Adv Machine Learning Lab 3

Yash Pawar

08/10/2020

Implement the greedy and epsilon-greedy policies in the functions.

```
GreedyPolicy <- function(x, y){

  # Get a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.
  action_ind = q_table[x,y,]
  max_ind = which(action_ind==max(action_ind))

  if (length(max_ind!=1)){
    select_ind = sample(max_ind,1)
  } else{
    select_ind = max_ind
  }
  max_action = select_ind

  return(max_action)

}


EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.
```

```
  action_ind = q_table[x,y,]
  max_ind = which(action_ind==max(action_ind))

  if (length(max_ind>1)){
    select_ind = sample(max_ind,1)
  } else{
    select_ind = max_ind
  }

  if (runif(1)>epsilon){
    select_eps = select_ind
  } else{
    select_eps = sample(c(1:4),1)
  }


  return(select_eps)

}
```

Implement the Q-learning Algorithm

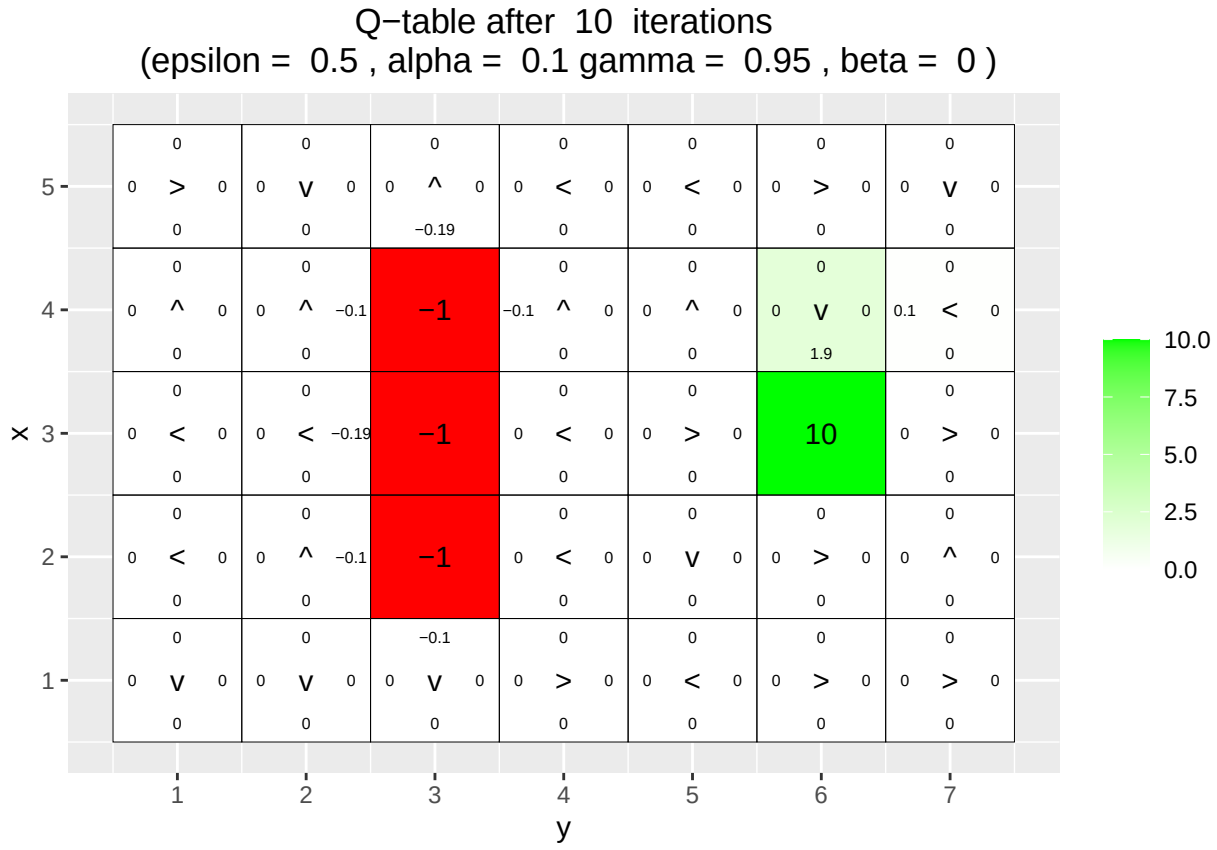
```
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
  # Args:
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting greedily.
  #   alpha (optional): learning rate.
  #   gamma (optional): discount factor.
  #   beta (optional): slipping factor.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   reward: reward received in the episode.
  #   correction: sum of the temporal difference correction terms over the episode.
  #   q_table (global variable): Recall that R passes arguments by value. So, q_table being
  #   a global variable can be modified with the superassigment operator <<-.

  # Your code here.

  S = start_state
  episode_correction = 0
  repeat{

    # Follow policy, execute action, get reward.

    # Follow policy returns the action
```

```
    action_ind = EpsilonGreedyPolicy(x = S[1],
                                     y = S[2],
                                     epsilon = epsilon)

    # trans model
    S_new = transition_model(x = S[1], y = S[2], action = action_ind, beta = beta)

    # Get reward
    reward = reward_map[S_new[1],S_new[2]]

    # Q-table update.
    max_ind = GreedyPolicy(S_new[1], S_new[2])


    correction = alpha*(reward+
                          gamma*(q_table[S_new[1],S_new[2],max_ind])-
                          q_table[S[1],S[2],action_ind])


    q_table[S[1], S[2], action_ind] <<- q_table[S[1],S[2],action_ind]+correction


    S = S_new
    episode_correction = correction + episode_correction
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
```

Environment A

1. After the first 10 iterations, it can be seen that the Q-table has not been updated much. However, the agent seems to have explored the terminal states as the rewards for the states adjacent to it have been updated.
2. After 10000 iterations, the agent has learned to completely ignore the terminal states with negative rewards. And the rewards closer to the terminal states with maximum reward seems to have maximized. However, it does not seem to have picked the direction that maximizes the reward for every state (even close to the terminal state). Thus, the policy does not seem to be optimal.
3. The agent learns explores multiple paths to get the positive reward. However, after running through multiple episodes, it converges to a single policy that maximizes the returns.

This can be explained by the fact that the agent is allowed to explore all the paths and not the only ones that maximize the reward. to make the agent strictly follow the path with positive reward, it needs to learn the policy that follows the path with maximum reward in each state. However, this would add a lot of bias and the agent, in that case will not explore all the states thus removing randomness.


Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
−1
10
−0.19
1.9
10.0
7.5
5.0
2.5
0.0


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
−1
10

Environment B

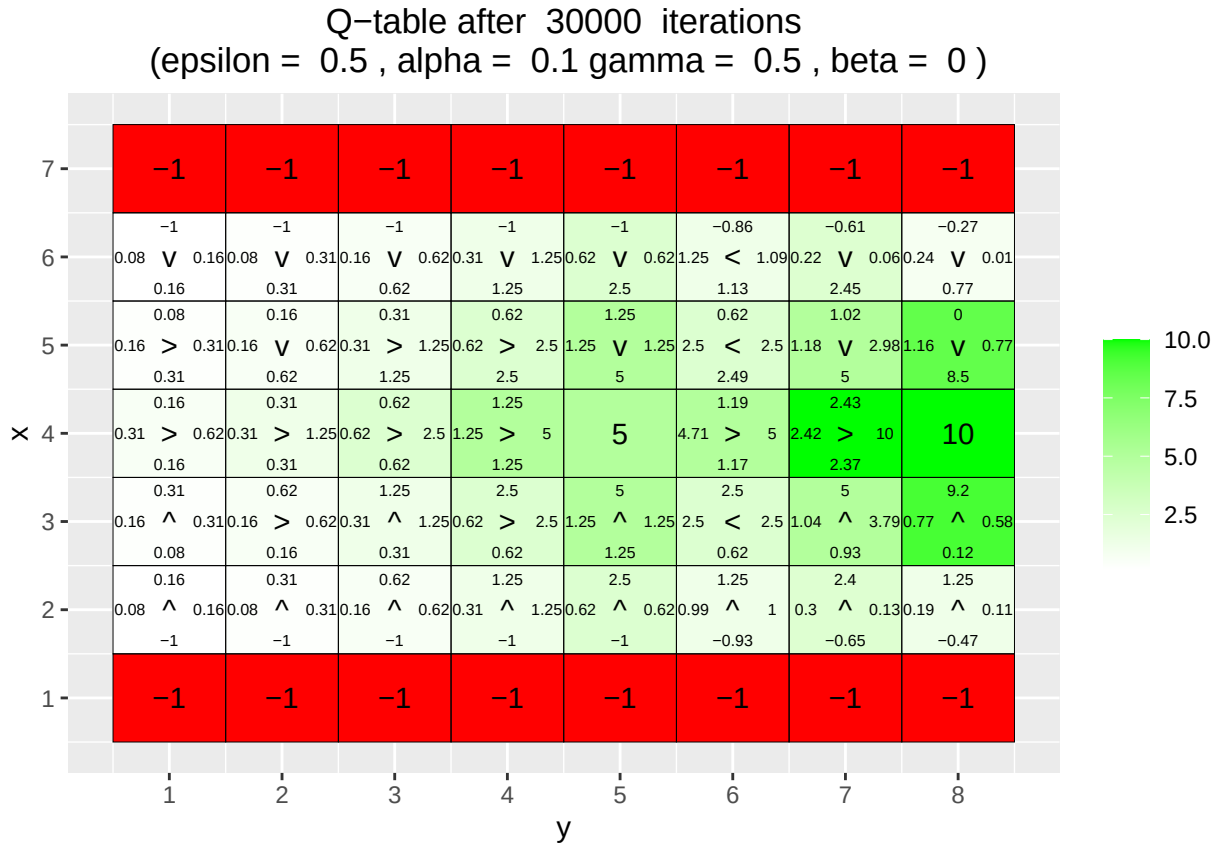
Epsilon = 0.5, gamma = c(0.5,0.75,0.95)

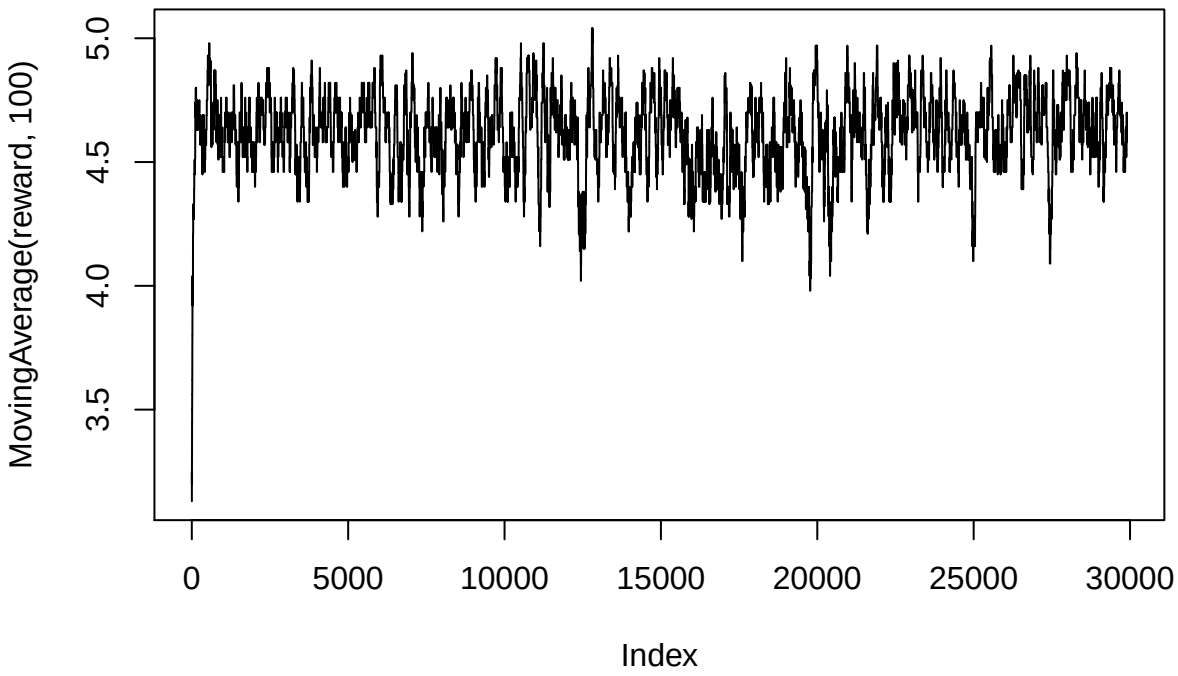
It is observed that higher gamma values allow future rewards i.e the expected returns in that case are high, whereas when the gamma values are low, immediate rewards are preferable.

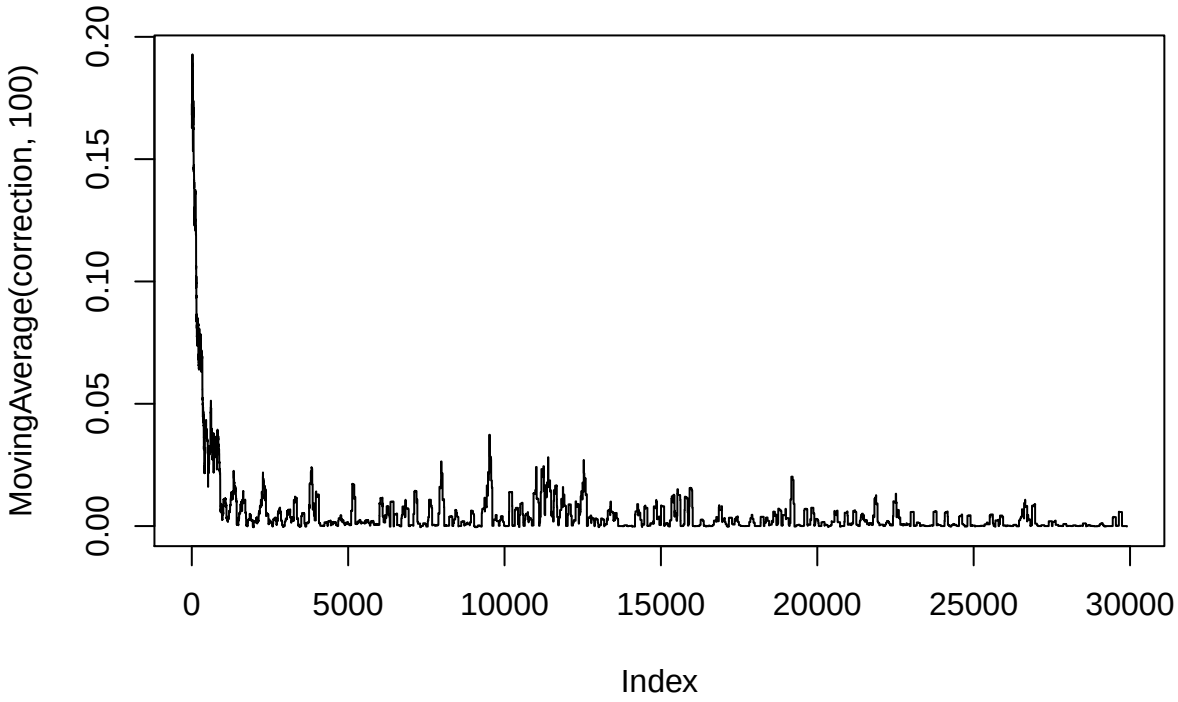
The actions are chosen at random in each states at random with probability 0.5

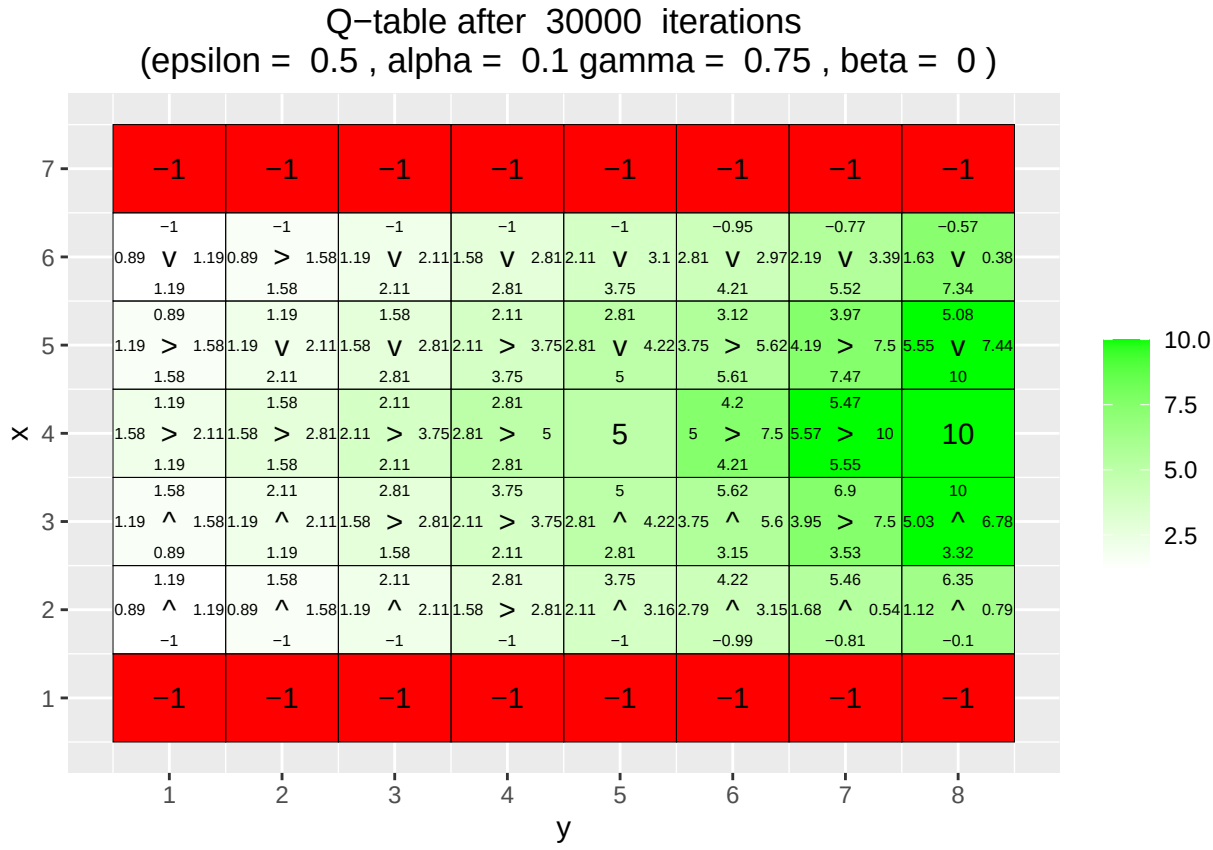
Thus we can observe from the graph that the expected returns at the 30000th iteration for gamma = 0.5 are not as high as compared to the ones with gamma = 0.95.

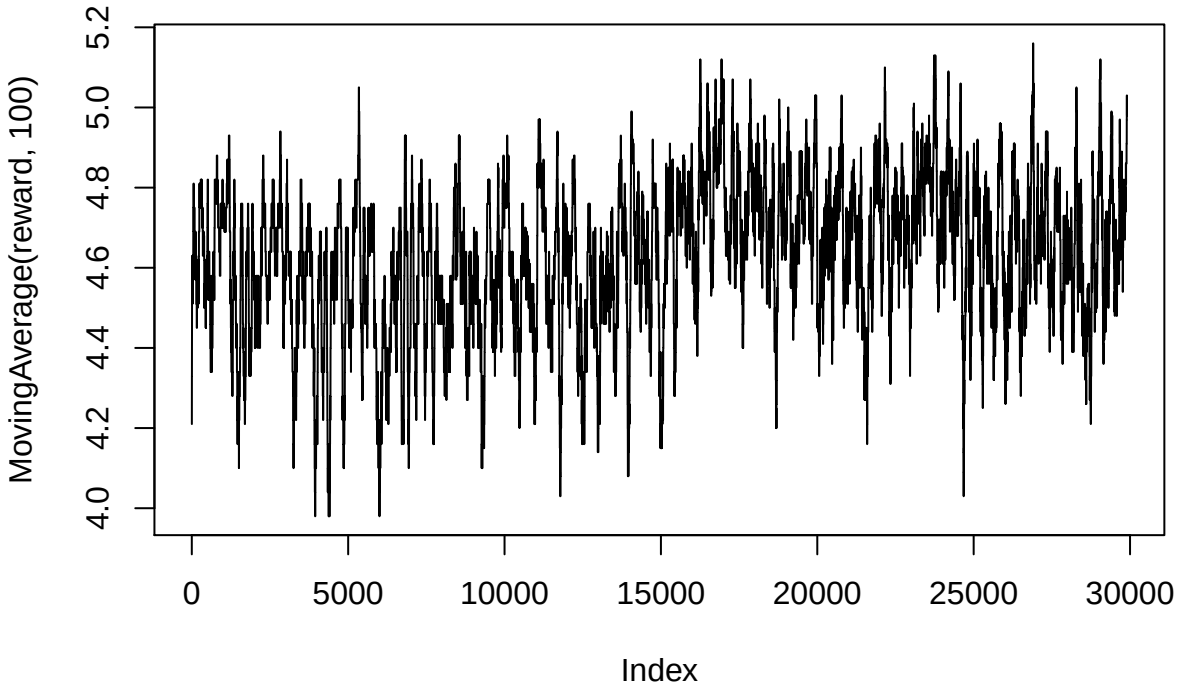
In case of gamma =0.95, the expected returns are maximum as it does not diminish the future rewards as significantly as for the ones with lower epsilon values. Thus we can see that the agent completely avoids the state with reward 5 when it converges to a final policy. Also it can be seen that the moving average for reward seems to have increased once it learned that avoiding the state 5 can maximize the rewards. Also the correction seems to have increased.

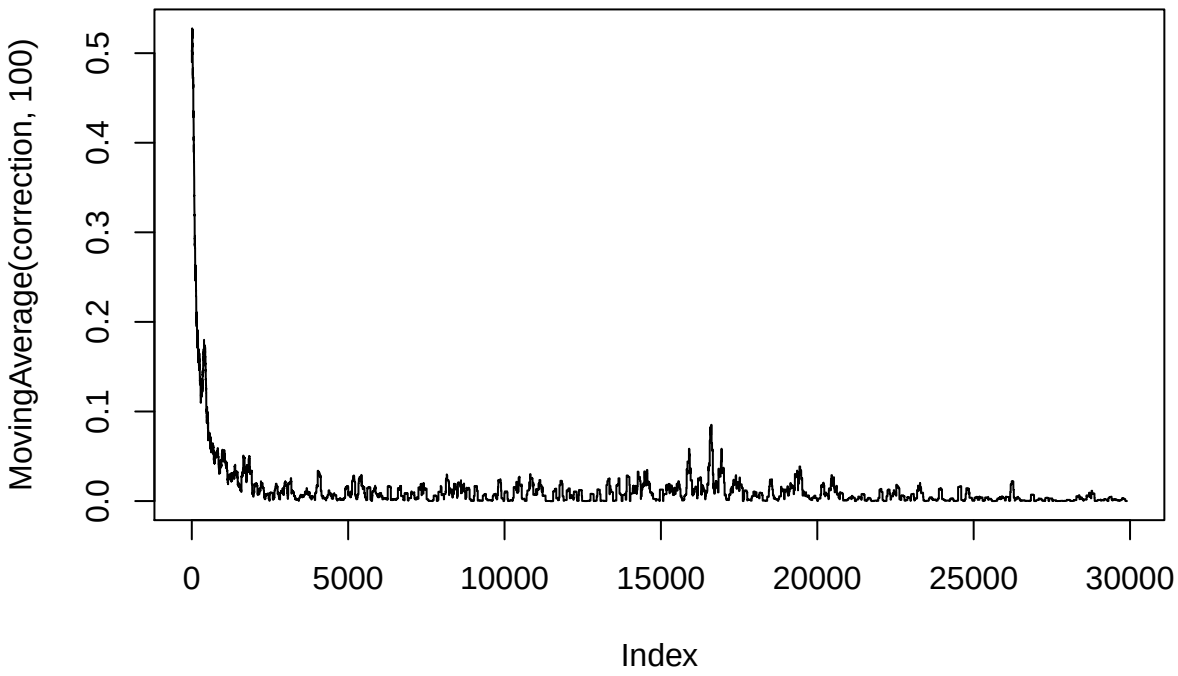

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

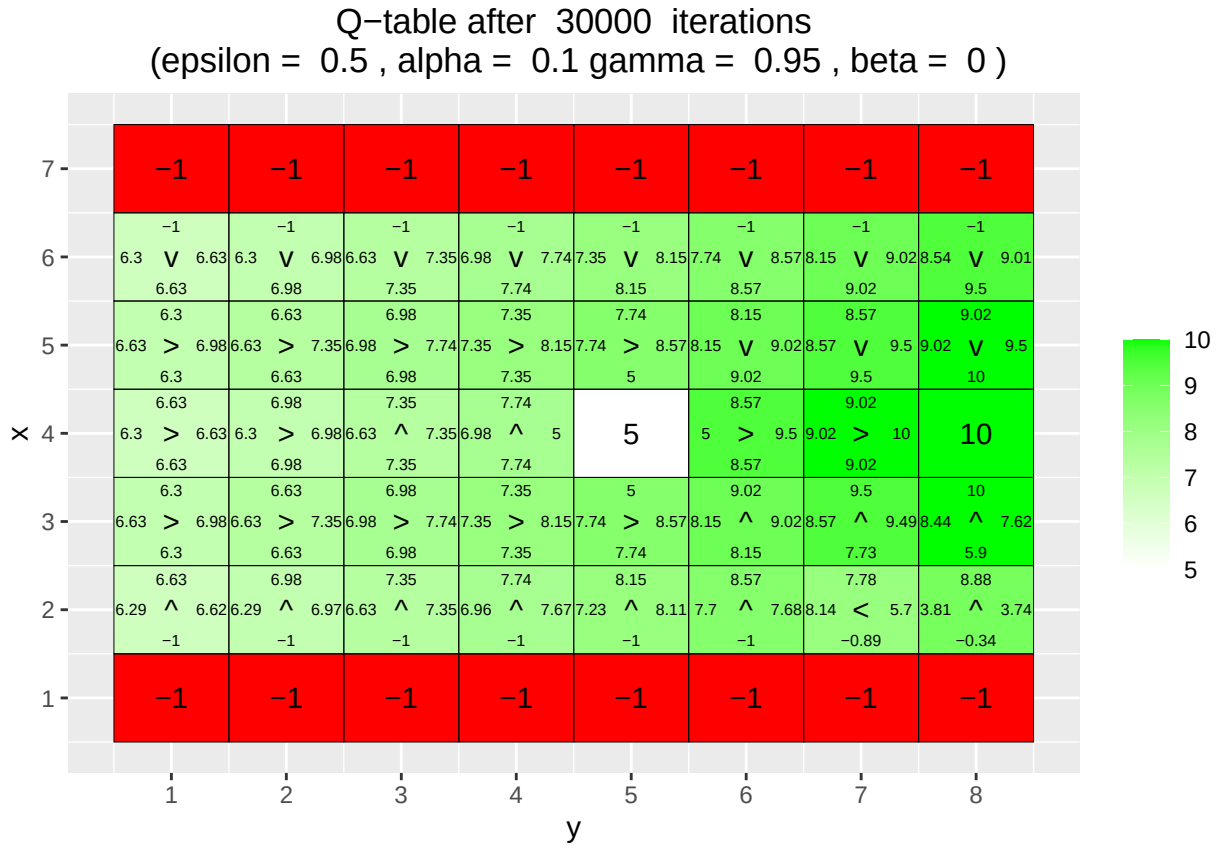

MovingAverage(reward, 100)
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

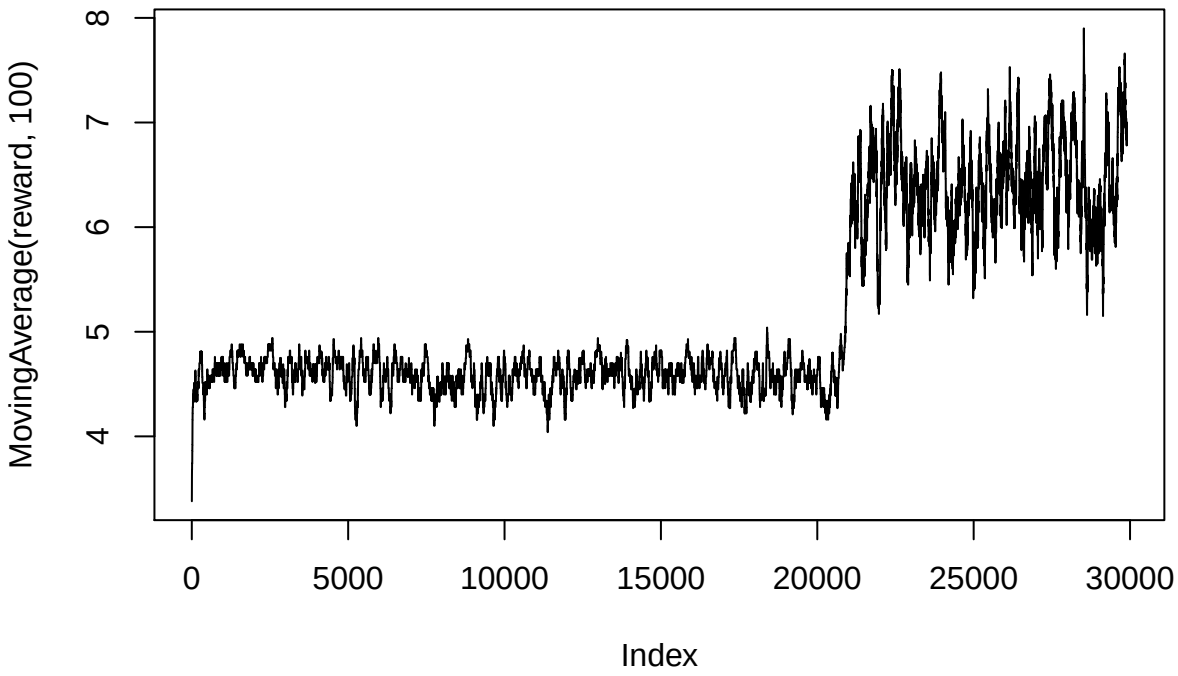

MovingAverage(correction, 100)
0.00
0.05
0.10
0.15
0.20
0
5000
10000
15000
20000
25000
30000
Index

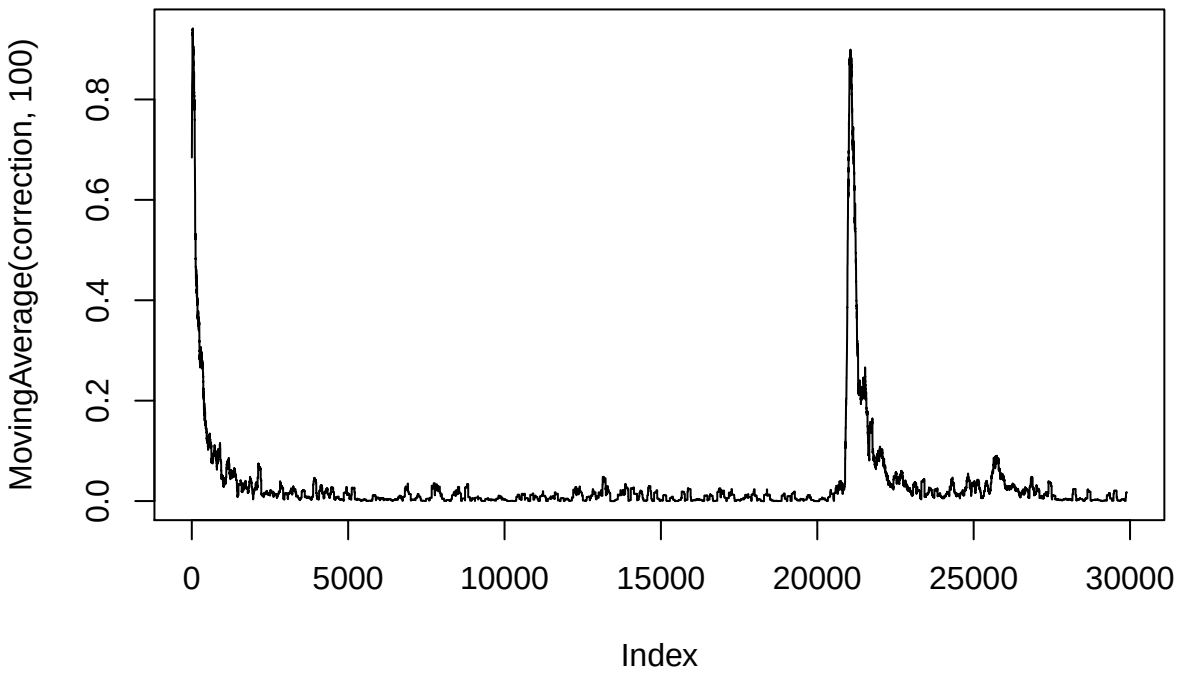

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
5.2
0
5000
10000
15000
20000
25000
30000
Index


MovingAverage(correction, 100)
0.0
0.1
0.2
0.3
0.4
0.5
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
10
9
8
7
6
5
−1
−1 6.3 V 6.63 6.63
−1 6.3 V 6.98 6.98
−1 6.63 V 7.35 7.35
−1 6.98 V 7.74 7.74
−1 7.35 V 8.15 8.15
−1 7.74 V 8.57 8.57
−1 8.15 V 9.02 9.02
−1 8.54 V 9.01 9.5
6.3 6.63 > 6.98 6.3
6.63 6.63 > 7.35 6.63
6.98 6.98 > 7.74 6.98
7.35 7.35 > 8.15 7.35
7.74 7.74 > 8.57 5
8.15 8.15 V 9.02 9.02
8.57 8.57 V 9.5 9.5
9.02 9.02 V 9.5 10
6.63 6.3 > 6.63 6.63
6.98 6.3 > 6.98 6.98
7.35 6.63 ^ 7.35 7.35
7.74 6.98 ^ 5 7.74
5
8.57 5 > 9.5 8.57
9.02 9.02 > 10 9.02
10
6.3 6.63 > 6.98 6.3
6.63 6.63 > 7.35 6.63
6.98 6.98 > 7.74 6.98
7.35 7.35 > 8.15 7.35
5 7.74 > 8.57 7.74
9.02 8.15 ^ 9.02 8.15
9.5 8.57 ^ 9.49 7.73
10 8.44 ^ 7.62 5.9
6.63 6.29 ^ 6.62 −1
6.98 6.29 ^ 6.97 −1
7.35 6.63 ^ 7.35 −1
7.74 6.96 ^ 7.67 −1
8.15 7.23 ^ 8.11 −1
8.57 7.7 ^ 7.68 −1
7.78 8.14 < 5.7 −0.89
8.88 3.81 ^ 3.74 −0.34
−1

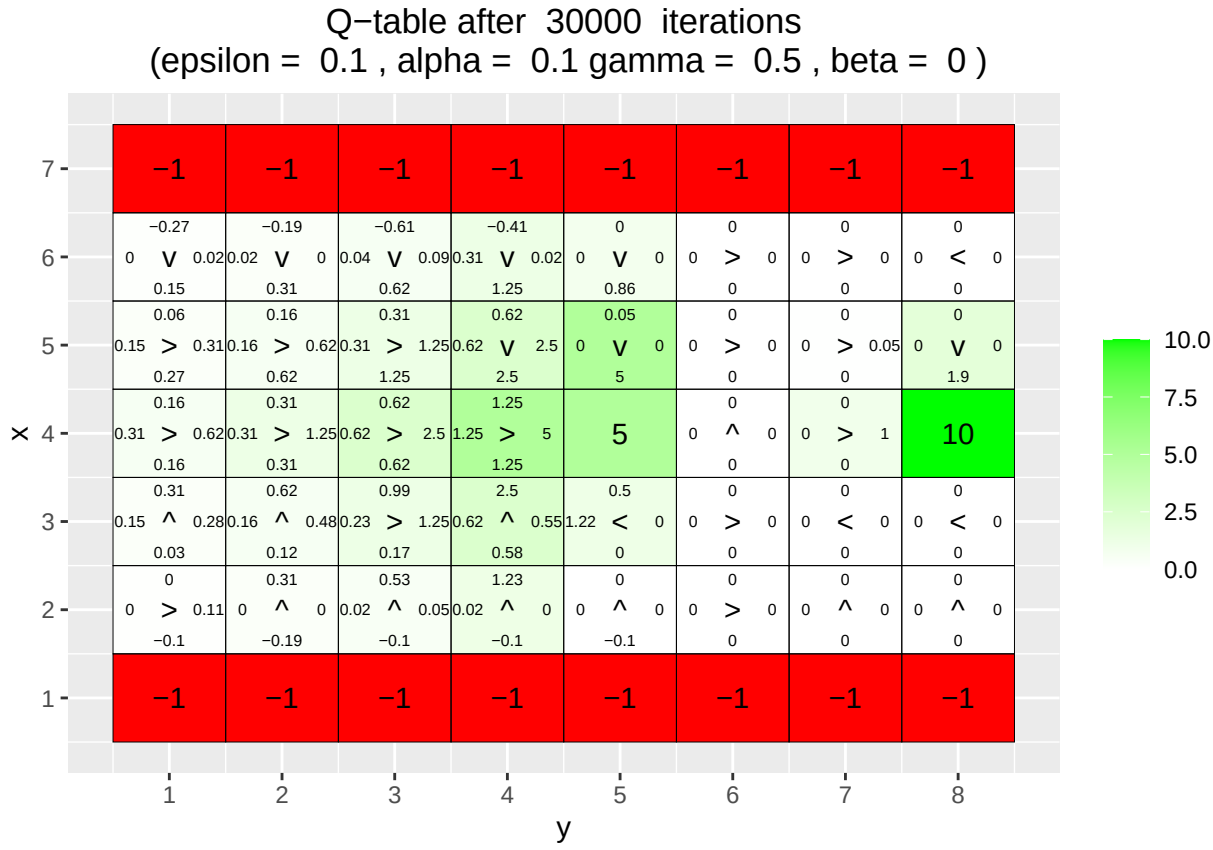

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index

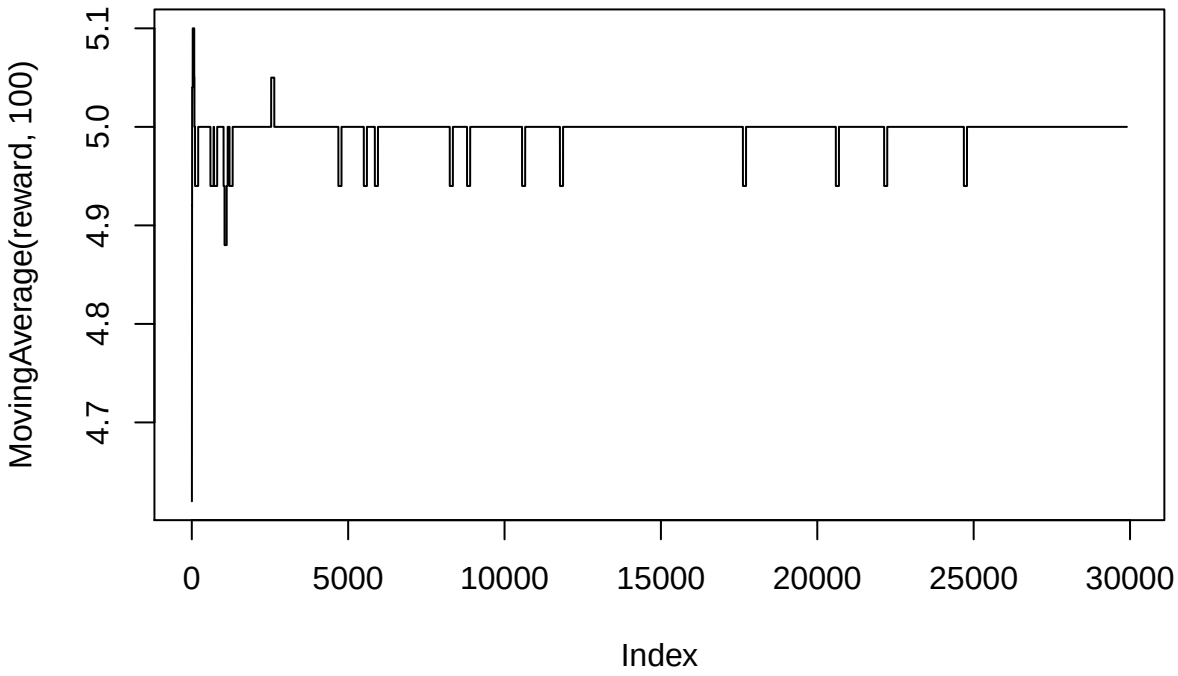


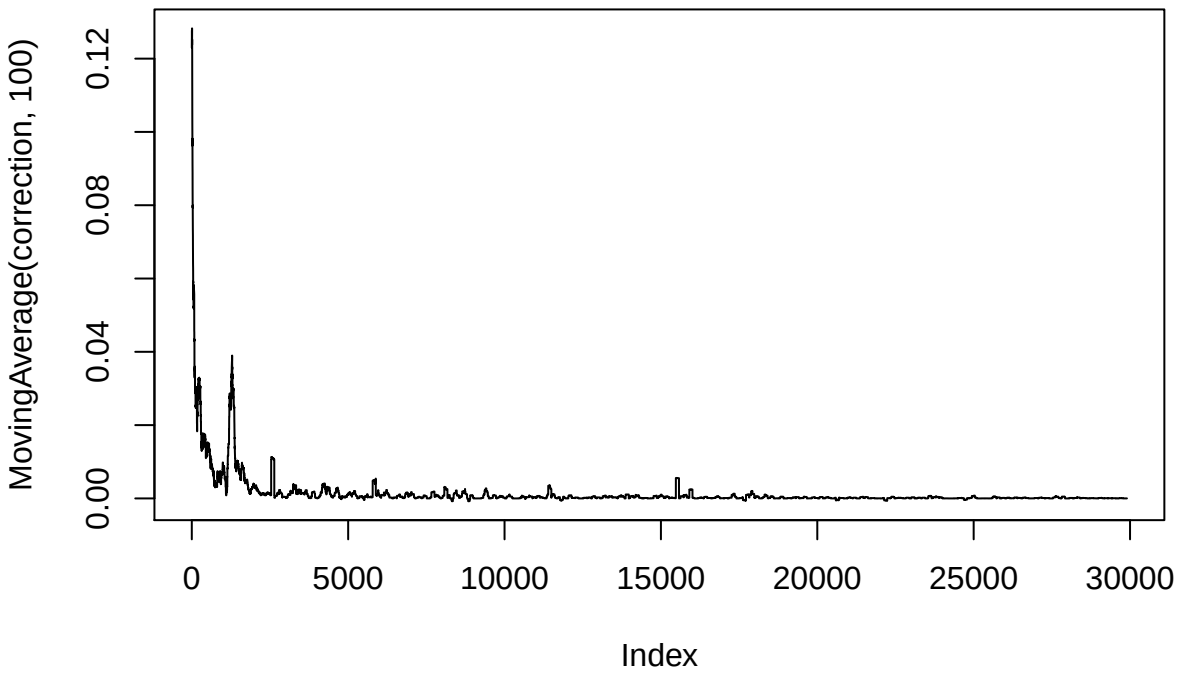
In all the cases below the epsilon = 0.1 signifies that it picks out random actions with probability 0.1 Thus, it implies that it picks out the action with maximum reward in each state predominantly and the reward Q values for the selected actions are very high as compared to the other actions in that state.

Epsilon = 0.1, gamma = 0.5

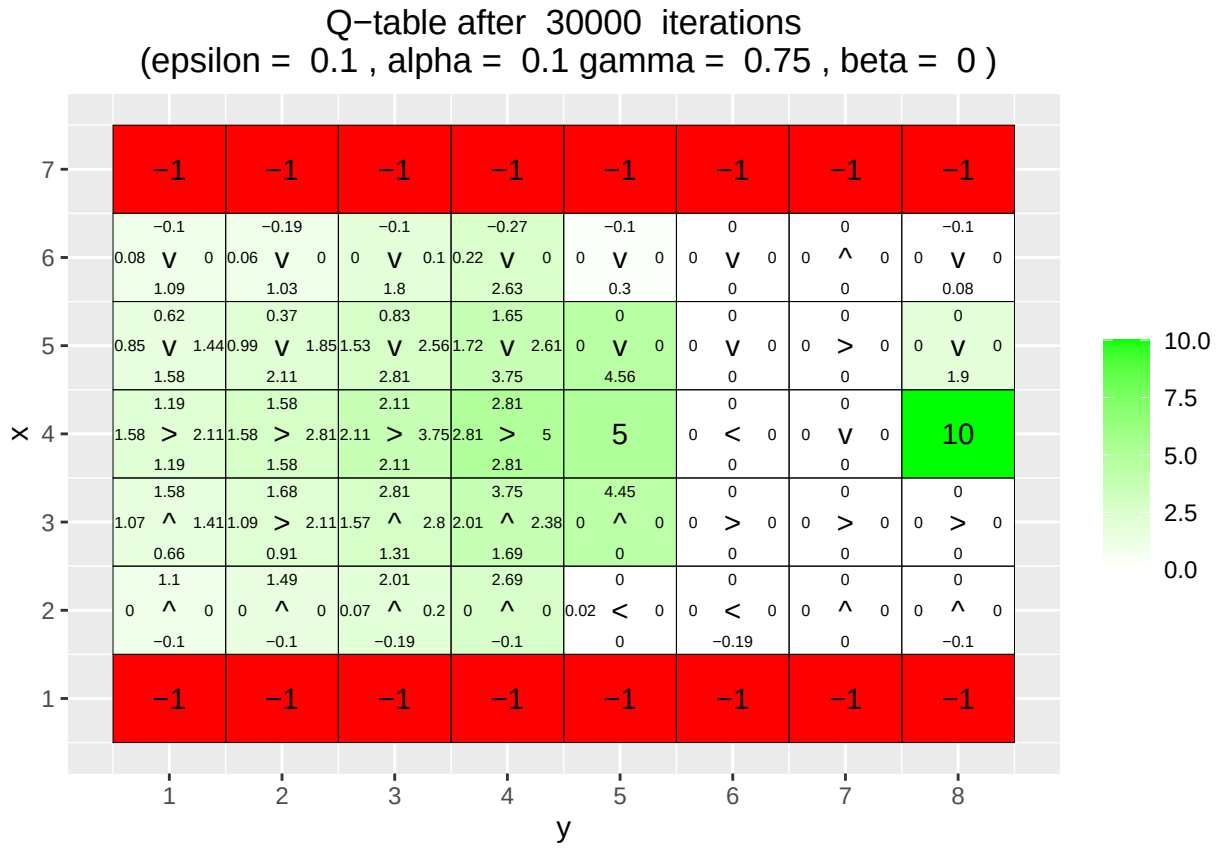
The moving avg for reward and correction in this case does not seem to have changed a lot and is quite stable at 5. This is because it does not explore different states with a very high probability and its motion is restricted only in the direction that maximizes the Q-value.

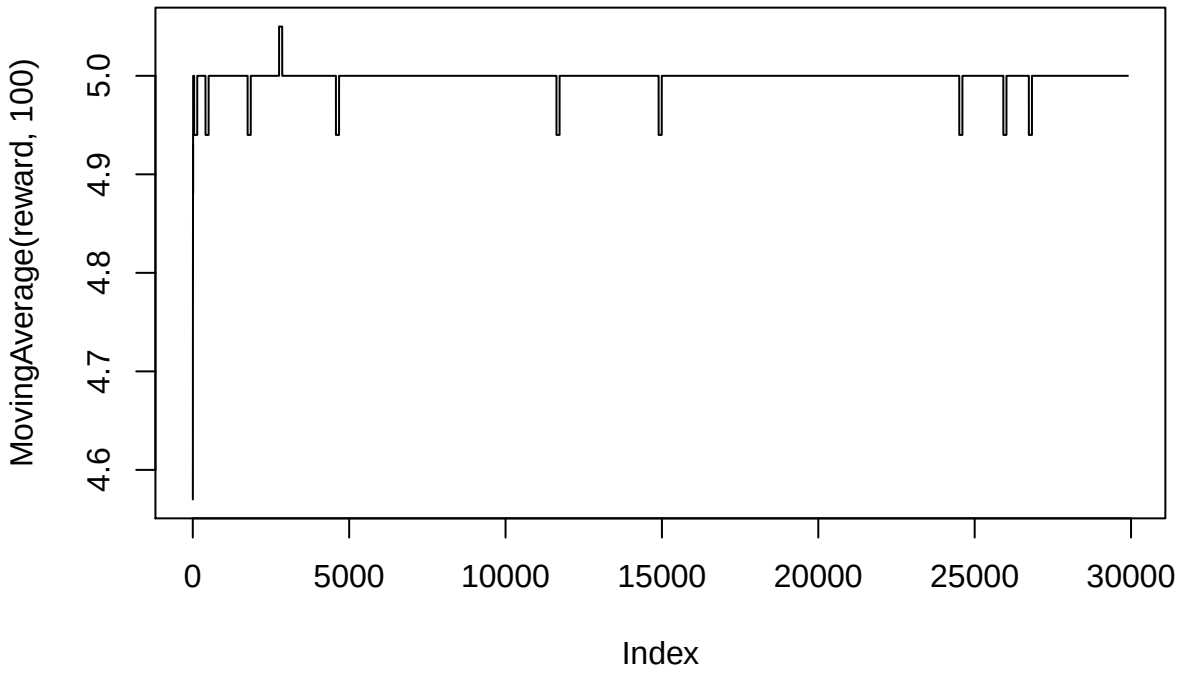

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

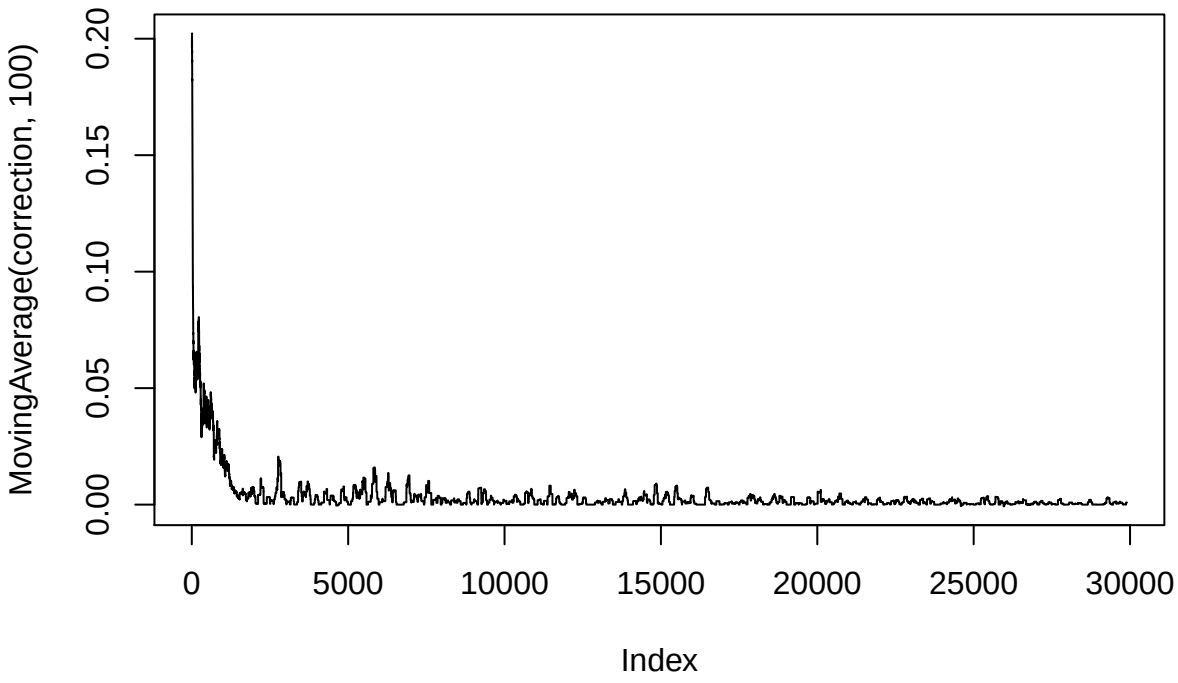

MovingAverage(reward, 100)
4.7
4.8
4.9
5.0
5.1
0
5000
10000
15000
20000
25000
30000
Index


MovingAverage(correction, 100)
0.00
0.04
0.08
0.12
0
5000
10000
15000
20000
25000
30000
Index

Epsilon = 0.1, gamma = 0.75

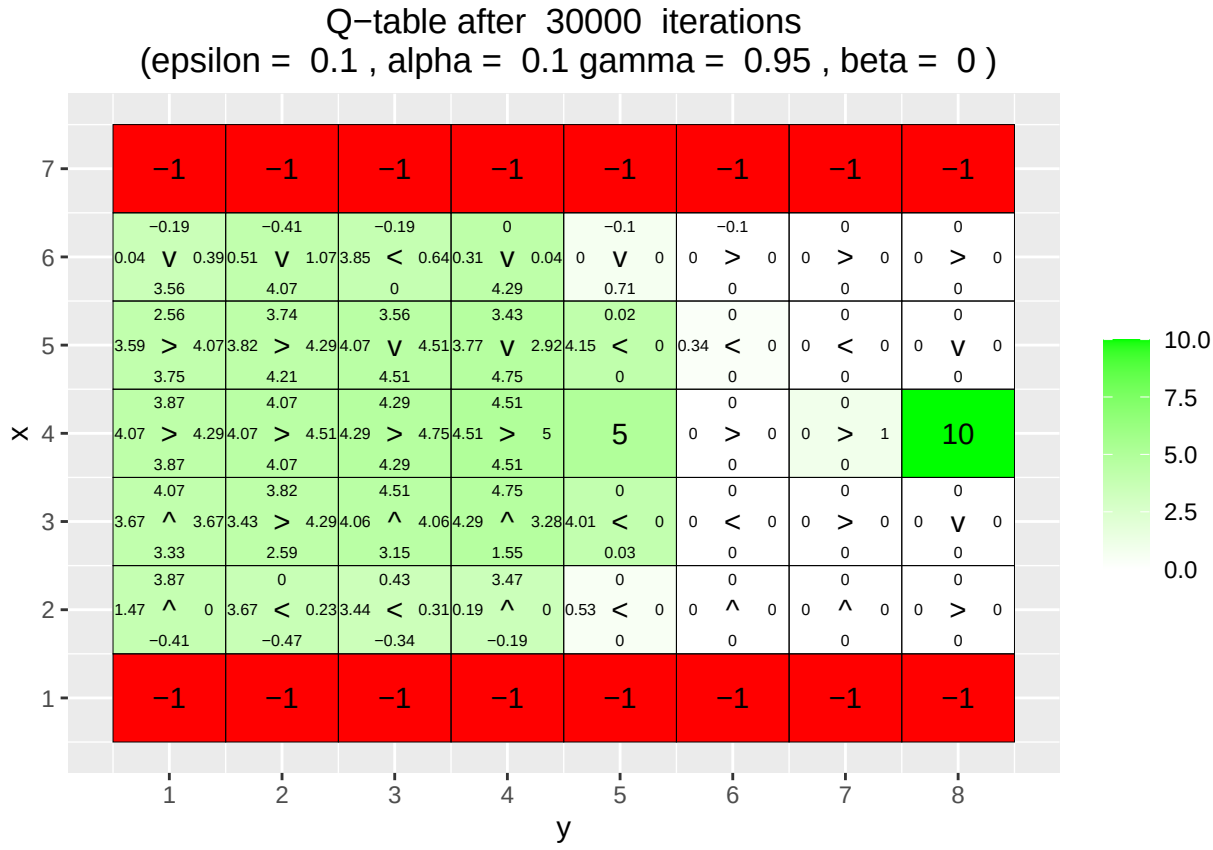


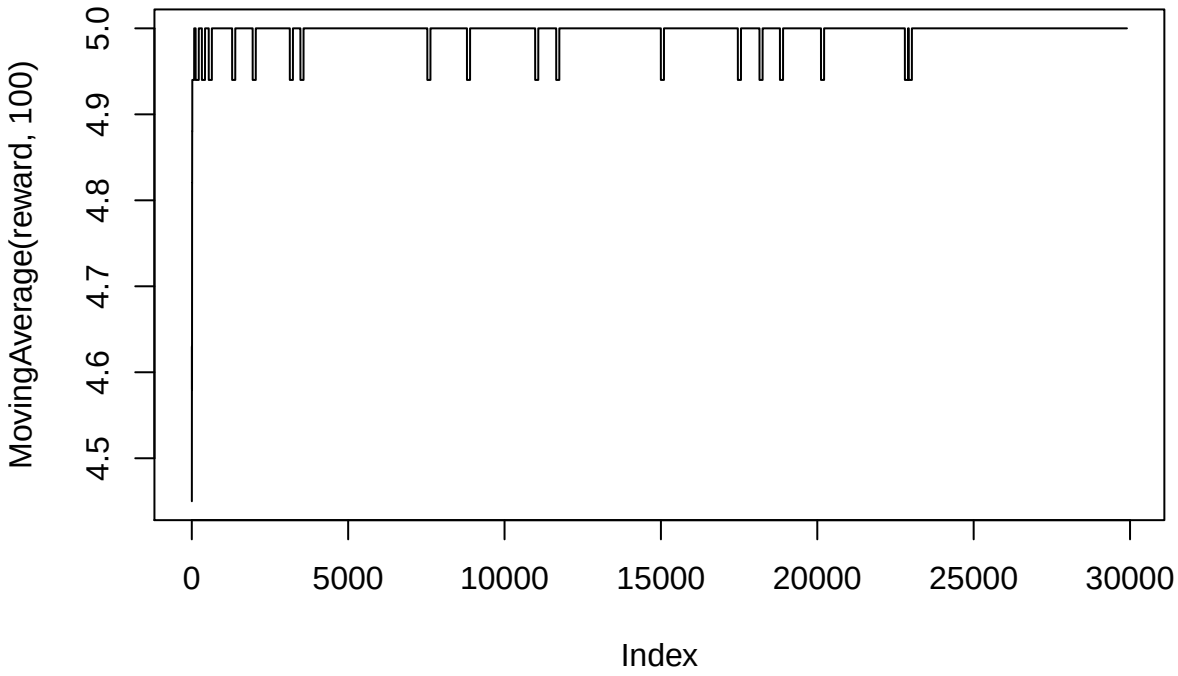


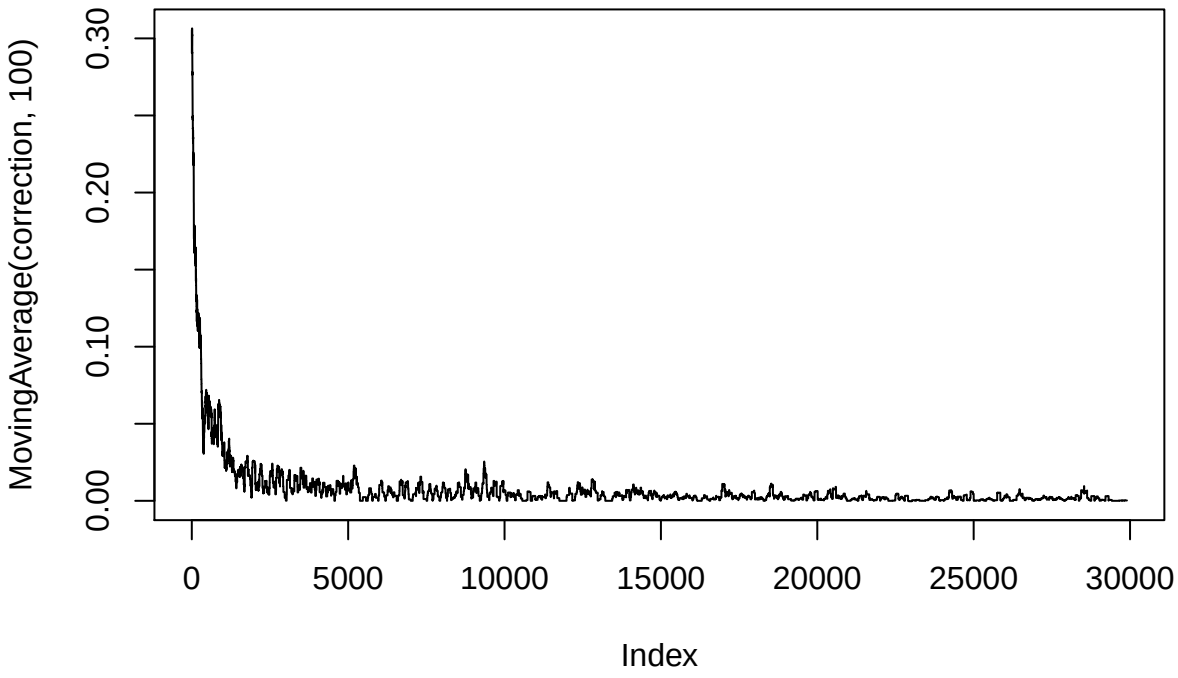


Epsilon = 0.1, gamma = 0.95

In this case it can take up future rewards to maximize the returns, thus it explores some more states as compared to the ones with $\epsilon = (0.5, 0.75)$.


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
−1
10

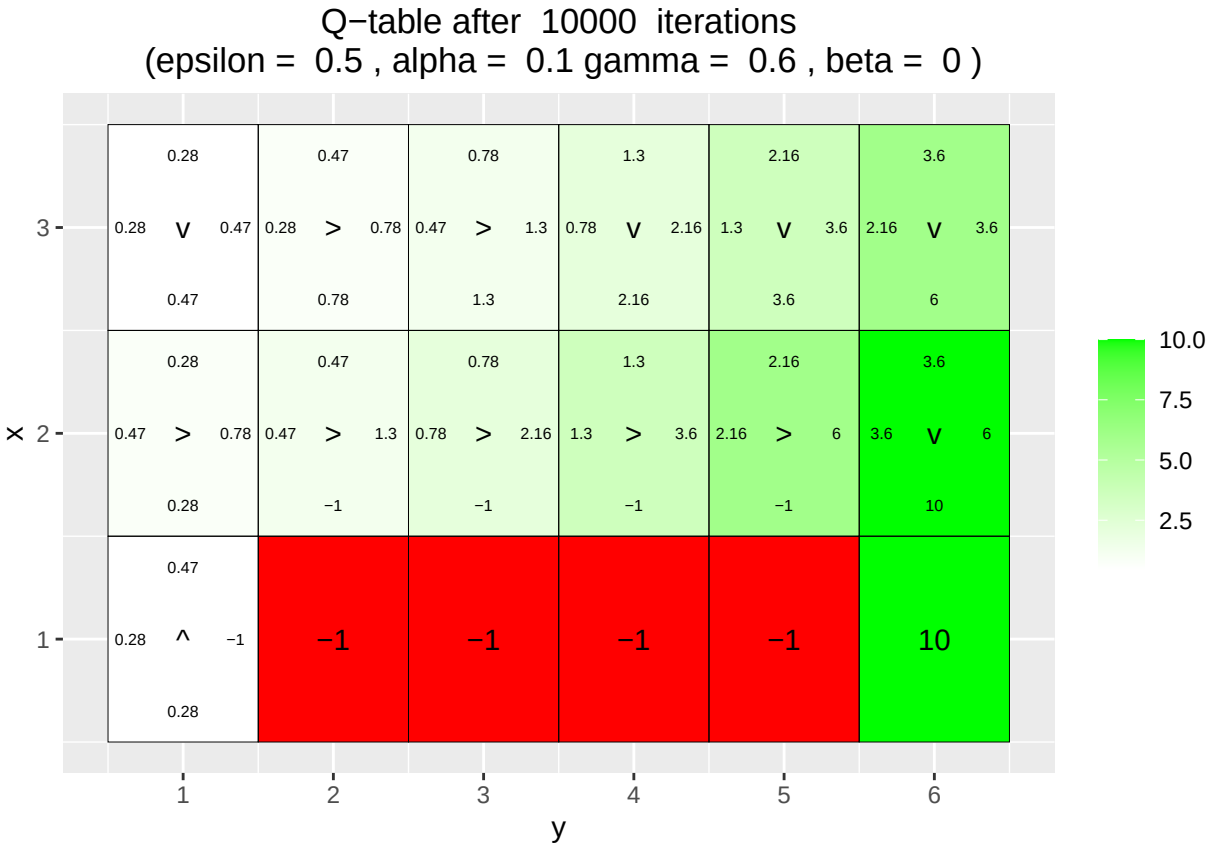

MovingAverage(reward, 100)
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

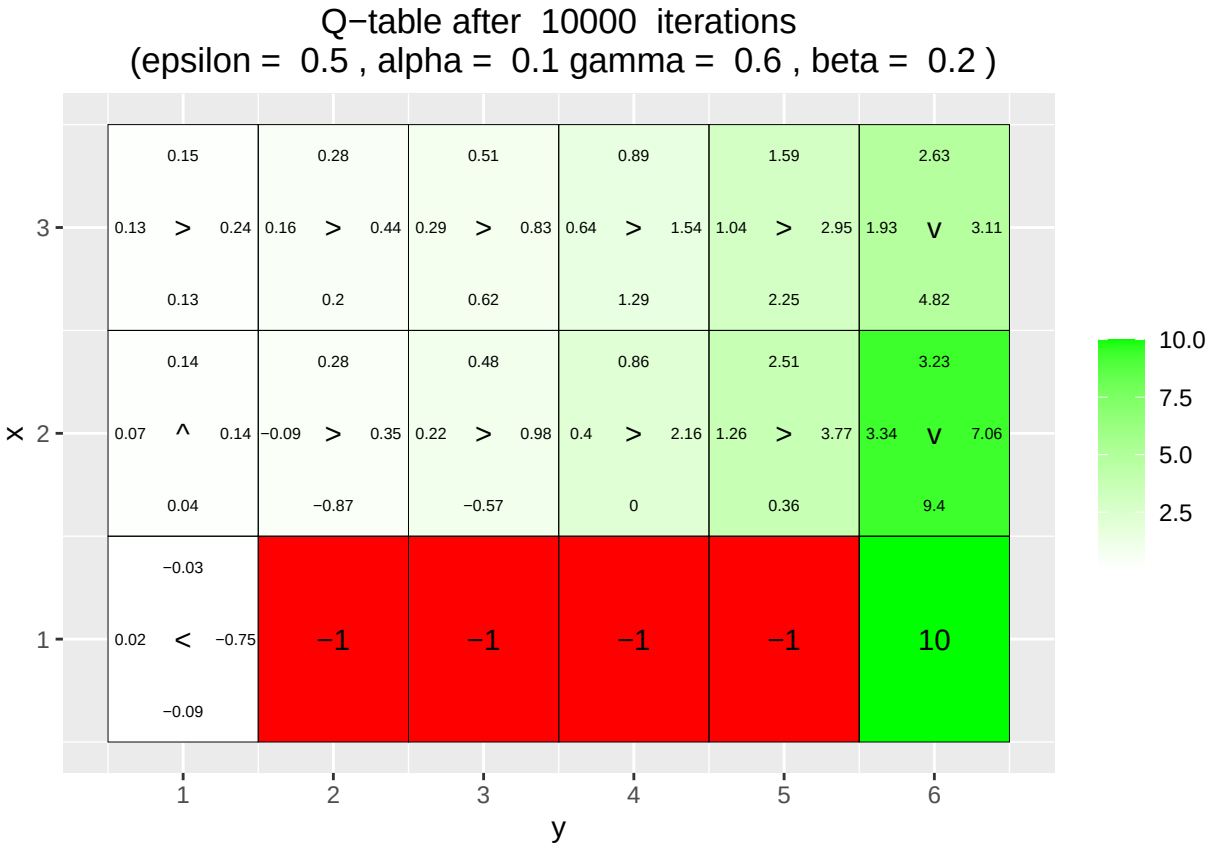


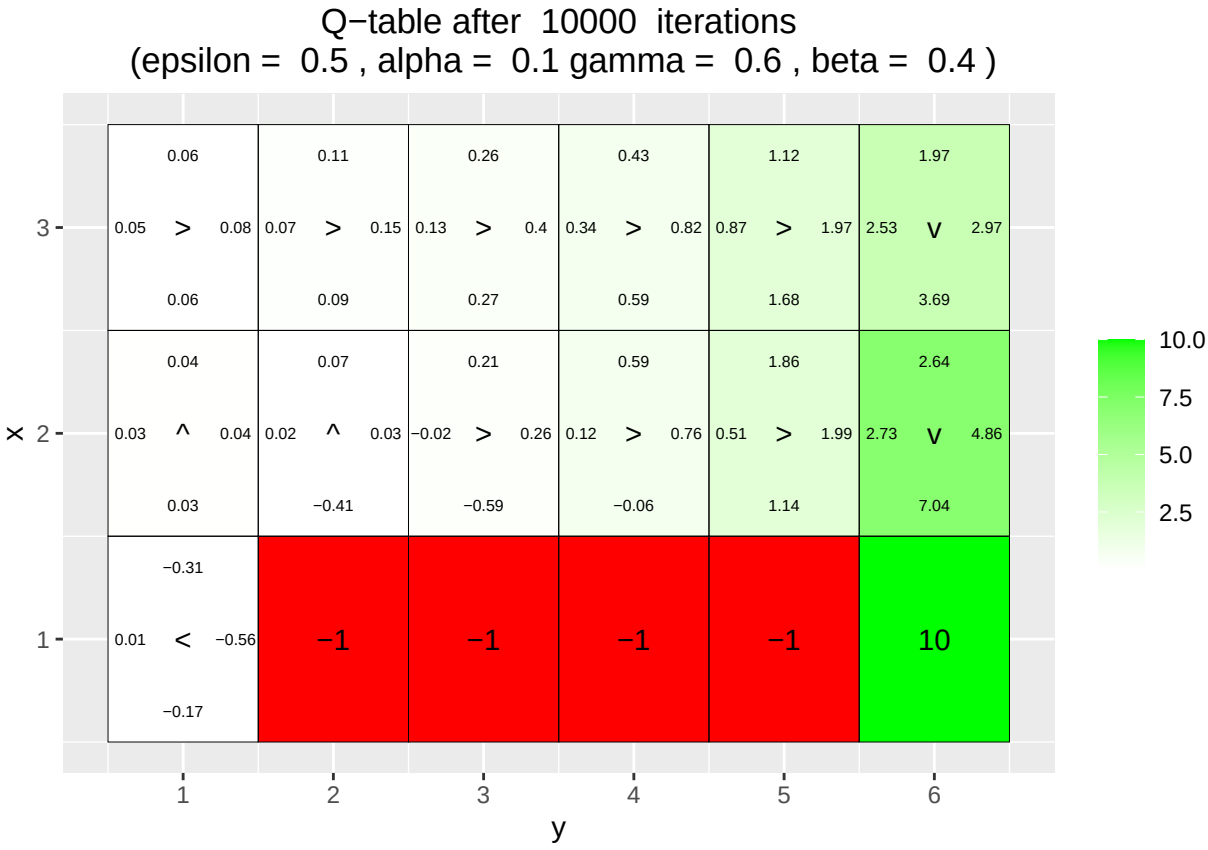
Environment C

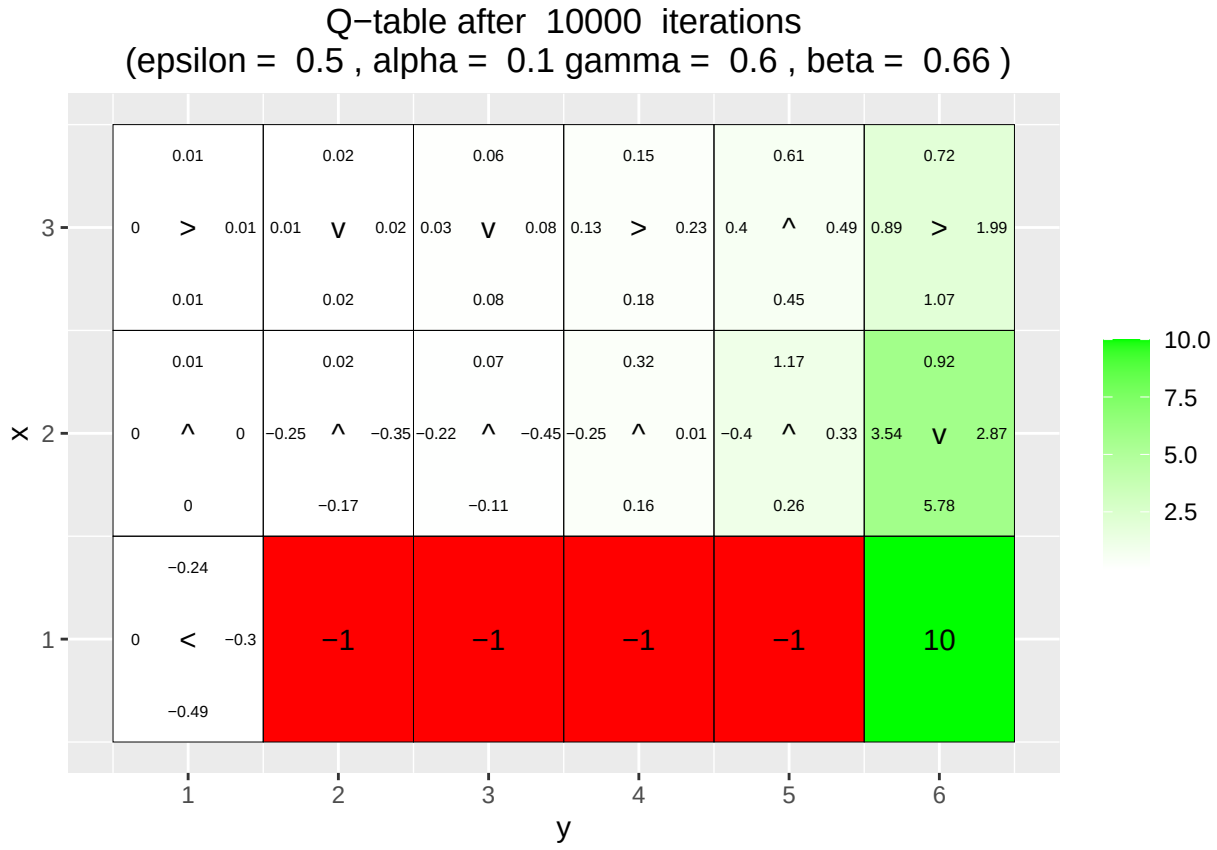
Higher β values indicate lower possibility of following the intended action. As seen from the graphs forβ=0 indicate that it follows the intended action with probability 1. So, the returns are maximized in this case as it does not slip to the neighboring states.

As the β value increases the returns are comparatively lower.


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)
0.28
0.28 v 0.47
0.47
0.47
0.28 > 0.78
0.78
0.78
0.47 > 1.3
1.3
1.3
0.78 v 2.16
2.16
2.16
1.3 v 3.6
3.6
3.6
2.16 v 3.6
6
0.28
0.47 > 0.78
0.28
0.47
0.47 > 1.3
−1
0.78
0.78 > 2.16
−1
1.3
1.3 > 3.6
−1
2.16
2.16 > 6
−1
3.6
3.6 v 6
10
0.47
0.28 ^ −1
0.28
−1
−1
−1
−1
10
3
2
1
x
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.15
0.13 > 0.24
0.13
0.28
0.16 > 0.44
0.2
0.51
0.29 > 0.83
0.62
0.89
0.64 > 1.54
1.29
1.59
1.04 > 2.95
2.25
2.63
1.93 v 3.11
4.82
0.14
0.07 ^ 0.14
0.04
0.28
−0.09 > 0.35
−0.87
0.48
0.22 > 0.98
−0.57
0.86
0.4 > 2.16
0
2.51
1.26 > 3.77
0.36
3.23
3.34 v 7.06
9.4
−0.03
0.02 < −0.75
−0.09
−1
−1
−1
−1
10
x
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
0.06
0.05 > 0.08
0.06
0.11
0.07 > 0.15
0.09
0.26
0.13 > 0.4
0.27
0.43
0.34 > 0.82
0.59
1.12
0.87 > 1.97
1.68
1.97
2.53 v 2.97
3.69
0.04
0.03 ^ 0.04
0.03
0.07
0.02 ^ 0.03
−0.41
0.21
−0.02 > 0.26
−0.59
0.59
0.12 > 0.76
−0.06
1.86
0.51 > 1.99
1.14
2.64
2.73 v 4.86
7.04
−0.31
0.01 < −0.56
−0.17
−1
−1
−1
−1
10
x
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5



Environment D

Training Environment

Yes, Agent seems to have learned a good policy as it converges to the terminal state when its checked on the validation states. It can be observed that if the agent starts at a random state, it converges to the goal.

Also, the policy has been learned taking into consideration any random states as the goal. It is not biased in the way it learns the policy.

The Q-learning will not solve this task as it optimizes a policy only for the specific environment. So if the learned policy is tested on the some other environment that would not yield optimal results.

Validation Environment

Environment E

No, The agent has not learned an optimal policy. In this case it has been trained with goals present in the first row. So in a way, the learned policy is biased.

When compared with environment D, this policy does not necessarily converge to the goals (when validated).

Figure 1: "Environment D (Training)"

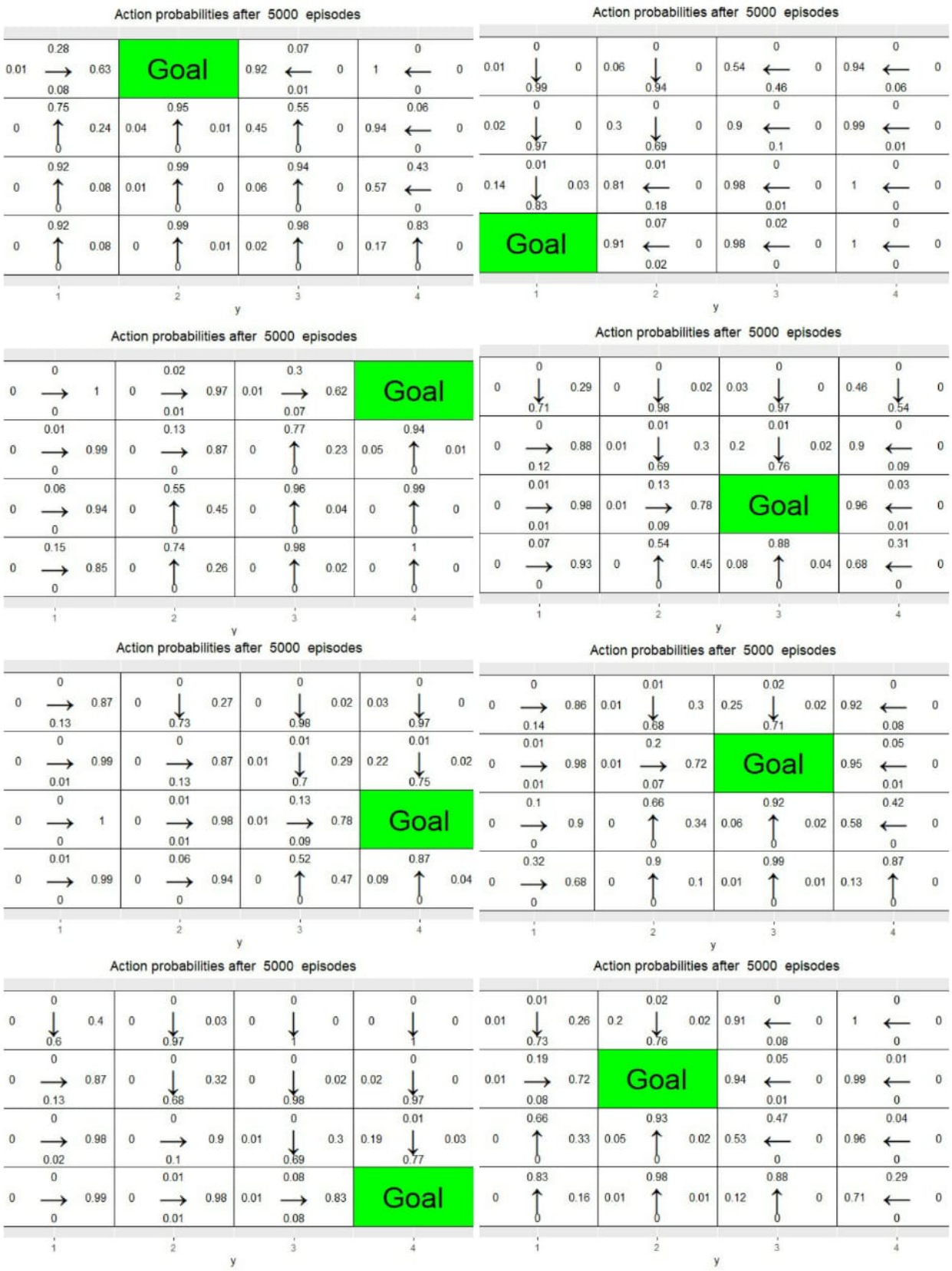


Figure 2: "Environment D (Validation)"

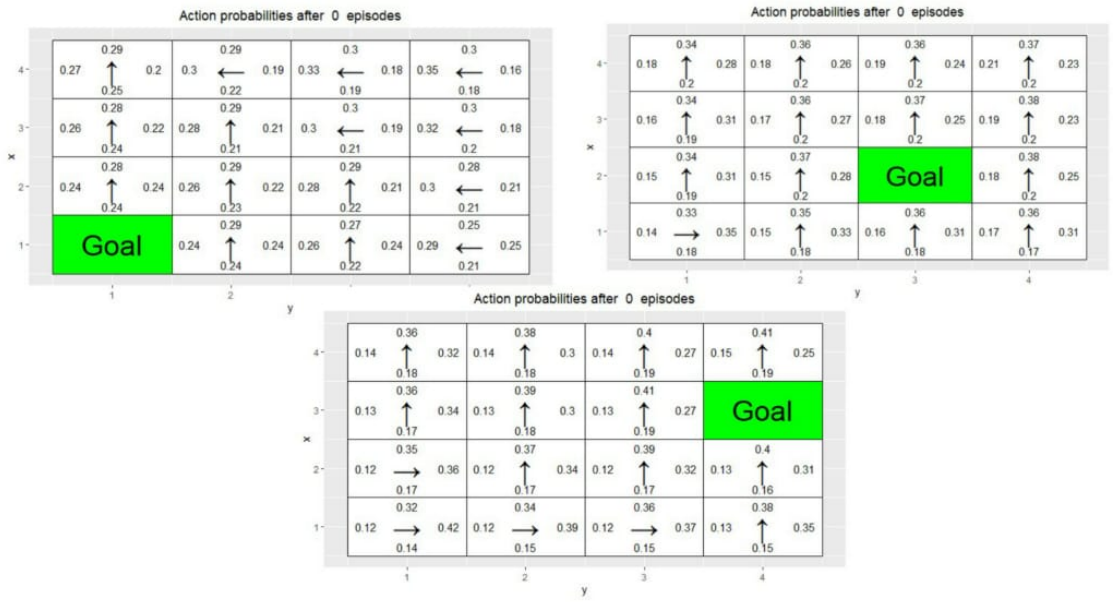


Figure 3: "Environment E (Training)"

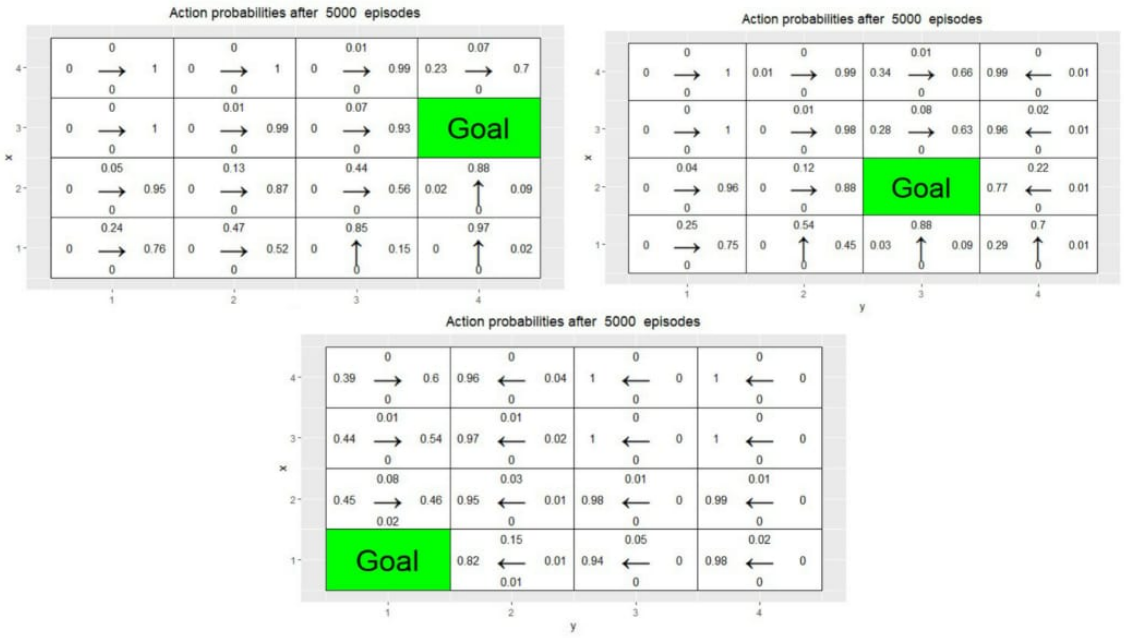


Figure 4: "Environment E (Training)"