## [**CS-7641**](https://gatech.instructure.com/courses/224746) **Markov Decision Processes**

**Why did I choose these MDPs? Why are they interesting?**

1. Frozen Lake-

I used the Frozen Lake environment from OpenAI Gym. I chose to use 4X4 so it was easy to visualize and track each iteration . As it is a grid problem, it is also easy to draw and check how different policies work on the grid.

1. Forest Management-

I used the Forest management from mdptoolbox-hiive as the non grid world problem with 600 states. I feel this kind of problem is more suitable as a real world problem with fire and reward probability. As the first problem’s policy could be also tried manually, forest management is much more difficult and closer to what problems will be in the real world which will need to be solved using reinforcement learning.

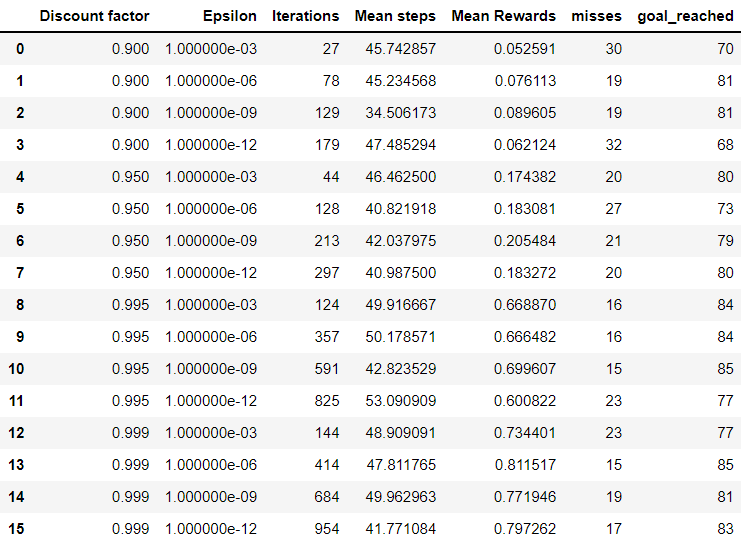
**Frozen Lake 4 X 4**

**Value Iteration**

1. Tuning parameters (Discount factor and Epsilon):

The first parameter to tune was Discount factor (i.e. gamma) which controls how important future rewards are. It was observed that the larger the gamma value, the more the agent cares about long term rewards. It was observed that larger values of gamma were performing better than smaller gamma values as agent was focusing on getting rewards (i.e. to the goal state) in earlier iterations of the training phase.

1. How did I decide the convergence criteria?

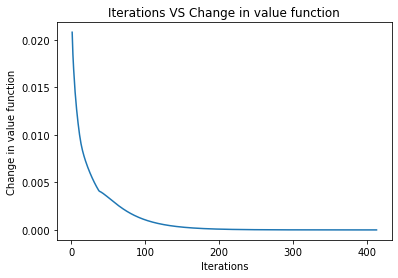
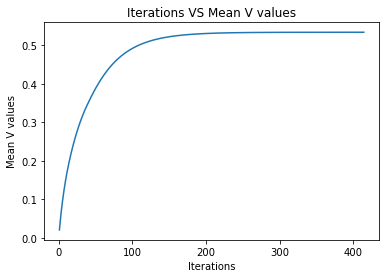
As value iteration needs a stopping condition, Epsilon was used to decide when to stop the iterations. 

When the maximum difference between the value functions of previous iteration and current iteration is equal or below epsilon, it means that the policy has stopped changing and we can say that value iteration has converged to the optimal policy. The epsilon value if too low or too high was not returning optimal policy.

Also, for each set of parameters, I ran the retrieved policy 100 times to see how it performs and made a note of the misses (i.e. the policy failed and went into the hole) and goal\_reached (i.e. if the policy could successfully find the goal).

It was observed that **epsilon = 1e-06 and discount factor = 0.999 was performing best**. Also, when the retrieved policy using these parameters was run 100 times, it reached the goal 85% of the times!

It was observed that, as iterations progressed, the rewards were increasing but after a certain value, i.e. 200-300 iterations the values stopped increasing. Also, it was seen that as iterations progressed, the value function stopped changing, which indicates that the algorithm has converged to an optimal policy.

****

*Iterations VS Rewards for optimal policy Iterations VS change in value func for optimal policy*

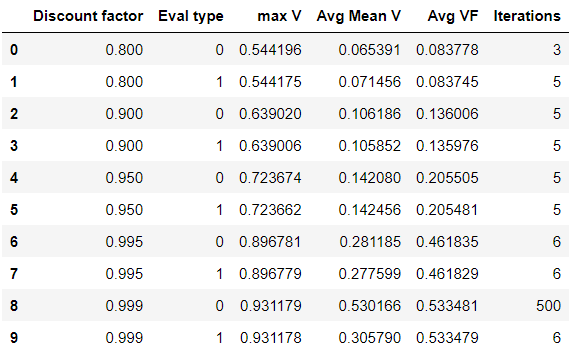
**Policy Iteration**

1. Tuning parameters (Discount factor and Eval type):

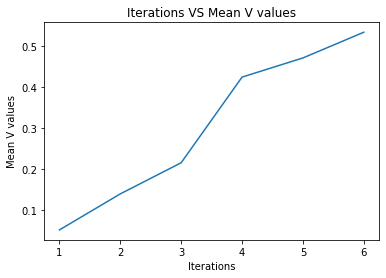
The first parameter to tune was Discount factor (i.e. gamma) which controls how important future rewards are. The larger the gamma value, the more the agent cares about long term rewards. Its behavior was very similar to that of value iteration.

1. How did I decide the convergence criteria?

For policy iteration, the algorithm starts with a predefined policy (in my case, all zeros) and the policy is updated and once the policy is not changing anymore as we go through iterations, we have converged and found the optimal policy.



With a **discount factor = 0.999 and eval type = 1** with 6 iterations, the algorithm converged to the optimal policy.



The rewards kept increasing as the iterations progressed.

Did value iteration and value iteration converge to the same policy?

Both the algorithms converged to the same policy i.e. [0, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0]

Which performed better?

|  | Discount factor  (Gamma) | Epsilon / Eval type | Iterations to converge | Policy |
| --- | --- | --- | --- | --- |
| Value Iteration | 0.999 | 1e-06 (Epsilon) | 414 | [0, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0] |
| Policy Iteration | 0.999 | 1 (Eval type) | 6 | [0, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0] |

It was clearly observed that Policy Iteration was faster in terms of time and iterations than Value iteration i.e. Policy iteration converges faster than Value iteration.

Value iteration starts with a random value function and as Policy iteration starts with a random policy, it is easier to jump from one policy to another than changing value functions which takes a lot of time

**Q Learning**

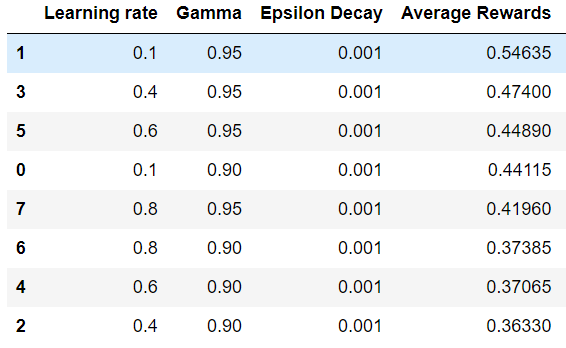
For Q learning, I wrote my custom code for Frozen lake rather than using that from mdptoolbox-hive as it was not very suitable for a small state problem.

Tuning parameters:

1. I tuned the learning rate (i.e. alpha), gamma and epsilon decay. Low learning rate allows the model to learn slowly over the states. Again gamma (i.e. the discount factor) plays a similar role as that of VI and PI. So higher gamma values performed better than lower values.
2. Exploration strategy?

For higher values of Epsilon (close to 1) there is more exploration which is needed at the beginning for the agent to learn. Once the agent is familiar we start using Epsilon decay and exploitation is used i.e the existing knowledge is used. Therefore, I started the Epsilon from 1 and used the Epsilon decay of 0.001 to

slowly change the Epsilon value so there is a lot more exploration at the start and then exploitation as we move further with iterations. This is the **epsilon-greedy method.**



It is clear that a **slow learning rate of 0.1 with high gamma of 0.95 and epsilon of 1 and low epsilon decay of 0.001** allows the Q learning to learn the state space more efficiently.

Testing the best Q Table from Q learning. Is it optimal?

After retrieving the Q table, I used this Q table to run the Frozen lake environment 100 times to see how on average it performed. **The agent reached the goal 82% of the time and went in hole 18% of the time**.

**Forest Management S=600**

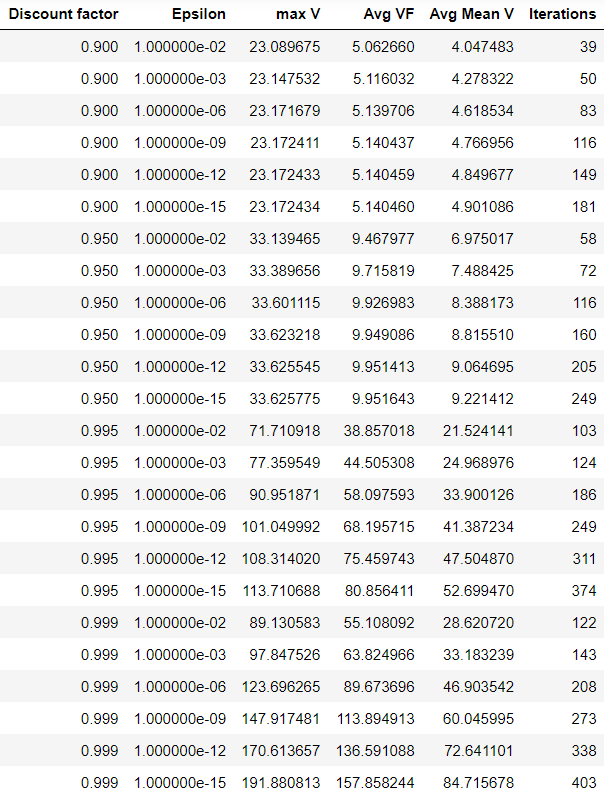
**Value Iteration**

1. Tuning parameters (Discount factor and Epsilon):

The first parameter to tune was Discount factor (i.e. gamma) which controls how important future rewards are. It was observed that for smaller values the rewards were low and the policy was not converging.

1. How did I decide the convergence criteria?

As value iteration needs a stopping condition, Epsilon was used to decide when to stop the iterations similar to that mentioned in Frozen Lake above. For higher epsilon values 1e-02, 1e-03, 1e-06 iterations were less and the algorithm did not converge. For lower values 1e-15, 1e-12 iterations were larger and policy was able to converge.

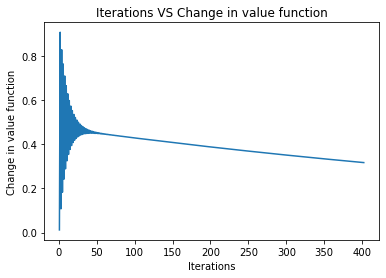
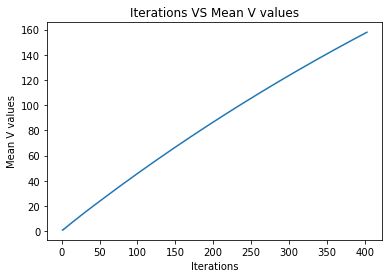


In the table Avg VF indicates the Average Rewards for that policy.

It was observed that higher value of gamma 0.999 and lower epsilon value of 1e-15 converged and optimal policy was found with highest rewards.

Iterations required were 403 for a state size of 600 and as I used the lowest Epsilon value the iterations required were high compared to others.

From the graphs below it is also seen that for the optimal policy, the rewards kept increasing as iterations progressed. While, from the second graph below it is seen that the difference in policy was decreasing indicating that the algorithm was converging to an optimal policy.



*Iterations VS Rewards for optimal policy Iterations VS change in value func for optimal policy*

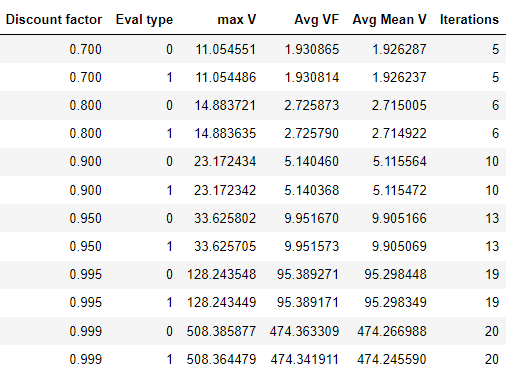
**Policy Iteration**

1. Tuning parameters (Discount factor and Eval type):

The first parameter to tune was Discount factor (i.e. gamma) which controls how important future rewards are. The behavior was similar to that in Frozen lake mentioned above.

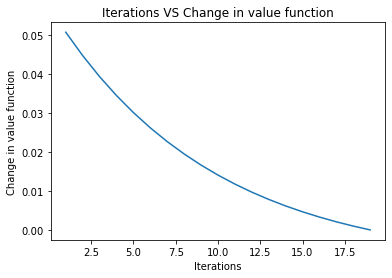
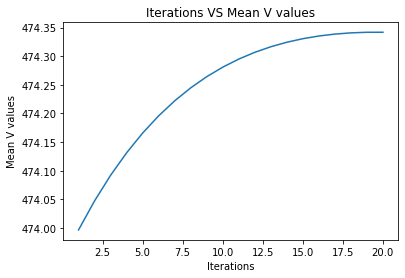
1. How did I decide the convergence criteria?

For policy iteration, the algorithm starts with a predefined policy (in my case, all zeros) and the policy is updated and once the policy is not changing anymore as we go through iterations, we have converged and found the optimal policy.



As seen in the table, a discount factor of 0.999 and eval type 1 with 20 iterations found the optimal policy. Avg VF (i.e. the rewards) was also highest for this set of parameters.

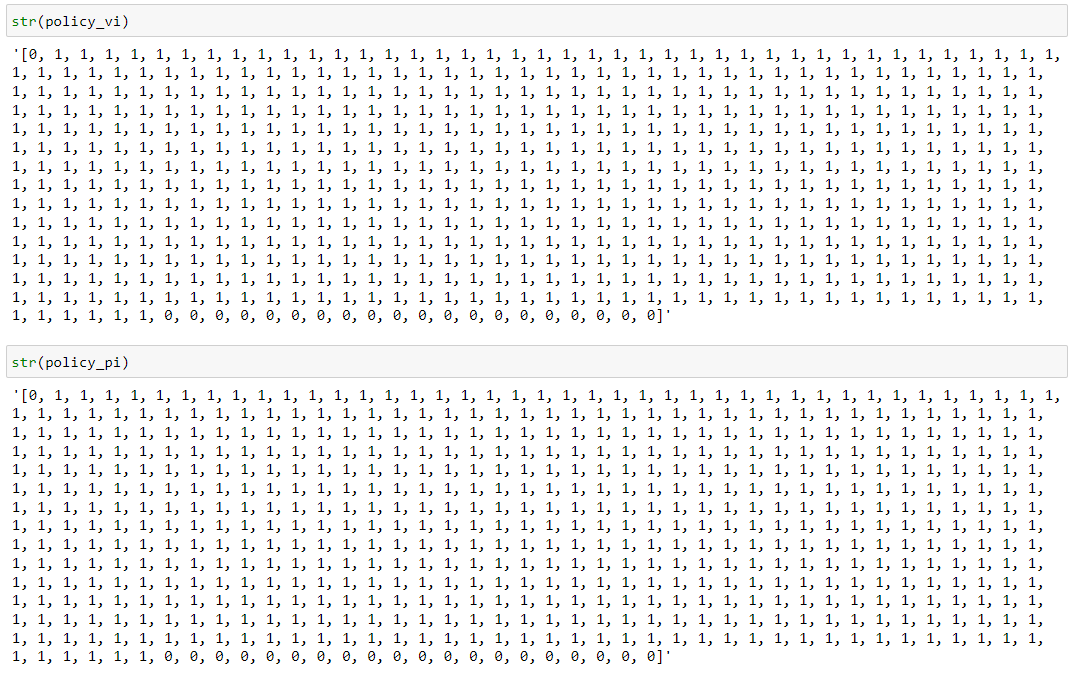
For the optimal policy retrieved, the rewards increased as iterations progressed and also in the second graph it is visible that the value function stopped changing indicating that the algorithm has converged.



*Iterations VS Rewards for optimal policy Iterations VS change in value func for optimal policy*

Did value iteration and value iteration converge to the same policy?

Both the algorithms converged to the same policy i.e.



This policy indicates that cutting and getting profit is more suitable given the fire probability and later on waiting for the forest to grow seems to return maximum rewards.

Which performed better?

|  | Discount factor (Gamma) | Epsilon / Eval type | Iterations to converge | Returns same policy |
| --- | --- | --- | --- | --- |
| Value Iteration | 0.999 | 1e-15 (Epsilon) | 403 | Yes |
| Policy Iteration | 0.999 | 1 (Eval type) | 20 | Yes |

It was clearly observed that Policy Iteration was faster in terms of time and iterations than Value iteration i.e. Policy iteration converges faster than Value iteration.

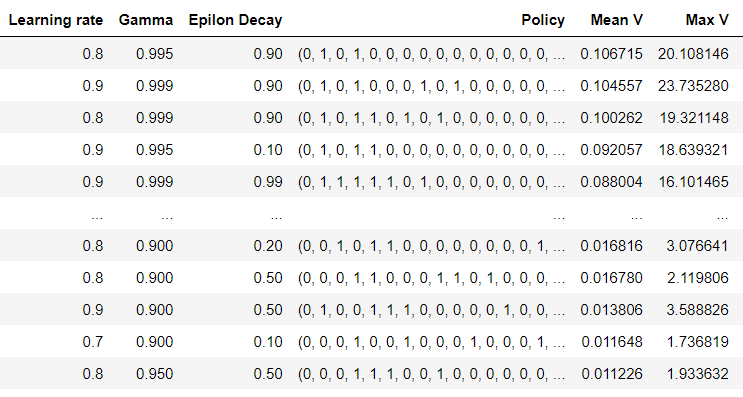
**Q Learning**

For Q learning I used the same parameter tuning method which is tuning learning rate, gamma and epsilon decay. I used the Q learning from the mdptoolbox-hive (which I have cloned and tweaked a little bit)

The difference observed was that a higher learning rate was required which indicates that Q values are updated faster. Also, a higher gamma value is more suitable as it returns more rewards.

A higher Epsilon decay performed better.

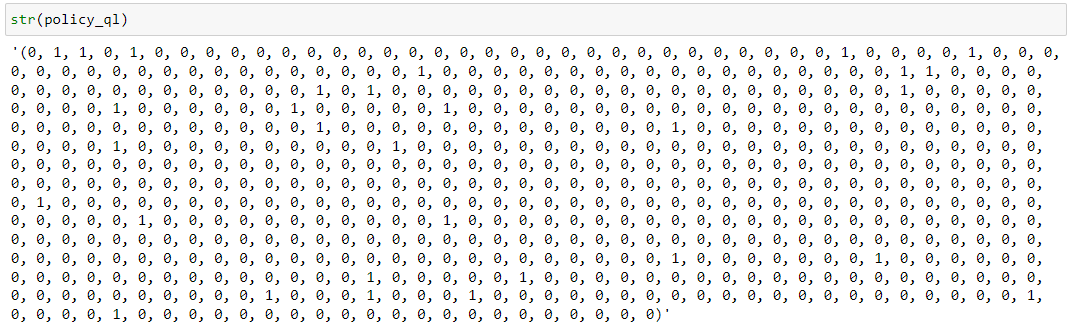
For frozen lake the way the epsilon was changing for each iteration was different (custom code). Here a higher value slowly decreases the exploration and more exploitation begins.



Tuning the Q learning to get high rewards using mdptoolbox needed more time.

The policy retrieved focused on not cutting the trees and rarely cutting them.

The policy is different of that retrieved from the VI and PI



**Conclusion**

1. Both MDPs being very different from each other took different training times and different parameter tuning.
2. Both Value iteration and Policy iteration return the same optimal policy.
3. Policy iteration takes less time and iterations to converge compared to Value iteration.
4. The Epsilon greedy approach worked well to balance between the exploration and exploitation.

* **Links referred**

1. <https://github.com/hiive/hiivemdptoolbox>
2. <https://pypi.org/project/mdptoolbox-hiive/>
3. <https://medium.com/deep-math-machine-learning-ai/ch-12-reinforcement-learning-complete-guide-towardsagi-ceea325c5d53>
4. <https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa#:~:text=Value%20function,agent%20starting%20from%20state%20s%20>.
5. <https://jacobhiggins.github.io/posts/2020/06/blog-post-1/>
6. <https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/>
7. <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>
8. <https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187>
9. <https://annisap.medium.com/searching-for-optimal-policies-in-python-an-intro-to-optimization-7182d6fe4dba>