Reinforcement Learning Assignment № 1

Ruslan Burakov, s1569105 11/02/2015

Note

All the code fragments are located in the end of the report in the appendix section. In order to reduce number of pages I don't attach unit tests.

1 Theoretical estimation

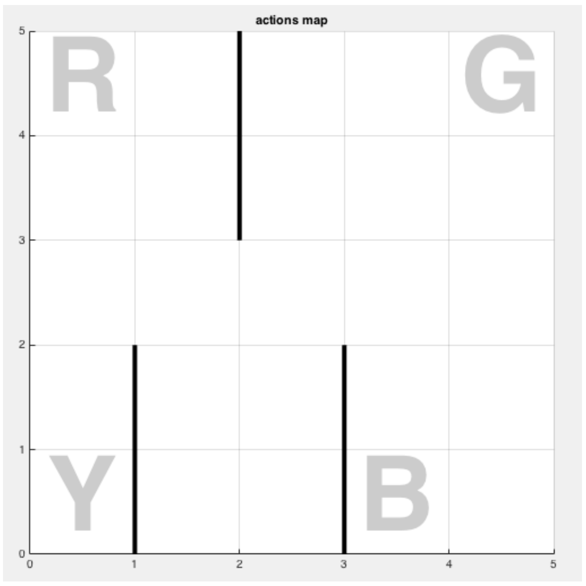


Figure 1: Taxi problem maze

1.1 Typical time horizon of the problem

The time horizon is the time needed for value functions in order to converge:

$$V = E(r_0 + \gamma r_1 + \gamma^2 r_2 + \dots) = E(\sum_{t=0}^{T} \gamma^t r_t) \tag{1}$$

Where E - means expectation, V - value function, r_t - immediate reward on the time-step t, γ - discount factor, and T - is either ∞ or expected duration of the episode. Here in the taxi problem we deal with the episodic problem as we here a terminal state (the passenger is dropped out in the target destination). The duration of the episode for the optimal policy is somewhere between 1 step which describes the best case when we start with the passenger within taxi in the exact drop-off location (for example in R location on Figure 1) and the worst scenario of $9*2 = 18$ when we start in Y location and the passenger in G, and drop-off location is Y again. The probability of the best case is small $4\frac{1}{25}\frac{1}{5}\frac{1}{4} \approx 0.008$ (using the number of possible states in the maze and the fact that there are only 4 best possible cases: R, G, Y, B) and the worst case probability is small as well $\frac{1}{25}\frac{1}{5}\frac{1}{4} \approx 0.002$. That is why I expect average time horizon to be close to 10 steps. Using formula for sum of geometric progression in the lecture it was shown that time horizon can be approximately connected to discount factor γ by $time\ horizon = \frac{1}{1-\gamma}$. Therefore, the γ is $1 - \frac{1}{T} = 1 - \frac{1}{10} = 0.9$

1.2 Mean (immediate) reward for a good policy

In the good policy agent doesn't bump into the walls or try to pick/drop off passenger in the wrong location so cumulative reward for one episode must be positive. There are two positive rewards in this problem: successful pick up and successful drop off. For successful pick on average we get $\frac{1}{5} * 0 + \frac{4}{5} * 1 = 0.8$ where 0 corresponds to the case when the passenger appears within the taxi from the beginning. The drop off reward is calculated using the time since last pick up $r_{dropoff} = \frac{10}{(time\ since\ pick)}$ and is determined by $4x5$ possible combinations of passenger and drop off locations. The knowledge of the number of optimal actions to do since pick up (for example to drop off, from G to R it will take 9 actions Figure1) allows to calculate average drop off reward $\frac{1}{16}(4*10+4*\frac{10}{9}+2*\frac{10}{5}+4*\frac{10}{8}+2*\frac{10}{6}) = 3.549$ (omitting the case when the passenger is already in the taxi). Summing reward for the pick and average reward for drop off gives **4.549** as our estimation of average cumulative (means sum of all received immediate rewards during one episode) reward in one episode.

1.3 maximal value of a state-action pair

The maximal value state action pair is the any of best cases discussed above. It is the taxi in the drop off position and the passenger within it and its action is drop off. In my implementation once the **correct** drop off location is reached the passenger is out of the taxi and its location becomes drop off location and this is reflected in the states. The reason why it is important is that Q-learning converges to the right policy but not necessary right values [SB98]. Here I consider case for G drop off location but it will be the same for any other drop off location. Q(taxi in G, passenger within taxi, drop off in G, action - drop off) = Q(G, IN, G, drop) and Q(taxi in G, passenger in G, drop off in G, action - pick up) = Q(G, G, G, pick). So in my case I when I try to drop off my Q(G, IN, G, drop) updated as following:

$$Q_{t+1}(G, IN, G, drop) = Q_t(G, IN, G, drop) + \alpha(r_{drop} + \gamma max_a Q_t(G, G, G, a) - Q_t(G, IN, G, drop)) \tag{2}$$

The maximum value of the next state (although in reality there is nothing after terminal state but in the tutorial examples provided to us it was solved in the same way) there will be Q(G, G, G, pick) and:

$$Q_{t+1}(G, IN, G, drop) = Q_t(G, IN, G, drop) + \alpha(r_{drop} + \gamma Q_t(G, G, G, pick) - Q_t(G, IN, G, drop)) \tag{3}$$

The update formula for Q(G, G, G, pick) (remembering that reward for pick up is 1):

$$Q_{t+1}(G, G, G, pick) = Q_t(G, G, G, pick) + \alpha(1 + \gamma Q_t(G, IN, G, drop) - Q_t(G, G, G, pick)) \quad (4)$$

Assuming that our Q values will eventually converge (t goes towards very big value) $Q_{t+1} = Q_t$ we will receive the following system of linear equations:

$$r_{drop} + \gamma Q(G, G, G, pick) - Q(G, IN, G, drop) = 0 1 + \gamma Q(G, IN, G, drop) - Q(G, G, G, pick) = 0 \quad (5)$$

Solving this system we obtain:

$$Q(G, IN, G, drop) = \frac{r_{drop} + \gamma}{1 - \gamma^2} \quad (6)$$

estimating average reward r for successful drop off similarly to previous section we get:

$$r_{drop} = (10 + 10/6 + 10/9 + 10/9) * \frac{1}{4} = 3.47 \quad (7)$$

After that substituting all the values into equation 6 we receive:

$$Q(G, IN, G, drop) = \mathbf{23.15} \quad (8)$$

1.4 number of trials

Well each of $5*5*5*4 = 500$ states must be visited at least several times. Let's set the number of times visited to the determined before time horizon of 10 steps. So roughly it is 5000 trials but of course it is worth investigating it more during computation

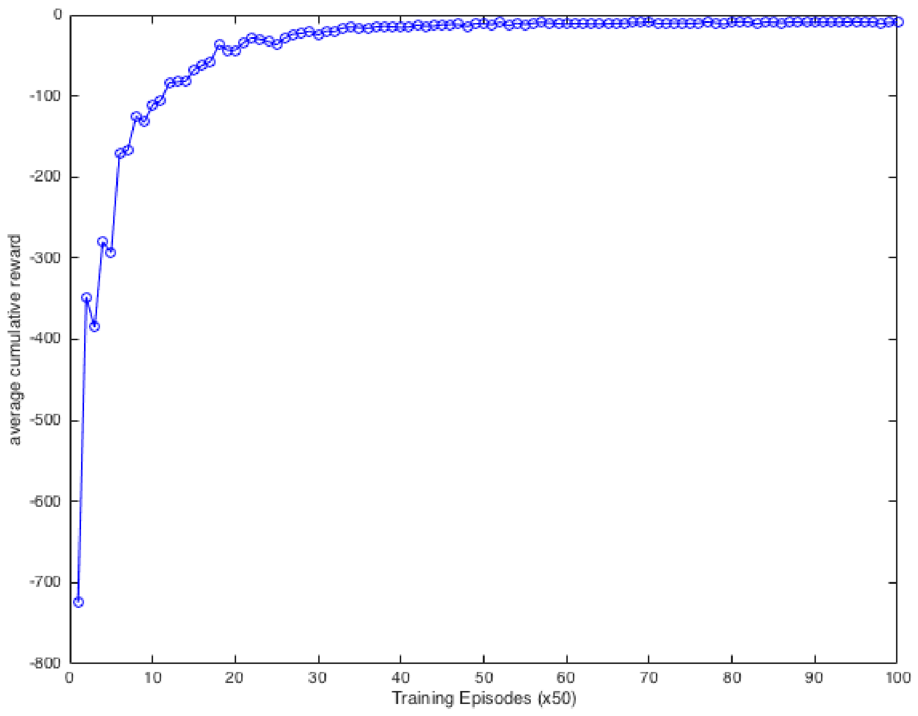
1.5 Stochastic vs Deterministic

We are using mixed exploring strategy (epsilon greedy, sometimes acting deterministically but with probability epsilon we are acting randomly). In the maze the results of our actions are deterministic given chosen action and current state. Given state representation optimal sequence of actions looks determined because all rewards are deterministic. So it points that it is deterministic problem but I note that from agent point of view the drop off reward is not always deterministic. Because the time since the last pick up is not a part of our state representation the agent can only guess and predict average reward for its drop off action although it is not driven from any distribution.

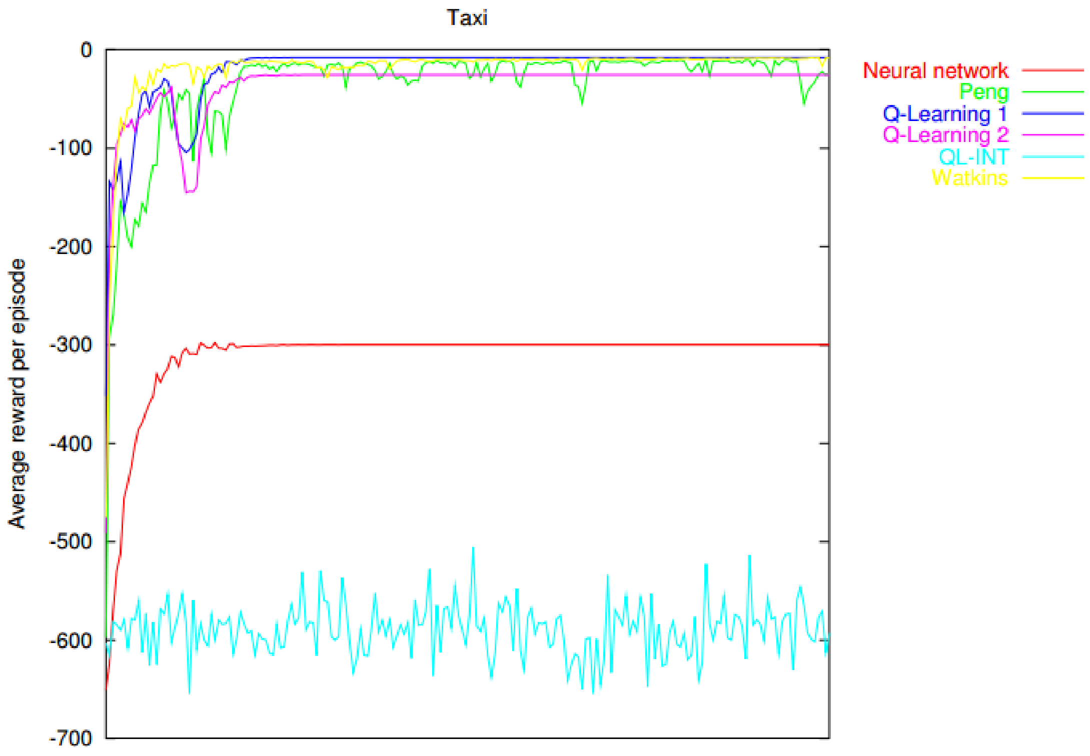
2 Solve the problem using Q-learning

2.1 Test correctness by comparing with known results

I wasn't sure whether my implementation (The Q learning code is in Q_learning_episode.m.) was right or wrong. That is why I have implemented reward function (code in trans_fun_paper.m) from the original benchmark [Dut+05] from which this homework was created in order to see the similarity between their results and my. I used parameters from there for QL1:



(a) Our reproduction of the paper [Dut+05] Q-learning 1 result



(b) Original plot from the paper [Dut+05] for taxi problem

Figure 2

ϵ	ϵ decay	γ	α	α decay
0.5	0.999	0.9	0.5	0.99999

The result obtained after 5000 episodes can be seen on Figure 2. Here cumulative reward is averaged per 50 episodes.

As it can be seen the results for Q-learning 1 are converging to optimal policy with average cumulative reward around -5 in episode after $40 * 50 = 2000$ time steps. In the paper [Dut+05] the different reward function is used and -5 is the expected average cumulative reward for it. So it means our simulation converges to optimal policy.
How Q values converge can be judged by the maximum absolute difference in Q values between two episode of the problem. This measure indicates the order of fluctuation in Q values during the experiment. I am using this metric instead of variance because it shows the performance of the most unstable state so it plays role of upper-bound. After that by plotting the average of this metric (over 50 episodes, for example), it is possible to judge how fast the Q values are converging. On the other hand, the policy may converge to optimal earlier.

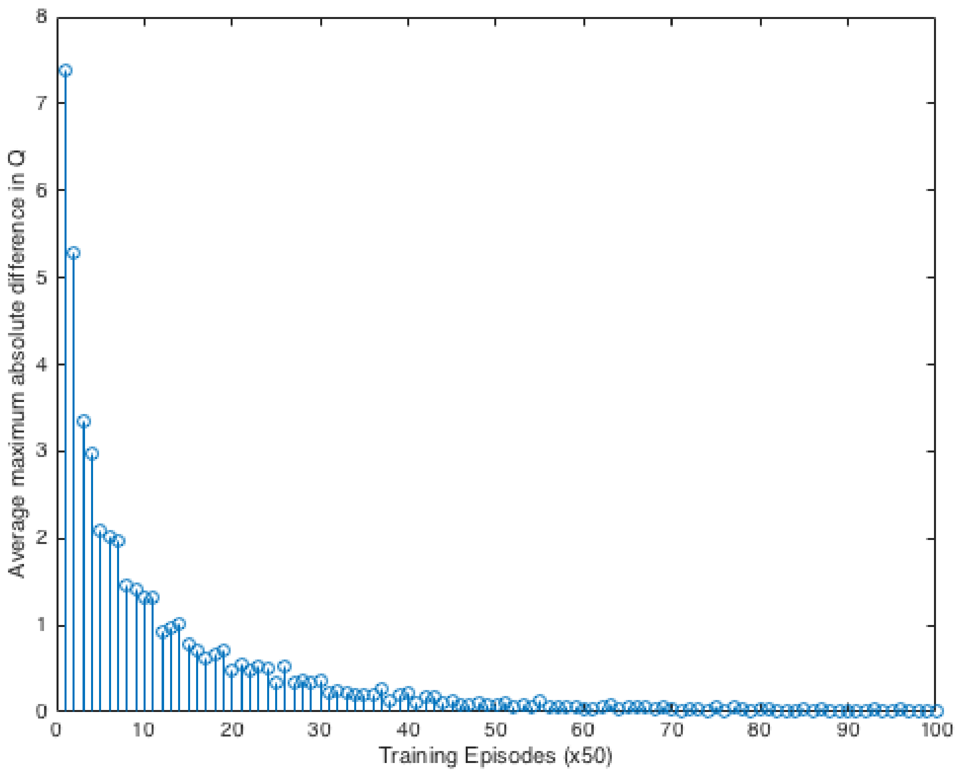


Figure 3: Q maximum absolute difference for reward function taken from paper [Dut+05]

Therefore, Figure 3 points out that after $45 * 50 = 2250$ time steps the Q-values magnitude of fluctuations goes below 0.4.
The match between paper results and my gave me additional confidence that q-learning implementation is correct.
Finally, I project final Q-values on the partially fixed state where passenger is within taxi and the drop off location is G on Figure 4. As it may be expected, the value in location G is the biggest

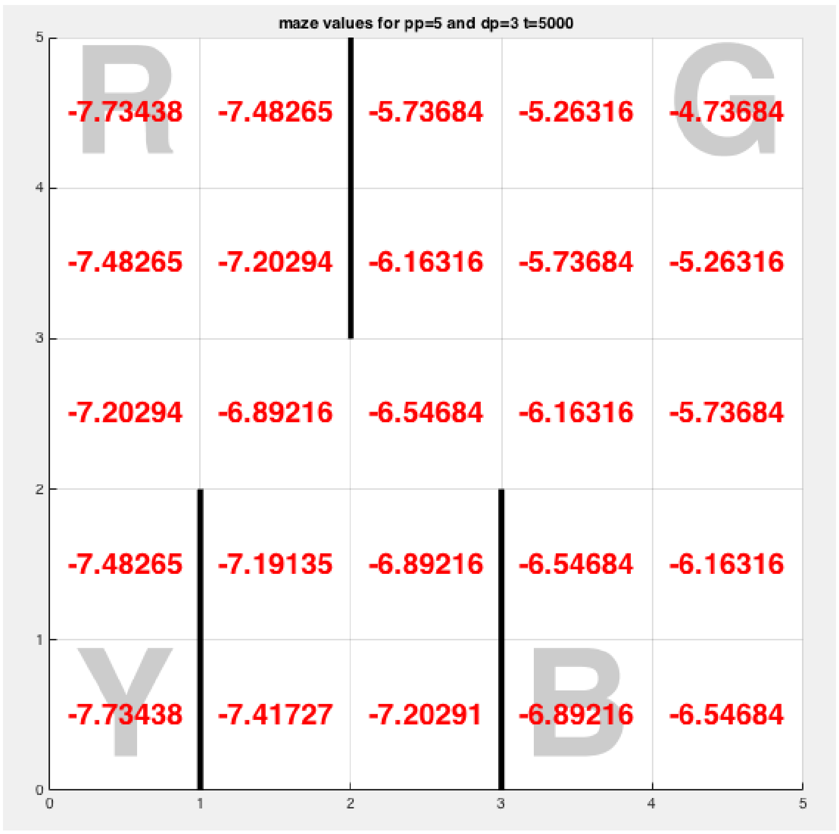


Figure 4: Values for case when passenger is within taxi and drop off location is in G for reward function [Dut+05]

one and all other values decreases very symmetrically from it depending on the distance.

2.2 Q learning performance with homework reward function

I tried Q-learning with parameters from the previous answer (with the exploration parameters such ϵ and α with corresponding exponential decays taken similarly from the paper [Dut+05] from previous section):

ϵ	ϵ decay	γ	α	α decay
0.5	0.999	0.9	0.5	0.9999

On Figure 5 (note here it is averaged over 100 episodes) it can be seen that our rough estimation of cumulative reward in one episode was correct. The average cumulative reward value is close to 3.5 and the predicted maximum value was 4.549.

In order to draw the maximum value case as 2-maze I project my final Q-values on the partially fixed state where passenger is within taxi and the drop off location is G. The values obtained are on Figure 6. You can see that in the location G which corresponds to maximum value state the value is equal to 22.95. This is quite close to our estimation from question 1 which is 23.15.

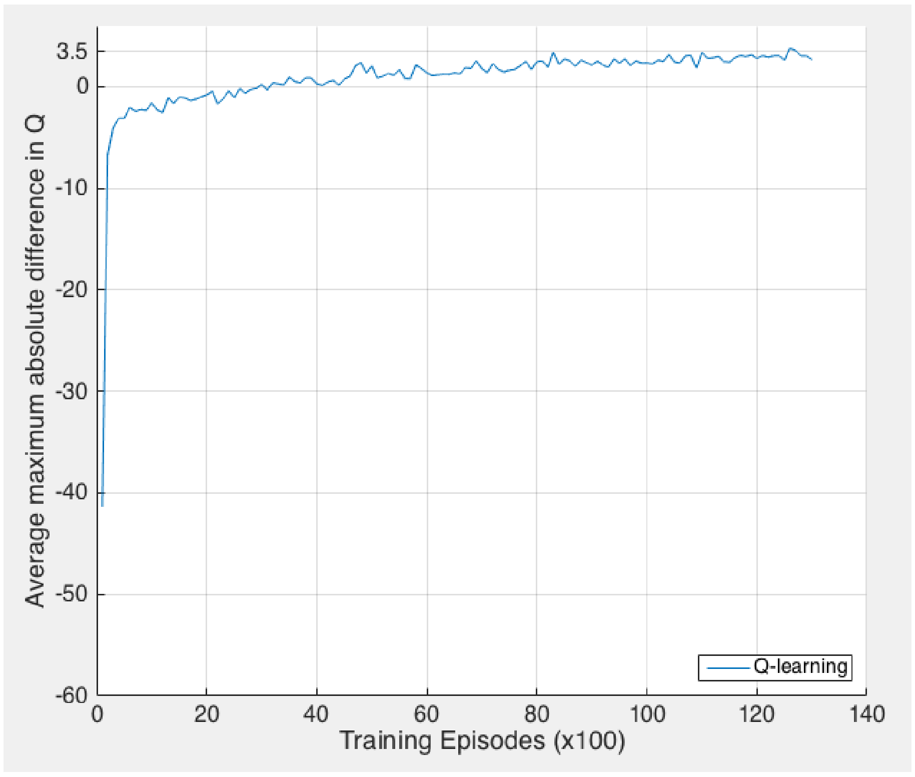


Figure 5: Q learning average cumulative reward for homework reward function

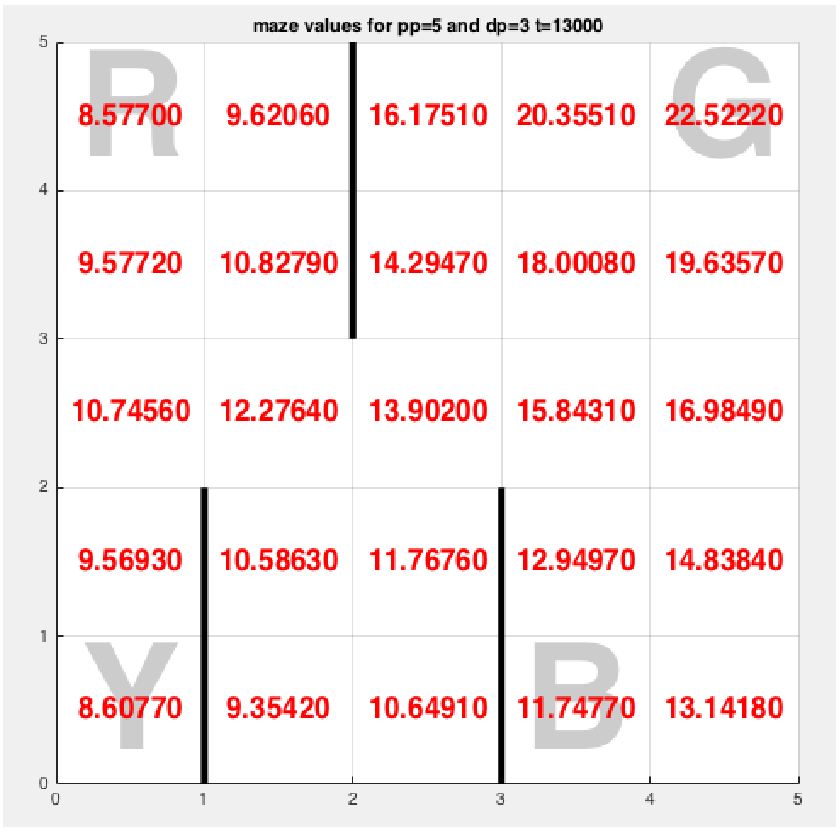


Figure 6: Values for case when passenger is within taxi and drop off location is in G for howework reward function

2.3 Represent your solution using an example

I use the same example as in previous question where passenger is within taxi and the drop off location is G on Figure 6. It can be seen that in contrast to results obtained with reward function from the paper [Dut+05] the values are much less symmetric depending on the distance to the G location. The possible explanation of this is that fluctuation of values in simulation with homework reward function are much bigger. It explained further in next question. The policy generated for this example is on Figure 7. The generated policy is optimal in the sense that it will try to reach drop off location as fast as possible.

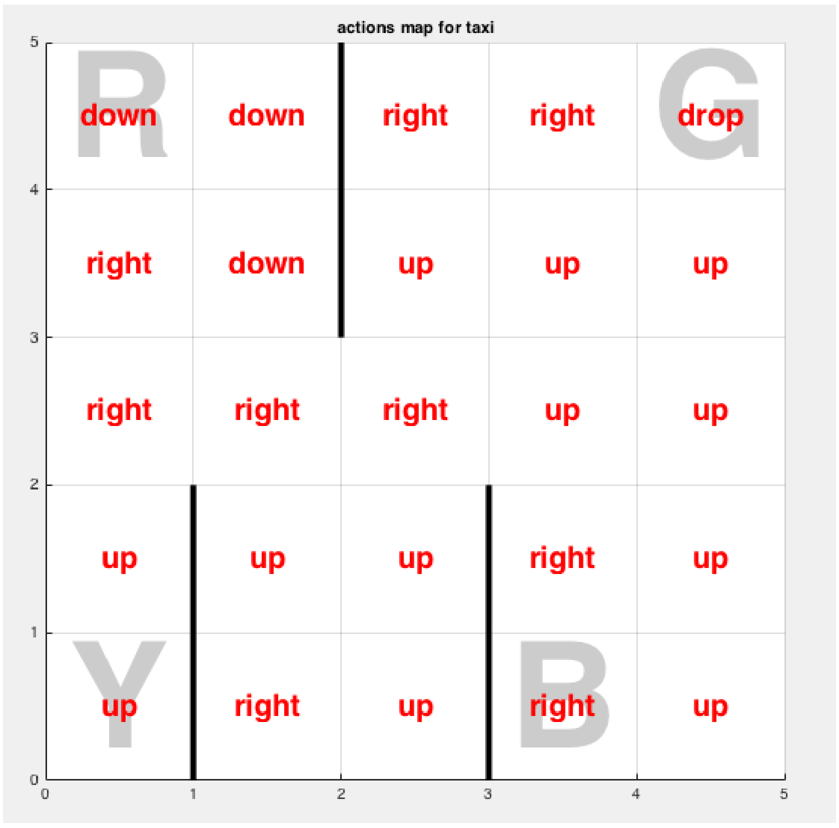


Figure 7: Best actions for case when passenger is within taxi and drop off location is in G

2.4 The time course of the values and the reward

On Figure 5 we can see that average cumulative reward converged (stabilized) on $100 * 100 = 10000$ time step. Approximately at the same time step maximum absolute difference on Figure 8 reaches the value of 0.7. However, not that the maximum fluctuation here is more than 5 times bigger than on Figure 3. This can be described by the fact that in the paper [Dut+05] the expected cumulative reward per episode is linear function of distance between pick up and drop off state, as they punish by -1 for each movement and give reward 0 for successful drop off. In our case the reward for successful drop off is 10/(time since pick) which is hyperbolic function and it changes much more suddenly. So depending on the random initial state in the episode the cumulative reward may vary significantly even with the optimal policy. In order to account such randomness the α parameter must be lower so more of the old values must be preserved and

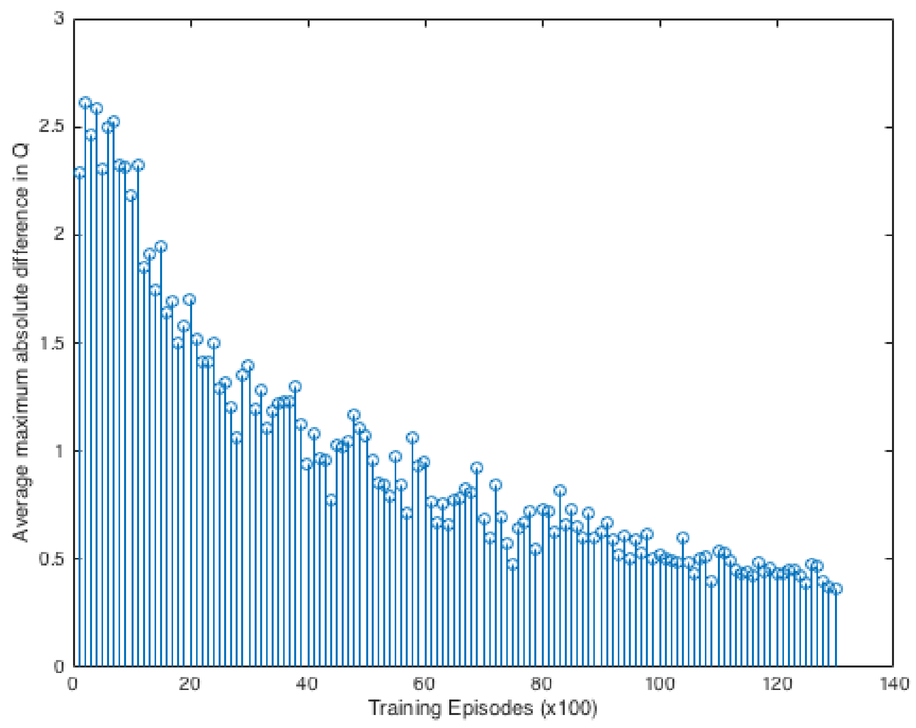


Figure 8: Q maximum absolute difference for homework reward function

the averaging of the Q-values will happen much smoother. But in that case the required number of time steps may increase drastically.

2.5 the convergence time depending on exploration strategy

Previously I was using exponential decays for exploration parameter ϵ of ϵ-greedy strategy. That is why I have decided to fix value of ϵ (exponential decay parameter equals to 1.0) and see what convergence time I will achieve. All other parameters are taken from previous question and fixed:

γ	α	α decay
0.9	0.5	0.9999

Judging by results obtained on Figure 9 the convergence time is roughly the same on the $3 * 100 = 300$ time step. Simulation with ϵ 0.5 converges to the cumulative reward close to -5 which is way below expected optimal one of 4.53. But this is expected as it acts randomly in half of the cases. The best result which is very close to the expected optimal cumulative reward is achieved by strategy with ϵ 0.01. This implies that in our problem even so little of exploration is enough to act nearly optimally. It can be described by the fact that we don't need to learn so much about all possible states. The most important information is that once we picked person we must reach drop off location with maximum speed.

After that I used greedy search approach and given that the most effective ϵ so far is 0.01 I have decided to find out what fixed value of α (interpolation parameter) will give the best results

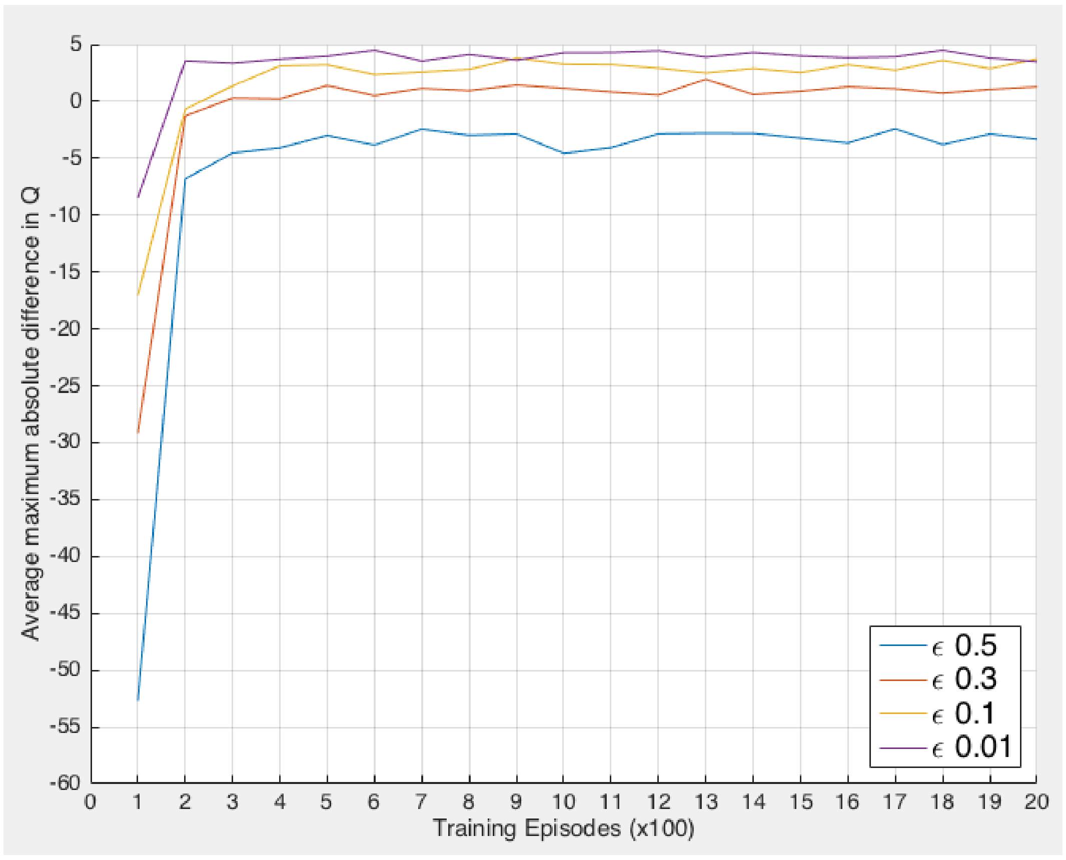


Figure 9: Average cumulative reward in one episode for different ϵ

γ	ϵ	ϵ decay	α decay
0.9	0.01	1.0	1.0

On Figure 10 the difference between difference α is almost negligible. Although α 0.9 converges slightly slower than others. This is expected as primal exploration parameter is ϵ and here with ϵ 0.01 we mostly update states greedily.
Finally, based on above results, I tried to get better average cumulative reward (closer to theoretical predicted one 4.549) by exploring for longer period of time.

ϵ	ϵ decay	γ	α	α decay
0.5	0.9998	0.9	0.5	1.0

As we can see on Figure 11 the average cumulative reward is much closer to the theoretically predicted one.

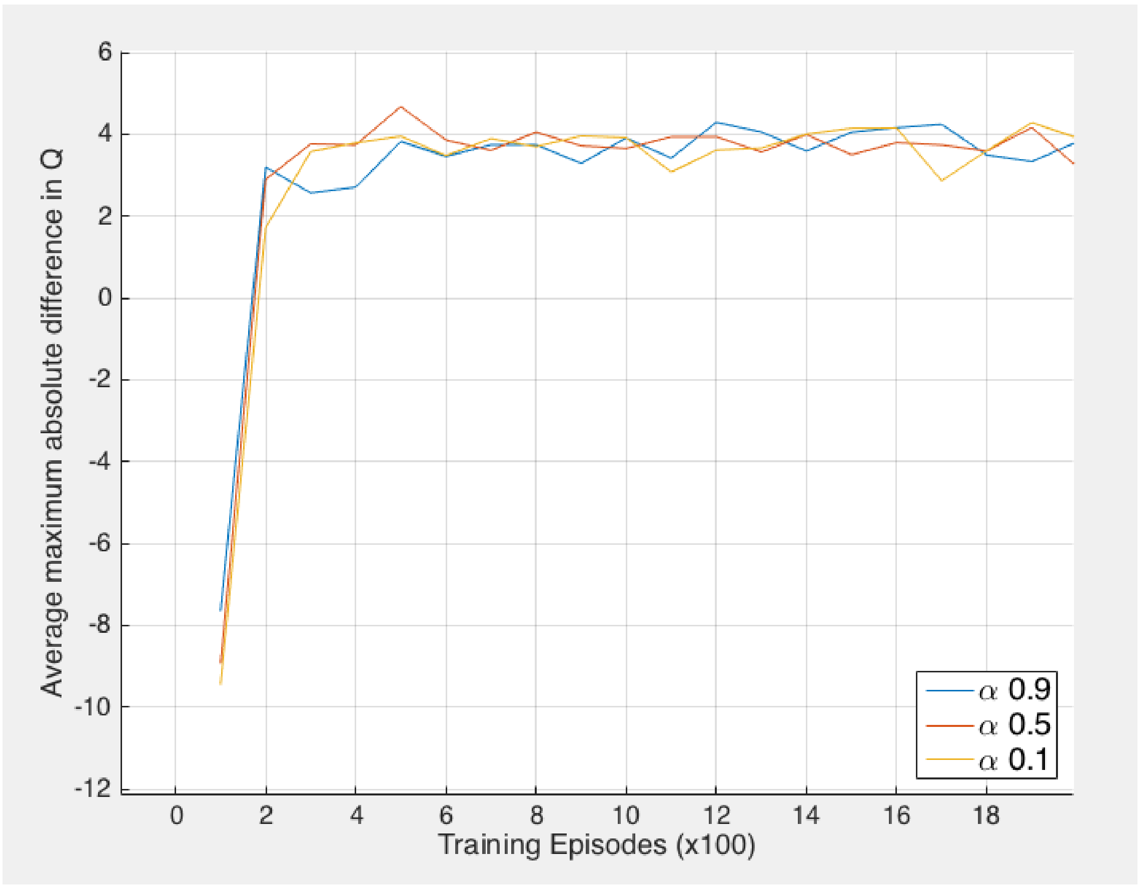


Figure 10: Average cumulative reward in one episode for different α

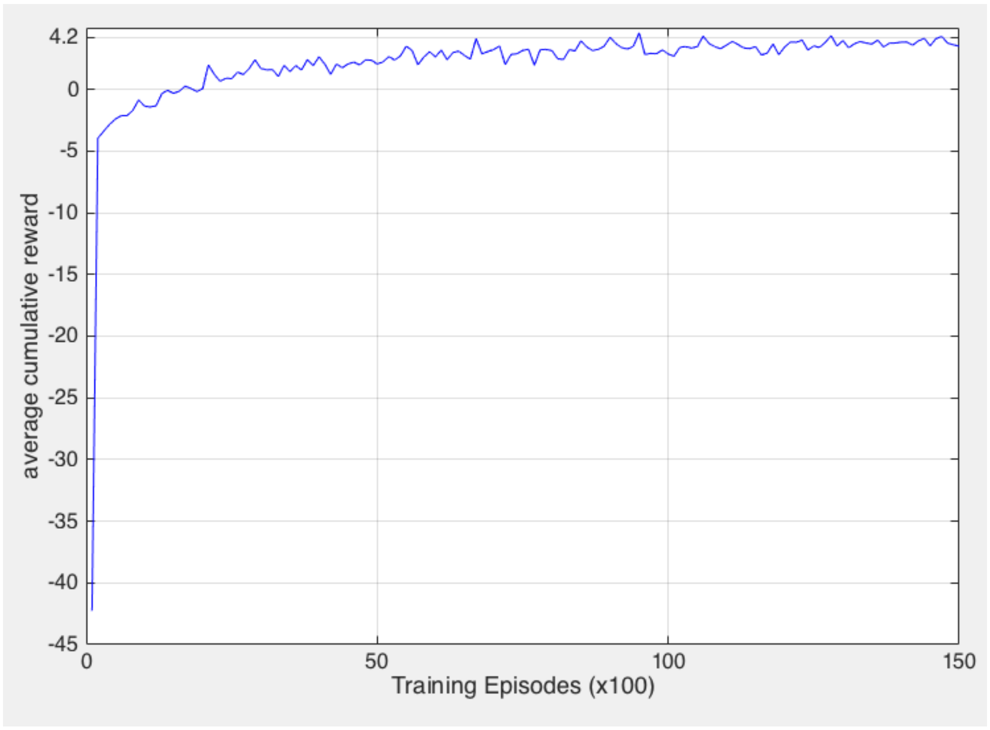


Figure 11: Average cumulative reward in one episode for Q-learning with longer exploring phase

3 SARSA

The SARSA code is in SARSA_episode.m. I am using initial parameters from section 2.2 for SARSA, so they can be comparable.

ϵ	ϵ decay	γ	α	α decay
0.5	0.9998	0.9	0.5	0.9999

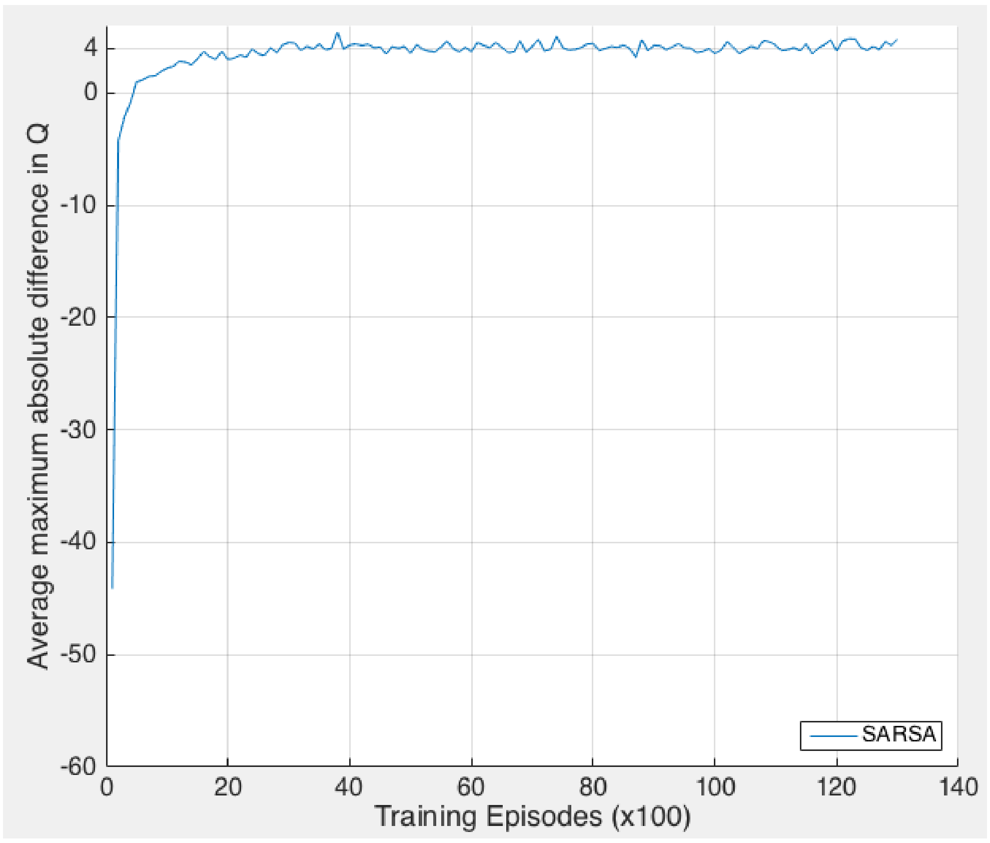


Figure 12: Average cumulative reward in one episode using SARSA

Figure 12 indicates that SARSA converges to approximately 4.1 cumulative reward which is not far from of the expected 4.549 for the taxi problem.

Values for case when passenger is within taxi and drop off location is in G are on Figure 13. In my case that epsilon is decaying parameter so it becomes small relatively quickly. For instance, $0.5 * (0.999)^3 910 = 0.01$. So on time step 3910 exploration rate is 0.01 and behaviour of SARSA and Q-learning becomes very similar as they both start to act very greedily. That is why on Figure 13 the value G converges to almost the same expected value of 23.15 as in case of Q-learning.

The same reasoning can be applied to describe similarity of Q maximum absolute difference generated by SARSA on Figure 14 to the Q maximum absolute difference dynamic obtained using Q-learning on Figure 8.

The policy generated by SARSA for the case when passenger is within taxi and drop off location is in G on Figure 15. The notable distinction is that in 3 cells $(\{2,5\},\{3,1\},\{3,5\})$ the action chosen differs from what we would consider optimal. It means that SARSA didn't explore

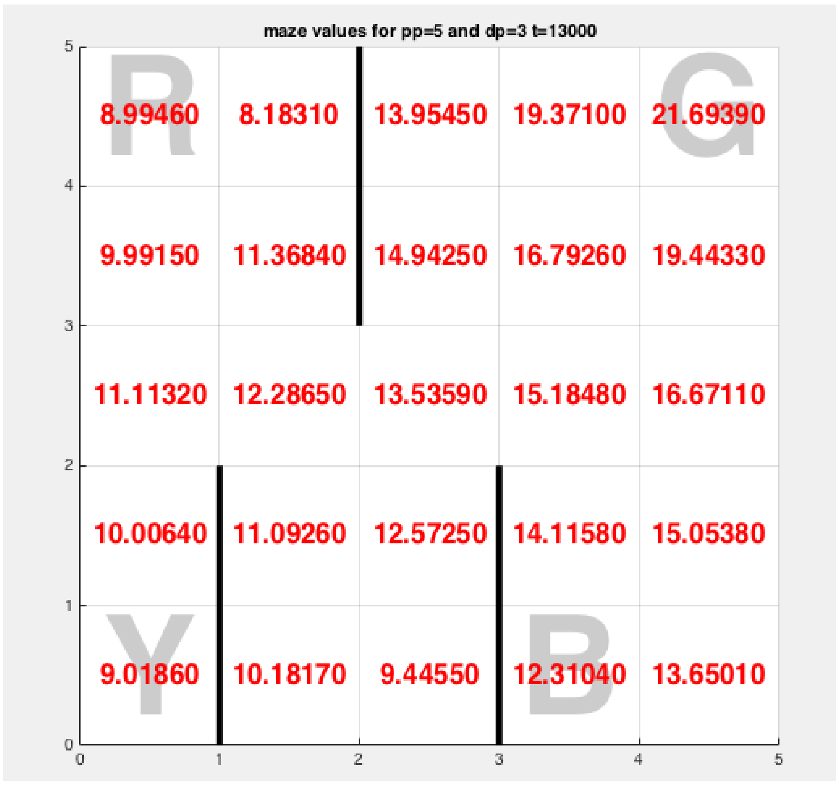


Figure 13: Final Values using SARSA for the case when passenger is within taxi and drop off location is in G

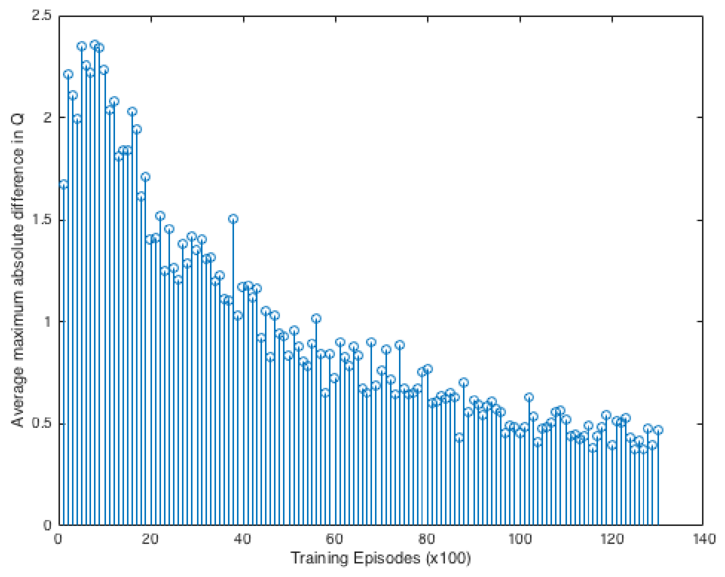


Figure 14: Q maximum absolute difference using SARSA

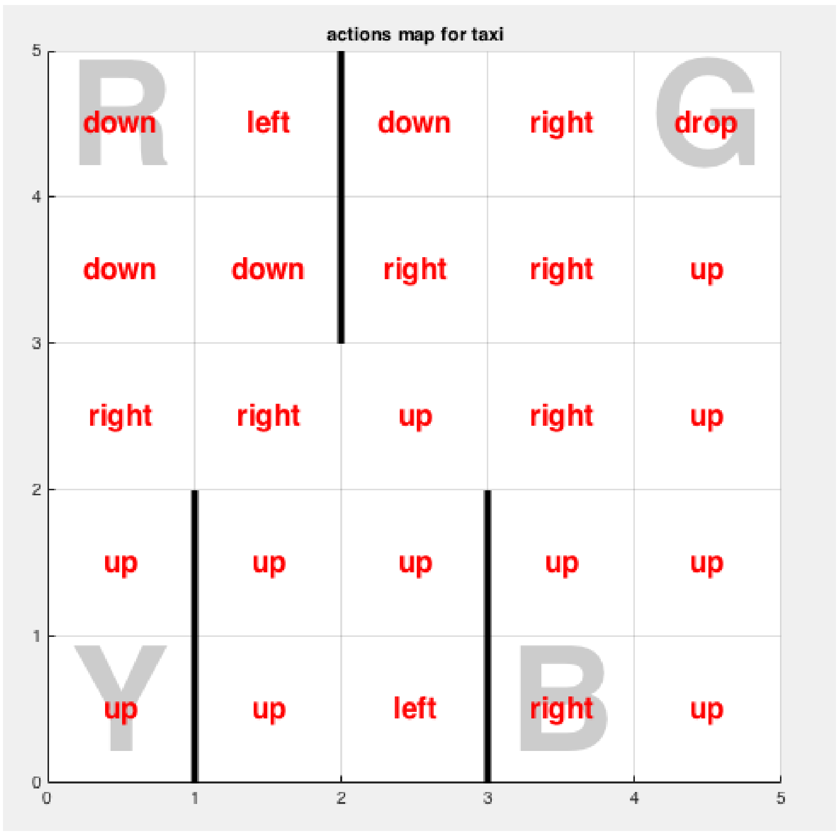


Figure 15: Generated policy using SARSA for the case when passenger is within taxi and drop off location is in G

these states enough. This can be describe by the fact that the probability of visiting these states is very small. Agent can appear in them only when exploring or when it was generated with the passenger within taxi in these particular cells and its drop off location is G, so it is $\frac{1}{25}\frac{1}{5}\frac{1}{4} \approx 0.002$. Due to this non-optimality doesn't noticeably affect average cumulative reward on Figure 12. Let's try another parameters with longer exploration (parameters the same as in the end of section 2.5).

ϵ	ϵ decay	γ	α	α decay
0.5	0.9998	0.9	0.5	1.0

The results on Figure 16 for average cumulative reward are much more closer to theoretical value of 4.549. And this time the generated policy on Figure 17 is be optimal one.

Finally, I have run tests to find best fixed ϵ value using parameters as in previous question

γ	α	α decay
0.9	0.5	0.9999

The results obtained on Figure 18 are very similar to those of Q-learning from previous question. The discussion why it can be like that in next question.

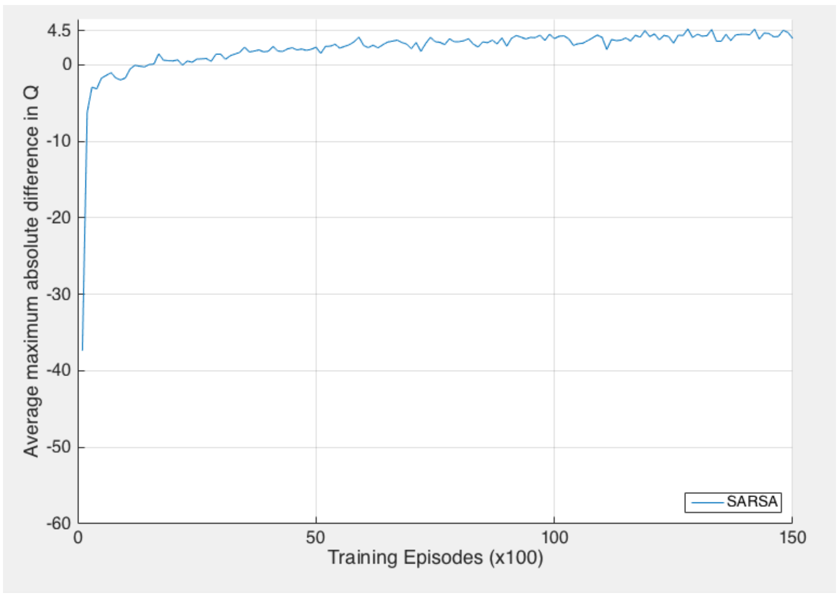


Figure 16: Average cumulative reward in one episode using SARSA with longer exploration phase

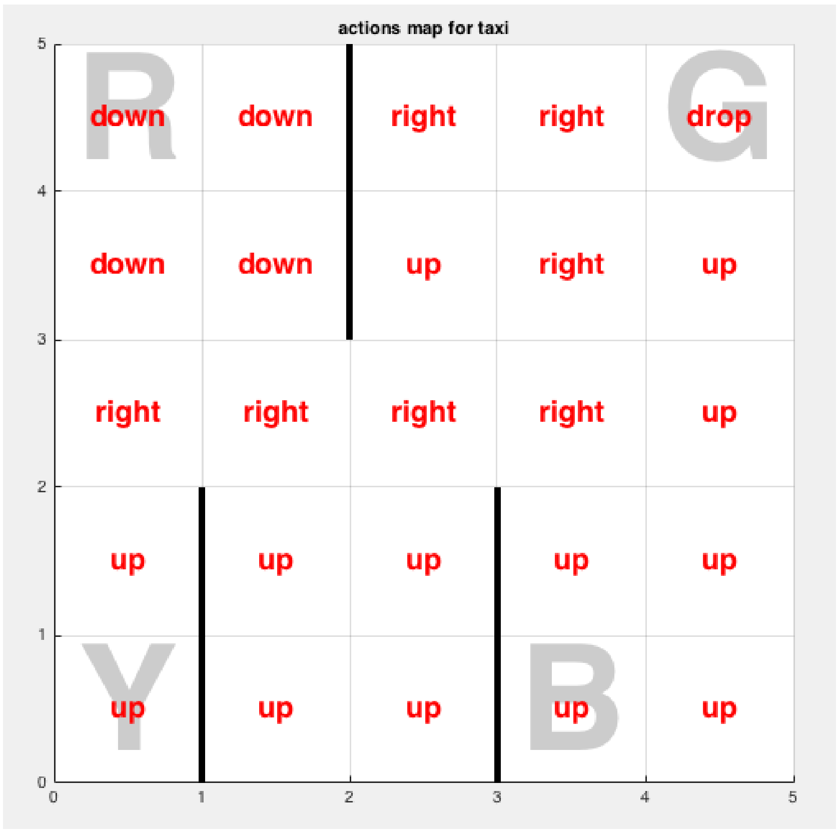


Figure 17: Generated policy using SARSA with longer exploration phase for the case when passenger is within taxi and drop off location is in G

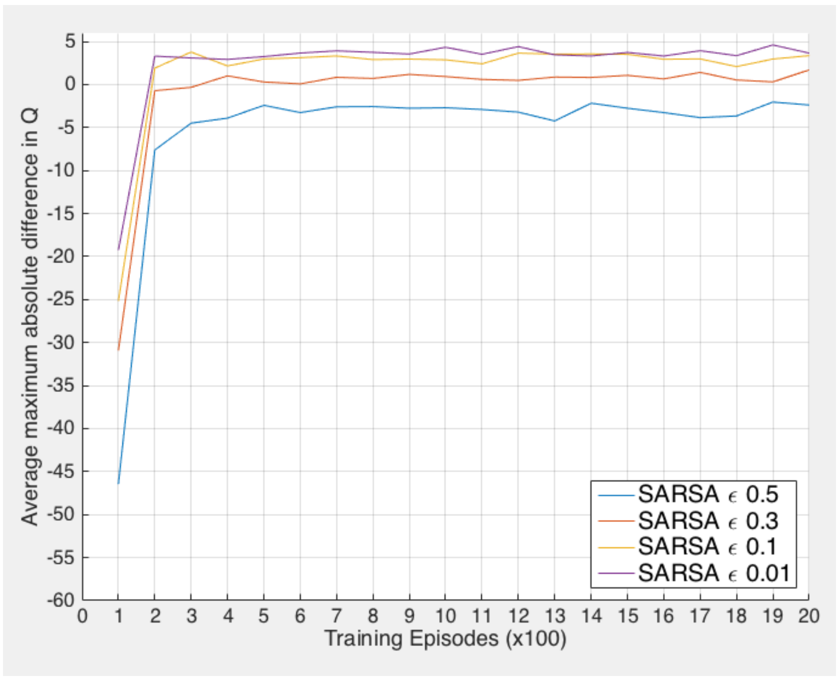


Figure 18: Average cumulative reward in one episode for different ϵ using SARSA

4 Comparison of Q-learning and SARSA

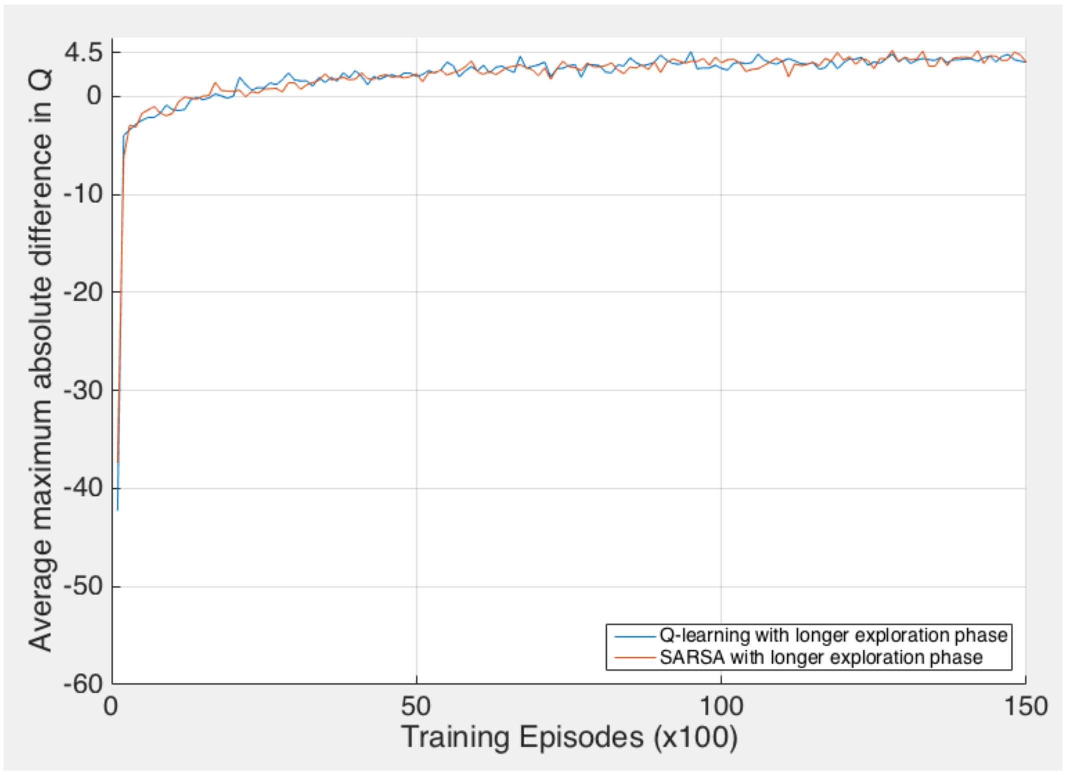
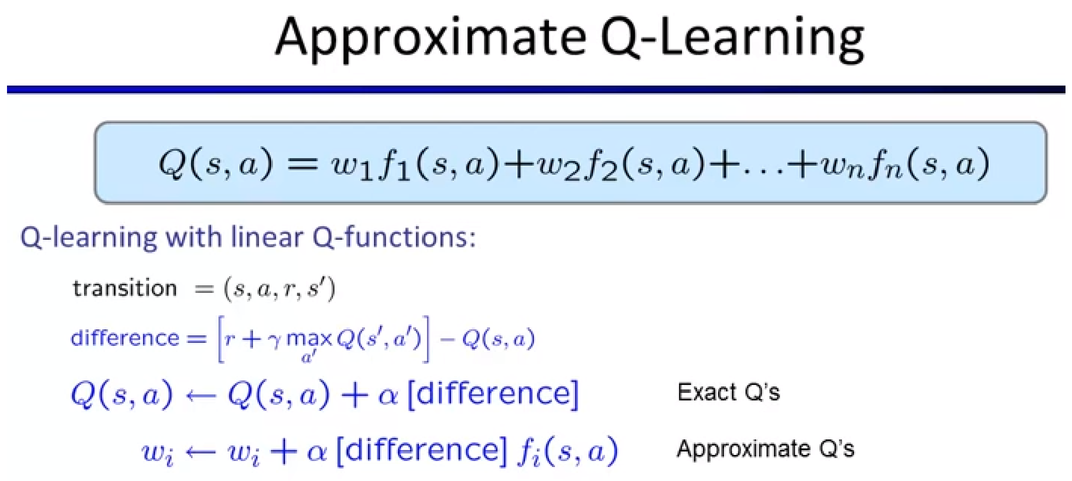


Figure 19: Average cumulative reward comparison for Q-learning and SARSA in the case with longer exploration phase

In this task the difference between Q-learning and SARSA is almost negligible in contrast to cliff example from [SB98]. Comparing results for the cases with longer exploration phases from previous sections on Figure 19 we can see how close obtained results are. Such performance can be explained by the fact that when ϵ goes to 0 both algorithms start to act similarly. Another reason is that taxi problem doesn't have adequate cliff analogue. We can unsuccessfully pick or drop passenger in almost any state, so SARSA couldn't learn to avoid some dangerous states knowing that it may act randomly like cliff example. The walls of the maze can't act as cliff as well, because the penalty for them is -1 the average reward is bigger than that. And once again in the maze almost in any state there are walls around the agent. Although probably in this problem SARSA requires longer exploration phase as Figure 15 shows SARSA wasn't able to converge to completely optimal policy with short exploration phase in contrast to Q-Learning on Figure 7.

5 Improvements

Accordingly to the Berkley A.I. lecture 11 Reinforcement Learning II [Kle12] we can compress the information in our task by using features instead of direct states and approximate Q-values on:



Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Q-learning with linear Q-functions:

transition $= (s, a, r, s')$

difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$ Exact Q's

$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$ Approximate Q's

So by choosing let's 3 features: distance in the maze to the passenger location, distance to the drop off position, whether passenger reached location or not (either 1 or 0) we can achieve much better convergence as algorithm will be able to generalize problem. For example, by picking up passenger in the position R, the agent will learn that picking up the passenger in general is good thing and it will adjust weight w corresponding for the distance to passenger location (during learning this weight should become negative as the lower distance to the passenger the bigger Q value must become). The magnitude of weight for the distance to the drop off position must be much smaller than magnitude of weight corresponding to distance to passenger. This should be like because there is no point of trying reaching drop off position without passenger (I assume that when passenger is within taxi the distance to the passenger is zero). And the weight corresponding to passenger reached desired location should become something huge and positive as it is our final goal. On the other hand, in this scheme it is assumed that we can determine distance in the maze, when in our case agent learns the position of locations: R, B, Y and G all by itself.

6 Conclusion

In this taxi problem we used reward function which is different from the one described in [Dut+05]. It behaves non-linearly depending on the time since last pick up and as it was discussed earlier in section 2.4. As it was described in section 2.4 this non-linear reward function results in situation when it is very hard for our Q function to reach the stage when the values remains static. On the other hand, it is minor distinguish as we are mostly interested in the generated policy and in taxi task with homework reward function it converges to the optimal one.
One more. in section 4 we found the possible reasons why Q-learning and SARSA act so similarly in the taxi problem

7 Appendix

Main launch code where simulation starts and all learning parameters such as α (note that in the code I use eta instead for shortness), γ and ϵ are specified:

```
function [Q, rewards_per_episode, max_Q_diffs] = rl_taxi_sim(trans_f, learning)
    %reinforcement learning taxi problem simulation. Can be found here:
    % Dutech, A., Edmunds, T., Kok, J., Lagoudakis, M., Littman, M., Riedmiller, M., Whiteson, S.
    % (2005) Reinforcement learning benchmarks and bake-offs II. Workshop at Advances in Neural
    % Information Processing Systems conference


    % define maze global variables
    define_maze_global();
    global X Y PSNG_POS DROP_POS A MAX_EPISODE_TIME DROP_LOCS

    %local paramaters
    % total number of learning trials
    T = 5000;
    eps = 0.5;
    eps_decay = 0.999;
    gamma = 0.9;
    eta = 0.5;
    eta_decay = 0.99999;
    over_episode = 50;
    %initialisation
    Q = 0.1*rand(X, Y, PSNG_POS, DROP_POS, A);

    rewards_per_episode = [];
    max_Q_diffs = [];

    for t=1:T
        [Q_new, total_episode_reward] = learning(Q, eta, gamma, eps, trans_f);

        rewards_per_episode = [rewards_per_episode, total_episode_reward];

        Q_diff = abs(Q_new - Q);
        max_diff = max(Q_diff(:));
        max_Q_diffs = [max_Q_diffs, max_diff];

        %paramaters decay
        eta = eta * eta_decay;
        eps = eps * eps_decay;

        %print and values to control converges
        if (rem(t, over_episode)==0)
           display(eps, 'current eps');
           display(eta, 'current eta');

           figure(1);
           avg_rewards = mean(reshape(rewards_per_episode, over_episode, ↩
               []));
           plot(avg_rewards, 'b');
           ylabel('average cumulative reward');
           xlabel(['Training Episodes (x', num2str(over_episode),')']);

           %maze values for a case when
           %the passenger within taxi and drop off point is in G or in my
           %representation state with pp=5 and dp=3
           MV = max(Q,[],5);
           MV = MV(:, :, 5, 3);
           MV = reshape(MV, X, Y);
           %round to 4 digits on the left
           MV = round(MV, 4);
           lbls_MV = grid_to_lbls(MV, '%2.5f');
           figure(2);
           draw_maze(lbls_MV, strcat('maze values for pp=5 and dp=3 t=', ↩
               num2str(t)));

           figure(3);
           max_diffs = mean(reshape(max_Q_diffs, over_episode, []));
           stem(max_diffs);
           xlabel(['Training Episodes (x', num2str(over_episode),')']);
           ylabel('Average maximum absolute difference in Q');
        end
        Q = Q_new;
    end
end
```

Defining supporting global variable which are the same for all simulations, such drop off locations, walls positions, etc. (except probably - sometimes I change it):

```
function [] = define_maze_global()
    global X Y PSNG_POS DROP_POS DROP_LOCS A WALLS LOC_NAMES ↩
        MAX_EPISODE_TIME
    X = 5;
    Y = 5;
    PSNG_POS = 5;
    DROP_POS = 4;
    % number of actions
    A = 6;
    walls = [[[1,1], [2, 1]];...
             [[1,2], [2, 2]];...
             [[2,4], [3, 4]];...
             [[2,5], [3, 5]];...
             [[3,1], [4, 1]];...
             [[3,2], [4, 2]]];
    mirrored_walls = zeros(size(walls));
    mirrored_walls(:, [3, 1, 4, 2]) = walls(:, [1, 3, 2, 4]);
    WALLS = [walls; mirrored_walls];
    LOC_NAMES = {'Y', 'R', 'G', 'B'};
    DROP_LOCS = [[1,1]; [1,5]; [5,5]; [4,1]];
    MAX_EPISODE_TIME = 1500;
end
```

State representation of taxi problem:

```
function state = create_rnd_state()
    %x  - x position in the maze
    %y  - y position in the maze
    %pp - passenger position
    %dp - drop position
    global X Y PSNG_POS DROP_POS
    state = struct('x', randi(X),...
                   'y', randi(Y),...
                   'pp',randi(PSNG_POS),...
                   'dp',randi(DROP_POS));
end
```

```
function copy_s = copy_state(s)
    %s - state
    copy_s = struct('x', s.x,...
                    'y', s.y,...
                    'pp',s.pp,...
                    'dp',s.dp);
end
```

Support drawing utilities:

```
function lbls = grid_to_lbls( grid_values, varargin)
    %transpose because the first axe is X, and the second is Y
    %it is important for draw_maze
```

```matlab

    flat_V = reshape(grid_values',[],1);
    if length(varargin) == 0
        fmt = '%d';
    else
        fmt = char(varargin(1));
    end
    lbls = strtrim(cellstr(num2str(flat_V, fmt)));
end
```

```matlab
function [] = draw_maze(lbls, name)
    %lbls - labels - cell array of strings to display in the maze grid
    %size X*Y
    %name - plot title
    global WALLS X Y DROP_LOCS LOC_NAMES
    %draw location names with light green color

    hold on;
    grid on;
    h = findobj(gca,'Type','Text');
    delete(h);
    for i = 1:size(DROP_LOCS,1)
        %substract shifts for centering, must be in sync with font size
        dlx = DROP_LOCS(i,1) - 0.5;
        dly = DROP_LOCS(i,2) - 0.5;
        loc_name = LOC_NAMES(i);
        %
        text(dlx, dly, loc_name,...
            'Color', [0.8 0.8 0.8],...
            'FontWeight', 'bold', 'FontUnits', 'normalized', 'FontSize', ↩
               1/Y,...
            'HorizontalAlignment', 'center',...
            'VerticalAlignment', 'middle');
    end
    title(name);
    xlim([0 X]);
    ylim([0 Y]);
    set(gca,'XTick',0:X);
    set(gca,'YTick',0:Y);
    %to have nice grid cells as squares
    axis square;

    % grid domains
    xg = 1:X;
    yg = 1:Y;
    %label coordinates, substract 0.5 for centering
    [xlbl, ylbl] = meshgrid(xg - 0.5, yg - 0.5);
    text(xlbl(:), ylbl(:), lbls(:),'color','r',...
        'HorizontalAlignment','center','VerticalAlignment','middle',...
        'FontWeight', 'bold', 'FontUnits', 'normalized', 'FontSize', 1/(5*↩
            Y));
    %plot maze walls

    for i = 1:size(WALLS, 1)
        wall = WALLS(i, :);
        %centering
        wall = wall - 0.5;
        x1 = wall(1);
        y1 = wall(2);
        x2 = wall(3);
        y2 = wall(4);
        %vertical
        if y1 == y2
            wx = (x2 - x1)*0.5 + x1;
            plot([wx, wx], [y1 - 0.5, y1 + 0.5],'-k', 'LineWidth',4);
        end
        %horizontal
        if x1 == x2
            wy = (y2 - y1)*0.5 + y1;
            plot([x1 - 0.5, x1 + 0.5], [wy, wy],'-k', 'LineWidth',4);
        end
    end
    hold off;
    drawnow;
end
```

```matlab
function [ lbls_M ] = actions2str(lbls_M_raw)
    %lbls_M_raw - array of string cells, with numeric actions
    lbls_M = lbls_M_raw(1:numel(lbls_M_raw));
    map = {'up', 'down', 'left', 'right', 'pick', 'drop'};
    lbls_M(strcmp('0', lbls_M)) = {''};
    lbls_M(strcmp('1', lbls_M)) = {'up'};
    lbls_M(strcmp('2', lbls_M)) = {'down'};
    lbls_M(strcmp('3', lbls_M)) = {'left'};
    lbls_M(strcmp('4', lbls_M)) = {'right'};
    lbls_M(strcmp('5', lbls_M)) = {'pick'};
    lbls_M(strcmp('6', lbls_M)) = {'drop'};
    for a1=1:6
      for a2=1:6
        action_combo = a2 + a1 * 6;
```

```
            lbls_M(strcmp(num2str(action_combo), lbls_M)) = {[map{a1}, '/', ↩
                map{a2}]};
        end
      end
end
```

```
function [] = plot_policy(Q)
  define_maze_global()
  global X Y
  actions_map = zeros(X, Y);
  %initial state in the left bottom corner with
  %pasenger and must go to G
  s = create_rnd_state();
  s.pp = 5;
  s.dp = 3;

  for x=1:X
    for y=1:Y
      %perform action
       s.x = x;
       s.y = y;
      [v, a] = greedy(s, Q);
      actions_map(s.x, s.y) = a;
    end
  end

  lbls_actions = grid_to_lbls(actions_map);
  lbls_actions = actions2str(lbls_actions);
  figure(4);
  draw_maze(lbls_actions, 'actions map for taxi');
end
```

Choosing action:

```
function [v, a] = greedy(s, Q)
    %s - state, v - value of state s, a - next action
    [v, a] = max(Q(s.x, s.y, s.pp, s.dp, :));
end
```

```
function [v, a] = eps_greedy(s, Q, eps)
    %s - state, v - value of state s, a - next action, eps - epsilon
    global A
    [v, a] = greedy(s, Q);
    if (rand(1)<eps)
        a = randi(A);
    end;
end
```

Transfer-reward function:

```
function [ns, r] = trans_fun(s, a, drive_time)
    %s - state, a - action, ns - next state, r - reward
    %drive_time - time since passenger pick up
    global DROP_LOCS WALLS X Y

    % number of actions

    ns = copy_state(s);
    %default - no reward
    r = 0;
    % a == 1, go north, up
    if a == 1
        ns.y = ns.y + 1;
    end;
    % a == 2, go south, down
    if a == 2
        ns.y = ns.y - 1;
    end;
    % a == 3, go west, left
    if a == 3
        ns.x = ns.x - 1;
    end;
    % a == 4, go east, right
    if a == 4
        ns.x = ns.x + 1;
    end;
    %check for collisions
    hit_walls = any(ismember(WALLS,[s.x, s.y, ns.x, ns.y],'rows'));
    within_maze = 1 <= ns.x && ns.x <= X && 1 <= ns.y && ns.y <= Y;
    out_of_maze = ~within_maze;
    if hit_walls || out_of_maze
        %restore previous position
        ns.x = s.x;
        ns.y = s.y;
        r = -1;
    end;

    % a == 5, pick up passenger
    if a == 5
```

```
        %default failed
        r = -1;
        %check for already pick up
        if s.pp == 5
            return;
        end;
        %check for the same location of taxi and passenger
        psng_loc = DROP_LOCS(s.pp, :);
        if all([s.x, s.y] == psng_loc)
            ns.pp = 5;
            r = 1;
        end;
    end;
    % a == 6, drop off passenger
    if a == 6
        %default failed
        r = -1;
        %check for passenger presence
        if s.pp ~= 5
            return;
        end;
        %check for the same location of taxi and passenger
        drop_loc = DROP_LOCS(s.dp, :);
        if all([s.x, s.y] == drop_loc)
            ns.pp = s.dp;
            r = 10/drive_time;
        end;
    end;
end
```

Transfer-reward function from paper [Dut+05]

```
function [ns, r] = trans_fun_paper(s, a, drive_time)
    %s - state, a - action, ns - next state, r - reward
    %drive_time - time since passenger pick up
    global DROP_LOCS WALLS X Y

    % number of actions

    ns = copy_state(s);
    %default - no reward
    r = -1;
    % a == 1, go north, up
    if a == 1
        ns.y = ns.y + 1;
    end;
```

```matlab
    % a == 2, go south, down
    if a == 2
        ns.y = ns.y - 1;
    end;
    % a == 3, go west, left
    if a == 3
        ns.x = ns.x - 1;
    end;
    % a == 4, go east, right
    if a == 4
        ns.x = ns.x + 1;
    end;
    %check for collisions
    hit_walls = any(ismember(WALLS,[s.x, s.y, ns.x, ns.y],'rows'));
    within_maze = 1 <= ns.x && ns.x <= X && 1 <= ns.y && ns.y <= Y;
    out_of_maze = ~within_maze;
    if hit_walls || out_of_maze
        %restore previous position
        ns.x = s.x;
        ns.y = s.y;
        r = -1;
    end;

    % a == 5, pick up passenger
    if a == 5
        %default failed
        r = -10;
        %check for already pick up
        if s.pp == 5
            return;
        end;
        %check for the same location of taxi and passenger
        psng_loc = DROP_LOCS(s.pp, :);
        if all([s.x, s.y] == psng_loc)
            ns.pp = 5;
            r = -1;
        end;
    end;
    % a == 6, drop off passenger
    if a == 6
        %default failed
        r = -10;
        %check for passenger presence
        if s.pp ~= 5
            return;
        end;
        %check for the same location of taxi and passenger
        drop_loc = DROP_LOCS(s.dp, :);
        if all([s.x, s.y] == drop_loc)
            ns.pp = s.dp;
            r = 0;
        end;
    end;
end
```

Q-learning code:

```
function [Q_final, total_episode_reward] = Q_learning_episode(Q, eta, ↩
    gamma, eps, trans_f)
   %v - value, a - action, s - state, r - reward, q - Q-value
   %n in front of them means next
   global DROP_LOCS MAX_EPISODE_TIME
   s = create_rnd_state();
   %trial
   drive_time = 0;
   total_episode_reward = 0;

   for step=1:MAX_EPISODE_TIME
       %passenger is within taxi increase time
       if s.pp == 5
           drive_time = drive_time + 1;
       end

       %perform action
       [v, a] = eps_greedy(s, Q, eps);
       [ns, r] = trans_f(s, a, drive_time);
       total_episode_reward = total_episode_reward + r;

       %learning
       nv = max(Q(ns.x, ns.y, ns.pp, ns.dp, :));
       q = Q(s.x, s.y, s.pp, s.dp, a);
       Q(s.x, s.y, s.pp, s.dp, a) = (1-eta)*q + eta*(r+gamma*nv);

       %check for terminal conditions
       drop_loc = DROP_LOCS(s.dp, :);
       %when we drop off passanger in the drop of location,
       %previously we had passanger within taxi
       if s.pp == 5 && a == 6 && all([ns.x, ns.y] == drop_loc)
           break;
       end
       %update old state
       s = ns;
   end
```

```
    Q_final = Q;
end
```

SARSA code:

```
function [Q_final, total_episode_reward] = SARSA_episode(Q, eta, gamma, ↩
    eps, trans_f)
    %v - value, a - action, s - state, r - reward, q - Q-value
    %n in front of them means next
    global DROP_LOCS MAX_EPISODE_TIME
    s = create_rnd_state();
    %trial
    drive_time = 0;
    total_episode_reward = 0;
    %initial action
    [v, a] = eps_greedy(s, Q, eps);
    for step=1:MAX_EPISODE_TIME
        %passenger is within taxi increase time
        if s.pp == 5
            drive_time = drive_time + 1;
        end

        %perform action
        [ns, r] = trans_f(s, a, drive_time);
        total_episode_reward = total_episode_reward + r;

        %estimate next state
        [nv, na] = eps_greedy(ns, Q, eps);
        %learning
        q = Q(s.x, s.y, s.pp, s.dp, a);
        Q(s.x, s.y, s.pp, s.dp, a) = (1-eta)*q + eta*(r+gamma*nv);

        %check for terminal conditions
        drop_loc = DROP_LOCS(s.dp, :);
        %when we drop off passanger in the drop of location,
        %previously we had passanger within taxi
        if s.pp == 5 && a == 6 && all([ns.x, ns.y] == drop_loc)
            break;
        end
        %update old state
        s = ns;
        %new action
        a = na;
    end
    Q_final = Q;
end
```

References

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[Dut+05] Alain Dutech et al. “Reinforcement learning benchmarks and bake-offs II”. In: *Advances in Neural Information Processing Systems (NIPS)* 17 (2005).

[Kle12] Dan Klein. *UC Berkeley CS188 Intro to AI*. `http://ai.berkeley.edu/lecture_videos.html`. 2012.