CO424/BBB
Reinforcement Learning/Bits, Brain & Behaviour
Coursework 1

Charalambos Hadjipanayi - 01077219

Question 1: Understanding of MDPs

1.1

My personalised trace of states and rewards, generated using my CID (01077219), is

$$\tau = \ s_0 \ 1 \ s_2 \ 0 \ s_0 \ 1 \ s_0 \ 1 \ s_1 \ 0 \ s_0 \ 1 \ s_2 \ 1$$

1.2

Part a)

Given the trace we just observed, we can infer that the structure of the transition matrix ($\boldsymbol{P}$) and reward function ($\boldsymbol{R}$) of this process are

$$\boldsymbol{P} = \begin{bmatrix} 0.25 & 0.25 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{R} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where P_{ij} is the transition probability of going from state i to state j, and R_{ij} is the reward collected when going from state i to state j. Note that non-encountered rewards were given a value of 0, which does not affect results since their corresponding $P_{ij} = 0$. Matrix $\boldsymbol{R}$ can be reduced to $\boldsymbol{R} = [1, 0, 0]^T$, where each element is the reward collected upon departing from each state. Note that $S = [s_0, s_1, s_2]^T$. The minimal MDP graph that is consistent with these data is shown in Figure 1. This process is essentially a Markov Reward Process we have a fixed policy, i.e. fixed action for each state, and thus involves no decision making.

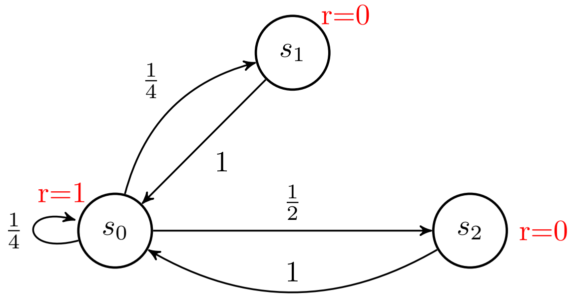


Figure 1: Minimal MDP graph consistent to generated trace.

The transitional probabilities of the process were estimated using sample proportions (or relative frequency of events) of transitions between the states from the trace observed. In other words, the probability P_{ij} is estimated by $\hat{P}_{ij}$:

$$\hat{P}_{ij} = \frac{n_{ij}}{\sum_{k=1}^{N} n_{ij}} \ \textit{where } n_{ik} = \textit{Number of transitions from i to k, and } N = \textit{number of states.}$$

The estimator $\hat{P}_{ij}$ is the Maximum Likelihood Estimator (MLE) of the transition probability P_{ij} (Carnegie Mellon University, Statistics Department, (2009) *Maximum Likelihood Estimation for Markov Chains Derivation of the MLE for Markov chains*). Given the trace, for state s_1 and s_2, there is only one transition out from these states, and this transition results in state s_0. This results in $P_{s_1,s_0} = P_{s_2,s_0} = 1$. Furthermore, there are 4 possible transitions from state s_0, two of which going to s_2, one back to itself and one to s_1. Thus, $P_{s_0,s_0} = P_{s_0,s_1} = 0.25$ and $P_{s_0,s_2} = 0.5$.

The use of MLE of transitional probability for the construction (estimation) of the Transitional probability matrix is valid under certain assumptions. Firstly, process is assumed to obey the Markov property. Additionally, we assume that we have discrete-time transitions, or trace is observed at evenly space time intervals. Finally, process is assumed to be time-homogeneous with $\boldsymbol{P}$ being the stationary distribution, so that $\boldsymbol{P}$ is the same after each step.
The reward function reduces to $\boldsymbol{R} = [1, 0, 0]^T$ because by following the trace, any transition from s_0 to any other state has a reward of 1, and also any transitions from state s_1 and s_2 result in reward of 0. This suggests that the rewards of the process (shown in the graph), are deterministic. It should be noted that the last reward from the trace after leaving state s_2 is 1, instead of the value of 0 we had before. This could be possible if either s_2 at this this point transitions to s_1 or back to itself, or if it transitions back to s_0, which in that case means that the reward function is stochastic. A final note for this latter case, is that if this case was true, transition from s_2 to s_0 would have 0.5 probability to get a reward or not.

Part b)

There are two approaches to compute the value of state s_0:
1) If we model the unknown process by an MRP from the generated trace, with the transition probability matrix and reward function specified in 1.2 part a), we can theoretically use the Bellman equation to find the state values:

$$\boldsymbol{v} = \boldsymbol{R} + \gamma \boldsymbol{P}\boldsymbol{v} \quad \text{with analytical solution} \quad \boldsymbol{v} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{R} \tag{1}$$

where $\boldsymbol{v}$ is the state value vector which is 3-dimensional.
However, the matrix $\boldsymbol{I} - \gamma \boldsymbol{P}$ is non-invertible when the discount factor $\gamma = 1$, which means that the expected returns for each state are infinite. This can be explained by the fact that when $\gamma = 1$ the process is "far-sighted" and immediate rewards have the same weighting as all future rewards in the calculation of the total return (R_t). For an MRP like this one we have which has no terminal states and goes on continually without limit, the return equation becomes problematic producing infinite returns. State values in this case can only be bounded if $\gamma < 1$.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \xrightarrow{\gamma=1} \sum_{k=0}^{\infty} r_{t+1+k} = \infty \tag{2}$$

2) If we consider the trace obtained as a sample trace from the unknown MDP, and we have no other prior knowledge of the process, we cannot identify if the trace given is a complete episode or a sample of a continuing task. If we assume that this is a complete episode, then we can apply Every-visit Monte Carlo on the trace to compute the value of s_0. Using MC prediction, we can approximate the state-value by the empirical mean return. In that case, MC prediction, with $\alpha = 1/(N(s_0))$ (where $N(s_0)$ is the number of visits to state s_0), gives us $\hat{V}(s_0) = 2.5$. The result from this is not reliable since s_2 is also visited twice in the trace and cannot be an absorbing state in order for the trace to terminate. However, if we do not assume the trace to be a complete episode, then MC prediction cannot be applied since it can only be applied to episodes that terminate. In that case, if we apply instead the TD(0) method and initialise the estimates of all state-values to 0, then using the trace we have we can update the values in one iteration only. By setting learning rate α to 1, and using update equation (3), we can obtain that $\hat{V}(s_0) = 2$.

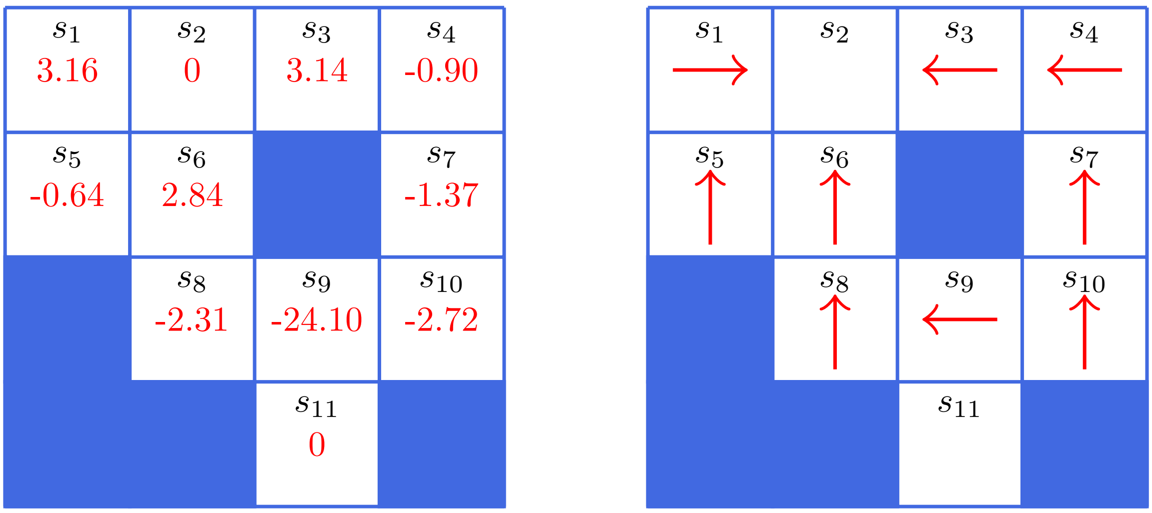
$$V(S_t) \leftarrow V(S_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(S_t)] \tag{3}$$

MC prediction estimate minimizes the mean-square-error while TD(0) gives the maximum likelihood estimate of the Markov model generating the data. However, MC prediction estimate will not be valid if trace is not a full episode. Note that $a = 1$ was chosen to maximise learning rate since we have only one trace.

Question 2: Understanding of Grid Worlds

2.1
My personalised reward state is $\mathbf{s_2}$, $\mathbf{p = 0.35}$, $\boldsymbol{\gamma} = \mathbf{0.25}$ and $\mathbf{q} = \frac{\mathbf{1-p}}{\mathbf{3}}$. (From [x,y,z]=[2,1,9])

2.2
The optimal policy and optimal state-values were obtained using the policy iteration algorithm implemented in Matlab (Appendix A). This algorithm consists of policy evaluation and policy improvement steps. The threshold θ used in policy evaluation for convergence of state-values was set to 0.001. Also, due to the 4 possible directions of motion, 4 transition probability matrices were used, one for each action, while using only one reward matrix. In policy evaluation, values were initialized to 0 and any updates to state-values were made from previous-iteration estimates.The policy iteration algorithm was run until there was no further change in value functions and policy, i.e. when convergence occurs. Figure 2 shows the optimal values and optimal policy for this Grid World example.



(a) Optimal value functions

(b) Optimal policy

Figure 2: (a) Optimal state-value function, rounded to 2 decimal places and (b) Optimal policy, where arrows indicate optimal action direction for each state. (stochastic policy).

2.3
The term $\pi(a|s_9)$, denotes the policy probability distribution for all actions in state 9, while $p(a, s_9)$ denotes the probability distribution of starting in s_9, taking action a and ending in any successor states s' of s_9. However, in the optimal policy obtained, the action (a) executed is **West**, which is therefore the optimal action. This is summarised below:

$$\pi(a|s_9) = \begin{cases} 1 & \text{,for } a = \text{West} \\ 0 & \text{,for } a = \text{North,East,South} \end{cases} \quad \text{and} \quad p(West, s_9) = \begin{cases} p & \text{,for } s' = s_8 \\ q & \text{,for } s' = s_9, s_{10}, s_{11} \end{cases}$$

We know that the agent in the optimal policy follows the path towards the rewards state that will maximize its expected return. In state s_9 the agent has 4 possible actions. If the agent moves south, it will reach the penalty state s_{11} and if agent moves north, it will stay in the same position as before with an extra cost of -1. Thus, the two paths that are possible to reach state s_2 (the reward state) is to either execute West or East. Since $V(s_8) > V(s_{10})$, this means that moving towards state 8 will result in higher expected return, thus the agent will execute West.
The probability p determines how successful an action will be in moving in the desired direction, while q is the probability in moving in the remaining three directions. A value of $p = 0$ effectively means that the action is completely unsuccessful while a value of $p = 1$ means that the action

is completely successful in moving in the optimal direction. My personalised $p = 0.35$, while $q \approx 0.2167$. This means that there is a higher probability of succeeding in moving towards desired direction following a particular action, comparing to the probability of ending in an undesired state. The fact that s_8 has the maximum value from all adjacent states to s_9 and succeeding in our intended move is more likely than failing means effectively that West is the optimal action from s_9. It should be noted that if $p < q$, then moving West would be the worst action. Also, if $p = q = 0.25$, then the probability of going to desired direction or not, is the same thus at the optimal policy all actions will be equiprobable.
A value of $\gamma = 0$, means that actions are only influenced by the immediate rewards, and state-values are equal to the sum of immediate rewards weighted by their transition probabilities. In that case, the agent's optimal actions at s_9 would be to go either West, East and North as these actions result in immediate reward of -1. On the contrary, if $\gamma = 1$, future rewards have the same weighting as immediate rewards, therefore agent's optimal action would be to go West in shortest path towards the reward state to obtain the maximum return. It is worth noting that as the value of γ increases, the value of s_9 decreases. This can be seen from the Bellman Optimality equation where $V^*(s) = \sum_{s'} P_{ss'}(R_{ss'} + \gamma V^*(s'))\ ,\ for\ \gamma > 0$ and $^*(s) = \sum_{s'} P_{ss'}R_{ss'}\ for\ \gamma = 0$. As γ increases, values of successor states of s_9 are affected more by the penalty state, which makes $V^*(s')$ more negative. In my case, $\gamma = 0.25$, and thus by following the optimal path, the present value of reward from the reward state is $(1/4)^2 * 10 = 0.625$. This is a non-zero reward that increases the total return of s_9. Therefore, the ideal action from s_9 will be West, so that it reaches the reward state with minimum steps.

2.4

The optimal policy obtained with my personalised γ and p is a deterministic policy. Actions from each state, indicated as arrows, in the optimal policy, point towards the reward state s_2 and away from penalty state s_{11}, while avoiding directions towards walls or boundaries of grid. The direction of these actions is towards the shortest path to s_2. The exceptions are s_5 and s_{10}, which have two possible shortest paths to reach s_2, but move preferentially towards states with higher state values. Effects of γ and p were studied using Matlab (Appendix B).
Using $\gamma = 0.25$ and varying p from 0.25 to 1, the optimal policy is the same. However, when the value of $p = 1$, the policy for s_5 and s_{11} becomes stochastic. At $p = 0.25$, all actions for all states are stochastic with probability of selection being 0.25 each. For $p < 0.25$ all policies will be stochastic and almost be the same. In the case of $p = 0$, the policy will be the exact inverse of policy at $p = 1$, i.e. the actions are chosen such that the agent does not move towards the reward state. Using $p = 0.35$ and varying γ from 0 to 1, the optimal policy would change only at $\gamma = 0$, where in that case, as explained in 2.2, the actions are based only on immediate rewards of states, which will make the policy stochastic. In fact, for any value of $\gamma \neq 0$, the policy will be the same for all p values from 0.25 to 1 (apart from 0.25 and 1 exactly). Finally for the special case of $\gamma = 0$ and $p = 1$, the only non-stochastic policies will be for s_1, s_3 and s_6, which are the closest to the reward state.
The optimal values obtained with my personalised γ and p are shown to be positive only for states next to reward state. The state-values increase from the penalty state to the reward state, with maximum increase in the direction of shortest path to s_2. Another feature noticed, is that although s_1, s_3 and s_6 have the same distance to s_2, value of s_6 is lower than the other two, since it is closer to s_{11}. It is worth noting that $V(s_1) \approx V(s3)$, since they are equidistant to s_2 and s_{11}. Using Matlab simulations, the lowest value of all states occurs at $p = 0.25$. As p increases, the values increase and reach a maximum at $p = 1$. It should be noted that for any state, the maximum possible value would be 10, and this will be the value of states s_1, s_3 and s_6 in the case that both γ and p are equal to 1. Also, for $p = 0.35$, as γ increases from 0 to 1, the values of states being closer to the penalty state, or those equally close to penalty and reward state, will decrease. However, the values of other states will also be affected.

Appendix A - Matlab code used in Question 2 Part 2

1. Defining the MDP process as an object

```
%Using an object that encapsulates data and
%the operations performed on that data.

classdef GridWorld
    properties
           % Specifying the states
           States_names = ["s1", "s2", "s3", "s4", "s5", "s6",...
                          "s7","s8","s9","s10","s11"];
           S = 11;

           %Specifying Probabilities
           %Probability of succeding in moving in the desired direction
           p_gw=7/20;
           %Probability of moving to any of the other cardinal directions.
           q_gw=13/60;

           % Specifying actions
           % Actions are: {"N","E","S","W"} --> {0,1,2,3}
           Action_names= ["N","E","S","W"];
           A = 4;

           %Matrix indicating absorbing states
           %  STATES -->       1  R  3  4  5  6  7  8  9  10   C
           Absorbing_states = [0, 1, 0, 0, 0, 0, 0, 0, 0,  0,  1];
    end

    methods
         %Function below defines the four transition probability matrices
         %One matrix for each action.
         function T_north = TN(obj)
             p = obj.p_gw;
             q = obj.q_gw;

           %For action of going North
           T_north = [p+q  q    0    0    q    0    0    0    0    0    0;
                       0    1    0    0    0    0    0    0    0    0    0;
                       0    q    p+q  q    0    0    0    0    0    0    0;
                       0    0    q    p+q  0    0    q    0    0    0    0;
                       p    0    0    0    2*q  q    0    0    0    0    0;
                       0    p    0    0    q    q    0    q    0    0    0;
                       0    0    0    p    0    0    2*q  0    0    q    0;
                       0    0    0    0    0    p    0    2*q  q    0    0;
                       0    0    0    0    0    0    0    q    p    q    q;
                       0    0    0    0    0    0    p    0    q    2*q  0;
                       0    0    0    0    0    0    0    0    0    0    1;];
         end

         function T_east = TE(obj)
             p = obj.p_gw;
             q = obj.q_gw;

           %For action of going East
           T_east = [ 2*q  p    0    0    q    0    0    0    0    0    0;
                       0    1    0    0    0    0    0    0    0    0    0;
                       0    q    2*q  p    0    0    0    0    0    0    0;
```

```
                         0    0    q    p+q  0    0    q    0    0    0    0;
                         q    0    0    0    2*q  p    0    0    0    0    0;
                         0    q    0    0    q    p    0    q    0    0    0;
                         0    0    0    q    0    0    p+q  0    0    q    0;
                         0    0    0    0    0    q    0    2*q  p    0    0;
                         0    0    0    0    0    0    0    q    q    p    q;
                         0    0    0    0    0    0    q    0    q    p+q  0;
                         0    0    0    0    0    0    0    0    0    0    1;];
        end

        function T_south = TS(obj)
            p = obj.p_gw;
            q = obj.q_gw;

         %For action of going South
         T_south = [2*q  q    0    0    p    0    0    0    0    0    0;
                         0    1    0    0    0    0    0    0    0    0    0;
                         0    q    p+q  q    0    0    0    0    0    0    0;
                         0    0    q    2*q  0    0    p    0    0    0    0;
                         q    0    0    0    p+q  q    0    0    0    0    0;
                         0    q    0    0    q    q    0    p    0    0    0;
                         0    0    0    q    0    0    2*q  0    0    p    0;
                         0    0    0    0    0    q    0    p+q  q    0    0;
                         0    0    0    0    0    0    0    q    q    q    p;
                         0    0    0    0    0    0    q    0    q    p+q  0;
                         0    0    0    0    0    0    0    0    0    0    1;];
        end

        function T_west = TW(obj)
            p = obj.p_gw;
            q = obj.q_gw;

         %For action of going West
         T_west = [ p+q  q    0    0    q    0    0    0    0    0    0;
                         0    1    0    0    0    0    0    0    0    0    0;
                         0    p    2*q  q    0    0    0    0    0    0    0;
                         0    0    p    2*q  0    0    q    0    0    0    0;
                         q    0    0    0    p+q  q    0    0    0    0    0;
                         0    q    0    0    p    q    0    q    0    0    0;
                         0    0    0    q    0    0    p+q  0    0    q    0;
                         0    0    0    0    0    q    0    p+q  q    0    0;
                         0    0    0    0    0    0    0    p    q    q    q;
                         0    0    0    0    0    0    q    0    p    2*q  0;
                         0    0    0    0    0    0    0    0    0    0    1;];
        end

      %This merges all transition matrices into a 3D matrix
      function TM = transition_matrix(obj)
          TM =zeros(obj.S,obj.S,obj.A);
          TM(:,:,1) = TN(obj);
          TM(:,:,2) = TE(obj);
          TM(:,:,3) = TS(obj);
          TM(:,:,4) = TW(obj);
      end

      %Obtains a specific probability from matrix TM
      function prob = transition_function(obj,prior_state,action,post_state)
          TM = transition_matrix(obj);
          prob = TM(prior_state,post_state,action+1);
      end
```

```
        %Defining the Reward matrix
        function R_matrix = reward_matrix(obj)
            %Since given any action, all possible directions are available,
            %then we will only need a single reward matrix of the process.

           R_matrix = [-1   10   0   0  -1   0   0   0   0   0    0;
                        0    0   0   0   0   0   0   0   0   0    0;
                        0   10  -1  -1   0   0   0   0   0   0    0;
                        0    0  -1  -1   0   0  -1   0   0   0    0;
                       -1    0   0   0  -1  -1   0   0   0   0    0;
                        0   10   0   0  -1  -1   0  -1   0   0    0;
                        0    0   0  -1   0   0  -1   0   0  -1    0;
                        0    0   0   0   0  -1   0  -1  -1   0    0;
                        0    0   0   0   0   0   0  -1  -1  -1 -100;
                        0    0   0   0   0   0  -1   0  -1  -1    0;
                        0    0   0   0   0   0   0   0   0   0    0;];
        end

        %Obtaining the specific reward
        function reward = reward_function(obj,prior_state,post_state)
            R_matrix = reward_matrix(obj);
            reward = R_matrix(prior_state,post_state);
        end
    end
end
```

2. Definition of Policy Evaluation algorithm

```
function value_estimates = policy_eval(obj,policy,gamma)
    %Initialisation
    num_states = obj.S;
    num_actions = obj.A;
    value_estimates = zeros(num_states,1);
    theta = 0.001; ∆ = 2*theta;

     while(∆>theta)
            values_old = value_estimates;
            for i=1:num_states
                value_estimates(i)=0;
                for a=1:num_actions
                    value_cum=0;
                    for s=1:num_states

                        value = ...
                        policy(i,a)*(transition_function(obj,i,(a-1),s)*...
                        (reward_function(obj,i,s) + gamma*values_old(s)));
                        value_cum = value_cum + value;
                    end
                    value_estimates(i)=value_estimates(i)+value_cum;
                end
            end

        for i=1:length(value_estimates)
            k(i) = abs(values_old(i) - value_estimates(i));
        end
        ∆ = max(k);
    end
end
```

3. Definition of Policy Improvement algorithm

```
function [policy_new,policy_stable] = ...
          policy_improv_v2(obj,value_estimates,policy_old,gamma)

    policy_stable = true;
    num_states = obj.S;
    num_actions = obj.A;
    new_values=zeros(num_states,num_actions);
    new_value_estimates= zeros(num_states,1);

    for a=1:num_actions
        for i=1:num_states
               new_value_estimates(i)=0;
               for s=1:num_states
                    value =...
                    transition_function(obj,i,a-1,s)*...
                    (reward_function(obj,i,s) + gamma*value_estimates(s));
                    new_value_estimates(i)=new_value_estimates(i)+value;
               end
               new_values(i,a)=new_value_estimates(i);
        end
    end

    %We want to store all actions that produce a maximum state-value.
    [maximum,¬] = max(new_values');

    policy_new = zeros(num_states,num_actions);

    for i=1:num_states
        if (i≠2)&&(i≠11)
            [¬,best_actions]=find(new_values(i,:)==maximum(i));
            num_possible_actions = length(best_actions);
            for j=1:length(best_actions)
                policy_new(i,best_actions(j)) = 1/(num_possible_actions);
            end
        end
    end

    counter=0;
    for k=1:length(policy_old)
       if policy_old(k)==policy_new(k)
          counter=counter+1;
       end
    end

    if counter==num_states
        policy_stable = false;
    end
end
```

4. Implementation of Policy Iteration algorithm

```
close all;clear all; clc;

%% Defining the MDP

%Calling the object into this file
import GridWorld
MDP = GridWorld;

gamma = 0.25;

%% Initial policy definition
initial_policy = 0.25*ones(MDP.S,MDP.A);
initial_policy(2,:)=0; initial_policy(11,:)=0;

%% Running the policy evaluation algorithm - We use in-place updates of ...
    value function
estimated_values = policy_eval(MDP,initial_policy,gamma);

%% Implementing policy improvement
[policy_new,policy_stable] = ...
    policy_improv_v2(MDP,estimated_values,initial_policy,gamma);

%% Implementing Policy Iteration algorithm to obtain the optimal policy ...
    and optimal values

policy = initial_policy;
num_iter=0; %To check how many times it is run

while policy_stable == true
    num_iter=num_iter+1;
    new_policy = policy;
    values = policy_eval(MDP,new_policy,gamma);
    [policy,policy_stable] = policy_improv_v2(MDP,values,new_policy,gamma);
    policy_stable;
end
```

Appendix B - Matlab code used in Question 2 Part 4

1. Studying the effects of p on optimal values and optimal policy

```
close all;clear all; clc;

%% Defining the MDP

%Calling the object into this file
import GridWorld
MDP = GridWorld;
gamma = 0.25;
p=[0:0.05:1];
q=(1-p)./3;

for index=1:length(p)
    MDP.p_gw=p(index);
    MDP.q_gw=q(index);
    initial_policy = 0.25*ones(MDP.S,MDP.A);
    initial_policy(2,:)=0; initial_policy(11,:)=0'
    policy = initial_policy;
    num_iter=0; %To check how many times it is run
    policy_stable = true;
    while policy_stable == true
        num_iter=num_iter+1;
        new_policy = policy;
        values = policy_eval(MDP,new_policy,gamma);
        [policy,policy_stable] = ...
            policy_improv_v2(MDP,values,new_policy,gamma);
        policy_stable;
    end
    optimal_policy(:,:,index) = policy;
    optimal_values (:,:,index) = values;

       if (p(index)==1)||(p(index)==0)
        p(index)
        optimal_policy(:,:,index)
       end
end

figure;
for j=1:MDP.S
        opt_val_vec = optimal_values(j,:,:);
        for i=1:length(p)
            opt_val(i) = opt_val_vec(i);
        end

        if (j≠2)&&(j≠11)
            plot(p,opt_val,'LineWidth',1)
            hold on
        end
end
xlabel('p value')
ylabel('Value of state')
ylim([-35 10])
title('Effect of p on state-values, while gamma=0.25')
legend('s1','s3','s4','s5','s6','s7','s8','s9','s10', 'Orientation',...
       'horizontal','Location','best')
```

2. Studying the effects of γ on optimal values and optimal policy

```
close all
clear all
clc

%% Defining the MDP

%Calling the object into this file
import GridWorld
MDP = GridWorld;

p=0.35;
q=(1-p)/3;
MDP.p_gw = p;
MDP.q_gw = q;

%% Initial policy definition
initial_policy = 0.25*ones(MDP.S,MDP.A);
initial_policy(2,:)=0; initial_policy(11,:)=0;

%% Implementing Policy Iteration algorithm to obtain the optimal policy ...
    and optimal values

gamma = [0:0.1:1];
for index=1:length(gamma)
    policy_stable=true;
    policy = initial_policy;
    num_iter=0; %To check how many times it is run
    while policy_stable == true
        num_iter=num_iter+1;
        new_policy = policy;
        values = policy_eval(MDP,new_policy,gamma(index));
        [policy,policy_stable] = ...
            policy_improv_v2(MDP,values,new_policy,gamma(index));
    end
    optimal_policy(:,:,index) = policy;
    optimal_values (:,:,index) = values;
end

figure;
for j=1:MDP.S
        opt_val_vec = optimal_values(j,:,:);
        for i=1:length(gamma)
            opt_val(i) = opt_val_vec(i);
        end

        if (j≠2)&&(j≠11)
            plot(gamma,opt_val,'LineWidth',1)
            hold on
        end
end

xlabel('gamma value')
ylabel('Value of state')
title('Effect of gamma on state-values, while p=0.35')
legend('s1','s3','s4','s5','s6','s7','s8','s9','s10', 'Orientation',...
        'horizontal','Location','best')
```

3. Graphs obtained from these simulations, showing effects of γ and effects of p on optimal solutions

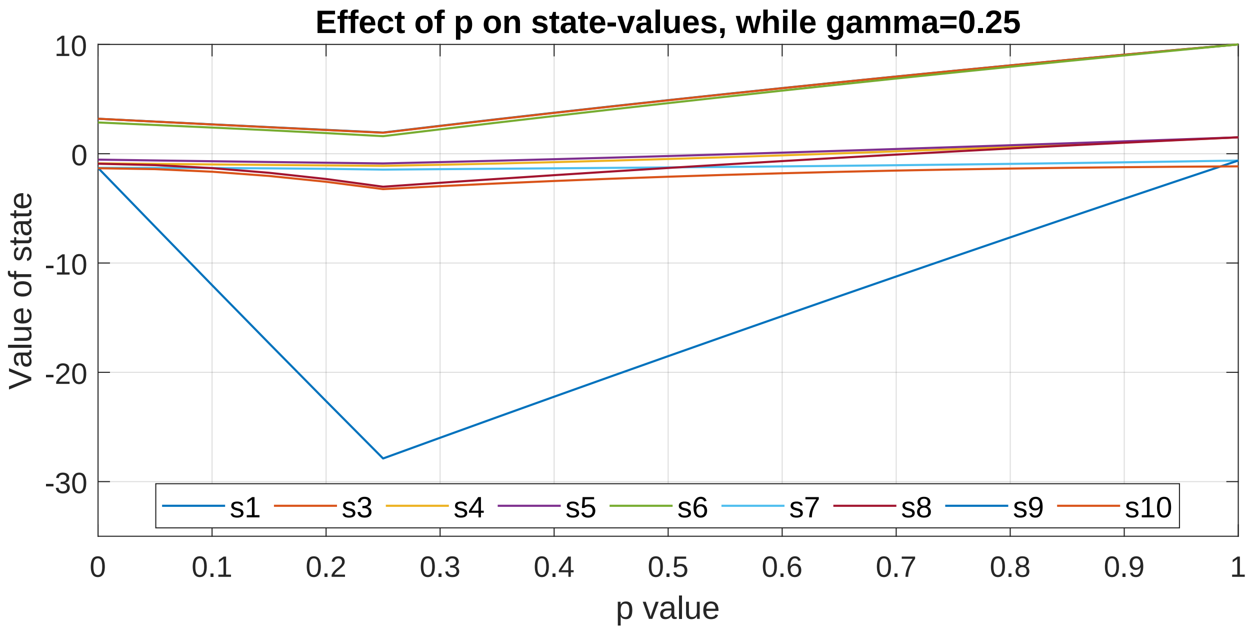


Figure 3: Effect of p on the state values of all states, while $\gamma = 0.25$. $V(s_2)$ and $V(s_{11})$ are not shown since they are always 0.

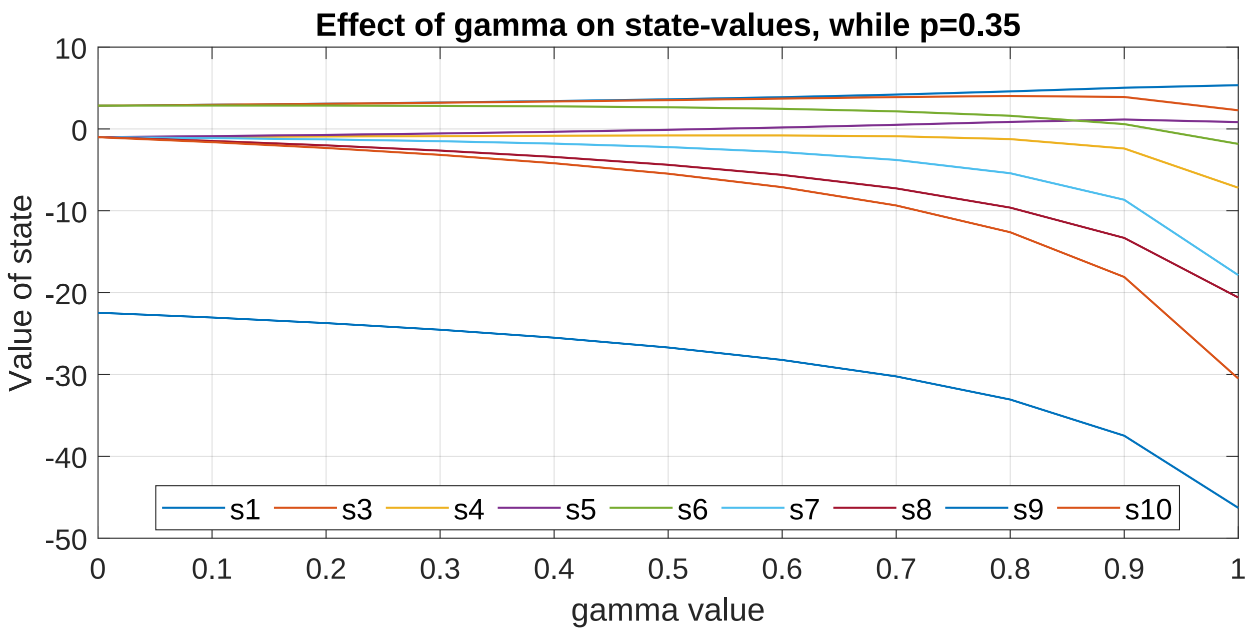


Figure 4: Effect of γ on the state values of all states, while $p = 0.35$. $V(s_2)$ and $V(s_{11})$ are not shown since they are always 0.