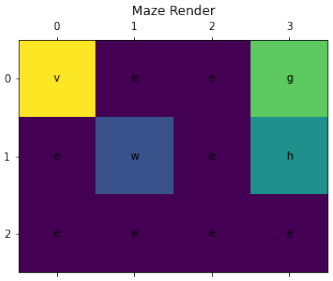
CS698R: Deep Reinforcement Learning

Mid-Semester Exam

Name: Shiven Tripathi
Roll NO.: 190816

Solution to Problem 1: Random-Maze Environment Implementation

1. Testing RME
   Rendered Maze



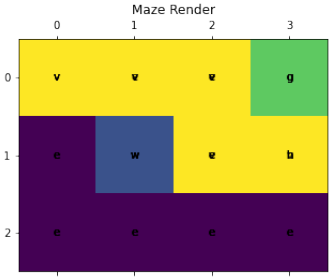
Testing with predefined steps to reach:

(a) Goal:

```
rme.reset(start_loc=0)
rme.step(1)
rme.step(1)
rme.step(1)
rme.step(1)
print(rme.trace)
    Trace: [0, 1, 2, 3, 3]
```

(b) Hole:

```
rme.reset(start_loc=0)
rme.step(1)
rme.step(1)
rme.step(2)
rme.step(1)
rme.render()
print(rme.trace)
```



Solution to Problem 2: RME Optimal Policy via Dynamic Programming

1. Policy Iteration:

   (a) An entire description of the environment in the form of state, action and transitions was generated as the 'P' data structure for the environment

   (b) We started with an arbitrary left facing policy and tested that policy evaluation worked appropriately

   (c) For the given environment, we didn't achieve convergence for Policy Iteration due to runtime errors in Policy Improvement Step

2. Value Iteration

   (a) We attained convergence for Value Iteration in just 23 iterations

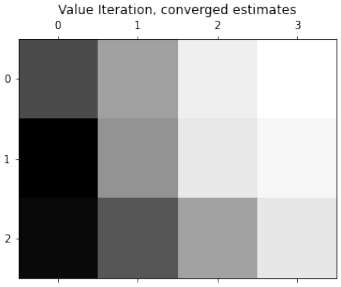
   (b) Problem 2 Notebook contains a rendered play through with the optimal policy discovered by VI

   (c) The Optimal Policy is:

   ```
   [[1. 1. 1. 1.]
   [3. 1. 1. 3.]
   [1. 1. 1. 3.]]
   ```

   where 0: Left, 1: Right, 2: Down, 3: Up

   (d) This figure shows a heat map for the value estimates. Observe that terminal states have near zero value at convergence, as we expect to be true



3. Comparison of PI and VI

   (a) PI did not converge for this particular environment

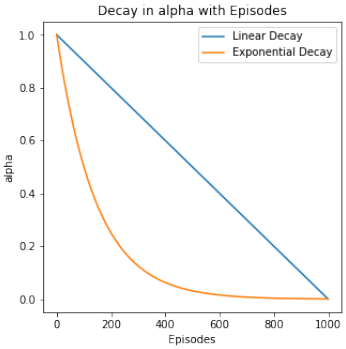
   (b) Both methods can return different policies, with changes near states which don't actually end up acted on, eg: terminal states

Solution to Problem 3: RME Prediction with MDP Unknown

1. Generate Trajectory: We test the trajectory generator with an arbitrary move right policy, which ends at the goal state

```
Step: 0
State: 0 Action: 1 Reward: -0.04 Next State: 1
Step: 1
State: 1 Action: 1 Reward: -0.04 Next State: 2
Step: 2
State: 2 Action: 1 Reward: 1 Next State: 3
```

2. Decay Alpha Step Parameter: We test the Decay of Alpha from 1 to 0.001 in 1000 time steps, both exponentially and linearly



3. Monte Carlo Prediction: We test Monte Carlo Prediction using the Optimal Policy found from Value Iteration (in report above)

```
Testing FVMC
FVMC Estimates:  [0.8854804  0.88191744 0.88675993 0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
Testing EVMC
EVMC Estimates:  [0.87099653 0.87373725 0.88843873 0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
```

Since the policy is defined for hard actions, we don't see other states being encountered and hence their values don't end up being corrected

4. TD Prediction: We test TD Prediction using the Optimal Policy found from Value Iteration (in report above)

```
Testing TD
TD Estimates:  [0.89650608 0.94007408 0.99120294 0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
```

Since the policy is defined for hard actions, we don't see other states being encountered and hence their values don't end up being corrected

5. n-step TD Learning Prediction: We test n step TD Prediction using the Optimal Policy found from Value Iteration (in report above). As mentioned, we used n = 3 and other values as specified in the instructions

```
Testing N Step TD
N Step TD Estimates:  [0.70132103 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
```

Since the policy is defined for hard actions, we don't see other states being encountered and hence their values don't end up being corrected

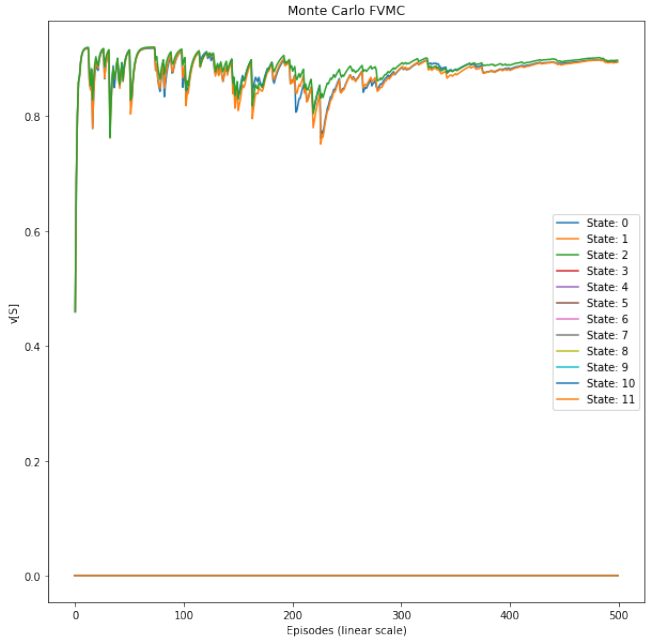
6. TD(lambda) Prediction: We test TD Lambda Prediction using the Optimal Policy found from Value Iteration (in report above).

```
Testing TD Lambda
TD Lambda Estimates:  [0.89618244 0.94463419 0.9911973  0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
E:  [0.         0.         1.42795945 0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
```

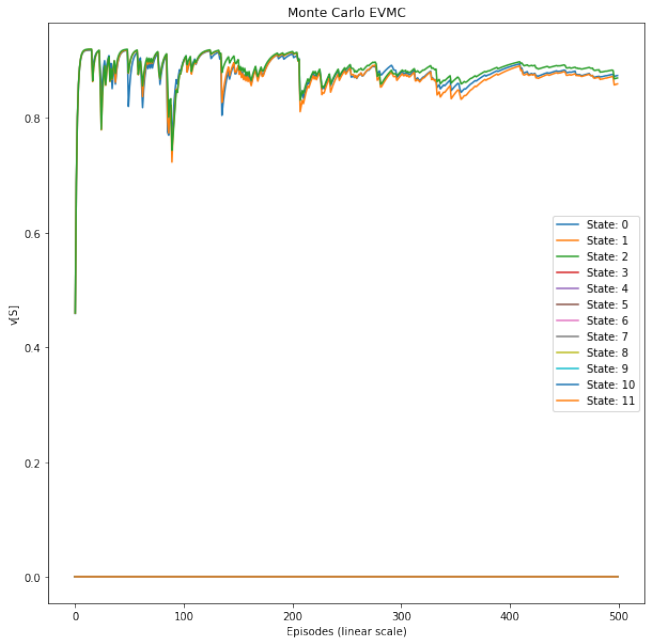
Since the policy is defined for hard actions, we don't see other states being encountered and hence their values don't end up being corrected

7. True Value Calculation: We can see good agreement between values obtained using different methods, except for n step TD which fails to update beyond the first entry. The actual correct values correspond to the Dynamic Programming Solution of the environment

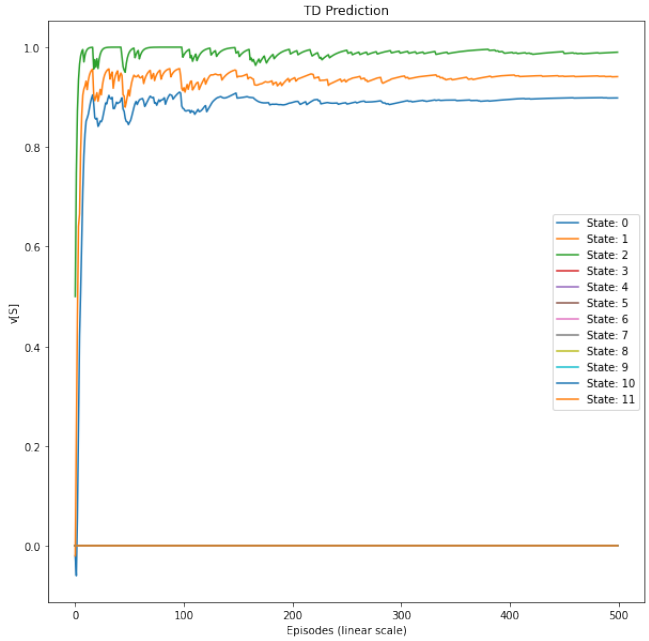
8. MC-FVMC Estimate



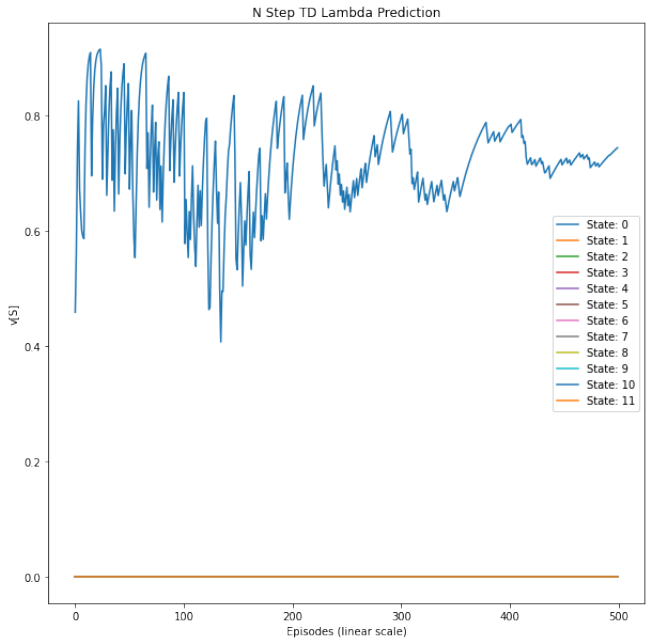
9. MC-EVMC Estimate



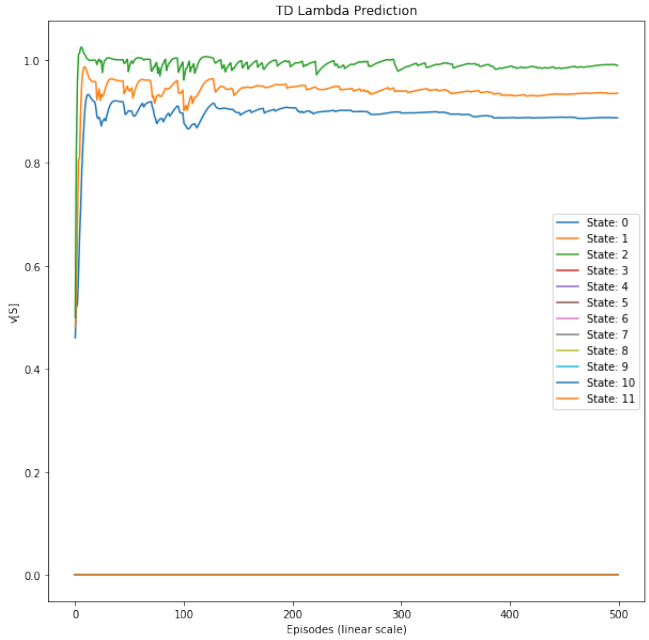
10. TD Estimate



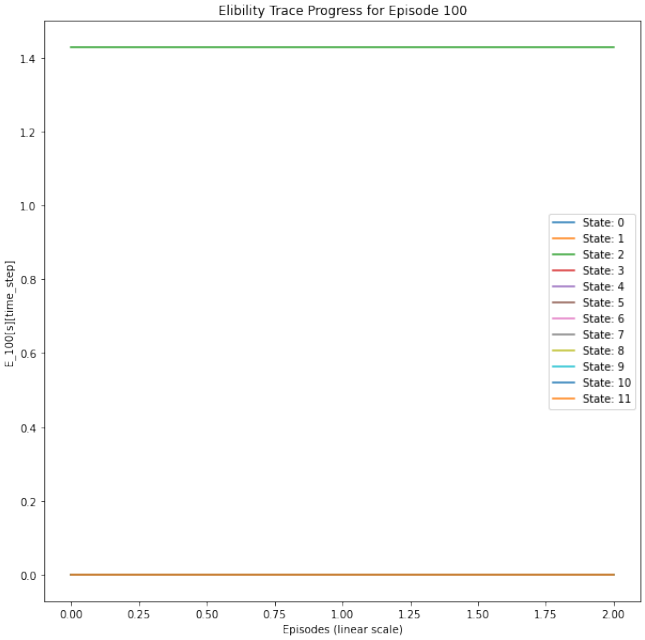
11. n-step TD Estimate



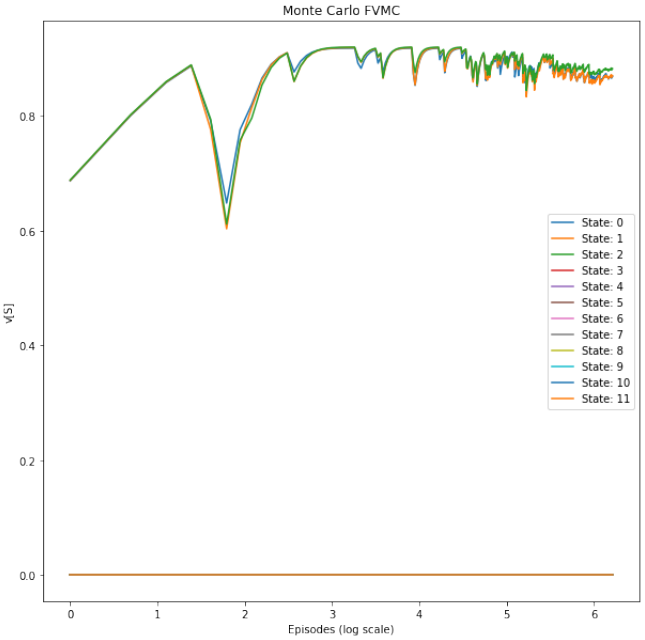
12. TD(lambda) Estimate



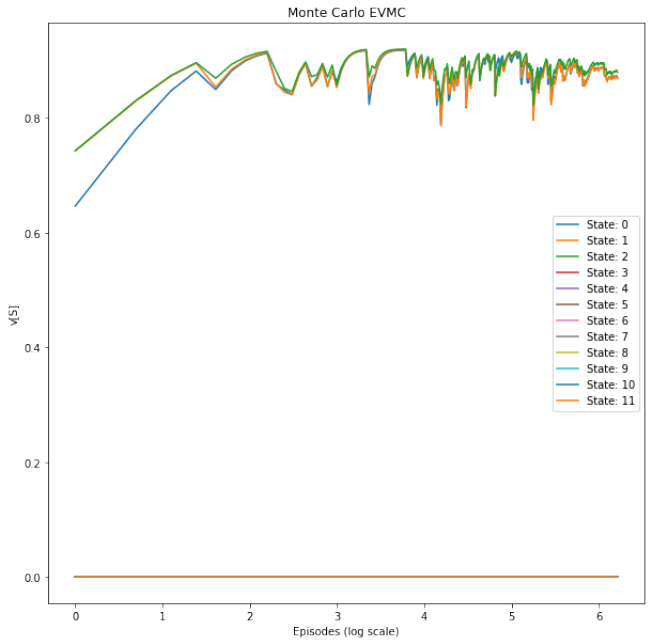
13. Eligibility Trace in TD(lambda)



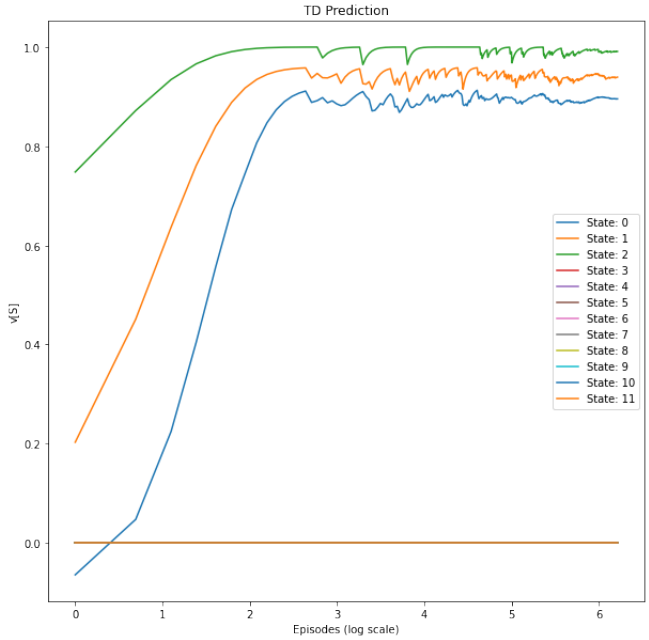
14. MC-FVMC Estimate (log scale)



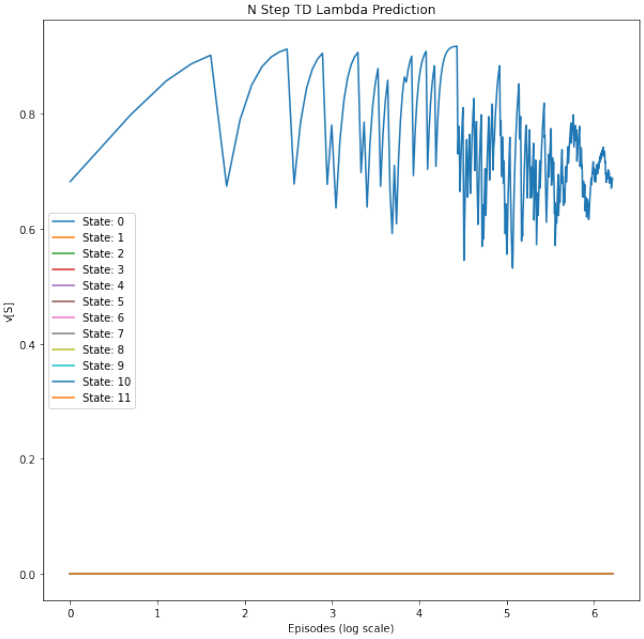
15. MC-EVMC Estimate (log scale)



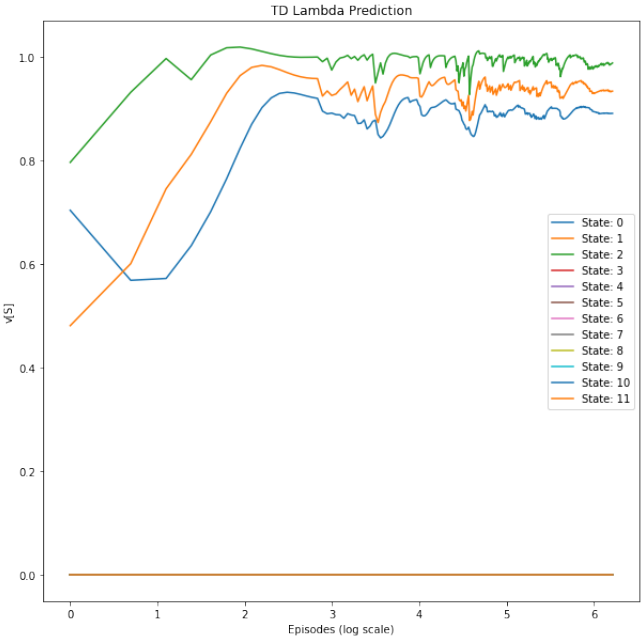
16. TD Estimate (log scale)



17. n-step TD Estimate (log scale)



18. TD(lambda) Estimate (log scale)



19. Comparison

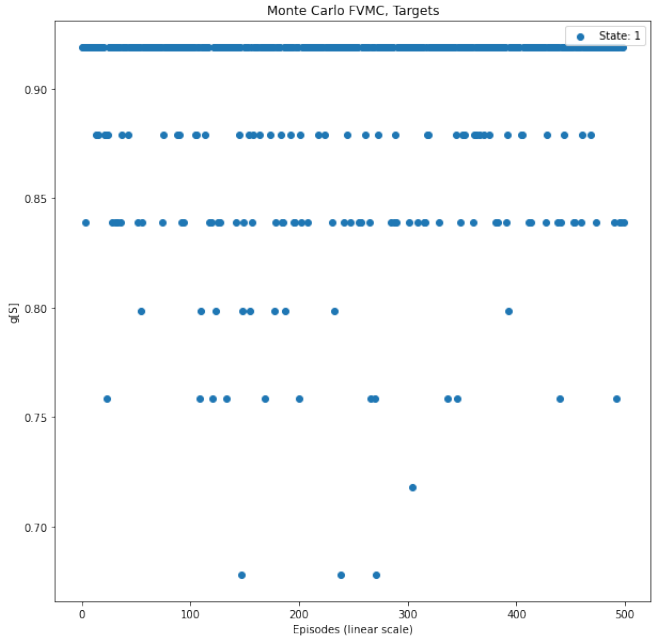
    (a) We can see good agreement between the Monte Carlo, TD methods

    (b) We find that TD Estimates are better at separating values for similar states and give clearer indications of the actual value functions, much earlier than the iterations we performed for Monte Carlo methods

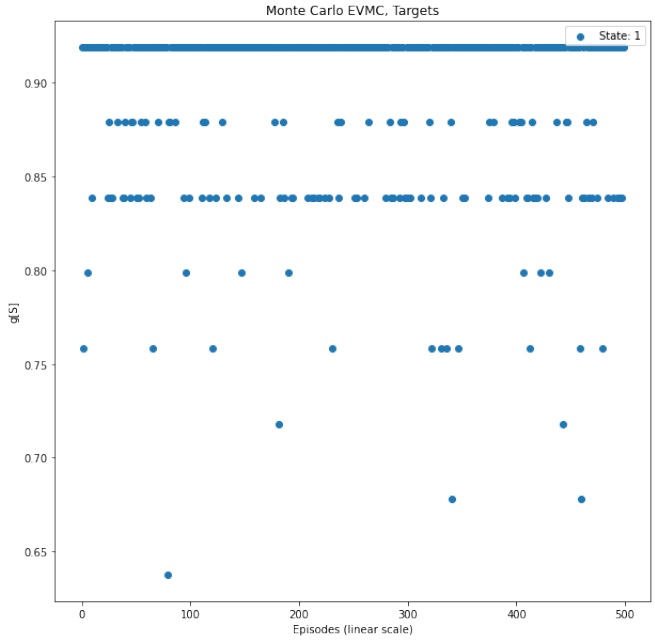
    (c) Most of the difference between the three TD Methods is only apparent for the initial epsiodes, and after that all methods converge to similar values

    (d) It is easier to observe these starting differences on the log scale plots. We see that TD(lambda) performs the best from the starting, while TD(0) has erroneous values at the start. Monte Carlo methods by virtue of sampling entire trajectories, fair better but at a much greater computational cost

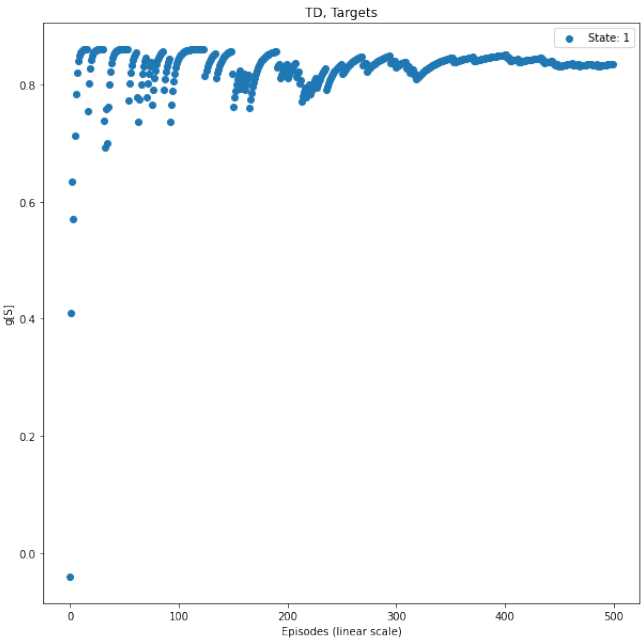
20. MC-FVMC Target



21. MC-EVMC Target



22. TD Target



23. Comparison

    (a) We see that TD methods are stabler to calculating target values from the start

    (b) No clear patterns can be observed for targets in the Monte Carlo methods except the fact that for most of the episodes exactly correct values are discerned, while due to randomness of the environment, some episodes have severely incorrect estimates

    (c) Overall we can say Targets in TD are stabler and monotonically attain the true values, while this stability is missing for Monte Carlo Targets