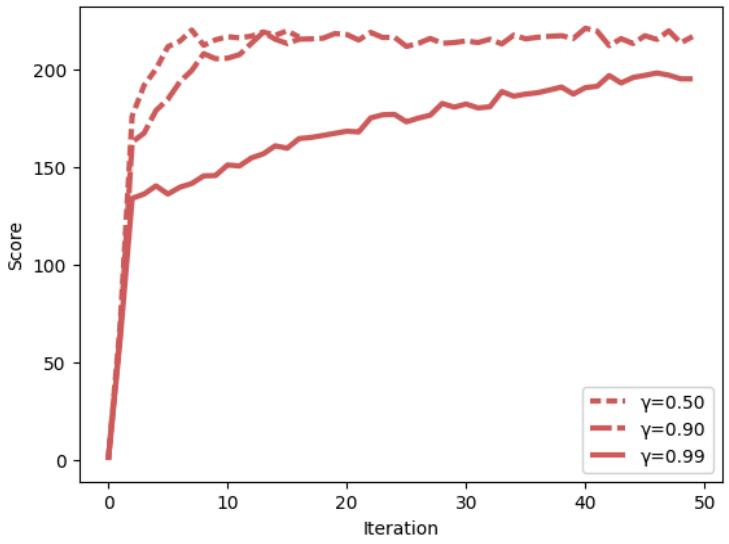
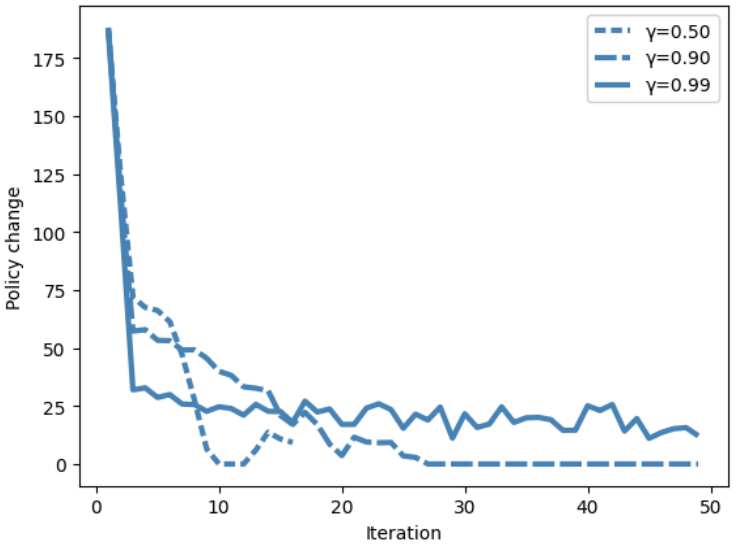
Reinforcement Learning Report
Reinforcement Learning is where an autonomous agent takes appropriate actions by sensing the environment and learns simultaneously to achieve its goals. In RL, the agent learns by trial and error (Q-learning algorithm), receiving feedback in the form of penalties based on the actions it takes. The problem here is agent should not only consider the immediate next reward associated with the state transitions rather the goal should be to maximise long-term rewards this can be done by considering all possible solutions and discounting back their rewards to the present state and this opens up a new world of MDPs and Q-learning solutions which can make agent better at solving real-world problems.

MDP:
1. Reaching Arm: The Reaching Arm can be modelled as MDP due to its sequential decision-making ability and the stochasticity involved in the environment which is very similar to Openai gym's Reacher [1]. The Reaching Arm can rotate in all possible directions restricted by one angular unit while joints in the arms help reach the target point by the effector. If the arm moves closer to the target then it receives the reward based on the distance between the effector and the target point. Since this is an infinite game once the effector reaches the target point robot receives the rewards while the target point is moved to another random new location. Hyperparameters for this are the number of joints (i.e. number of arm pieces minus one), the location of the target point, rewards for reaching the target point, and the angle between two arm pieces. A large enough angle should be required to have ideal learning for the robot (100 steps as the number of resolutions increases). Increasing the number of arms exponentially adds to the number of possible states because it gets multiplied by the angular resolution. This makes the MDP interesting as simple as it looks initially can be tedious and complex from computational resources and evaluation to reach the target point in combination with the states, angular resolution, and rewards. For the Assignment, I chose 2 arms with 20 angular resolutions and 6 target resolutions which counted for 14K states while another one with the configuration of 2 arms with a resolution of 60 and a target of 10 different random points which resulted in 360 Thousand states and 3.6 Million combinations of state-action pairs (results are present in the code).

Value Iteration: Value Iteration is a reinforcement learning algorithm used for solving MDPs like Forest Management and Reaching Arm. The main aim of VI is to find the optimal value function which represents the expected cumulative reward the agent can receive from a given state following the optimal policy. The value of each state can be defined as the maximum expected cumulative reward that can be received from the state onwards. VI is trying to optimize for value (cumulative reward) from the initial state to the final state. The final state can be defined as when the difference between the current value and the next value is less than the threshold or the maximum number of iterations has been reached. Since VI works iteratively, at convergence we can find the optimal policy by selecting the states that maximize the value function for each state.

VI for Reaching Arm: I ran VI on the reaching arm where 2 arms with an angular resolution of 20 and a target resolution of 6 with 14 Thousand states while setting discount factors as 0.5, 0.90 and 0.99.

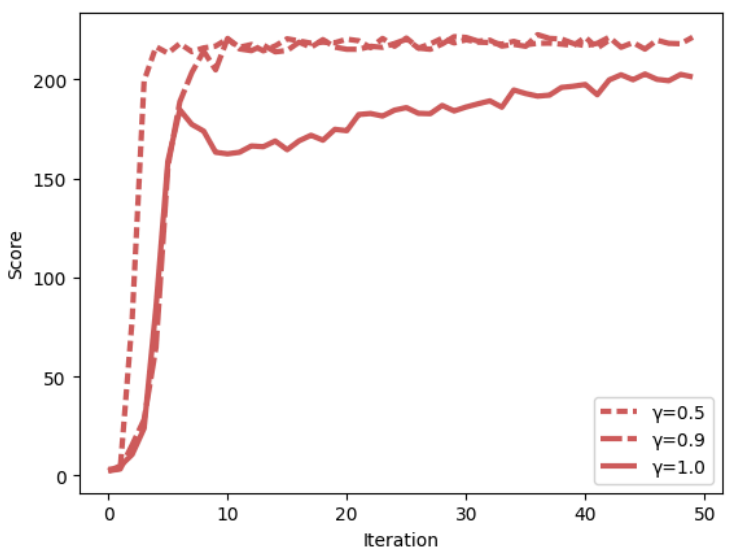


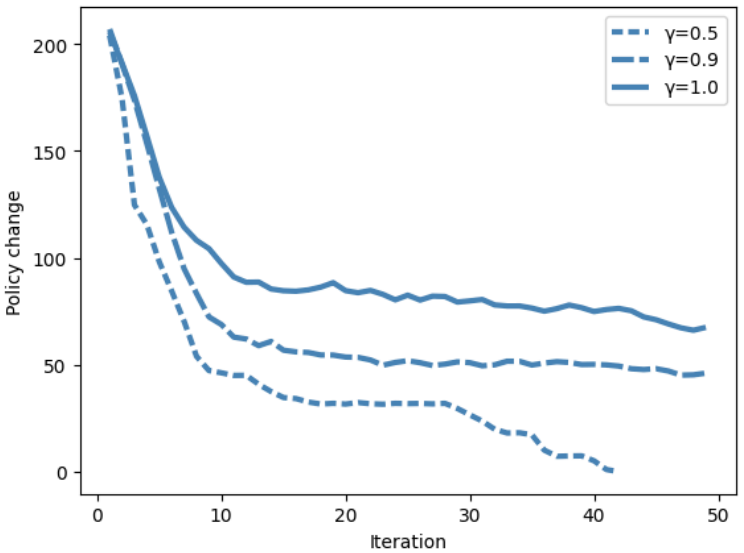


The initial expectation is that when the discount factor is high, policy should converge faster because a high discount factor emphasizes future rewards over immediate rewards. A high discount rate also encourages exploitation over exploration and since the agent values future rewards it exploits good actions with high rewards leading to quick convergence. So when we have a discount rate of 0.90 we are expecting the reaching arm to perform well as compared to 0.99 and 0.5 because 0.9 seems reasonable where convergence can be achieved quickly without overshooting and oscillating between different policies. As per the graph for VI, the discount factor of 0.5 leads to convergence in only 16 iterations as compared to the discount factor of 0.90 and 0.99 which consumed the max_iter hyperparameter. For discount factor 0.90 and 0.99 VI consumed all the max_iter while the threshold of 0.001 was not reached. This points out that for balanced discount factor VI policy converges faster rather than extreme values. Also, the Policy change graph points out that a discount factor of 0.5 has a less policy change which suggests that a lower discount value prefers exploitation over exploration which helps reaching arm MDP to converge with the minimum threshold and reach all target resolutions. VI with a discount factor of 0.9 and 0.99 consumed most of the computational resources till it reached the max_iter parameter and still was unable to converge while the policy change was high which shows that the exploration has stopped the VI algorithm from reaching the convergence point.

Policy Iteration: Policy Iteration iteratively improves the policy until the convergence to optimal policy has been reached where the expected maximum cumulative reward is highest. Policy iteration starts with a random or predefined policy. Policy evaluation is done by estimating the value function associated with taking actions based on the policy. If convergence is not achieved then improve the policy by selecting actions recommended by the policy to maximise rewards for the current value function. Repeat this until convergence where we define convergence such that when current policy and improved policy are the same. PI assures to converge to the optimal policy for finite MDP like Forest Management and Reaching Arm. In each iteration, the policy is either improved or remains the same and the value function converges as policy evaluation progresses.

PI for Reaching Arm: I ran PI on the reaching arm where configurations were 2 arms with an angular resolution of 20 and a target resolution of 6 with a discount factor of 0.5, 0.9 and 0.99 to check if it behaves similarly concerning VI for Reaching Arm.



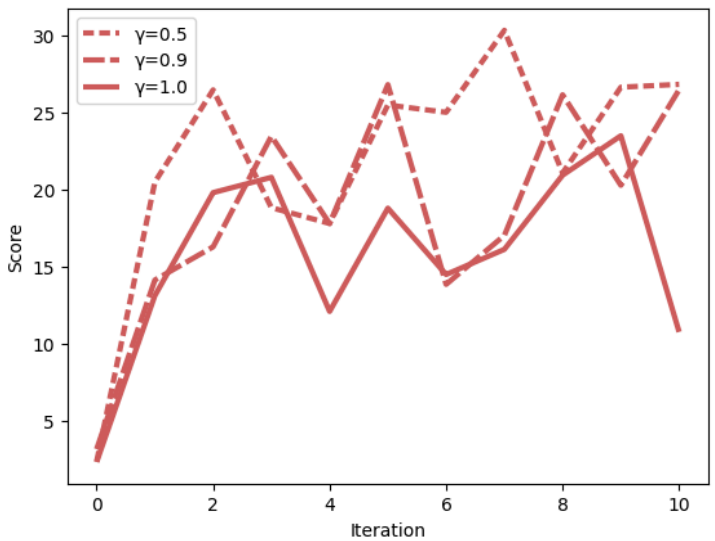


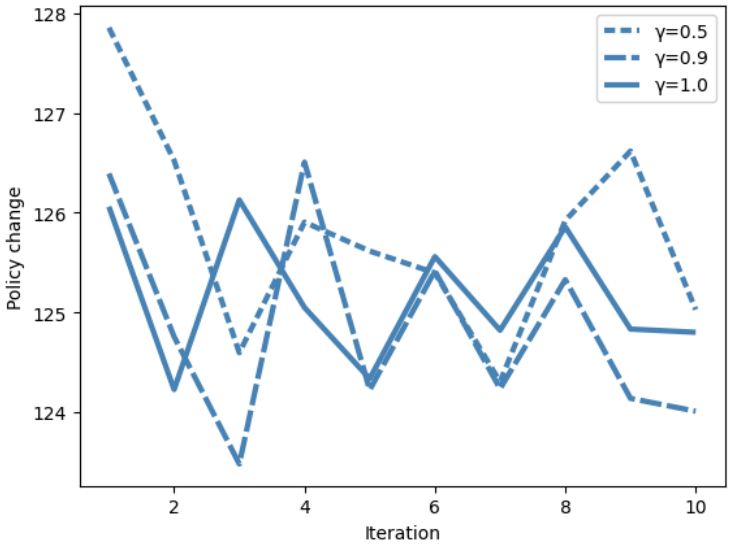
As per observations from VI discount factor of 0.5 should again converge rapidly since the 0.9 and 0.99 values were exploding and overshooting the MDP hence their performance will be similar but poor when compared with the 0.5 discount value. As the Score vs. Iteration graphs point out the initial expectations were correct the discount factor of 0.5 reached near to convergence point rapidly within the first 5 iterations but we can see that the line oscillates for about 35 iterations to converge. Discount factors of 0.9 take time to explore the MDP and finally converge with a score of about 210 but take more computational resources. The poor performance of PI with hyperparameter gamma=0.99 displays clear overshooting of MDP and does not converge to optimal policy within 50 iterations. The policy vs. Iteration graph displays there is a small amount of change in the policy from the policy evaluation and policy improvement steps which help the discount factor of 0.5 to converge rapidly without exploiting resources and delaying rewards by the value functions. Since exploitation is preferred over exploration when we set the discount

factor to 0.9 or 0.99 we can observe the PI function remains almost flat from 15 to 50 iterations not achieving the convergence point. PI for Reaching Arm with gamma=0.9 and 0.99 are not suitable values because the difference between the current policy and the evaluated policy is high enough which is the reason for not reaching the convergence point and completing all target resolutions by the Arm. When updating the policy based on the current value function, a high gamma can cause the algorithm to favour policies that promise high long-term rewards, even if they are suboptimal in the short term which is required for convergence and for that reason PI has not converged in less number of iterations. A discount factor of 0.5 is good to go for Reaching Arm like reinforcement problems.

Q-Learning: Q-learning is a reinforcement algorithm that converges to the optimal policy for an MDP without requiring explicit knowledge of the environment's transition probabilities. Q-learning belongs to the class of temporal difference (TD) learning methods and is based on the concept of action-value functions and for that reason it is the model-free approach that learns by interacting with the environment through exploring and exploiting. The goal of Q-learning is to learn the optimal action-value function which represents the cumulative rewards for taking action for that state and then following the optimal policy thereafter. And the optimal policy is represented by the maximizing of the expected cumulative reward. Q-learning balances exploration and exploitation using an exploration strategy, such as epsilon-greedy.

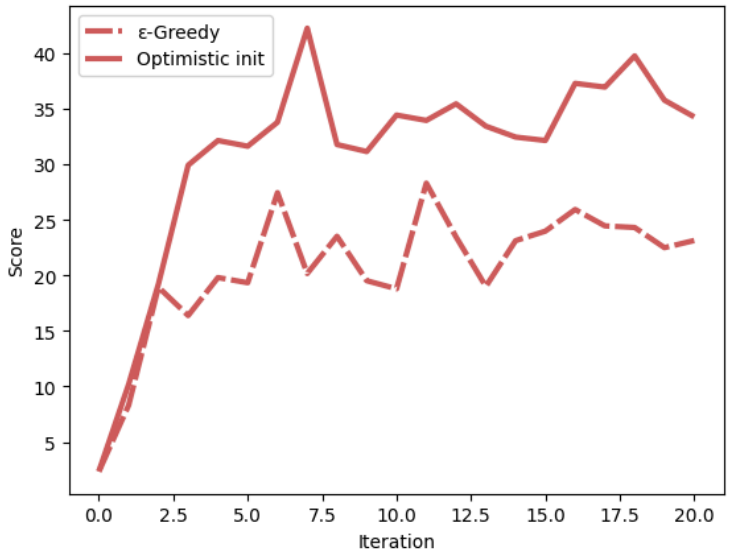
QL for Reaching Arm: Q-learning is the commonly used algorithm for solving MDPs and hence there are high expectations. We have many hyperparameters for Q-learning and most of them have been optimized to get the best results from the experimentation. The epsilon greedy strategy is used to set the exploration-exploitation of the algorithm and as per understanding and experimentation, it is set to 0.2 and 0.05 because those values resulted in higher scores for cumulative rewards. The learning rate alpha has also been tested with different values like 0.01, 0.04, 0.1, 0.3, 0.5, and 0.9 where the observation was that there is no significant difference in the learning speed and convergence of the Q-learning algorithm and for that reason, I chose alpha=0.5. While running experiments for discount factor we observed interesting results because ideally, we expected that since Q-learning is a robust algorithm whose performance will be high for Reaching Arm but that was not true. The Scores vs. Iteration graph points out that the discount factor of 0.5 leads to better performance and faster convergence. While the higher values of gamma have exhibited poor convergence ability the reason for this is the same the exploration won't allow the algorithm to converge optimally resulting in degradation in performance. While the Policy change vs. Iteration graph is too volatile reason behind this is Q-learning and epsilon-greedy exploration will make large changes in the policy and that's why requires a large number of iterations for converging.

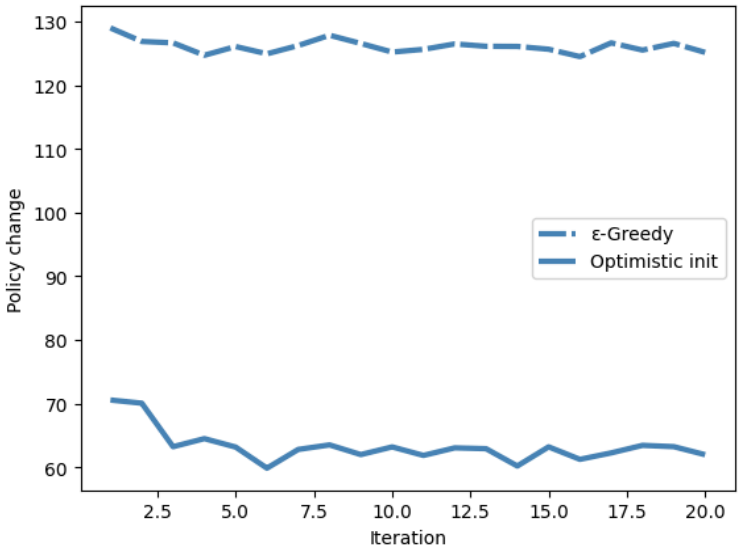




The gamma=0.5 and 0.9 have similar scores for the tenth iteration but since their past performance has been oscillating which restricts us from making the Q-learning algorithm as reliable for the reinforcement learning problem of Reaching Arm. Similar results can be derived from the policy change graph as expectations were that the policy change should reduce but here the graph displays a high delta between the current policy and the evaluated policy and the reason for this is the nature Q-learning algorithm which prefers trial and error to reach optimal state while this is restricting from convergence and achieving high cumulative rewards. To improve the performance of the Q-learning algorithm I tried Optimistic Initialization which helps in improving performance.

Optimistic initialization is a technique used in Q-learning to emphasize exploration by initializing the Q-values with optimistic estimates (in our case 5) which is high enough to allow the algorithm to explore unvisited states. As we can observe in the graph the Optimistic Initialization performed 2 times better than the epsilon greedy Q-learning algorithm subject to other hyperparameters like learning rate and discount factor as the same. The reason for this is when epsilon is set to 0.05 and q_init value to 5 then Q-learning does not consider cumulative expected rewards but focuses on exploration which helps the algorithm to bypass the suboptimal policies making Optimistic Initialization a robust algorithm. We can observe that the policy doesn't change for higher iteration value and that displays the importance of delaying rewards and exploration of state-action pair to reach optimal policy for convergence. Thus Q-learning is a good reinforcement learning algorithm but requires optimization in its design like restricting the exploitation-exploration part to understand the underlying reinforcement problem and pass from non-convergent policy to reach optimal state and increase cumulative rewards.





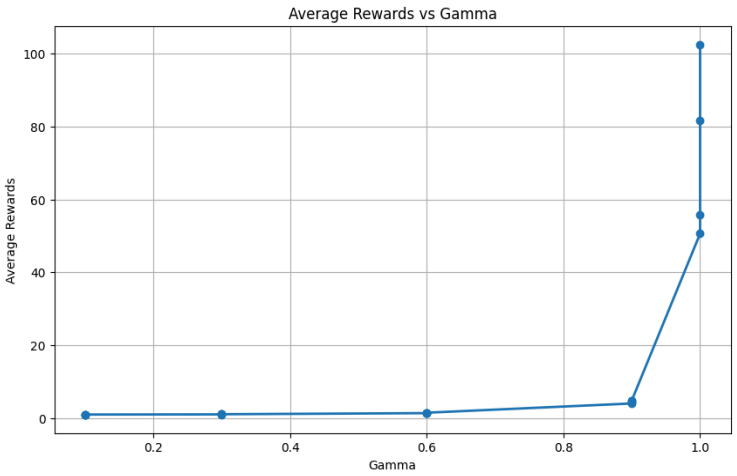
Comparison: The performance of Value iteration with gamma as 0.5 was highest because it consumes the lowest iterations to converge while rewards were the most considering the explore-exploit strategy and extracting most from the agent without changing the policy to reach the optimal solution. The same can be said for Policy Iteration where the iteration to converge was more than VI and this consumed more computational power and time for that reason, PI ranked second out of three reinforcement learning algorithms. There were most of the hopes from the Q-learning algorithm for the Reaching Arm because QL is designed on trial and error methods but the scores are too volatile pointing out that the Reaching Arm is a problem that shouldn't be solved with Q-learning. When modifications are made in Q-learning with Optimistic Initialization, performance has somewhat increased which is still less compared to VI and PI and hence it requires deterministic solutions that can reach the target resolutions properly to accumulate cumulative rewards

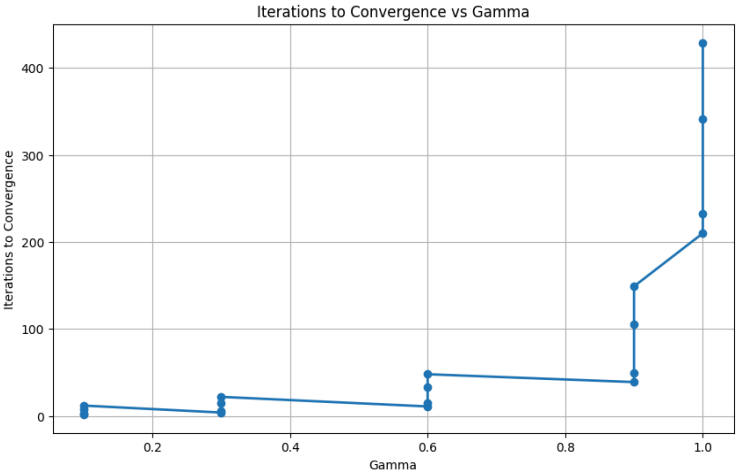
Conclusion: Value Iteration and Policy Iteration help achieve optimal solutions in the limited number of iterations to complete all 6 target resolutions defined by the 2 arms and 20 angular resolutions. Q-learning scores are low suggesting improvement which can lead to future scope of reinforcement learning while exploring other algorithms like SARSA, REINFORCE, and Actor-Critic for Reaching Arm. When we have a Reaching Arm-like reinforcement learning problem with limited computing power and accuracy that has to be optimized then we should prefer the Value Iteration algorithm. But when we have ample computational resources and want to explore all possible cases we can prefer policy iteration to find out how the algorithm discovers the problem structure and help reach convergence by maximizing rewards. The Q-learning algorithm should only be preferred when the accuracy and computational time are not a concern and we want to explore the trial and error cases from the agent in the reinforcement environment to achieve convergence.

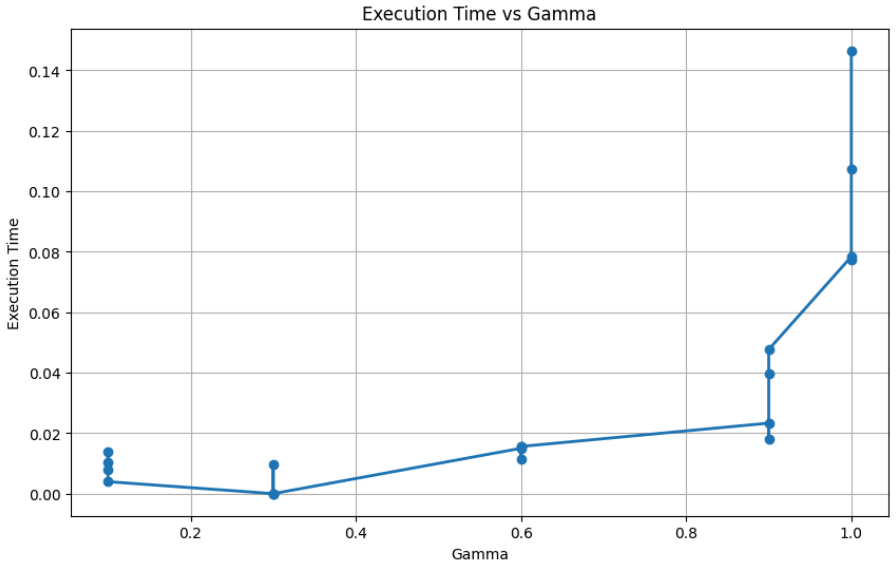
2. Forest Management: Forest Management is a classic non-grid world problem where agents need to find a balance between "Cut" i.e. cutting the forest versus "Wait" and not cutting the forest. The wait action will lead to a forest fire with a probability of p each year which will reset the environment back to its initial state whereas the cut action will lead to a reward of getting profit in terms of money by selling wood from trees. The discount in this case

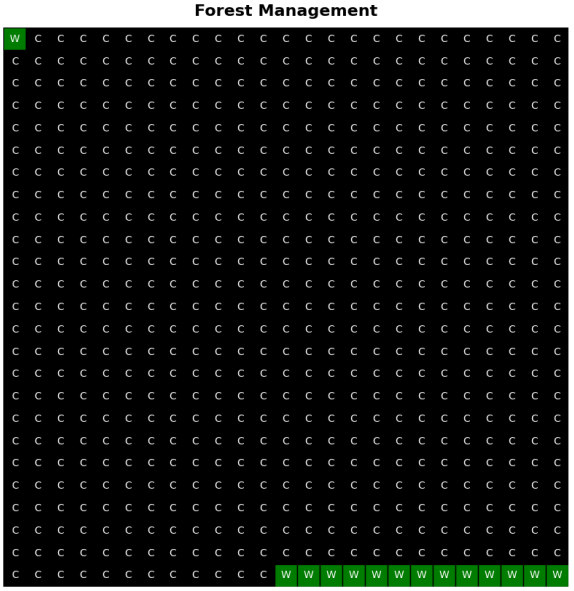
can lead to interesting results while backtracking what possible solutions the agent could have taken to maximise profit for value iteration, policy iteration and Q-learning is different which determines the importance of future rewards compared to immediate rewards (wait v/s cut). A higher value of gamma places more emphasis on future rewards but the probability p will destroy those rewards if the forest is found in fire while in our case we chose 0.9. The reason for choosing Forest Management as MDP for this assignment is due to the ambiguity concerns where the agent can optimize rewards by opting for wait action which leads to forest fire and reset to the initial state or to cut the tree. Hyperparameter epsilon determines the stopping criterion for reinforcement algorithms where it represents the maximum tolerated error between iterations. The maximum No. of iterations limits the number of iterations allowed for convergence in both Policy Iteration and Value Iteration and we chose 1 million which is the sweet spot between computational resources and time.

Value Iteration for Forest Management:





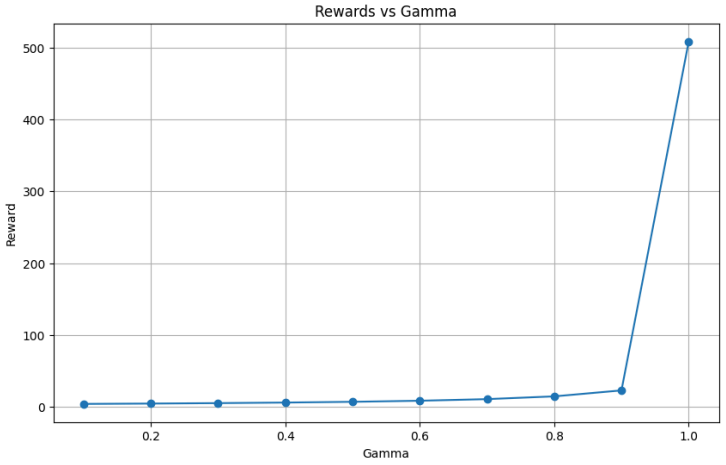


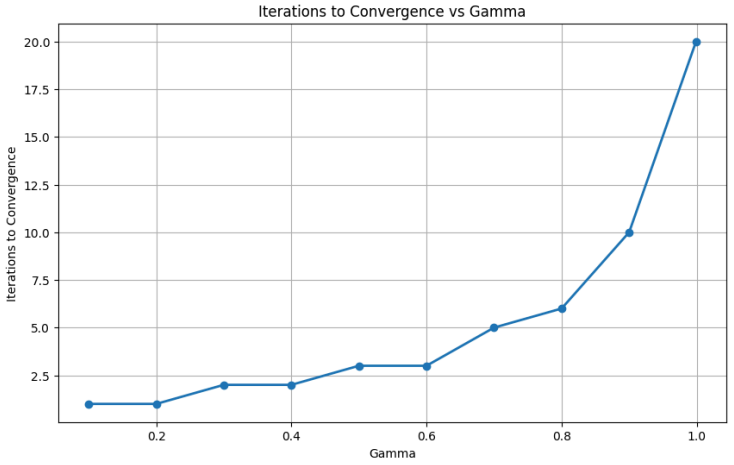


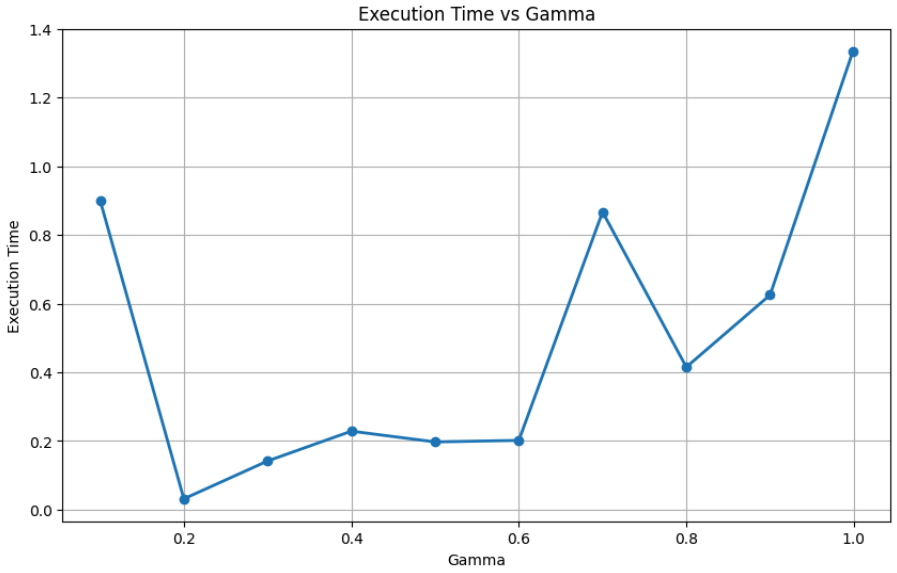
Value Iteration has been run for different values of gamma [0.1, 0.3, 0.6, 0.9, 0.999] and epsilon [1e-2, 1e-3, 1e-8, 1e-12] which act as hyperparameters for VI in the case of forest Management. Initial Expectations are quite high from the VI since it performed well on Reaching Arm, especially for the discount factor of 0.5 should outperform other values because it is the perfect balance between exploration-exploitation of the MDP. Since the max_iter value has been set to 100K I assume that the VI should always converge to an optimal solution without reaching the max_iter number. As Average Rewards vs. Gamma (discount factor) displays for gamma=0.999 the rewards are maximum while for lower discount factor cumulative rewards are in the range of 0-10. The reason for this fact can be related to the graph of Iterations to Converge vs. Gamma where we can see that as the discount value increases it requires a higher number of iterations to converge to the optimal policy. Thus VI performed well when gamma=0.999 receiving 238 cumulative rewards but it require about 429 iterations to converge. This means the discount factor has

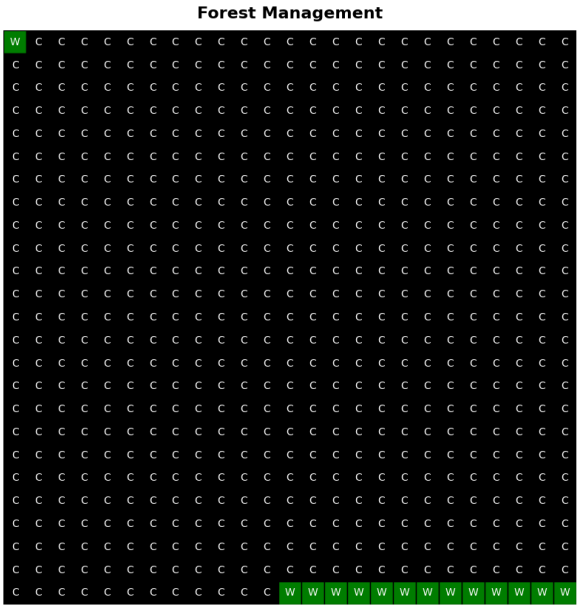
been set to extreme (0.999 in our case) which encourages the algorithm to explore over exploiting for cumulative reward function while accelerating convergence without overshooting. Execution time vs. Gamma graphs point that as we decrease the epsilon value, execution time and iterations increase slightly since reaching the threshold takes time and VI has to find better solutions without getting stuck in an oscillating path. The Forest Management graphs show that the upper left and a few bottom right have been marked with a “wait” state while other remaining states from 625 have been marked with “cut”.

Policy Iteration for Forest Management: Policy Iteration is very similar to VI and hence as per observations from VI for Forest Management we can expect that the PI algorithm should ideally perform in similar terms with gamma=0.999. Here we have gamma as a hyperparameter ranging from 0.1 to 1.0 with 0.1 as an interval among other values.





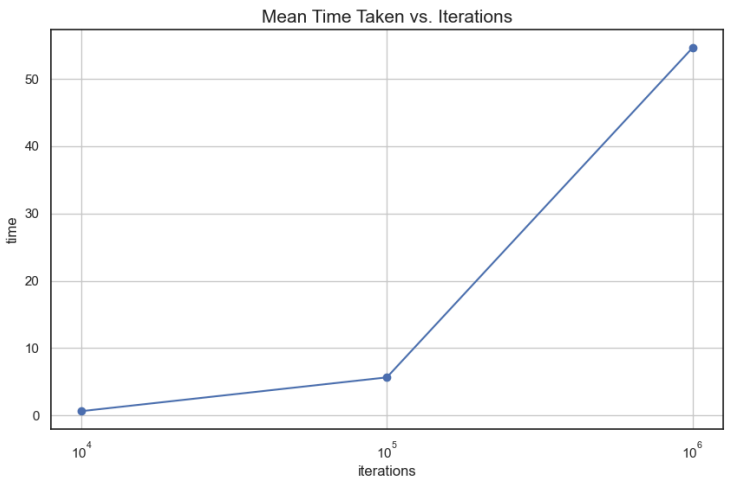




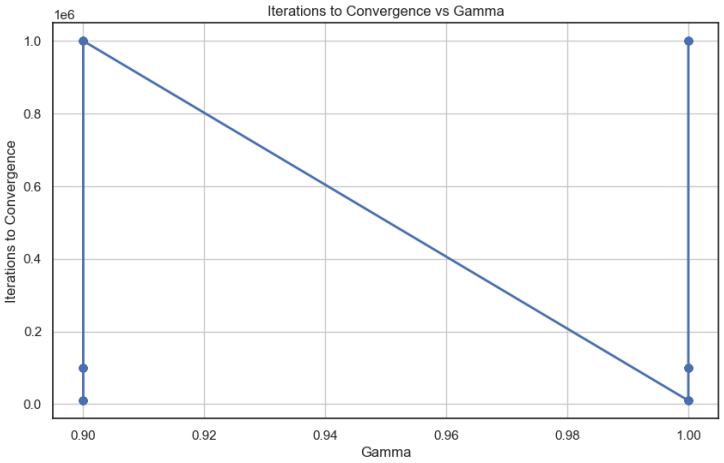
As we can observe from the Rewards vs. Gamma graph the curve is mostly flatish for rewards but as we get close to 0.9 and beyond we see a steep rise in rewards which is valued at 508. This reward value is far higher than the rewards received in VI for FM reason for this is implicitly the epsilon value when set to a lower number will continue running the algorithm iteratively for VI which explores all possible states until the threshold value has been reached. But in PI, epsilon has been set to 0.0001 by default which helps in reaching high rewards and not exploring other states unnecessarily. This phenomenon when combined with the discount factor always prioritises the rewards that help achieve great performance in terms of reward while the total iterations required was 20 where the max_iter value was 100K. The iterations graph is strictly increasing for large gamma values but still, it is very much lower to VI and the reason for this is as stated above the epsilon value when set in a lower range we expect the algorithm not to explore much terrain and get to the convergence point only by optimizing for cumulative rewards. The execution time

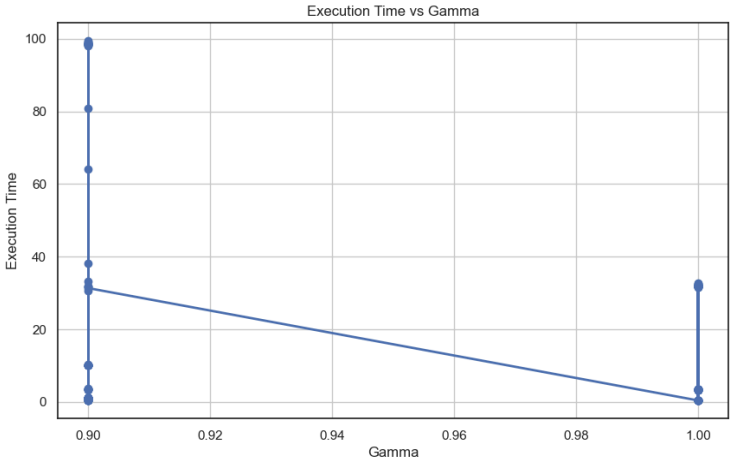
vs. Gamma graph points out that when the gamma value increases, execution time is getting higher. But when we compare the timings of best-performed hyperparameters of VI [gamma=0.999,epsilon=1e-12] and PI [gamma=0.999] we can say that VI though receiving lower rewards due to more exploration has executed the algorithm in lower time than PI resulting in same final output which can be seen from below visualization. Thus Both VI and PI are converging to the same state irrespective of the algorithm which points that the solution generated by the reinforcement learning algorithms is perfectly aligned with the problem.

Q-Learning for Forest Management: Q-learning is a widely used reinforcement learning algorithm that when applied to Forest Management should generate results that are superior in quality as compared to VI and PI, the reason for stating this is because q-learning is the model-free algorithm that doesn't require prior knowledge of state probabilities and rewards while its performance is good for complex and uncertain problems like Forest Management.



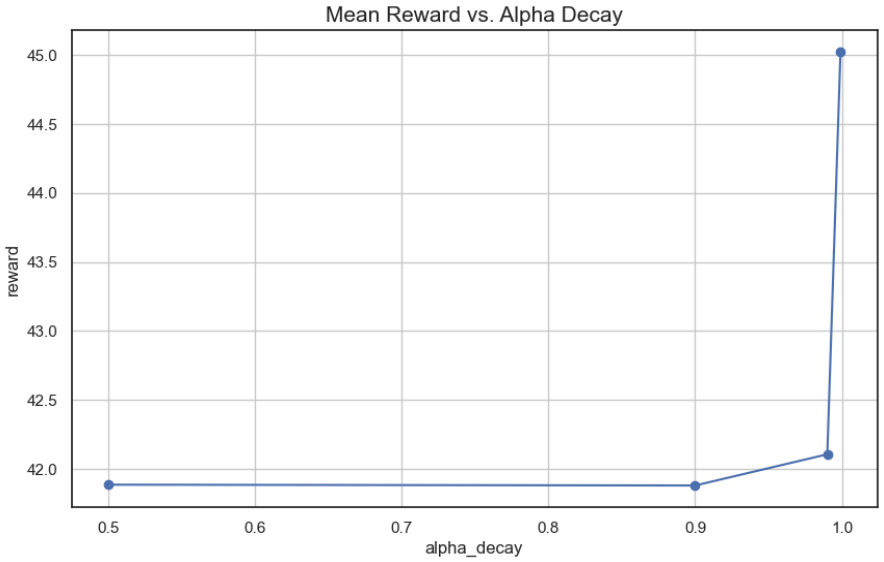
The initial expectation from Q-learning is that we have hyperparameters like gamma, alpha, alpha_decay, epsilon and iterations and performance will be superior when gamma is at its highest value i.e. 0.999 in our case. Since Q-learning is a trial and error module we have learning rates alpha and alpha_decay which control the overfitting of agents in the environment and from Deep Learning we have insights that when the learning rate is minimum then the agent learns the structure of the problem and when the learning rate is decayed as iterations increase, help achieve greater rewards. Q-learning is an expensive algorithm in terms of computational time and resources and hence we expect it to take time to reach convergence.

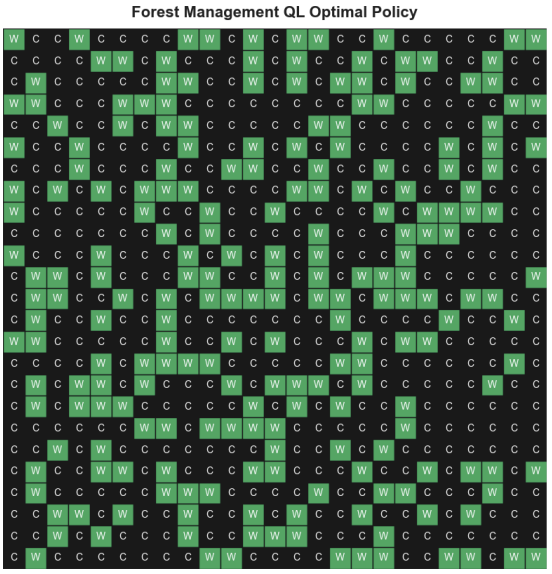




As the Mean Rewards vs. Iteration graph shows that rewards increase as we increase iterations, there is a possibility of running this algorithm for 1e7 but, I limited my report to 1e6 which still took about 60 seconds (average) to run one iteration. Mean Rewards vs. Alpha decay suggests that as the decay value increases rewards increase because the agent learns the terrain appropriately which helps in collecting cumulative rewards that are discounted back by the gamma to reach convergence. Iterations to Converge vs. Gamma displays that when iterations hyperparameter with values as [1e4, 1e5, 1e6] we can observe that none of the iterations converges below its assigned hyperparameter value which also explains that even if the agent might not have achieved a complete

solution while due to limiting the iteration size algorithm has been concluded. As the gamma value increases the execution time increases which is expected from our prior knowledge of reinforcement learning. The Forest Management visualization explains how the “cut” and “wait” states could have been arranged for a 625 sq. size forest while this solution can be improved by increasing the iteration number. The Q-learning agent has done a good job finding the optimal states to maximize the reward to 234 points where the best policy hyperparameters were Gamma = 1.00, Alpha = 0.10, Alpha Decay: 0.999, Epsilon Decay: 0.990, Iterations: 1.0E+06. The results could have been improved by setting 1e7 and 1e8 as iteration numbers but it takes an infinite amount of computational resources and time to complete all combinations and get more refined results as we got in VI/PI algorithms.





Comparison: Value iteration with gamma=0.999 performed the best in its segment which points out that forest management of 625 sq. unit is a problem that requires more exploration to understand and find ideal states for cut and wait to maximize the cumulative rewards. But the reward score of 100 is less as compared to Policy Iteration which scored 508 but the execution time is well inchecked with the number of iterations as it consumes less computational power. Policy Iteration on the other hand can iteratively change its policy to converge to an optimal solution with maximum cumulative reward but consumes more time to reach the convergence point because gamma=0.999 which encourages exploration and optimizes for rewards that’s why it reaches a high reward score. Q-learning took exceptionally high computational time and maximum iterations of 1e6 to display forest management’s cut and wait for decisions taken by the agent via the trial and error method. QL yielded a reward similar to policy iteration and this result was contributed by the hyperparameter of alpha and alpha_decay which adjusted the learning rate of the agent for reinforcement of the Forest Management problem.

Conclusion: As per Comparison we can clearly state that if the computational power and time are limited while the cumulative reward could be on the lower end then we should prefer Value iteration if the reinforcement learning problem resembles Forest Management. When we have ample computational time and there is no limitation for reward then we can choose Q-learning with the best set of hyperparameters to achieve decent performance and optimal solution where the agent is opting for model-free transitioning or reward knowledge. Though this methodology will result in substandard rewards the behaviour of how the agent will react could help us understand more about the reinforcement problem and the potential advanced solutions for the same.

References

[1]. https://www.gymlibrary.dev/environments/mujoco/reacher/
[2]. Assignment 1 writeup.
[3]. Assignment 2 writeup.
[4]. Assignment 3 writeup.