(resultant state). Value of reward is as per stated in question. As this table is very large so only those R(s,a,s’) are stored in code for which T(s,a,s’) are non-zero.

Simulator: Implementation of different things like next state based on current state and current action, stochastic effect of actions, rewards for each action. A class MDP is made in code which have functions to support these stochastic effects of applications. Also it stored these states, actions, transition model and reward model.

Instance: An instance of the problem is defined uniquely by the grid(possible depots, its size and the walls it contains) and the destination depot on which the passenger has to be dropped finally to end the episode. On this instance, the initial location for taxi can be any grid while for the passenger it can be any depot.

## Value Iteration

Implemented value iteration by using the formula

V\_k+1(s) <- max over all a(actions available on s) (∑(over states s’) T(s,a,s’) \* [R(s,a,s’) + Y.V\_k(s’)] ), Y is discount factor. (in code V\_k+1=cvfn, V\_k=pvfn).

Termination of iteration when max-norm error becomes less than epsilon.

1. For discount factor = 0.9 and epsilon = 0.01, number of iterations required was 22.

The value of epsilon chosen is 0.01 because for discount factor of gamma, the value of reward +20 carried over to upto 20 states look ahead(going to passenger than going to drop location may take 10+10 actions in best case stochastic effects in a 5\*5 grid where these locations are farthest apart from each other). So the effect of +20 reward would reduce to (gamma^20)\*20. For gamma =0.9 this value would be approximately 2.43. But with stochastic effects this would reduce even further and thus it is better to have a even lower epsilon (< 2.43\*0.85) to account for these stochastic effects. Also, we have to run these value iterations for different discount factors and through above mentioned approximate calculations it can be observed that epsilon=0.01 holds good enough for gamma> 0.7.

1. Discount factor = 0.01 => 3 iteration

Discount factor = 0.10 => 4 iteration

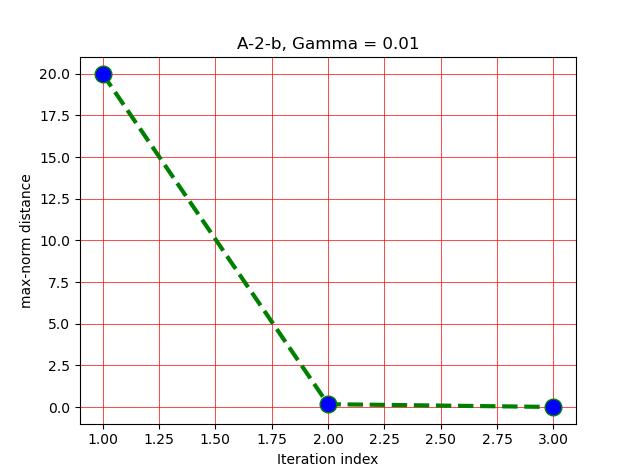
Discount factor = 0.50 => 9 iteration

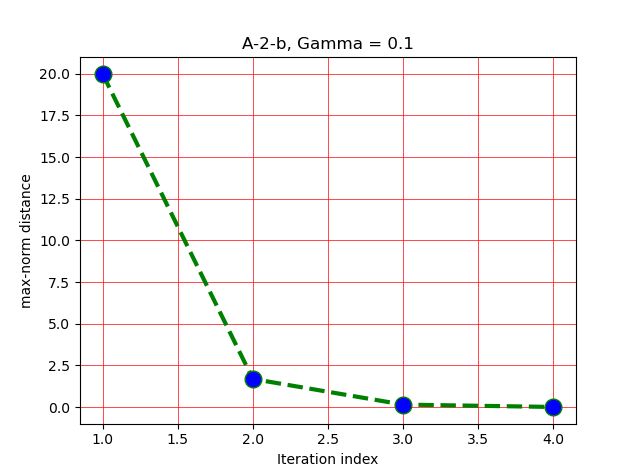
Discount factor = 0.80 => 16 iteration

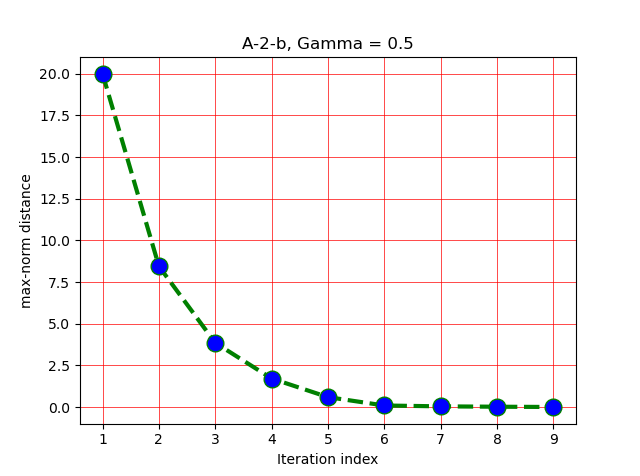
Discount factor = 0.99 => 32 iteration

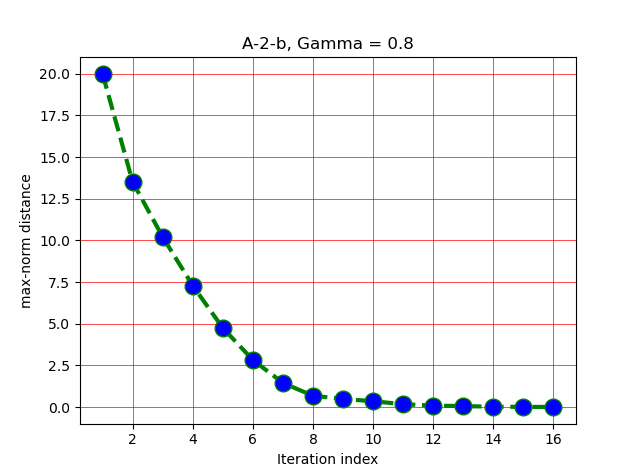
\*\* Epsilon was fixed to 0.01 for all above runs.

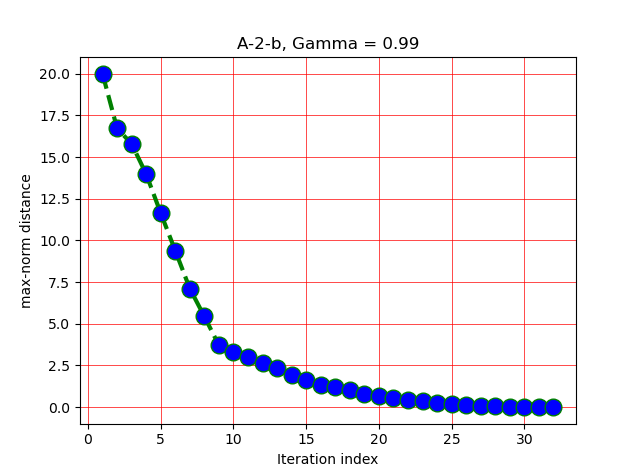
\*\* PLOT the graph b/w iteration index(x-axis) and max-norm(y-axis) for each and describe observation. \*\*









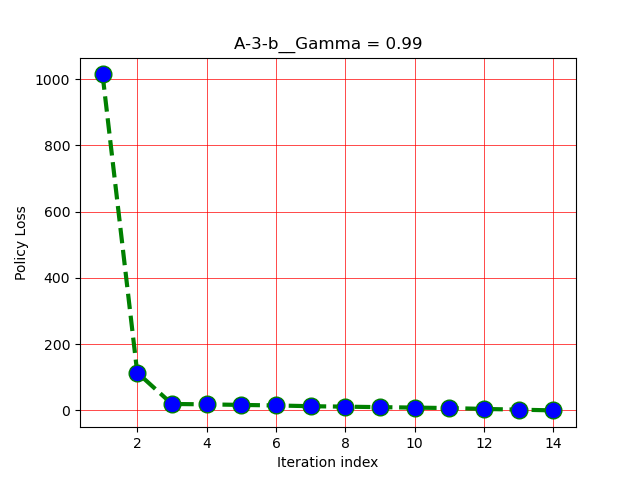
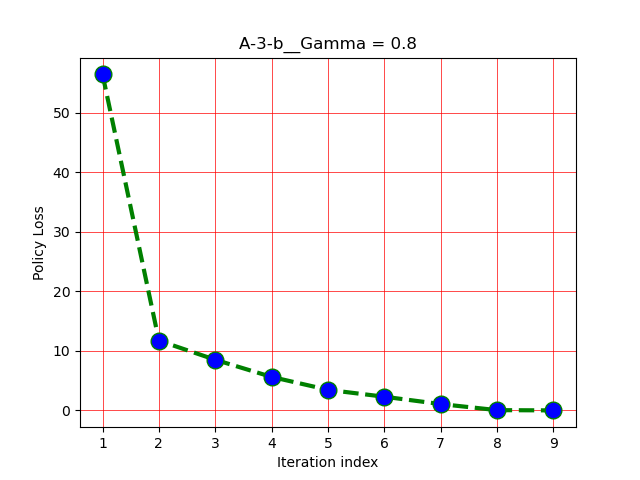
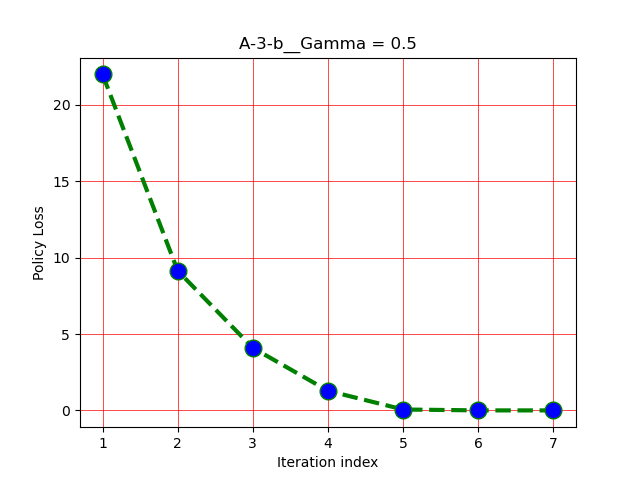
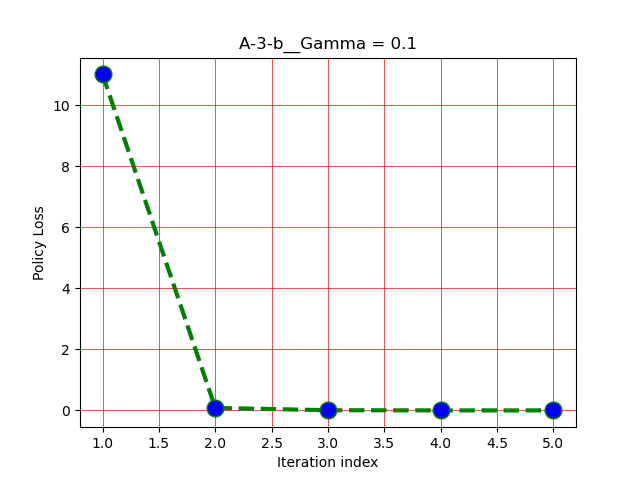
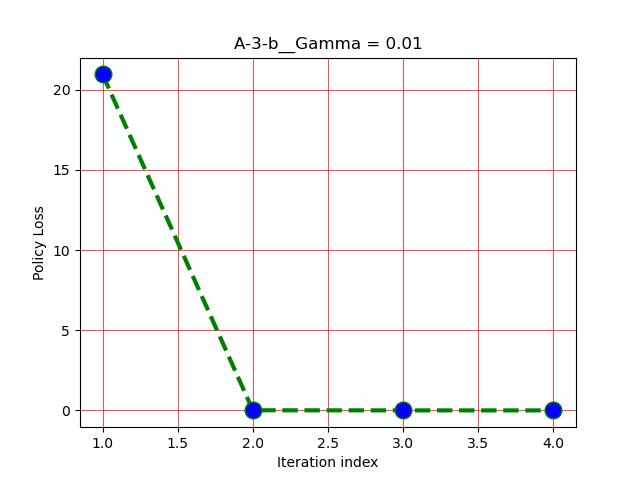


From above graphs we can see that as discount increases max-norm decrease slowly.

This happens because as we increase the value of discount factor then there is a larger effect of future rewards on the current value function and hence the change in value function is larger which implies that the max-norm error would be large and hence it converges slowly to the optimal value.

1. For the first time both are giving the same policy but if we repeat algorithm for many time we will find higher discount factor gives better policy.

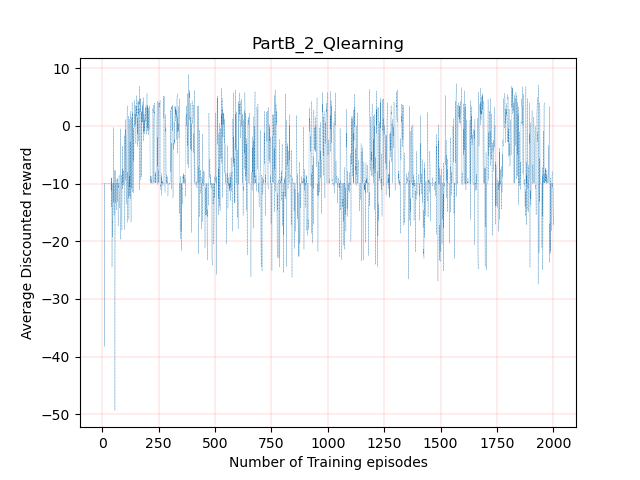
## Policy Iteration

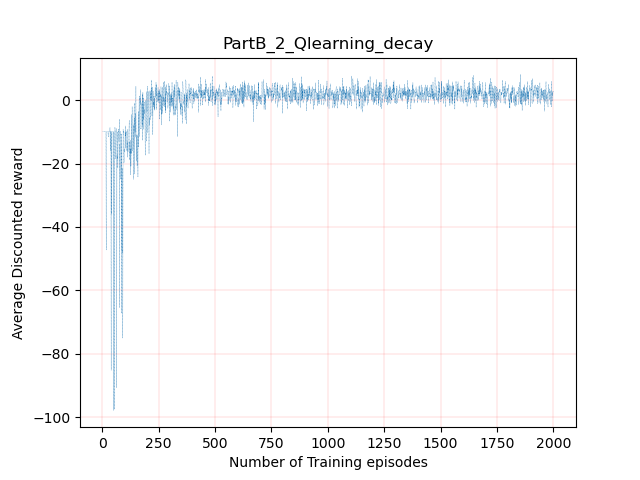


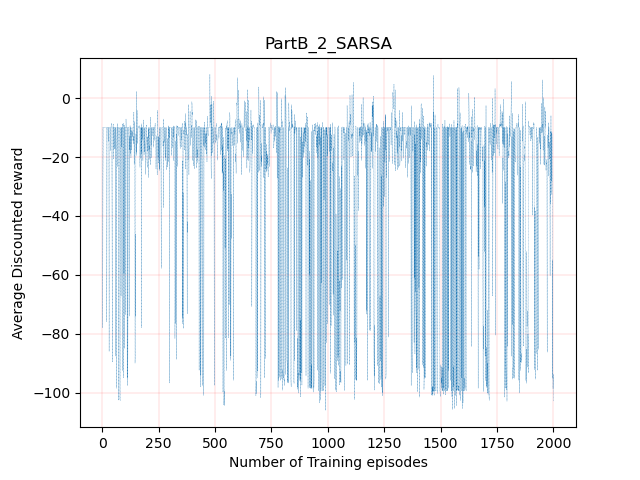
Lower the discount factor faster it converge to zero policy loss.

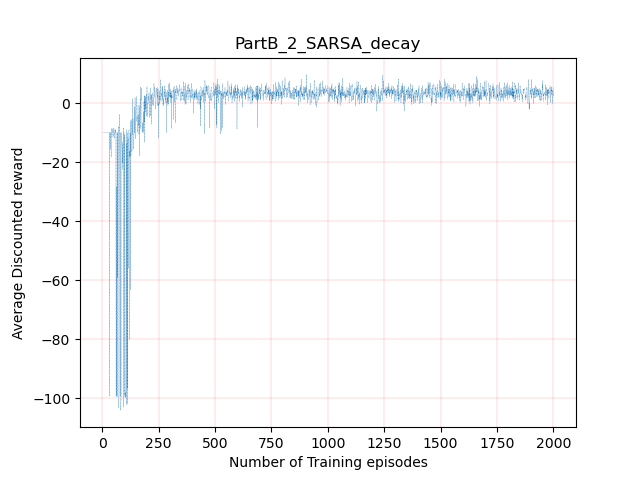
# Part B: Incorporating Learning

## 2)









## 4)

