Advanced Machine Learning Lab3 Individual Report

Akshath Srinivas(akssr921)

2023-09-22

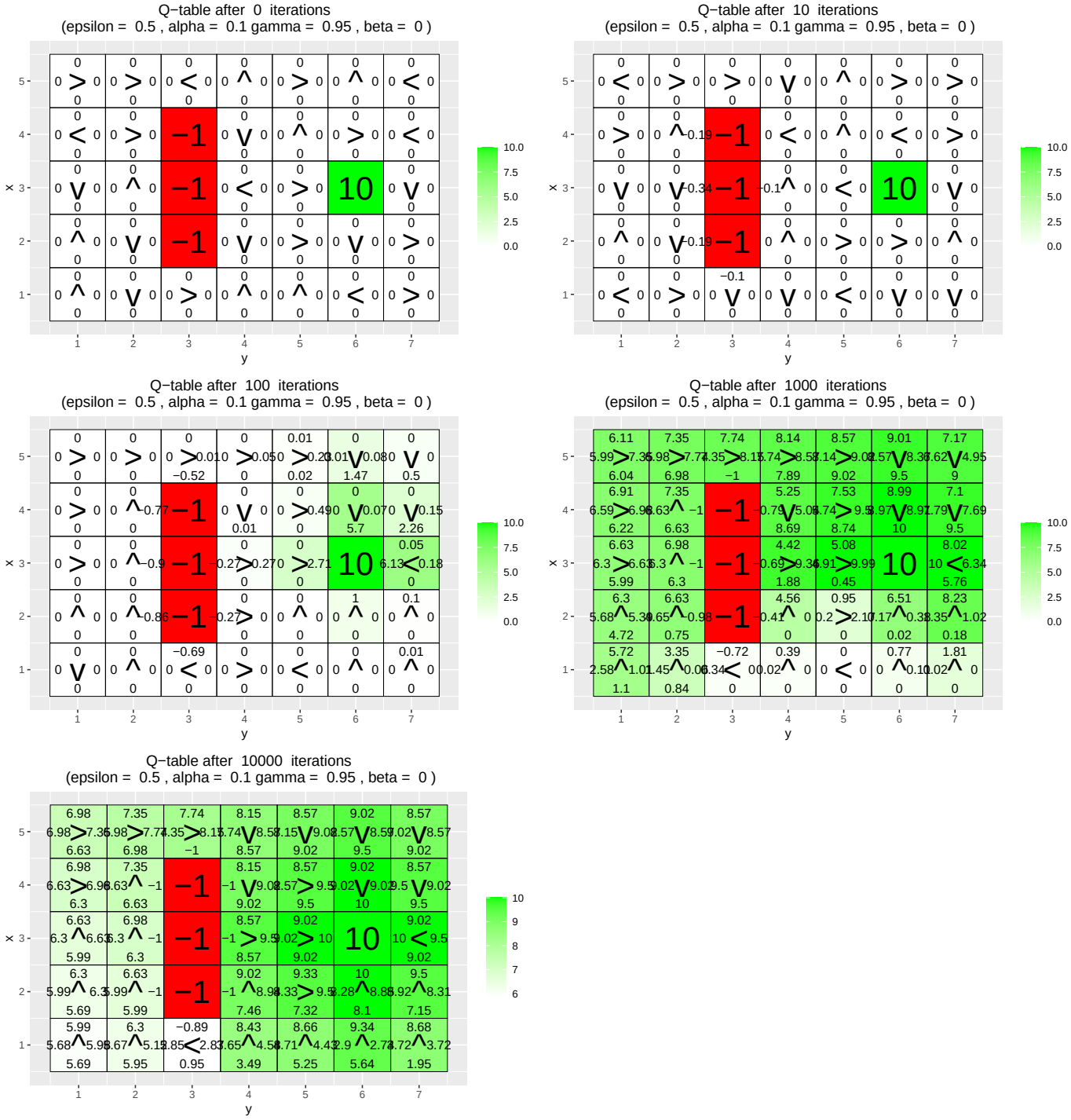
```
# By Jose M. Peña and Joel Oskarsson.
# For teaching purposes.
# jose.m.pena@liu.se.

##########################################################################################################
# Q-learning
##########################################################################################################

# install.packages("ggplot2")
# install.packages("vctrs")
library(ggplot2)

arrows <- c("^", ">", "v", "<")
action_deltas <- list(c(1,0), # up
                      c(0,1), # right
                      c(-1,0), # down
                      c(0,-1)) # left

vis_environment <- function(iterations=0, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){

  # Visualize an environment with rewards.
  # Q-values for all actions are displayed on the edges of each tile.
  # The (greedy) policy for each state is also displayed.
  #
  # Args:
  #   iterations, epsilon, alpha, gamma, beta (optional): for the figure title.
  #   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.
  #   H, W (global variables): environment dimensions.

  df <- expand.grid(x=1:H,y=1:W)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,1],NA),df$x,df$y)
  df$val1 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,2],NA),df$x,df$y)
  df$val2 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,3],NA),df$x,df$y)
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,4],NA),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y)
    ifelse(reward_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward_map[x,y]),df$x,df$y)
  df$val5 <- as.vector(foo)
```

```
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,max(q_table[x,y,]),
                                     ifelse(reward_map[x,y]<0,NA,reward_map[x,y])),df$x,df$y)
  df$val6 <- as.vector(foo)

  print(ggplot(df,aes(x = y,y = x)) +
          scale_fill_gradient(low = "white", high = "green", na.value = "red", name = "") +
          geom_tile(aes(fill=val6)) +
          geom_text(aes(label = val1),size = 4,nudge_y = .35,na.rm = TRUE) +
          geom_text(aes(label = val2),size = 4,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 4,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 4,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 10) +
          geom_tile(fill = 'transparent', colour = 'black') +
          ggtitle(paste("Q-table after ",iterations," iterations\n",
                        "(epsilon = ",epsilon,", alpha = ",alpha,"gamma = ",gamma,", beta = ",beta,")")) +
          theme(plot.title = element_text(hjust = 0.5)) +
          scale_x_continuous(breaks = c(1:W),labels = c(1:W)) +
          scale_y_continuous(breaks = c(1:H),labels = c(1:H)))

}


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.
  maxact_values <- which(q_table[x, y, ]==max(q_table[x, y, ]))
  if(length(maxact_values)<=1){
    return_max_act=maxact_values
  }else{
    return_max_act=sample(maxact_values, 1)
  }
  return(return_max_act)

}

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.
  if (runif(1) < epsilon) {
    # Random action with probability epsilon
    return(sample(1:4, size = 1))
  } else {
    # Greedy action with probability 1 - epsilon
    return(GreedyPolicy(x, y))
  }

}

transition_model <- function(x, y, action, beta){

  # Computes the new state after given action is taken. The agent will follow the action
  # with probability (1-beta) and slip to the right or left with probability beta/2 each.
  #
  # Args:
  #   x, y: state coordinates.
  #   action: which action the agent takes (in {1,2,3,4}).
  #   beta: probability of the agent slipping to the side when trying to move.
  #   H, W (global variables): environment dimensions.
  #
  # Returns:
  #   The new state after the action has been taken.

  delta <- sample(-1:1, size = 1, prob = c(0.5*beta,1-beta,0.5*beta))
  final_action <- ((action + delta + 3) %% 4) + 1
  foo <- c(x,y) + unlist(action_deltas[final_action])
  foo <- pmax(c(1,1),pmin(foo,c(H,W)))

  return (foo)
}

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 randomly.
  #   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.

  # Initialize episode-specific variables
  episode_correction <- 0
  sta = start_state

  repeat {
    x <- sta[1]
    y <- sta[2]
    # Choose an action using epsilon-greedy policy
    action <- EpsilonGreedyPolicy(x, y, epsilon)

    # Store the current state-action pair
    current_q <- q_table[x, y, action]

    # Perform the action and transition to a new state
    new_state <- transition_model(x, y, action, beta)
    new_x <- new_state[1]
    new_y <- new_state[2]

    # Get the reward for the new state
    reward <- reward_map[new_x, new_y]

    # Compute the maximum Q-value for the new state
    max_q <- max(q_table[new_x, new_y, ])

    # Update the Q-value for the current state-action pair
    di<- reward + ((gamma * max_q) - current_q)
    q_table[x, y, action]<<-q_table[x, y, action] + alpha * di

    # Transition to the new state
    sta = new_state

    # Update the correction term
    episode_correction <- episode_correction + di

    # Check if the episode has ended (reached a terminal state)
    if (reward != 0)
      return(c(reward, episode_correction))
  }

}
```

Environment A



Question 1

- After 10 iterations very few values in the Q-table has been filled and the agent has learned Q table values near red region i.e not to go to states having negative one reward.

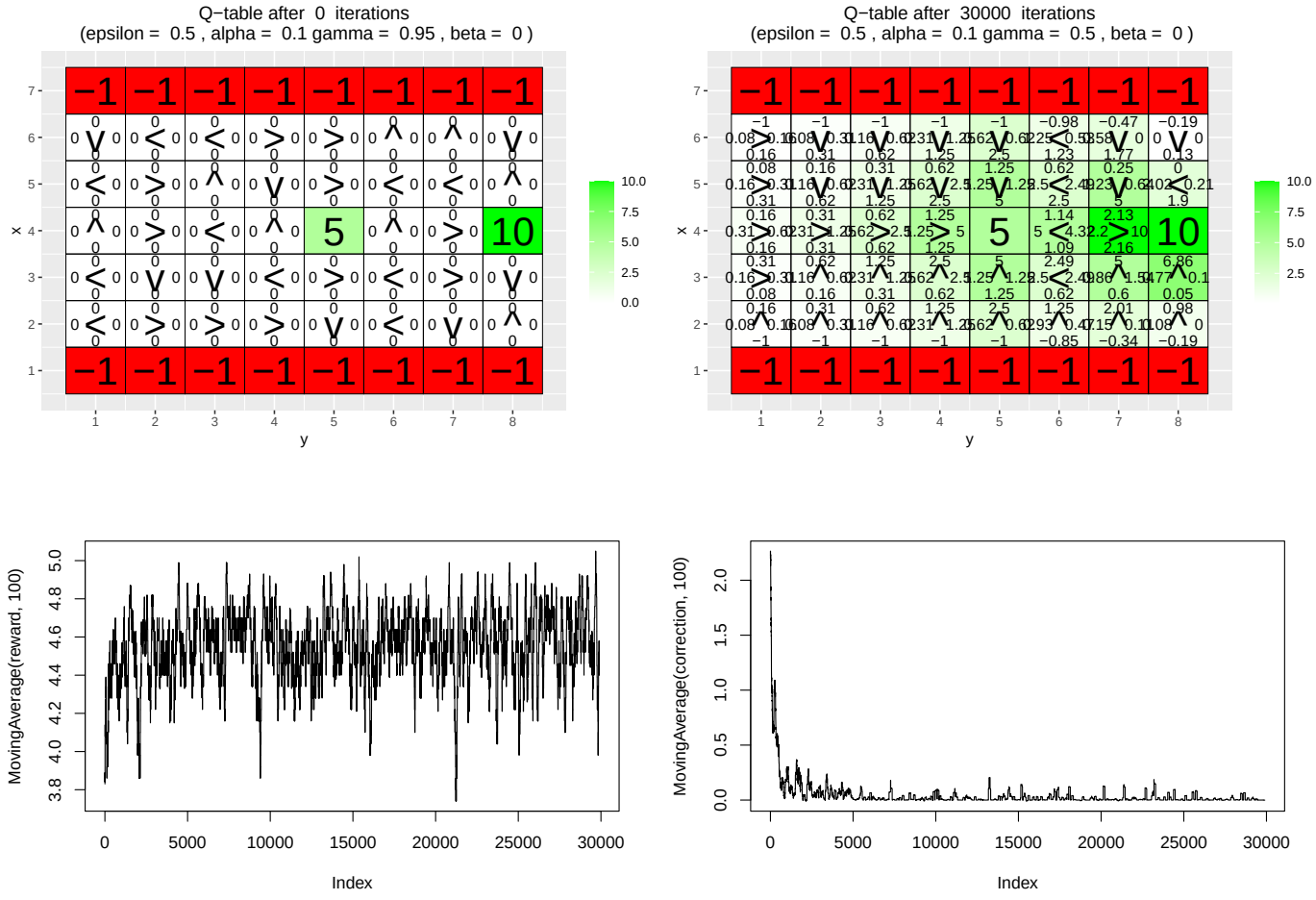
Question 2

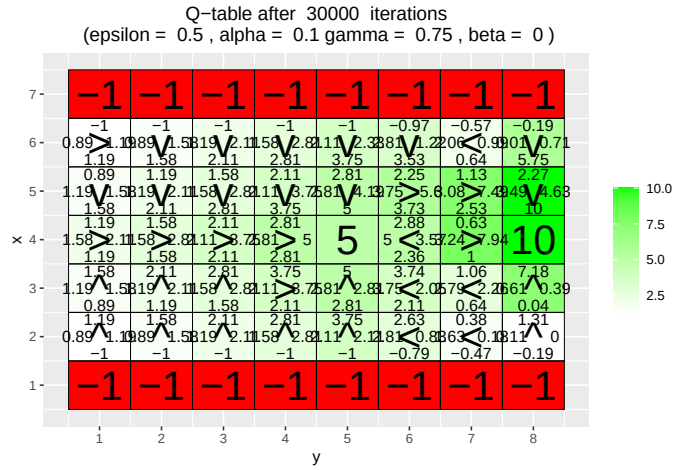
- The final greedy policy appears to be optimal for the majority of states, as it enables the agent to maximize its expected discounted return while reaching the target. However, there are instances where the policies for certain states are not optimal. For instance, if the agent initiates at positions (5,1) or (5,2), it has the potential to take a different path, moving downwards to reach the target instead of going upwards. This suboptimality arises because the agent does not actively explore alternative paths. It is possible that increasing the value of epsilon in the exploration strategy could help identify the optimal paths for all states.

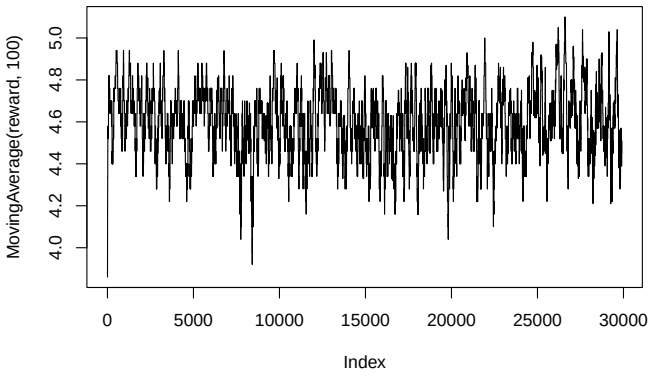
Question 3

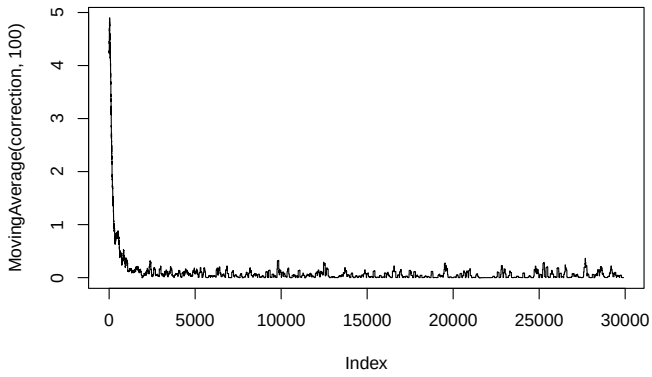
The reason behind the agent's inclination to go above the negative values might be from its early exploration behavior, particularly evident at iterations 100 and 1000 (due to epsilon-greedy strategy), chose the upper path. This initial preference for the upper path led to higher Q-values, eventually influencing the policy to favor that path. This preference remained because the bottom path was explored less frequently. To address this problem, one potential solution could involve increasing the value of epsilon to encourage more exploration below the negative values or adding more iterations to learn the policy further.

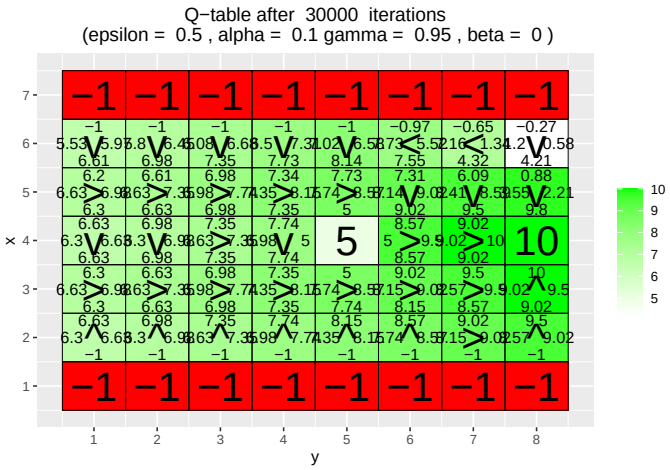
Environment B

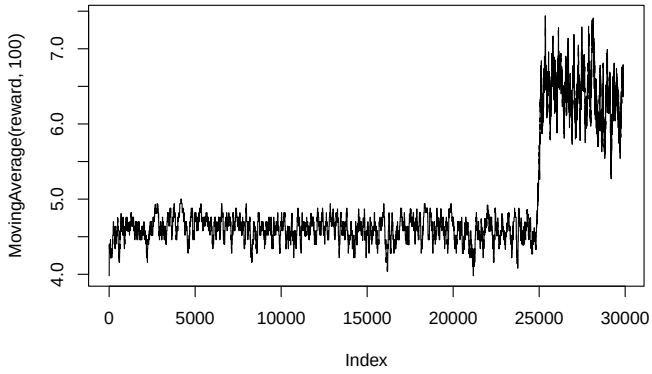


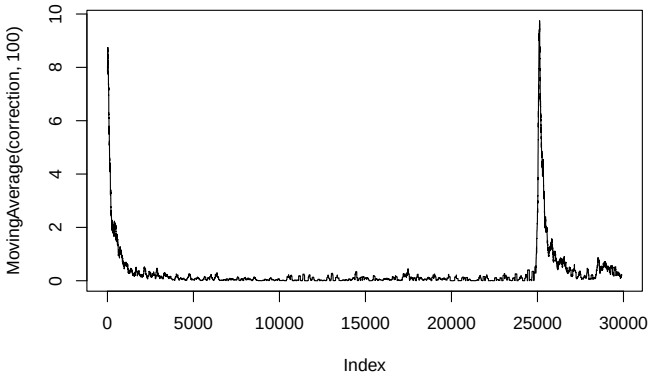












For $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$ and $\gamma = (0.5, 0.75, 0.95)$

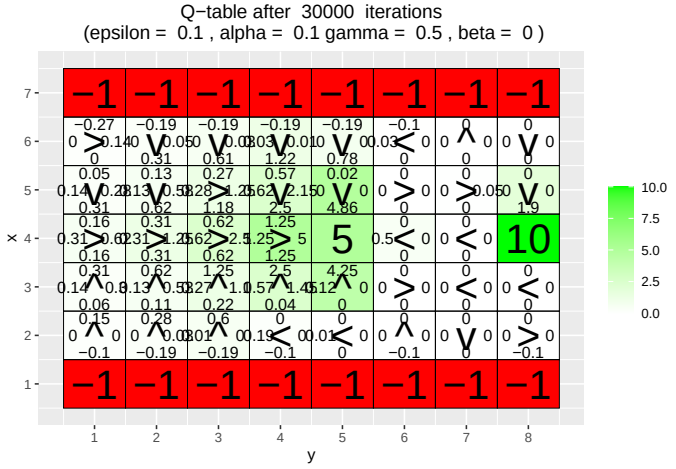
From the Moving average plot of rewards for the given configuration, we can see that agent has on average, learnt the path to reward block 5. This is because the agent prioritizes short term returns more than the long term returns because of the discount factor.

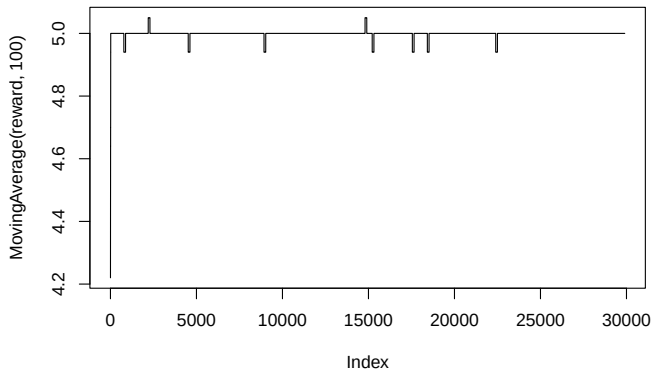
The corrections represents the sum of the difference between the predicted Q-value for a state-action pair and the updated Q-value based on the observed rewards and estimated future rewards. Furthermore, it provides insight into how much the Q-values are being updated during the learning process in each episode.

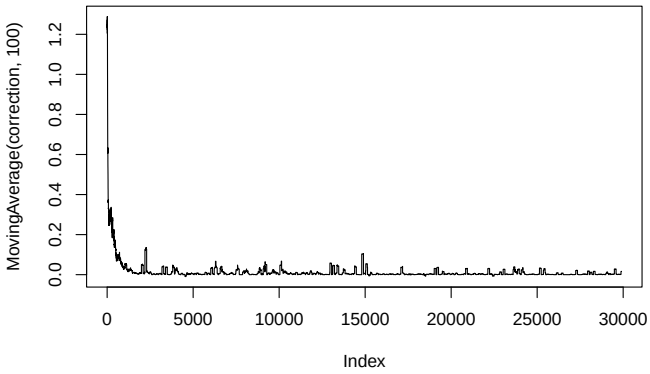
From the Moving average plot of corrections for the given configuration, we can see that in the initial few 1000 episodes, the correction is very high given that the agent is still learning the optimal policy(Q-table values) signifying unstableness. However, after 5000 episodes, we can say that the q-values have stabilized and we observe only minor fluctuations in the corrections henceforth.

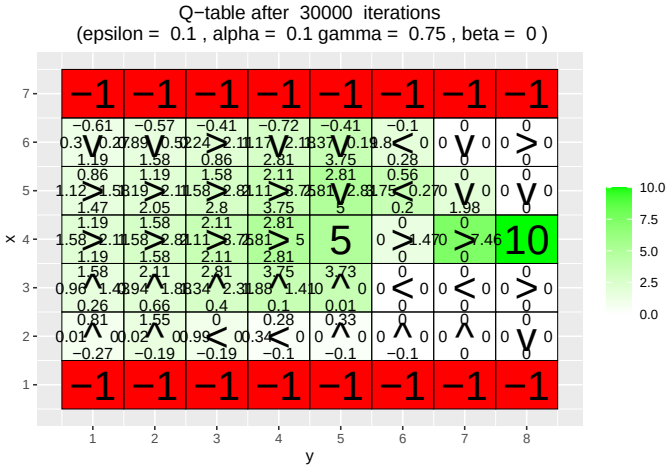
When $\gamma = 0.75$ the behavior of the agent with respect to moving average of rewards and corrections are similar to the previous case. However, now the agent assigns higher q-values to actions leading to reward block 10 in the vicinity of reward block 5. This signifies that the agent is now starting to delay its immediate rewards given the increase in the discount factor.

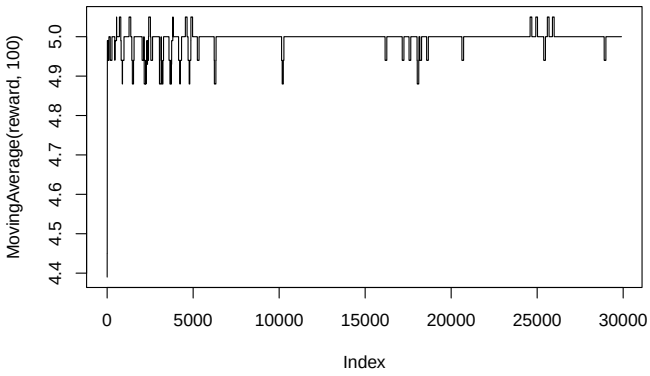
When $\gamma = 0.95$, we can see from the moving average plots for rewards and corrections, that after 2500 episodes the agent starts to acknowledge the terminal state of reward 10. Since, the discount factor is higher the agent does not prioritize immediate rewards i.e., reward block 5 and hence always tries to reach the reward block 10.

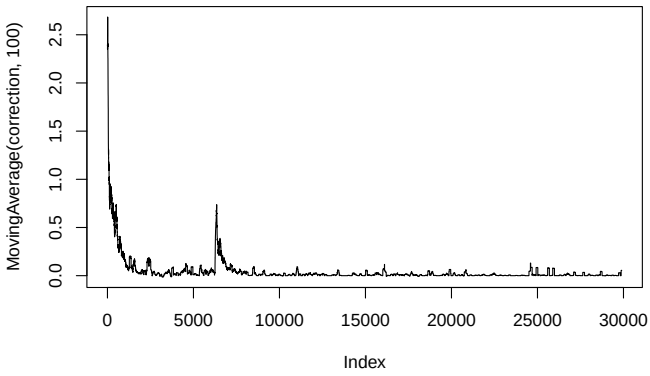


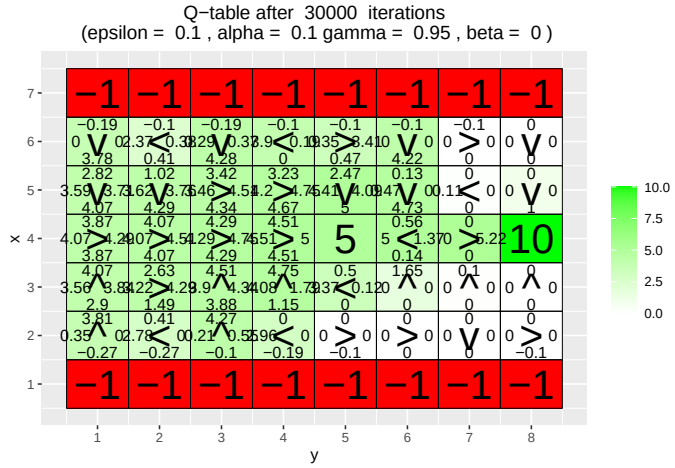


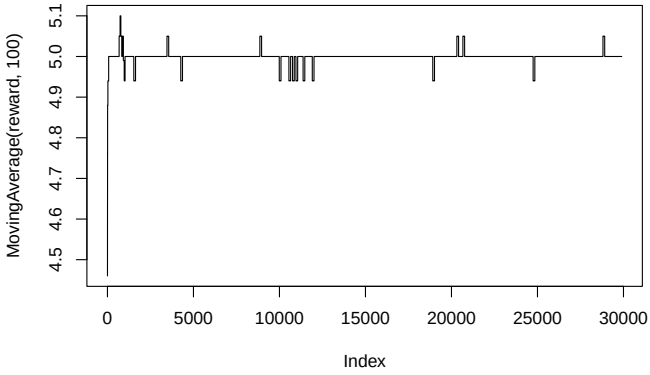


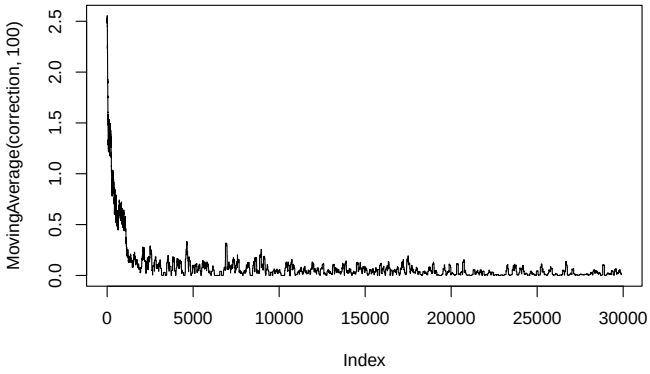










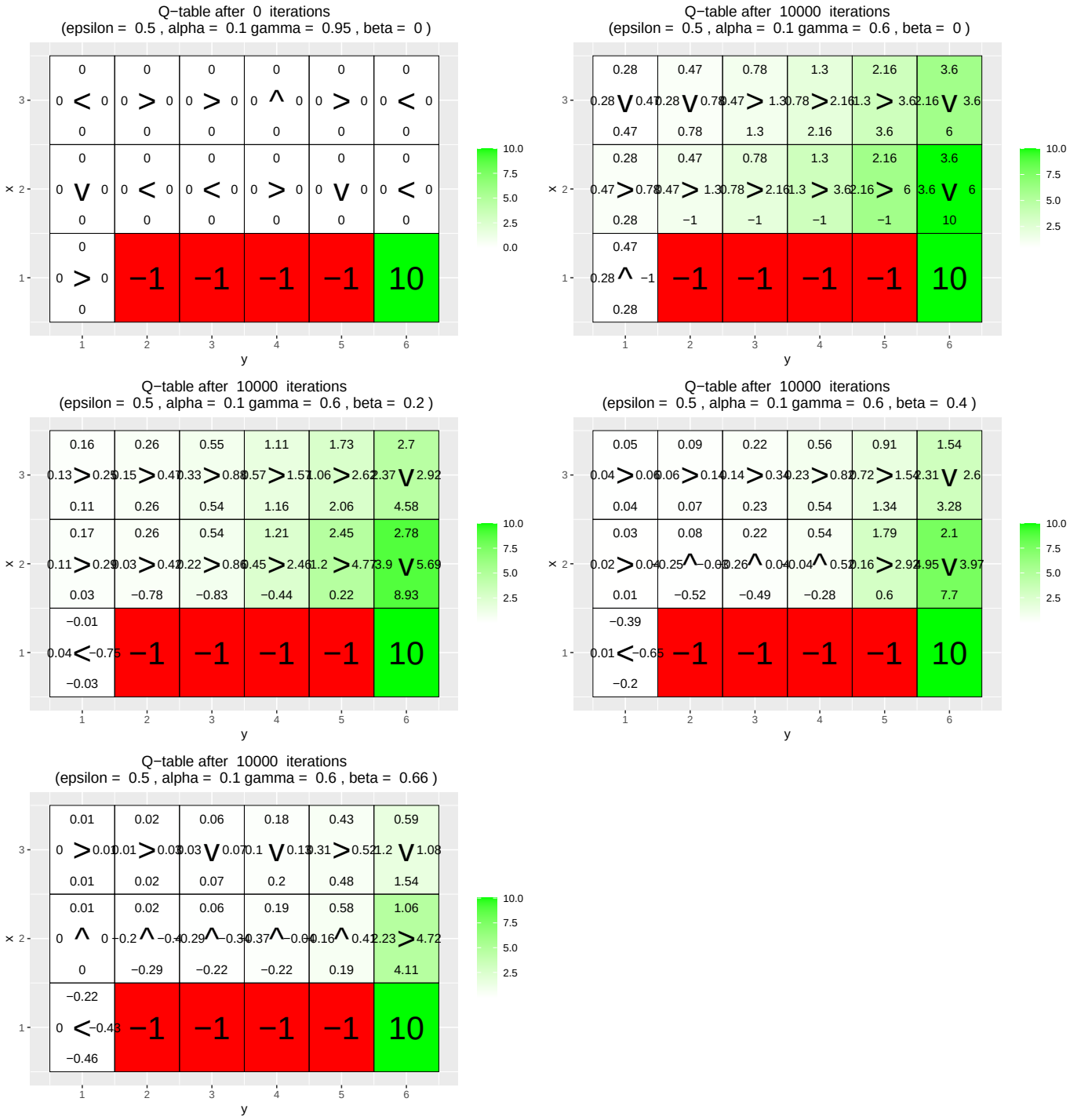


For $\epsilon = 0.1$, $\beta = 0$, $\alpha = 0.1$ and $\gamma = (0.5, 0.75, 0.95)$

When $\epsilon = 0.1$, the agent becomes non exploratory in the sense that once it reaches the terminal state of reward block 5, it starts to exploit the path with the highest q-values leading to the same reward block and doesn't explore for another terminal state i.e., in this case reward block 10.

Because of this exploitative nature of the agent, increasing the discount factor doesn't change its behavior as it almost always reaches the terminal state of reward block 5. The same can confirmed from the moving average plots for rewards.

Environment C



When $\beta = 0$, there is no chance of "slipping" and the learned policy is optimal.

However, as the slipping factor increases, the agent finds it difficult to learn an optimal policy. In the case of $\beta = 0.66$ the agent is not able to learn an optimal policy because 66% of all its actions are sub-optimal.

From the environment plot, we can observe that the algorithm assigns low expected returns to actions which are actually optimal and this is due to the agent "slipping" into the negative reward blocks.

Environment D

Sub Question 1: Has the agent learned a good policy? Why / Why not ?

Yes, it has learnt the optimal policy based on the final probability distributions of actions for each state. That is, in almost all cases it takes the path that minimizes the categorical cross-entropy loss by optimizing the weights via Stochastic Gradient Descent.

Another point to note is that the training goals span across the environment and the validation goals are in the vicinity of training goals. Because of this, the model is able to generalize well to the validation goals.

Sub Question 2: Could you have used the Q-learning algorithm to solve this task ?

Q-learning(model-free solution) would have resulted in a bad policy learning since it is usually applied to environments where the end goal is fixed. In this case since the start and end goals are not fixed, we would need a model-based solution.

Environment E

Sub Question 1 & 2: Has the agent learned a good policy? Why / Why not ?

In this environment the agent has not learned a good policy because, its training was explicitly based on the top row. It cannot generalize well when the validation goals end up being anywhere other than the top row.

Whereas, as mentioned above, the agent performs/generalizes better in Environment D due to the spread in training goals across the environment.