CS698R: Deep Reinforcement Learning

Mid-Semester Exam

Name: Shreyansh Pachauri
Roll NO.: 200954

Solution to Problem 1: Random-Maze Environment Implementation

1. To test the correct implementation of the I generated the trajectory of the Random movement Policy and Always Top Policy multiple times and checked the trajectories obtained.

```
generateTrajectory(env)
```

```
[(1, (8, 'Top', -0.04, 4)),
 (2, (4, 'Top', -0.04, 1)),
 (3, (1, 'Top', -0.04, 1)),
 (4, (1, 'Top', -0.04, 2)),
 (5, (2, 'Top', 1, 3))]
```

Figure 1: Always Top Policy

Solution to Problem 2: RME Optimal Policy via Dynamic Programming

1. There are Two Initial Policies. First one was the "Always Top" Policy and the second one was "Random Movement Policy". The results for Policy Iteration Algorithm are as follows:

```
PolicyIteration(env)[1]
```

```
{0: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 1: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 2: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
 3: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 4: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 5: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 6: {'Top': 0, 'Right': 0, 'Left': 1, 'Bottom': 0},
 7: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 8: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 9: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 10: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 11: {'Top': 0, 'Right': 0, 'Left': 0, 'Bottom': 1}}
```

Figure 2: Initial Policy = "Always Top" for Policy Iteration

```
(array([0.53025943, 0.53025943, 0.53025943, 0.53025943, 0.53025943,
        0.53025943, 0.53025943, 0.53025943, 0.53025943, 0.53025943,
        0.53025943, 0.53025943]),
 {0: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  1: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  2: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
  3: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  4: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  5: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  6: {'Top': 0, 'Right': 0, 'Left': 1, 'Bottom': 0},
  7: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  8: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  9: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  10: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  11: {'Top': 0, 'Right': 0, 'Left': 0, 'Bottom': 1}})
```

Figure 3: Initial Policy = "Random Movement" for Policy Iteration

2. Again for Value Iteration we have same two Initial Policies

```
ValueIteration(env, policy=default_policy)

{0: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 1: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 2: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
 3: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 4: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 5: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 6: {'Top': 0, 'Right': 0, 'Left': 1, 'Bottom': 0},
 7: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 8: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 9: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
 10: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
 11: {'Top': 0, 'Right': 0, 'Left': 0, 'Bottom': 1}}
```

Figure 4: Value Iteration for "Always Top"

```
(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 {0: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  1: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  2: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
  3: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  4: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  5: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  6: {'Top': 0, 'Right': 0, 'Left': 1, 'Bottom': 0},
  7: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  8: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  9: {'Top': 0, 'Right': 1, 'Left': 0, 'Bottom': 0},
  10: {'Top': 1, 'Right': 0, 'Left': 0, 'Bottom': 0},
  11: {'Top': 0, 'Right': 0, 'Left': 0, 'Bottom': 1}})
```

Figure 5: Value Iteration for "Random Walk"

3.

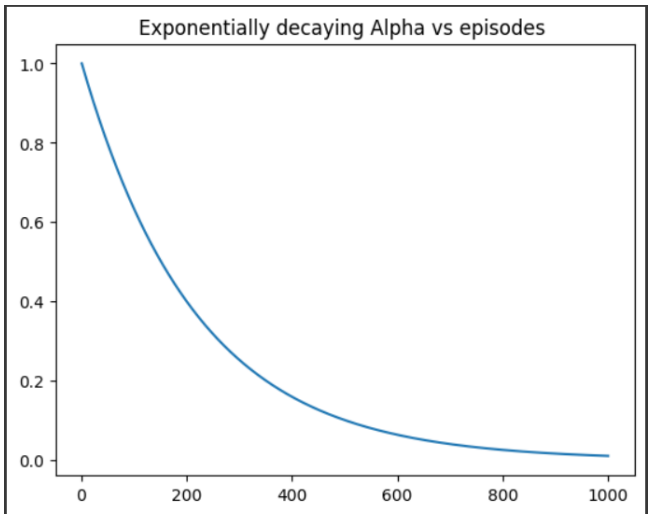
Solution to Problem 3: RME Prediction with MDP Unknown

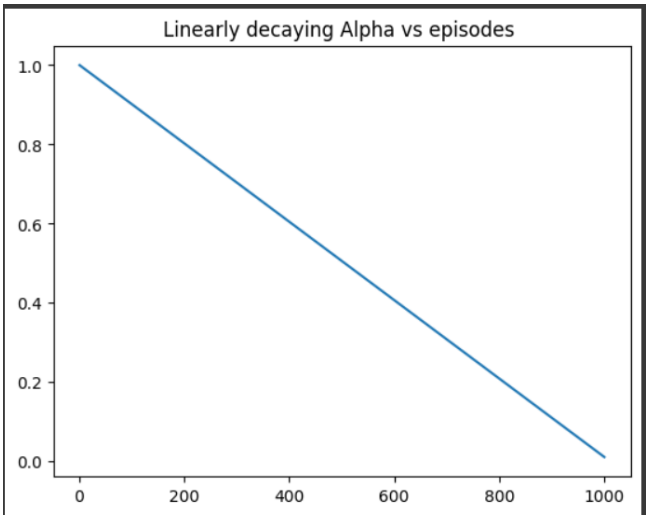
1. The generate trajectory function is created:

```
def generateTrajectory(env, policy = random_policy, maxSteps=50):

  trajectory = []

  state = env.reset()[0]["agent"]

  i=0

  for _ in range(maxSteps):
    i = i+1
    action = max(policy[state], key = policy[state].get)
    next_state, reward, terminated, truncated, info = env.step(action)
    trajectory.append((i, (state, action, reward, next_state["agent"])))
    state = next_state["agent"]
    if terminated:
      break

  if not terminated:
    return []

  return trajectory

generateTrajectory(env)
```

Figure 6: Generate Trajectory

2. Solution 3.2





3. Solution 3.3

```
v1 = MonteCarloPrediction(env)[0]
v1
```

```
array([ 0.        ,  0.27151208,  0.39067643,  0.        , -0.16319853,
        0.        ,  0.03611094,  0.        , -0.22124397, -0.45167337,
       -0.08025853, -0.88328505])
```

4. Solution 3.4

```
v2 = TemporalDifferencePrediction(env)[0]
v2
```

```
array([ 0.        , -0.10787927,  0.20420851,  0.        , -0.16865512,
        0.        ,  0.05910215,  0.        , -0.25838285, -0.3759415 ,
       -0.11254661, -0.86998372])
```

5. Solution 3.5

```
v3 = nStepTemporalDifferencePrediction(env=env, policy=default_policy)[0]
v3
```

```
array([ 0.        , -0.43587709, -0.17131145,  0.        , -0.53017683,
        0.        , -0.13145131,  0.        , -0.57161289, -0.59931518,
       -0.39591623, -0.23834497])
```

6. Solution 3.6

```
TDLambdaPrediction(env)[0] # TD Lambda over 50000 episodes
```

```
array([ 0.        , -0.13510276,  0.31170665,  0.        , -0.19555978,
        0.        ,  0.04639679,  0.        , -0.2463109 , -0.3657551 ,
       -0.12512428, -0.84967187])
```

7. Solution 3.7 To find the True Value of various states I ran all the agents over 50000 iterations and averaged them out since as the number of iterations tend to infinity the estimated values tend to the true values (By the law of large numbers) and hence the true values were found. I also estimated the Standard Deviation in the values in all the states got through all the agents

```
Calculate the true value of state for each state. Implement it in the code and show the full derivation in the report.

[110] v1 = MonteCarloPrediction(env, firstVisit=True)[0] #FVMC over 50000 episodes
      v2 = MonteCarloPrediction(env)[0]    #EVMC over 50000 episodes
      v3 = TemporalDifferencePrediction(env)[0] #TD Learning over 50000 episodes
      v4 = nStepTemporalDifferencePrediction(env)[0] #n-Step TD Learning over 50000 episodes
      v5 = TDLambdaPrediction(env)[0] # TD Lambda over 50000 episodes
```

The averaged-out standard deviations and the true values are as follows respectively:

```
print(rowstd)
print(mean)
```

```
[0.         0.17716229 0.1134746  0.         0.04619189 0.
 0.02049302 0.         0.04884287 0.06175088 0.05284055 0.0735926 ]
[ 0.         -0.10818966  0.21093021  0.         -0.25320101  0.
  0.02936661  0.         -0.29697022 -0.47990254 -0.1625246  -0.78787469]
```

8. Solution 3.8

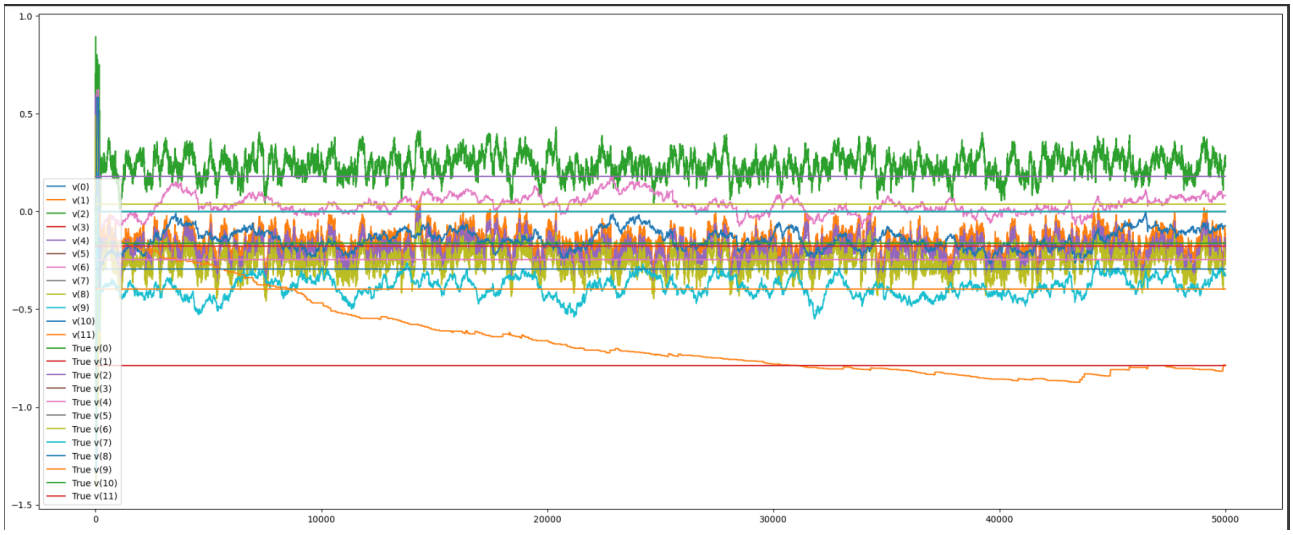


Figure 7: 3.8

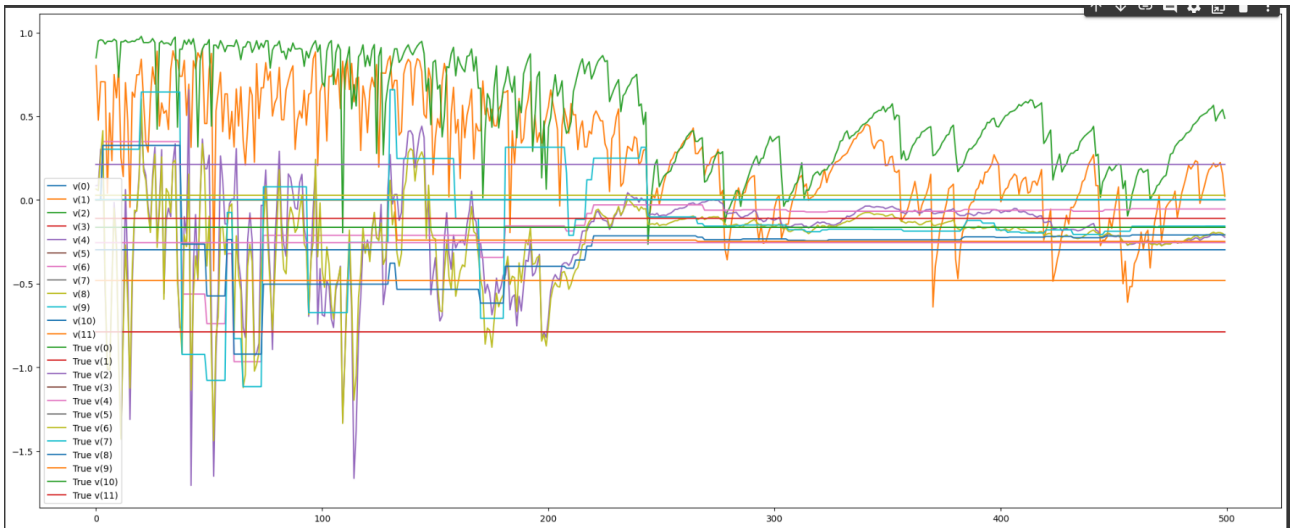


Figure 8: MC-EVMC Reward vs Episodes

9. Solution 3.9

10. Solution 3.10

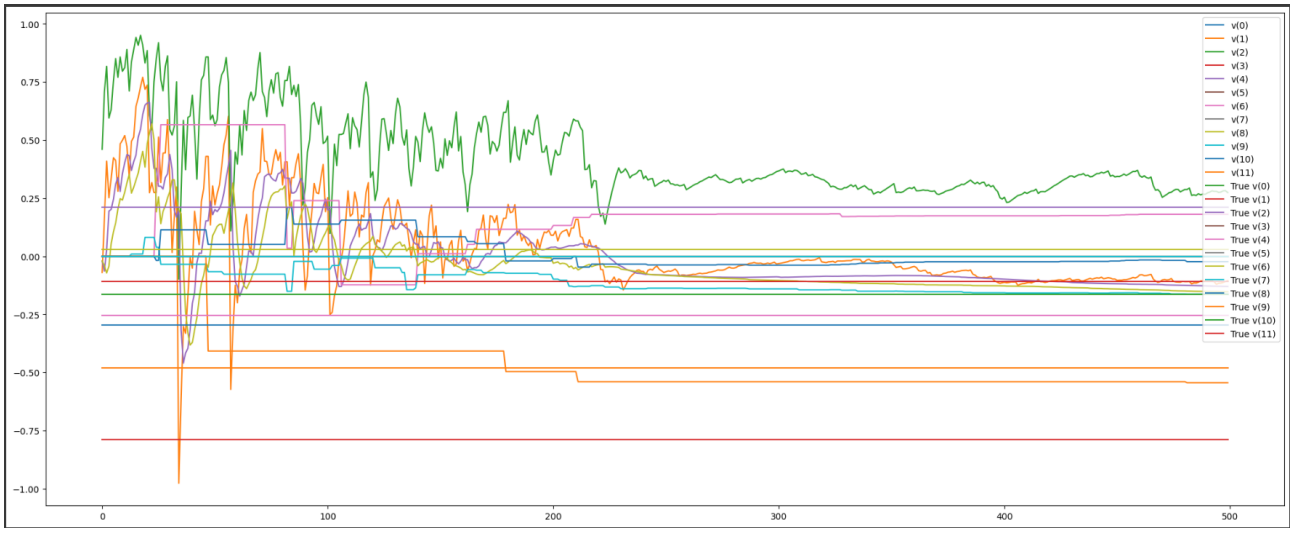


Figure 9: TD Estimate Reward vs Episodes

11. Solution 3.11

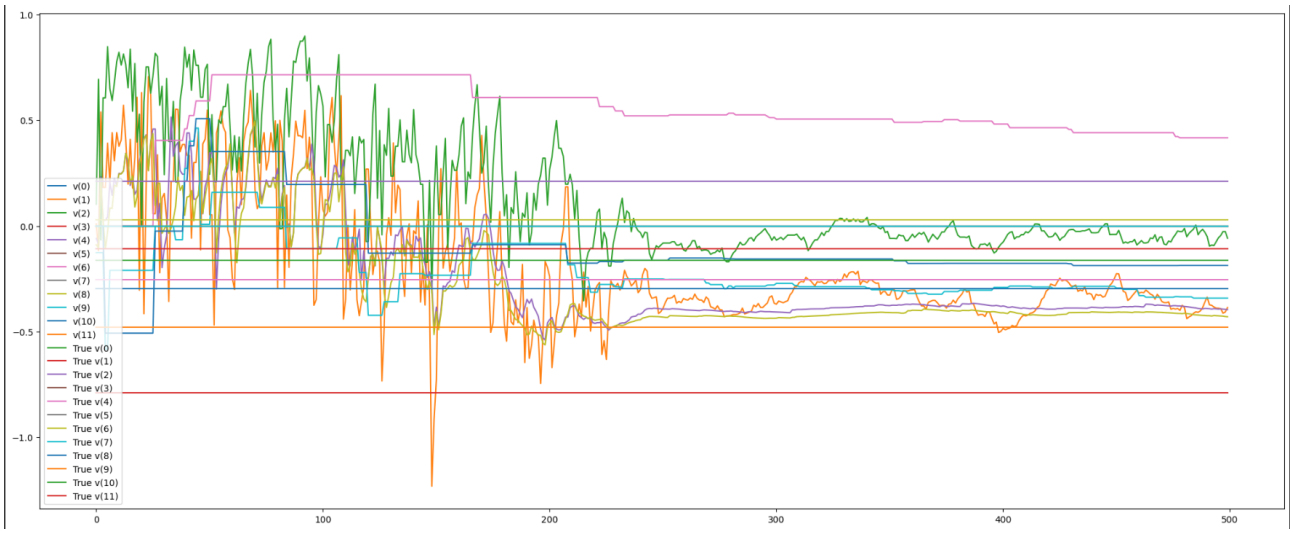


Figure 10: n-step TD Learning Reward vs Episodes

12. Solution 3.12

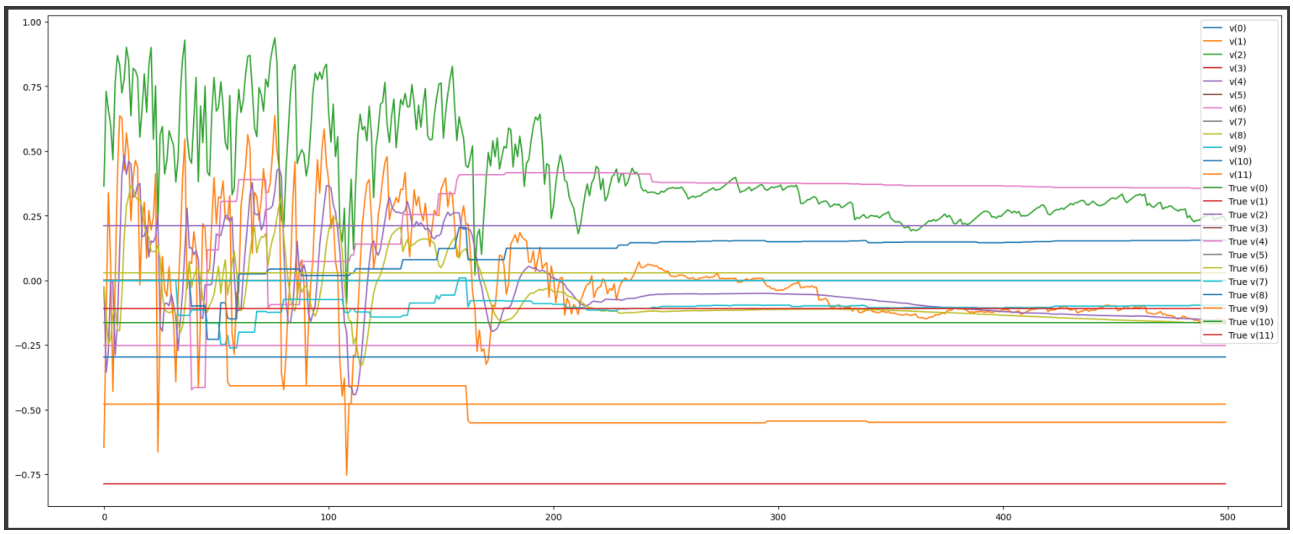


Figure 11: TD(lambda) Reward vs Episodes

13. Solution 3.13

14. Solution 3.14

15. Solution 3.15

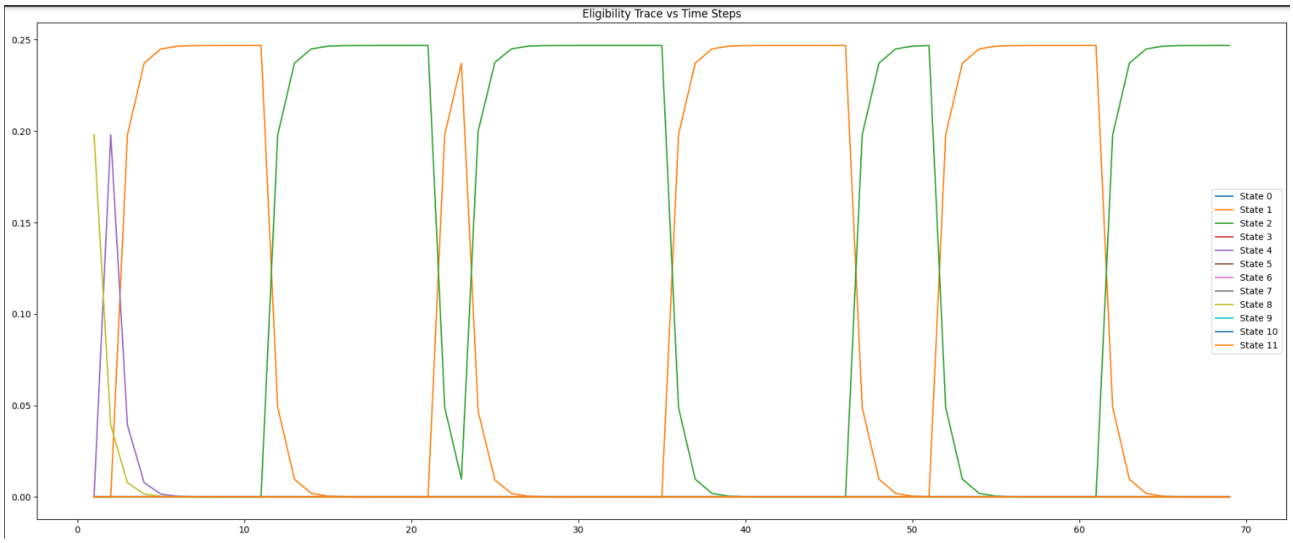


Figure 12: Eligibility Trace vs Time steps

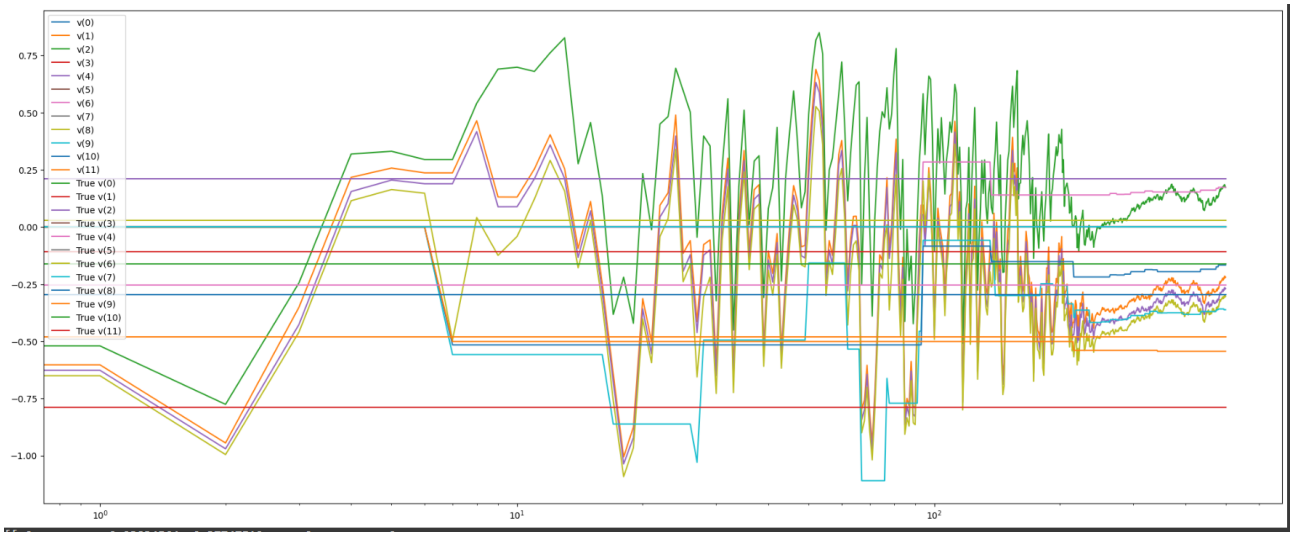


Figure 13: MC-FVMC Log scale

16. Solution 3.16
17. Solution 3.17
18. Solution 3.18

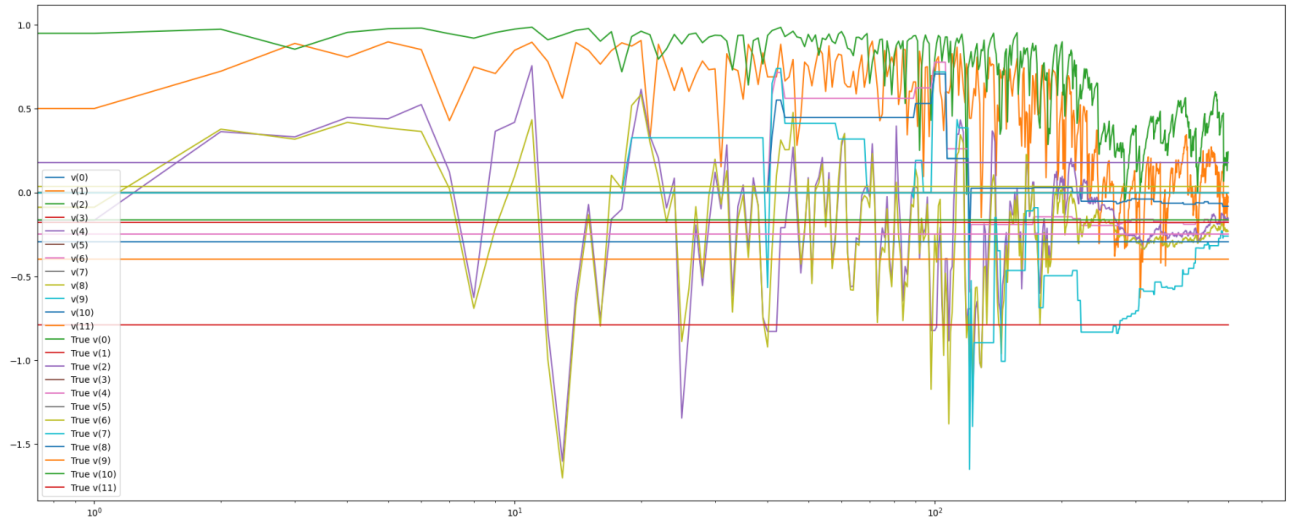


Figure 14: MC-EVMC Log scale

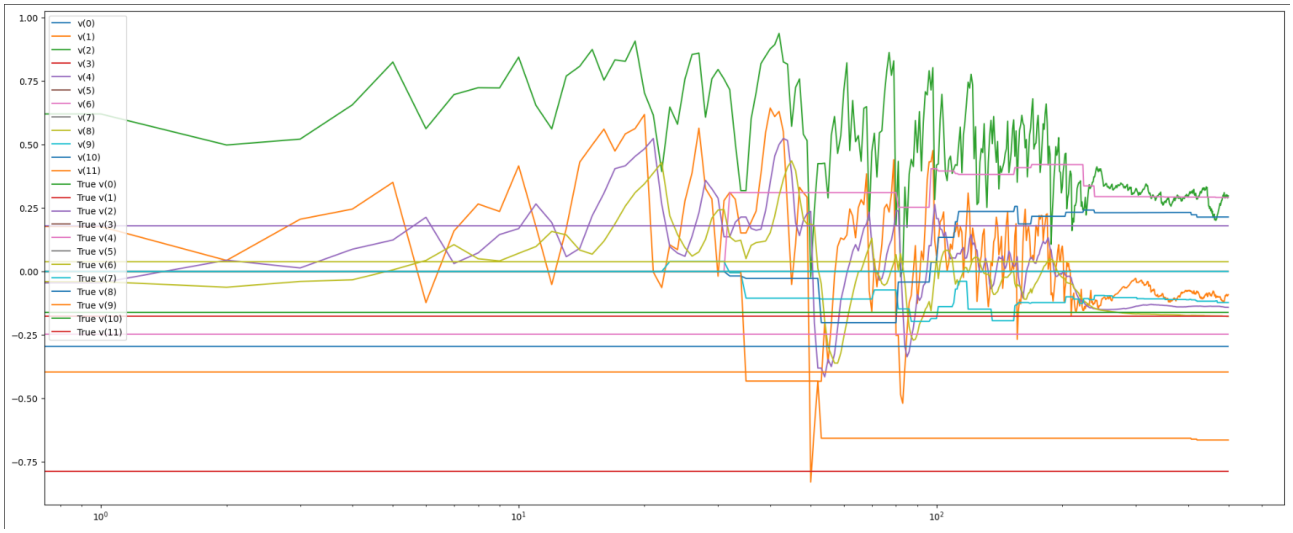


Figure 15: TD Log scale

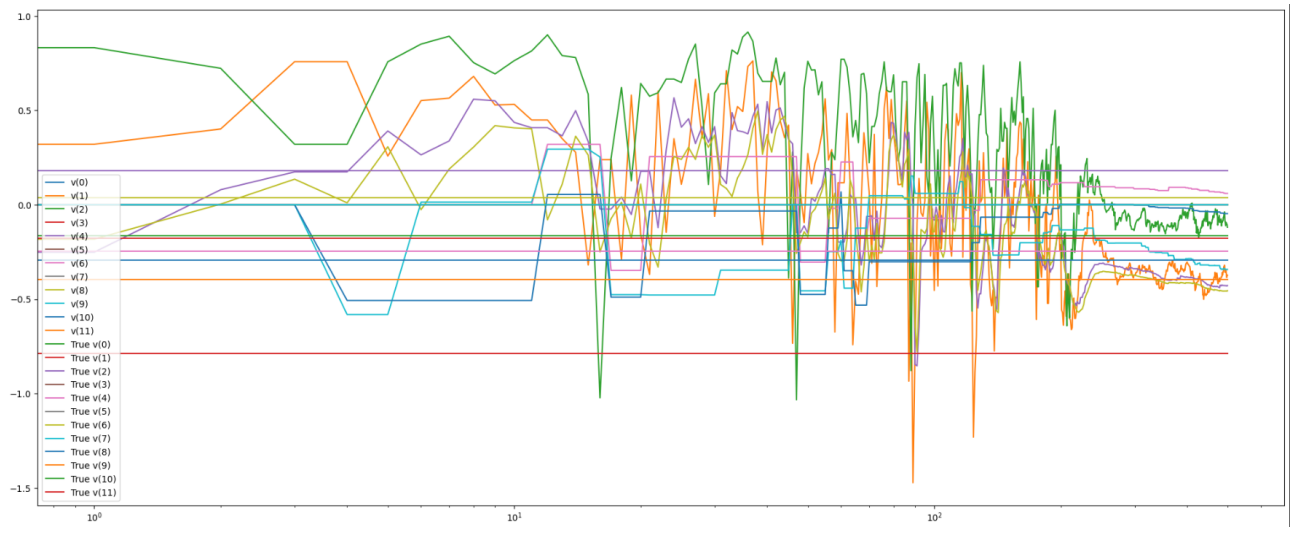


Figure 16: n-step TD Log scale

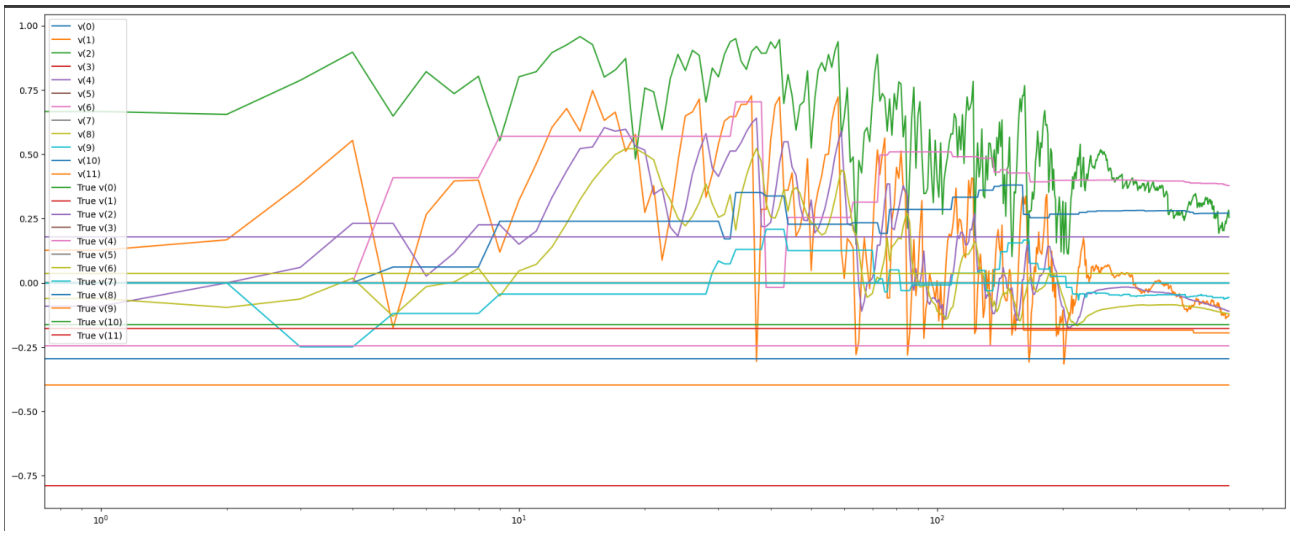


Figure 17: TD(lambda) Log scale