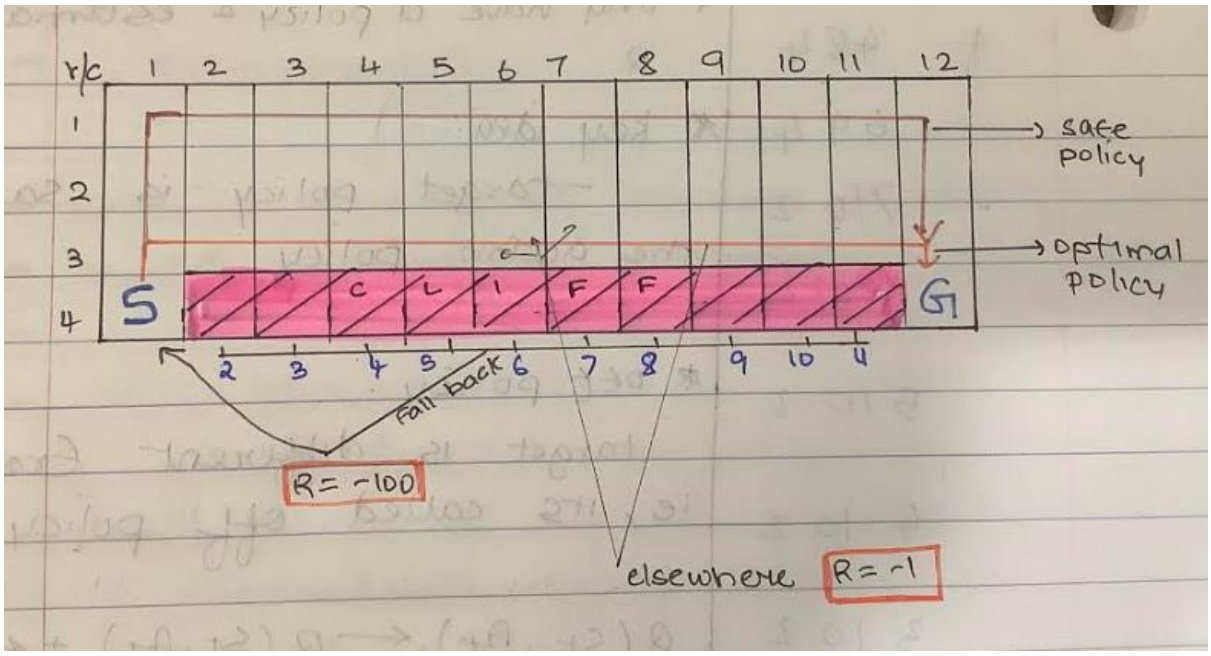
Homework 6

Environment Set up:

In this homework, we have a cliff world set up, in which we aim to traverse from the start index/node (4,1) to end index/node (4,12). In between, we have a cliff where if our agents fall into those index(s), we have to restart from the 'start' node all over again, and we get a reward of –'100'. All other states/index(s) are assigned a reward of '-1'. This reward is assigned because we aim to traverse the Cliff-Grid World in the least possible number of steps.

This all can be better understood by observation of the below figure :



Also, here we are implementing three different algorithms : SARSA, Q-learning and expected SARSA

Algorithm explanation

Initializations

We first assign/initialize the various values. These include gamma, which is set to '1' as it is not discounted here. Then we set the step size as 0.5. Further, we assign the action-value function; here, we make use of MATLAB's multidimensional arrays, wherein the various actions, 'up,' 'down,' 'right,' left,' are represented by the numbers 1,2,3,4 respectively, are also defined. This manner of initiation allows us to choose the best action for our index. Other parameters introduced here include the number of episodes (which is set to 500).

SARSA:

For our 500 episodes, we will run the SARSA policy algorithm. Here we first select our best action based on our policy. The policy that we use is E-greedy policy where 90% of the time, we choose the best action (maximum) and 10 % of the time, and we select a random action. Also, here we define our starting step-index (which is (4,1)). Next, we begin our first episode and use a while loop to continue the episode until we reach the end node /index. So, once we choose our first action, we need to update our current state-based on that action. I will explain the various steps below :

- For the action "UP" we would only need to decrement a row value by 1
- For the action " down" we would need to increment the row value by 1
- Next, for the action of " Right, "we would need to increment a column by 1
- And lastly for the action of "left," we would need to decrement a column by 1

Once we have done this because this is a cliff world, we check if the agent has reached the cliff indices if he has, we would assign it a reward of '-100' and send it back to the starting node. Otherwise, we assign a reward of '-1'. This assignment is done using a switch case in the code. Next, we also check for boundary conditions, and if the agent has breached them, we bring it back in by assigning it to the closest column or row that it has crossed. Additionally, to make things clearer, the cliff conditions indexes include the following :

(Row, Column):(4,2),(4,3),(4,4),(4,4),(4,5),(4,6),(4,7),(4,8),(4,9),(4,10)(4,11)

Once these have been checked and the reward has been assigned, we sample the next action, also using the e-greedy policy. Then update the Q matrix using the given formula :

Q(S[t],A[t])<---(Q(S[t],A[t])+alpha*(R[t+1]+gamma(Q(S[t+1],At[t+1]))-Q(S[t],A[t]))) .

Also, we find the next action based on our policy itself over here. This helps us update the previous states and perform policy improvement after every time step/step. Also, the policy used is used to help us determine the next action that we would take to help estimate and predict policy evaluation. Once we update the Q values, the newly calculated actions, index values are passed onto the next step to redo all the steps all over again.

These steps are terminated when we reach the final index (4,12). Now we use these computed Q values to do another episode and so on. We continue this for 500 episodes and then redo this for 20 additional loops. This helps us achieve a smoother curve. We then sample 20 of them to find their mean over the 10000-reward sum that we obtained and plot this sum vs. the episodes.

Q learning :

The methodology remains very similar to SARSA in Q learning except for a few key differences. I will go through the whole procedure again and highlight the differences ahead.

For our 500 episodes, we will run the Q-learning policy algorithm. Here we first select our best action based on our policy. The policy that we use is E-greedy policy, where 90% of the time, we choose the best action (maximum) and 10 % of the time, and we select a random action. Also, here we define our starting step-index (which is (4,1)). Next, we begin our first episode and use a while loop to continue the episode until we reach the end node /index. So, once we choose our first action, we need to update our current state-based on that action. I will explain the various steps below :

- For the action "UP" we would only need to decrement a row value by 1
- For the action " down" we would need to increment the row value by 1
- Next, for the action of " Right, "we would need to increment a column by 1
- And lastly for the action of "left," we would need to decrement a column by 1

Once we have done this because this is a cliff world, we check if the agent has reached the cliff indices if he has, we would assign it a reward of '-100' and send it back to the starting node. Otherwise, we assign a reward of '-1'. This is done using a switch case in the code. Next, we also check for boundary conditions, and if we agent has breached it, we bring it back in by assigning it the closest column or row that it has crossed. Additionally, to make things clearer, the cliff conditions indexes include the following :
(Row, Column):(4,2),(4,3),(4,4),(4,4),(4,5),(4,6),(4,7),(4,8),(4,9),(4,10)(4,11)

Once these have been checked and the reward has been assigned, we sample the next action, also using the e-greedy policy. Then update the Q matrix using the given formula :
Q_1(r_1,c_1,action_1) = Q_1(r_1,c_1,action_1) +alpha_1*(R_1 + gamma_1*value-Q_1(r_1,c_1,action_1))
Here we sample our next action as the maximum among the given actions in the Q matrix and use that in place of again using our policy matrix to estimate the actions. The rest remains the very same.
This helps us update the previous states and perform policy improvement after every time step/step. Also, the policy used is used to help us determine the next action that we would take to help estimate and predict policy evaluation. Once we update the Q values, the newly calculated actions, index values are passed onto the next step to redo all the steps all over again.

These steps are terminated when we reach the final index (4,12). Now we use these computed Q values to do another episode and so on. We continue this for 500 episodes and then redo this for 20 additional loops. This helps us achieve a smoother curve. We then sample 20 of them find their mean over the 10000-reward sum that we obtained and plot this sum vs. the episodes.

Expected SARSA:

The methodology remains very similar to Q learning, except for a few key differences. I will go through the whole procedure again and highlight the differences ahead.

For our 500 episodes, we will run the Expected SARSA-learning policy algorithm. Here we first select our best action based on our policy. The policy that we use is E-greedy policy where 90% of the time we choose the best action (maximum) and 10 % of the time, and we select a random action. Also, here we define our starting step-index (which is (4,1)). Next, we begin our first episode and using a while loop to continue the episode until we reach the end node /index. So, once we choose our first action, we need to update our current state-based on that action. I will explain the various steps below :

- For the action "UP" we would only need to decrement a row value by 1
- For the action " down" we would need to increment the row value by 1
- Next, for the action of " Right, "we would need to increment a column by 1
- And lastly for the action of "left," we would need to decrement a column by 1

Once we have done this because this is a cliff world, we check if the agent has reached the cliff indices if he has, we would assign it a reward of '-100' and send it back to the starting node. Otherwise, we assign a reward of '-1'. This is done using a switch case in the code. Next, we also check for boundary conditions, and if we agent has breached it, we bring it back in by assigning it the closest column or row that it has crossed. Additionally, to make things clearer, the cliff conditions indexes include the following :
(Row, Column):(4,2),(4,3),(4,4),(4,4),(4,5),(4,6),(4,7),(4,8),(4,9),(4,10)(4,11)

Once these have been checked and the reward has been assigned, we sample the next action, also using the e-greedy policy. Then update the Q matrix using the given formula :

Q_2(r_2,c_2,action_2) = Q_2(r_2,c_2,action_2) +alpha*(R_2 + gamma*v_sum-Q_2(r_2,c_2,action_2));

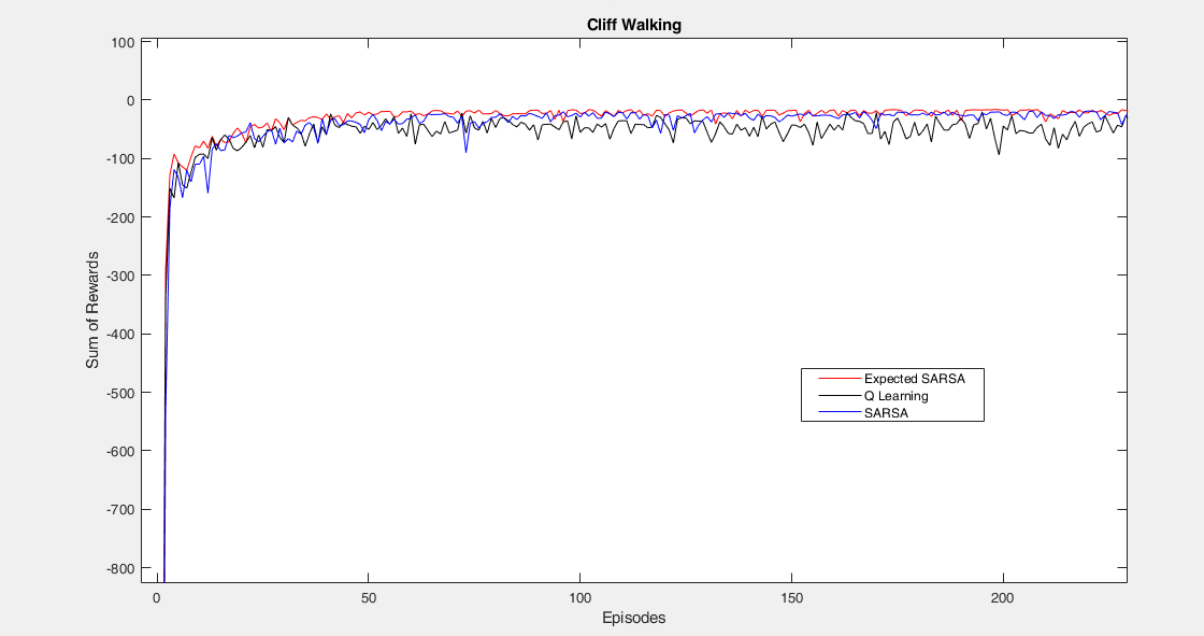
Here our V_sum is a product of the policy estimation of the various actions. Instead of sampling from the policy, we form an estimate of the actions and select the maximum among them. The rest remains the same.

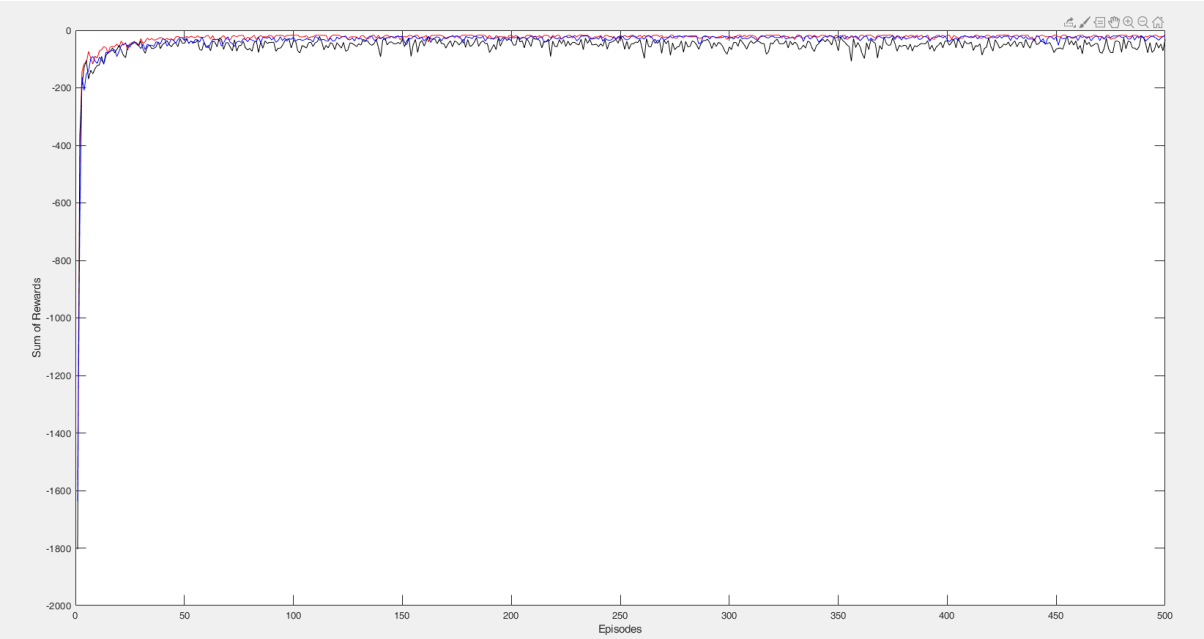
This helps us update the previous states and perform policy improvement after every time step/step. Also, the policy used is used to help us determine the next action that we would take to help estimate and predict policy evaluation. Once we update the Q values, the newly calculated actions, index values are passed onto the next step to redo all the steps all over again.

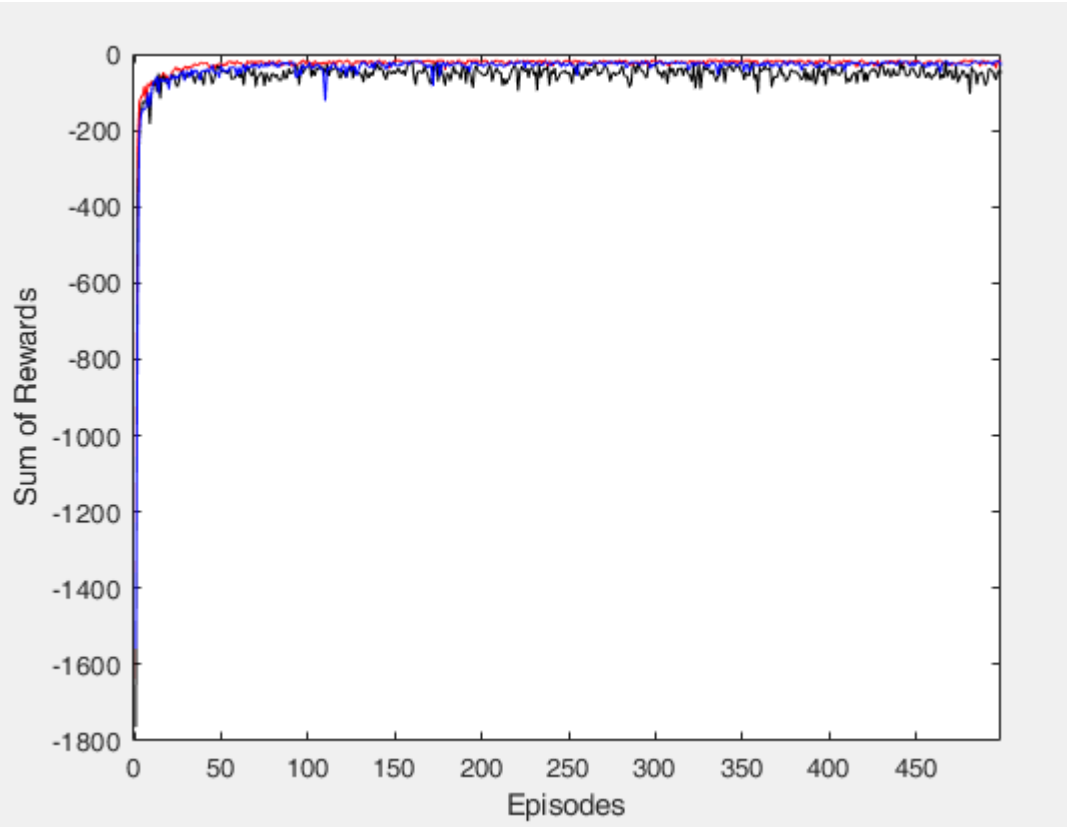
These steps are terminated when we reach the final index (4,23). Now we use these computed Q values to do another episode and so on. We continue this for 500 episodes and then redo this for 20 additional loops. This helps us achieve a smoother curve. We then sample 20 of them find their mean over the 10000-reward sum that we obtained and plot this sum vs. the episodes.

Plots:

I have shown a zoomed-in view as well to highlight the various algorithms and their results






0
-200
-400
-600
-800
-1000
-1200
-1400
-1600
-1800
Sum of Rewards
0
50
100
150
200
250
300
350
400
450
Episodes