Ahmet Narman

CID: 01578741

MSc. Human and Biological Robotics

Machine Learning and Neural Computation: Coursework 1

1) CID=01578741, p=0.7, γ=0.25.

2) A function (find_value) was created for calculating the value function for any given policy for the grid world. It takes some required parameters and the policy for which the value function will be calculated. A loop runs that would find the value of every state by doing the weighted sum (weights are policy probabilities) of the immediate rewards and discounted future values which corresponds to this equation:

$$V^{\pi}(s) = \sum_{a \epsilon A} \pi(a,s) \sum_{s' \epsilon S} P^{a}_{ss'}(R^{a}_{ss'} + \gamma V^{\pi}(s'))\ s_1$$

This equation was calculated using element-wise matrix multiplication. We have started with a zero value function and it got updated with each cycle of the loop. With this method, the value function converges after some iterations. The loop was stopped when the change in the value function between two cycles fell below 0.00001 in terms of mean squared distance. The value function for the unbiased policy that was found using this function is given here:

State	s_1	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}
Value	-0.95	-3.83	-1.29	-1.13	-3.65	-1.65	-1.33	-1.36	-1.33	-1.33	-1.33	-1.33

3.a) Knowing the transition matrix and the policy, the given sequence probabilities can be calculated. A function (trace_probability) was created that would calculate the likelihood of a trace happening given a policy. The function calculates the probability of going from one state to the next for all states in the trace and multiplies these to find the overall trace likelihood. The way the probability of going from one state to the next is calculated is by weighted sum (policy probabilities are the weights) of all transition which corresponds to the following equation:

$$P(s,s') = \sum_{a \epsilon A} \pi(a,s)\, T(a,s,s')$$

With this function, the probabilities of given sequences in an unbiased policy was found as:

State Transitions	{s14, s10, s8, s4, s3}	{s11, s9, s5, s6, s6, s2}	{s12, s11, s11, s9, s5, s9, s5, s1, s2}
Likelihood	0.0061	0.00025	0.0000038

3.b) A function (give_bias) was written that would increase the probability of actions in the policy that has the maximum transition probability of going from one state to the next for all states in these sequences. The function would get the transition matrix, a prior policy and a sequence as arguments to return a posterior policy for which the likelihood of the given sequence is higher. For each state in a sequence, the function would find the action that will give the maximum transition probability to the next state, and increase the probability of that action, which makes that transition more probable. An unbiased policy was gone through this function three times for each sequence. The resulting policy gives the following likelihoods:

State Transitions	{s14, s10, s8, s4, s3}	{s11, s9, s5, s6, s6, s2}	{s12, s11, s11, s9, s5, s9, s5, s1, s2}
Likelihood	0.058	0.00044	0.000059

For this policy, the states have following actions that have the maximum probability:

State	s_1	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}
Value	E	W	N	N	-	N	N	N	N	W	-	N

Note that the resulting policy is still probabilistic, not deterministic, and as s_7 and s_{13} was never visited in these sequences, their policies remained unbiased.

4.a) To generate traces, a generic function was written (generate_trace) that would take the policy and other required parameters to produce a trace. This function randomly assigns an initial state (one of $s_{11},\ s_{12}, s_{13}, s_{14}$). Then it starts a loop that will be terminated when an absorbing state was reached. In the loop, it choses an action given the current state and the policy. Depending on the action, it chooses a successor state based on the transition probabilities for that state-action pair. Depending on the state-action-successor state combination, the function finds the corresponding reward from the reward matrix. States, actions and rewards are then added to the trace and the loop starts again until an absorbing state was reached. This function was called 10 times for the unbiased policy and the generated traces are given here (increase the font size to see better):

S14,W,-1,S13,W,-1,S12,S,-1,S11,S,-1,S11,S,-1,S11,E,-1,S12,N,-1,S12,E,-1,S13,E,-1,S14,S,-1,S14,S,-1,S14,W,-1,S13,N,-1,S13,E,-1,S14,E,-1,S14,E,-1,S14,N,-1,S10,S,-1,S14,W,-1,S14,W,-1,S10,W,-1,S10,N,-1,S08,W,-1,S07,E,-1,S08,S,-1,S07,W,-10

S14,S,-1,S14,W,-1,S10,E,-1,S10,W,-1,S10,E,-1,S10,E,-1,S10,N,-1,S08,E,-1,S08,N,-1,S04,N,-1,S04,W,-10

S12,S,-1,S12,S,-1,S12,W,-1,S11,W,-1,S11,S,-1,S12,N,-1,S12,S,-1,S12,S,-1,S12,N,-1,S12,N,-1,S12,E,-1,S12,N,-1,S13,S,-1,S13,W,-1,S12,N,-1,S12,E,-1,S13,N,-1,S13,E,-1,S14,W,-1,S13,W,-1,S12,S,-1,S13,S,-1,S13,S,-1,S13,W,-1,S12,S,-1,S12,S,-1,S12,E,-1,S13,N,-1,S12,W,-1,S12,N,-1,S12,W,-1,S11,N,-1,S09,N,-
1,S05,E,-1,S06,S,-1,S06,W,-1,S05,W,-1,S05,W,-1,S05,N,-1,S01,S,0

S13,S,-1,S13,S,-1,S13,W,-1,S13,S,-1,S14,W,-1,S14,S,-1,S13,S,-1,S14,E,-1,S14,N,-1,S10,S,-1,S14,E,-1,S14,S,-1,S14,N,-1,S10,N,-1,S08,N,-1,S04,W,-10

S13,W,-1,S12,N,-1,S12,N,-1,S11,N,-1,S09,W,-1,S09,E,-1,S09,N,-1,S09,W,-1,S05,W,-1,S05,W,-1,S05,N,-1,S01,S,-1,S05,E,-1,S06,N,0

S14,N,-1,S10,W,-1,S10,N,-1,S10,E,-1,S14,N,-1,S10,E,-1,S08,W,-1,S10,N,-1,S08,W,-1,S04,E,-1,S04,E,-1,S04,W,-1,S04,W,-10

S11,N,-1,S11,N,-1,S11,E,-1,S11,N,-1,S09,W,-1,S09,W,-1,S05,E,-1,S06,W,-1,S05,N,-1,S05,S,-1,S06,S,-1,S06,N,0

S12,N,-1,S11,W,-1,S09,E,-1,S09,W,-1,S09,E,-1,S11,S,-1,S12,N,-1,S12,W,-1,S12,S,-1,S12,S,-1,S12,W,-1,S11,W,-1,S11,E,-1,S11,S,-1,S11,E,-1,S12,N,-1,S12,N,-1,S13,W,-1,S12,S,-1,S12,N,-1,S12,W,-1,S12,S,-1,S13,W,-1,S12,N,-1,S12,E,-1,S13,E,-1,S13,N,-1,S14,E,-1,S14,W,-1,S13,E,-1,S14,E,-1,S10,S,-1,S14,N,-
1,S10,S,-1,S14,W,-1,S13,W,-1,S12,E,-1,S13,S,-1,S13,S,-1,S13,W,-1,S12,N,-1,S12,E,-1,S12,S,-1,S13,E,-1,S13,S,-1,S13,E,-1,S14,E,-1,S14,E,-1,S14,N,-1,S10,W,-1,S08,E,-1,S08,E,-1,S10,S,-1,S14,N,-1,S13,W,-1,S12,E,-1,S13,E,-1,S13,W,-1,S12,S,-1,S12,W,-1,S11,E,-1,S12,N,-1,S12,N,-1,S11,S,-1,S11,N,-1,S09,E,-
1,S09,S,-1,S11,N,-1,S12,N,-1,S12,N,-1,S12,E,-1,S13,N,-1,S13,N,-1,S14,S,-1,S14,E,-1,S10,S,-1,S14,S,-1,S13,S,-1,S13,S,-1,S13,S,-1,S13,W,-1,S12,S,-1,S12,W,-1,S11,E,-1,S12,S,-1,S13,E,-1,S13,W,-1,S12,S,-1,S12,S,-1,S11,E,-1,S12,N,-1,S12,W,-1,S12,E,-1,S13,W,-1,S12,S,-1,S13,E,-1,S14,S,-1,S14,N,-1,S13,N,-
1,S13,S,-1,S13,N,-1,S13,S,-1,S13,S,-1,S12,W,-1,S11,E,-1,S12,E,-1,S13,W,-1,S12,E,-1,S13,W,-1,S13,N,-1,S13,E,-1,S14,E,-1,S14,W,-1,S13,W,-1,S12,E,-1,S13,S,-1,S13,W,-1,S13,S,-1,S14,E,-1,S14,N,-1,S10,E,-1,S14,E,-1,S10,E,-1,S10,S,-1,S10,N,-1,S08,S,-1,S10,S,-1,S14,E,-1,S10,E,-1,S10,S,-1,S10,E,-1,S10,W,-
1,S10,N,-1,S08,W,-1,S07,E,-1,S08,S,-1,S10,E,-1,S08,E,-1,S08,W,-1,S07,E,-1,S07,W,-1,S07,E,-10

S12,N,-1,S13,S,-1,S13,E,-1,S14,W,-1,S13,S,-1,S13,N,-1,S13,S,-1,S13,W,-1,S12,E,-1,S13,N,-1,S13,N,-1,S13,W,-1,S12,W,-1,S11,S,-1,S11,N,-1,S09,S,-1,S11,E,-1,S09,E,-1,S05,W,-1,S05,W,-1,S05,S,-1,S05,E,-1,S01,E,-1,S05,N,-1,S05,S,-1,S09,N,-1,S05,E,-1,S06,W,-1,S06,W,-1,S05,W,-1,S05,N,-1,S01,S,-1,S01,W,-
1,S05,E,-1,S01,E,0

S12,E,-1,S12,N,-1,S12,W,-1,S11,S,-1,S11,W,-1,S11,E,-1,S12,E,-1,S13,S,-1,S13,S,-1,S13,N,-1,S12,W,-1,S11,E,-1,S11,N,-1,S11,N,-1,S09,W,-1,S09,W,-1,S09,S,-1,S09,W,-1,S09,E,-1,S09,W,-1,S09,S,-1,S11,S,-1,S11,N,-1,S09,S,-1,S11,E,-1,S12,N,-1,S12,W,-1,S11,E,-1,S09,E,-1,S09,N,-1,S09,S,-1,S09,S,-1,S09,N,-
1,S09,S,-1,S11,S,-1,S11,S,-1,S11,S,-1,S12,S,-1,S11,W,-1,S11,W,-1,S09,N,-1,S05,S,-1,S05,W,-1,S01,N,-1,S01,E,0

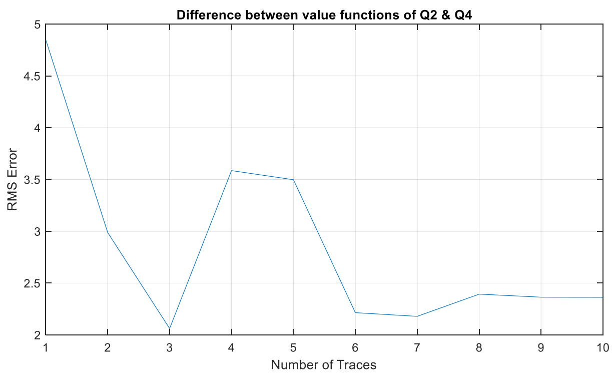
4.b) A function was written to do First-Visit Batch Monte-Carlo Policy Evaluation. The function gets a list of traces and a discount value as parameters and returns a value function (14*1 array, terminal states doesn't matter). For every trace in the list, the function finds the first occurrences of states, calculates the future returns for those states and appends them in the corresponding locations in a cell array of 14*1. A cell array was used to record returns for each trace because this way, we don't have to record how many times a state was visited, each cell can be appended individually which makes taking the average of returns easier. If a state that was visited before is visited again in the loop, the function skips that state. After the loop ends, the value function was assigned by averaging the return cell array (each cell was averaged individually). The resulting estimate of the value function is given below:

State	s_1	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}
Value	-0.89	-4.99	-1.33	-0.99	-1.60	-1.82	-1.33	-1.33	-1.33	-1.33	-1.33	-1.33

4.c) To see how well the value function is estimated, a difference measure of root mean squared error (RMSE) was chosen between the original value function and the value function estimate. The mathematical representation of the difference is given below:

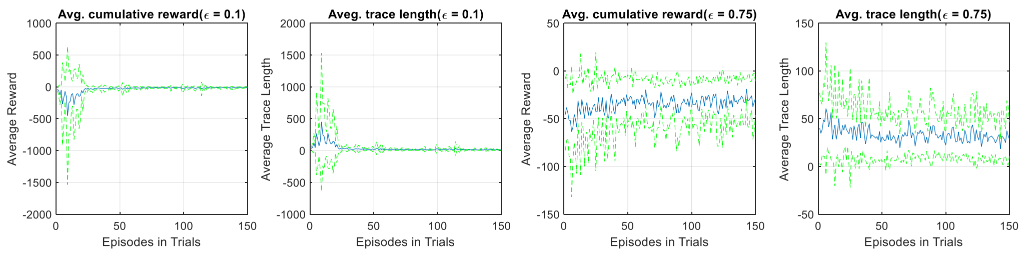
$$RMSE = \sqrt{\sum_i \left(V_i^{\pi^u} - \hat{V}_i^{\pi^u}\right)^2},$$

where 'i' is the state number. This measure was chosen because it shows the error in terms of distance and it converges to zero when the estimate is arbitrarily close to the original value function. This error was calculated for estimated value functions that are calculated by using only 1 generated trace up to using all 10 traces and the resulting error profile is given below:

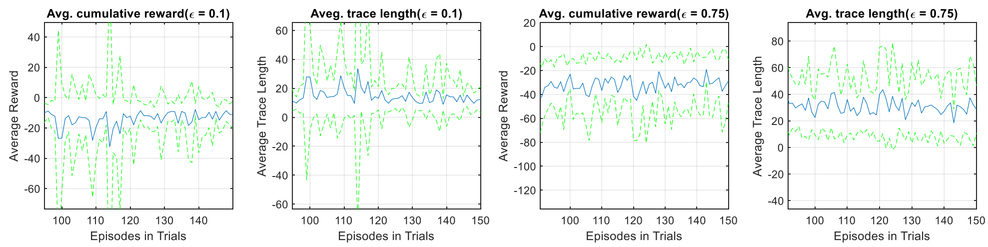


The error does not converge to zero, but it shows a decreasing profile on this plot. To see the error profile plot for 500 traces where the error converges near zero, check Appendix B.

5) A function (learn) was written to do ϵ-greedy first-visit Monte Carlo control. To implement a model free policy improvement, Q(s,a) function was used instead of the V(s) value function. A loop of predetermined length was implemented that would do policy evaluation and policy update each cycle. The policy evaluation idea in Q4 was used but this time, instead of V(s), Q(s,a) function was used (4*14). After each loop cycle, the return cell array was appended for visited states and the Q function was updated by averaging the return cell array. After every policy evaluation, a policy update was done that would find the action that has the maximum Q value for every state and update the policy according to ϵ-greedy method. After every policy update, the loop goes back to the policy evaluation stage with the updated policy and after every policy evaluation, the loop goes to policy update with a new Q function. Using this function, 20 trials of 150 episodes are done for ϵ=0.1 and ϵ=0.75, and the resulting average learning curves (blue) with ± standard deviations (green) were found like this:



We can see that generally the reward increases and trace length decreases for both over episodes. For ϵ=0.75, we can see the reward converges near -30 and trace length converges near 30, which is not a big difference considering they started around -45 and 45. This was the case because ϵ was too big (it wasn't greedy enough). On the other hand, for ϵ=0.1, we see the rewards and trace length explode after initialization. This is because during the exploration stage, suboptimal actions were assigned high probabilities which prevented episodes from terminating quickly. Yet, after the exploration process, the reward and trace length values converged to even better numbers compared to the previous case because of the low ϵ. It is not visible above but zooming in to the end of the graph, we see the reward converges near -15 and trace length converges near 15. Below is given the last 50 episodes of the above plots:



Appendix A: Matlab Code

```matlab
%%% Machine learning and Neural computation
%%% Coursework 1
%%% by Ahmet Narman
%%% CID: 01578741
%%% Instructor: Prof Aldo Faisal
%%% Imperial College London
%%% Nov 2018

clear all;
close all;

RunCoursework();

% The function that includes the coursework
function RunCoursework()
    %% Part 1

    CID = 01578741;
    p = 0.5 + 0.5*(4/10); % Probability to be used in the transition matrix
    gama = 0.2 + 0.5*(1/10); % The discount value

    %% Part 2

    [NumSt, NumAct, TrMat,RewMat, StName, AcName, AbsSt] ...
    = PersonalisedGridWorld(p);
    % For the reward and transition matrices the rows (first indice) are
    % successor states, the columns (second indice) are current states,
    % and the third indice is the taken action.

    % Unbiased policy function is denoted with the below matrix. This
    % matrix has the size of (NumActions, NumStates) because a policy is a
    % function of state,action pairs
    unbias = ones(NumAct, NumSt)*(1/4); % Unbiased Policy

    % Value function for the unbiased policy is found below
    Value = find_value(TrMat, RewMat, AbsSt, gama, unbias)

    %% Part 3

    % Part (a)
    % Sequences that were given to us
    sequence1 = [14 10 8 4 3];
    sequence2 = [11 9 5 6 6 2];
    sequence3 = [12 11 11 9 5 9 5 1 2];

    % The probabilities of these sequences for an unbiased policy
    unbiased_prob_seq1 = trace_probability(TrMat, unbias, sequence1)
    unbiased_prob_seq2 = trace_probability(TrMat, unbias, sequence2)
    unbiased_prob_seq3 = trace_probability(TrMat, unbias, sequence3)

    %Part (b)
    % Our function will modifiy the unbiased policy to improve the
    % probabilities of the three sequences given above. The policy update
    % is done three times, once for every sequence
    biased = give_bias(TrMat, unbias, sequence1);
    biased = give_bias(TrMat, biased, sequence2);
    biased = give_bias(TrMat, biased, sequence3);

    % The probabilities of these sequences for our biased policy
```

```
    biased_prob_seq1 = trace_probability(TrMat, biased, sequence1)
    biased_prob_seq2 = trace_probability(TrMat, biased, sequence2)
    biased_prob_seq3 = trace_probability(TrMat, biased, sequence3)

    %% Part 4

    % Part (a)
    Traces = cell(1,10); % 10 traces will be put in this cell array
    for i = 1:length(Traces)
        % Below function will generate a trace given the parameters
        trace=generate_trace(TrMat, RewMat, unbias, StName, AcName, AbsSt);
        fprintf('%s,',upper(string(trace))); % Printing individual traces
        fprintf('\n');
        Traces(i) = {trace}; % Adding traces to the array
    end

    % Part (b)
    % Below function is the estimated val. function for the unbiased policy
    MCvalue = policy_evaluation(Traces, gama) % Val. function calculated

    % Part (c)
    % The difference between the real value function and the estimated
    % value function was calculated using root mean squared error (RMS)
    diff = zeros(1,length(Traces)); % Difference between value functions
    for i = 1:length(diff)
        newValue = policy_evaluation(Traces(1:i), gama); % MC val. function
        diff(i) = sqrt(sum((Value - newValue).^2)); % RMS Err. calculation
    end

    figure; % Plotting the difference values
    plot(diff);
    title('Difference between value functions of Q2 & Q4');
    xlabel('Number of Traces');
    ylabel('RMS Error');
    grid on;

%% Part 5


    numEp = 150; % Number of episodes in each trial
    numTri = 20; % Number of trials
    % Below matrices hold rewards and trace lengths for every episode in
    % every trial for eps=0.1 and eps=0.75 case.
    trcMat1 = zeros(numTri,numEp);
    rewMat1 = zeros(numTri,numEp);
    trcMat2 = zeros(numTri,numEp);
    rewMat2 = zeros(numTri,numEp);

    % Because the learning process is stochastic, we are going to run the
    % learning process for multiple trials and take the average rewards and
    % trace lengths because of the variability between trials
    for i = 1:numTri
        % Two learning processes for eps=0.1 & eps=0.75
        [OptPol1,trc1,rew1] =...
            learn(numEp,gama,0.1,TrMat,RewMat,unbias,StName,AcName,AbsSt);
        [OptPol2,trc2,rew2] =...
            learn(numEp,gama,0.75,TrMat,RewMat,unbias,StName,AcName,AbsSt);
        trcMat1(i,:) = trc1;
        trcMat2(i,:) = trc2;
        rewMat1(i,:) = rew1;
        rewMat2(i,:) = rew2;
```

```
    end

    % Plotting the average rewards and trace lengths across episodes
    figure;
    subplot(1,4,1);
    plot(mean(rewMat1));
    hold on;
    plot(mean(rewMat1)+std(rewMat1), 'g--');
    plot(mean(rewMat1)-std(rewMat1), 'g--');
    title('Avg. cumulative reward(\epsilon = 0.1)');
    xlabel('Episodes in Trials');
    ylabel('Average Reward');
    grid on;

    subplot(1,4,2);
    plot(mean(trcMat1));
    hold on
    plot(mean(trcMat1)+std(trcMat1), 'g--');
    plot(mean(trcMat1)-std(trcMat1), 'g--');
    title('Aveg. trace length(\epsilon = 0.1)');
    xlabel('Episodes in Trials');
    ylabel('Average Trace Length');
    grid on;

    subplot(1,4,3);
    plot(mean(rewMat2));
    hold on;
    plot(mean(rewMat2)+std(rewMat2), 'g--');
    plot(mean(rewMat2)-std(rewMat2), 'g--');
    title('Avg. cumulative reward(\epsilon = 0.75)');
    xlabel('Episodes in Trials');
    ylabel('Average Reward');
    grid on;

    subplot(1,4,4);
    plot(mean(trcMat2));
    hold on
    plot(mean(trcMat2)+std(trcMat2), 'g--');
    plot(mean(trcMat2)-std(trcMat2), 'g--');
    title('Avg. trace length(\epsilon = 0.75)');
    xlabel('Episodes in Trials');
    ylabel('Average Trace Length');
    grid on;
end

%% Functions

function Value = find_value(T, R, absorbing, disc, policy)
    Value = zeros(14, 1); % Val. func. is immidiately all zero
    prevValue = zeros(14, 1); % To calculate the rate of change each loop
    irew = T.*R; % Immediate reward matrix
    stop = 0; % To stop the loop when the system converges

    while stop == 0

        invAbsorb=-1*(absorbing'-1);% Inverse of the absorbing array

        % Implementing an intermediate variable: Successor Value
        succValue = cat(3, disc*T(:,:,1)*Value, disc*T(:,:,2)*Value,...
            disc*T(:,:,3)*Value, disc*T(:,:,4)*Value);
```

```
        % The value function is updated here for each action
        Value=policy(1,:)'.*(succValue(:,:,1) + sum((irew(:,:,1)))');
        Value=Value+ policy(2,:)'.*(succValue(:,:,2)+ sum((irew(:,:,2)))');
        Value=Value+ policy(3,:)'.*(succValue(:,:,3)+ sum((irew(:,:,3)))');
        Value=Value+ policy(4,:)'.*(succValue(:,:,4)+ sum((irew(:,:,4)))');

        Value = Value.*invAbsorb; % Eliminating the terminate state values

        diff = sum((Value - prevValue).^2); % MSE between two iterations
        prevValue = Value; % Updating the preValue for the next cycle

        if diff<0.00001 % stop if the iteration difference is low enough
            stop = 1;
        end
    end
end

function prob = trace_probability(T, policy, seq)
    % The "policy" input should be a matrix of size [actions x states].
    % For every (state,action) pair, there is a corresponding probability.

    prob = 1; % Will be multiplied with state trans. probabilities

    for i = 1:length(seq)-1 % No need for calculation for the last state

        % We will find the probability of going from one state to the next
        % for every action and sum it, which will give the total
        % probability of going from seq(i) to seq(i+1)
        tProb = policy(1,seq(i))*T(seq(i+1),seq(i),1)+...
                policy(2,seq(i))*T(seq(i+1),seq(i),2)+...
                policy(3,seq(i))*T(seq(i+1),seq(i)+3)+...
                policy(4,seq(i))*T(seq(i+1),seq(i),4);
        prob = prob*tProb; % Trace probability is updated here
    end
end

function posterior = give_bias(T, prior, seq)
    % This function will take a transition matrix, a prior policy, and a
    % sequence; and will return a posterior policy for which the given
    % sequence has a higher probability of occuring. The way it does this
    % is increasing the probability of the action that gives the maximum
    % probability of going from current state to the next state in the
    % given sequence.

    posterior = prior; % Posterior policy will be modified
    a = 1; % Constant to adjust the probability of choosing an action

    for i=1:length(seq)-1
        [M, I] = max(T(seq(i+1), seq(i),:)); % Finding the optimal action
        posterior(I,seq(i))=posterior(I,seq(i))+a; % Increasing its prob.
        posterior(:, seq(i)) = posterior(:, seq(i))/(a+1); % Normalizing
    end
end

function trace = generate_trace(T, R, pol, SN, AN, Absorb)
    trace = {}; % The trace will be stored in this cell aray

    % Finding the initial state randomly (using uniform probability)
    s = rand*4+11; % Random number for choosing the initial state
    state = fix(s); % Now it is a random integer between 11 and 14
```

```matlab
    trace = [trace SN(state,:)]; % Adding the initial state to the trace

    terminate = 0; % Will terminate the loop if we reach an absorbing state
    while terminate == 0
        % Individual action probabilities will determine how likely an
        % action will be chosen in the condition below
        x = rand; % To be used in the policy when choosing an action
        if x<pol(1,state) % p(1)
            action = 1;
        elseif x<pol(2,state)+pol(1,state) % p(1)+p(2)
            action = 2;
        elseif x<pol(3,state)+pol(2,state)+pol(1,state) % p(1)+p(2)+p(3)
            action = 3;
        else % If the probability is higher, it is the 4th action
            action = 4;
        end

        trace = [trace AN(action)]; % Adding the action to the trace

        % Below line will find the successor states that has nonzero
        % probabilities so we don't have to go over all states
        nonzeroP = find(T(:,state,action));

        % Below loop will choose a successor state based on the current
        % state, the taken action and the corresponding successor state
        % probabilities on the transition matrix.
        y = rand; % Random number to be used in the transition matrix
        prob = 0;
        for i=1:length(nonzeroP)
            % Below probability is summed until finding the successor state
            prob = prob+T(nonzeroP(i),state,action); % Checking next prob.
            if y<prob % Checking the successor state probabilities
                postState = nonzeroP(i); % Successor state was found
                break % Breaking the loop, VERY IMPORTANT!
                % If you don't break the loop, this code will always give
                % the last state in 'nonzeroP'. You don't want this
            end
        end

        reward = R(postState, state, action); % Corresponding reward value
        trace = [trace reward SN(postState,:)]; % Updating the trace

        if Absorb(postState)==1 % If the absorbing state was reached
            terminate = 1; % Stop the trace
            trace = trace(1:length(trace)-1);
        end
        state = postState; % The current state in the next cycle
    end
end

function Value = policy_evaluation(traces, disc)
    % This function gets a list of traces (in 'cell' format) and the
    % discount value and returns the estimated value function for states
    % according to the First-Visit Batch Monte Carlo policy evaluation

    len=length(traces);
    Value = zeros(14,1);
    stateReturns = cell(14,1); %Cell array was used to append state returns

    for i=1:len % Work this loop for each TRACE
        visitedStates = []; % To implement the first visit MC in each trace
```

```
        tau = traces{i}; % The indiced trace, will be used in the next loop
        stLen = (length(tau))/3; % How many states are there in the trace
        rew = cell2mat(tau(3:3:length(tau))); % Reward array of the trace

        for j=1:stLen % Work this loop for each STATE in the trace
            state = str2num(traces{i}{3*j-2}(2:3));% Indice of the state

            if ismember(state, visitedStates)
                % If the state is visited before, do nothing (First-visit)
            else
                % If it is the first visit, calculate and append the return
                visitedStates = [visitedStates state];
                % Total discounted rewards is given below
                Ret=sum(rew(j:stLen).*(disc.^(0:length(rew(j:stLen))-1)));
                % Adding the reward to the return array
                stateReturns{state} = [stateReturns{state} Ret];
            end
        end
    end

    Value = cellfun(@mean, stateReturns); % Average return for each state

    % If a state has never been visited, the return array will be empty and
    % averaging in will give 'NaN'. Instead, we assign them zero
    x = find(isnan(Value));
    Value(x) = 0;
end

function [finalPolicy, TraceLen, Rewards] =...
    learn(NumIter, disc, eps, T, R, pol, SN, AN, Absorb) % Function start

    % This function implements the e-greedy first-visit Monte Carlo control
    % algorithm. It returns the policy at the end of the learning, the
    % trace length array that shows how trace length changed throughout
    % learning and the reward array for showing the change in rewards

    Returns = cell(length(AN), length(SN)); %Returns for state-action pairs
    Rewards = zeros(1,NumIter); % Reward for each episode
    TraceLen = zeros(1,NumIter); % Trace length for each episode
    Qfunc = zeros(length(AN),length(SN)); % State-Action Val. function
    policy = pol; % Policy to be updated when learning

    for i=1:NumIter
        % An episode is generated for each iteration
        episode = generate_trace(T, R, policy, SN, AN, Absorb);
        SAnum = (length(episode))/3; % State-Action amount in an episode
        visitedSA = []; % Visited state-action pairs in each episode
        rew =cell2mat(episode(3:3:length(episode))); %Reward of the episode
        Rewards(i) = sum(rew);
        TraceLen(i) = (length(episode))/3; % Trace length added

        for j=1:SAnum
            s = str2num(episode{3*j-2}(2:3)); % Curent state
            a = episode{3*j-1}; % Current action in letters (N,E,S,W)
            a = AN==a;
            a = find(a); % Current action in number (1,2,3,4)

            % Below notation is used to utilize the ismember() function
            % Integer value is the state, decimal value is the action
            s_a = str2num(strcat(num2str(s), '.',num2str(a)));
```

```
            if ismember(s_a, visitedSA)
                % If the state-action is visited before, do nothing
            else
                % If it is the first visit, calculate and append the return
                visitedSA = [visitedSA s_a]; % Updating the visited S-A
                % Total discounted rewards found below
                ret= sum(rew(j:SAnum).*(disc.^(0:length(rew(j:SAnum))-1)));
                Returns{a,s} = [Returns{a,s} ret]; % Appending the return
            end
        end

        Qfunc = cellfun(@mean, Returns); % Q(s,a) function updated
        x = find(isnan(Qfunc)); % To eliminate 'NaN' terms
        Qfunc(x) = 0;

        updatedState = []; % States for which the policy is updated
        for k=1:length(visitedSA)
            saTemp = visitedSA(k);
            sNew = fix(saTemp);
            if ismember(sNew, updatedState)
                % If the state policy is already updated, do nothing
            else
                % If not, implement eps-greedy algorithm
                updatedState = [updatedState sNew];
                [M, aStar] = max(Qfunc(:,sNew)); % The optimal action
                policy(:,sNew)= eps/length(AN); % Suboptimal actions
                policy(aStar, sNew) = policy(aStar, sNew)+1-eps;
            end
        end
    end
    finalPolicy = policy; % The policy that will be returned by the func.
end
```

Appendix B: Question 4.c result for 500 traces