COURSEWORK 1

IMPERIAL COLLEGE LONDON

Reinforcement Learning

Author:
Jonas Tjomsland
Msc Human and Biological Robotics
(CID: 01570830)

Date: November 16, 2018

1.

CID: 01570830
Personal p = 0.65
Personal gamma = 0.2

2.

The obtained value function for an unbiased policy. See Matlab code for reference, I used a tolerance of 0.01 for the policy evaluation.

State:	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Value:	-0.901	0	0	-3.694	-1.220	-1.041	-3.555	-1.500	-1.248	-1.263	-1.249	-1.249	-1.249	-1.250

Figure 1: The value function for an unbiased policy.

3.

a)

To calculate the likelihood of observing the three sequences, with an unbiased policy, we use the following two concepts. $P(a|s)$, the probability of choosing action a when in state s, known as the policy of the MDP. And secondly, $P^a_{ss'}$, the probability of ending up in state s' when starting in state s and executing action a, known as the transition probability. For example, to calculate the probability of going from state 14 to state 10 we would compute the following.

$$P(10) = P(10|N)P(N) + P(10|S)P(S) + P(10|W)P(W) + P(10|E)P(E) \tag{1}$$

Which for this example would be,

$$P(10) = p\frac{1}{4} + 0\frac{1}{4} + \frac{(1-p)}{2}\frac{1}{4} + \frac{(1-p)}{2}\frac{1}{4} \tag{2}$$

where p is the given probability for executing desired move and an unbiased policy. By doing this we include every possible way we could end up in state 10 starting in state 14. To find the likelihood of observing a sequence we iterate over all the states of the sequence, for each iteration calculating the probability of moving to the desired state. Then we take the product of all those probabilities times the probability of starting in the first state to get the likelihood of the whole sequence occurring. In the Matlab code $P^a_{ss'}$ is represented by the transition matrix and $P(a|s)$ by the policy. We see from equation (2) that for the unbiased case the likelihood of observing a certain sequence will always be the policy, $\frac{1}{4}$ in this case, to the power of number of states in the sequence plus one.

b)

To find a policy that performs better than the unbiased one I created a new function called "biased_likelihood". Very similar to the function used in a), it iterates over every state of the sequence and then every possible action for each state. The difference now is that for every state it checks which action that gives the best probability and then changes the state's policy to always choose that action. In my Matlab code you will see that this is done by always picking the policy which maximize the variable called "local_p", see Matlab code for reference.

However, a problem occurs when the same state is in more than one sequence or appear two times in the same sequence. That means that the best action isn't always the same one for all cases. For these three sequences the states 11, 5, 9 and 6 appear several times. Some of these states, like s11 and s9, the the algorithm handles automatically. To handle s6 and s5 I manually chose the most suitable action. The resulting policy for the involved states is presented below.

State:	S1	S2	S3	S4	S5	S6	S8	S9	S10	S11	S12	S14
Value:	E	Terminal state	Terminal state	W	E	E	N	N	N	N	W	N

Figure 2: The optimized policy for the three sequences.

4.

a)

Ten random selected traces from the MDP are presented in Appendix 1. The traces are generated by letting a agent act in the environment, choosing actions based on an unbiased policy and succeeding based on the probabilities in the transition matrix. The visited states, taken actions and resulting returns are printed as asked.

b)

To compute the estimated value function I created three functions in my code, all should be clearly written and well-commented. The algorithm does, as according to MC First Visit, calculate the value of a state as the discounted returns following that state's first occurrence. It does this for all ten traces and then store the value as the mean of the values from all traces. The estimated value function from the Monte Carlo First Visit policy evaluation is presented in Fig. 3.

State:	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Value:	-0.566	0	0	-4.171	-1.151	-0.950	-3.770	-1.320	-1.250	-1.260	-1.250	-1.250	-1.250	-1.250

Figure 3: The estimated value function from the Monte Carlo First Visit method.

c)

I used Mean Squared Error (MSE) as measure of similarity between the value function obtained in Q2 an the MC First Visit value function. MSE is a widely used measure of the quality of an estimate, it accounts for both the variance of the estimator and its bias. In Fig. 4 the MSE is plotted with respect to number of traces.

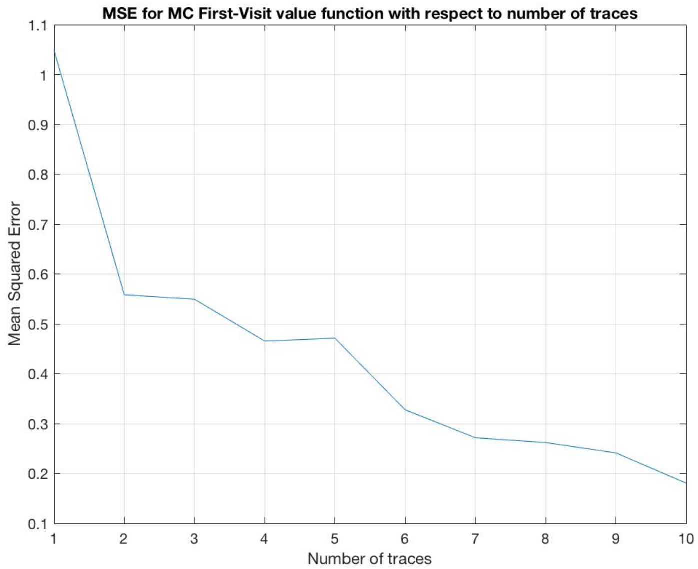


Figure 4: The estimated value function from the Monte Carlo First Visit method.

One can clearly see the trend of decreasing MSE, which implies higher similarity, as the number of traces increase. This example was one of the "better-looking" graphs, but all showed the same pattern of decreasing MSE. The reason for some of them looking abnormal is that ten traces still is a low number and in some of them a state may be observed only one time. In that case the reported value of that state won't be as accurate as if it had appeared in several traces.

5.

For question five I created three functions to simplify the implementation of the ϵ-greedy first-visit Monte Carlo control. The code should be well commented, but I'll briefly explain the algorithm.

The algorithm does 30 trials, for every trial it lets the agent learn for 1-100 episodes. For every number of episodes it append the resulting trace length and rewards to a cell matrix containing information about all trials. Then, after all 30 trials, the mean, as well as the standard deviation, for all trials are computed for both trace lengths and rewards.

It is clear from the results that a higher ϵ leads to a better learning curve. This may be because a larger ϵ would make the the possibility of the agent choosing the best action higher. The results are presented in Fig. 5-8.

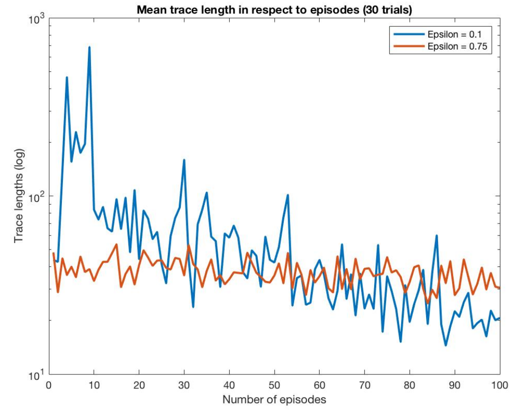


Figure 5: Instability in joint angles for spiral path.

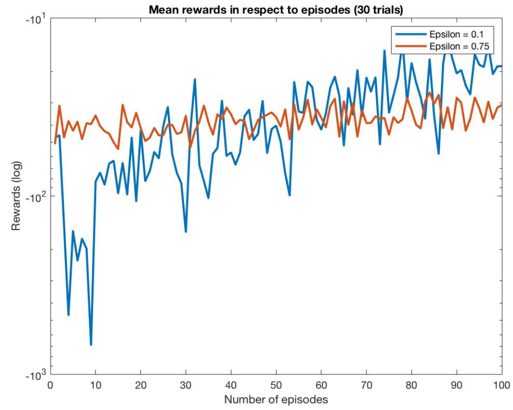


Figure 6: Instability in the pseudo inverse for spiral path.

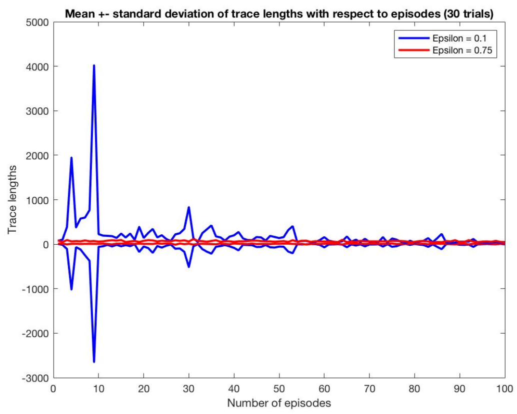


Figure 7: Instability in joint angles for spiral path.

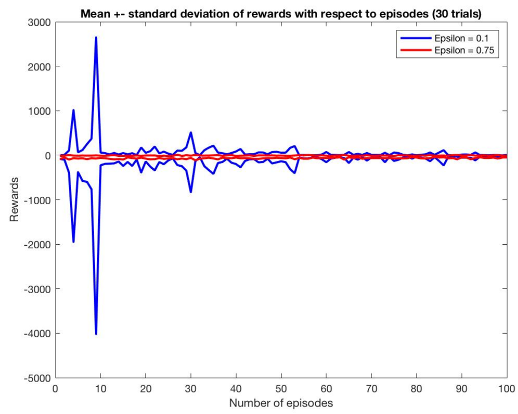


Figure 8: Instability in the pseudo inverse for spiral path.

Appendix

Traces from Q4:

1. S12,S,-1,S12,S,-1,S12,N,-1,S12,N,-1,S12,S,-1,S11,S,-1,S12,W,-1,S12,S,-1,S12,E,-1,S13,N,-1,S13,S,-1,S14,N,-1,S10,N,-1,S10,W,-1,S10,S,-1,S10,E,-1,S8,E,-1,S10,E,-1,S10,N,-1,S10,E,-1,S10,W,-1,S8,E,-1,S8,W,-1,S7,W,-1,S6,W,-1,S5,N,-1,S1,S,0
2. S12,S,-1,S12,E,-1,S13,W,-1,S12,N,-1,S11,W,-1,S11,N,-1,S12,W,-1,S11,N,-1,S9,S,-1,S11,N,-1,S9,E,-1,S9,E,-1,S9,N,-1,S5,S,-1,S9,S,-1,S11,W,-1,S11,S,-1,S12,W,-1,S11,W,-1,S11,W,-1,S11,S,-1,S11,S,-1,S12,E,-1,S13,S,-1,S12,W,-1,S12,E,-1,S13,E,-1,S13,S,-1,S13,W,-1,S12,W,-1,S11,W,-1,S11,W,-1,S11,N,-1,S9,N,-1,S5,S,-1,S6,N,-1,S5,W,-1,S9,S,-1,S11,S,-1,S11,S,-1,S11,W,-1,S11,W,-1,S11,E,-1,S9,S,-1,S11,W,-1,S9,W,-1,S9,S,-1,S11,W,-1,S11,S,-1,S11,N,-1,S9,W,-1,S11,E,-1,S12,N,-1,S12,N,-1,S13,N,-1,S14,E,-1,S14,W,-1,S13,S,-1,S13,N,-1,S13,N,-1,S13,E,-1,S14,W,-1,S14,N,-1,S10,S,-1,S14,S,-1,S14,E,-1,S14,N,-1,S10,E,-1,S10,E,-1,S10,E,-1,S10,S,-1,S10,N,-1,S8,W,-1,S7,N,-10

3. S12,S,-1,S12,E,-1,S12,W,-1,S11,S,-1,S11,W,-1,S11,N,-1,S9,N,-1,S5,E,-1,S9,E,-1,S5,S,-1,S6,W,-1,S5,E,-1,S9,S,-1,S9,N,-1,S5,S,-1,S9,N,-1,S5,E,-1,S6,S,-1,S7,S,-1,S7,E,-1,S8,W,-1,S7,S,-1,S8,E,-1,S10,W,-1,S8,N,-1,S4,E,-1,S4,N,-1,S4,W,-10

4. S11,E,-1,S12,E,-1,S13,W,-1,S13,S,-1,S13,N,-1,S13,S,-1,S13,W,-1,S13,N,-1,S13,N,-1,S12,S,-1,S12,E,-1,S13,N,-1,S12,E,-1,S13,S,-1,S13,W,-1,S12,S,-1,S13,E,-1,S14,N,-1,S10,W,-1,S10,W,-1,S10,E,-1,S8,S,-1,S10,N,-1,S8,W,-1,S10,W,-1,S14,S,-1,S14,W,-1,S13,W,-1,S12,E,-1,S13,W,-1,S12,W,-1,S11,S,-1,S11,W,-1,S11,N,-1,S12,N,-1,S12,N,-1,S12,N,-1,S11,N,-1,S9,N,-1,S9,E,-1,S5,W,-1,S5,S,-1,S9,N,-1,S5,S,-1,S9,W,-1,S9,E,-1,S9,N,-1,S5,W,-1,S5,S,-1,S9,E,-1,S5,S,-1,S9,E,-1,S9,W,-1,S9,W,-1,S9,W,-1,S9,S,-1,S11,N,-1,S11,N,-1,S12,S,-1,S12,S,-1,S12,N,-1,S12,S,-1,S12,W,-1,S11,N,-1,S12,N,-1,S12,N,-1,S12,E,-1,S13,S,-1,S14,N,-1,S10,E,-1,S10,S,-1,S14,N,-1,S13,E,-1,S14,S,-1,S14,W,-1,S14,S,-1,S14,W,-1,S13,N,-1,S13,S,-1,S13,E,-1,S13,E,-1,S14,E,-1,S14,N,-1,S10,E,-1,S10,N,-1,S8,E,-1,S10,N,-1,S8,W,-1,S7,E,-10

5. S13,N,-1,S14,W,-1,S14,S,-1,S14,N,-1,S10,E,-1,S8,E,-1,S10,N,-1,S10,N,-1,S10,W,-1,S8,E,-1,S10,S,-1,S10,N,-1,S8,S,-1,S10,W,-1,S10,S,-1,S10,N,-1,S10,N,-1,S8,S,-1,S10,S,-1,S10,N,-1,S8,W,-1,S7,W,-10

6. S14,N,-1,S14,W,-1,S13,E,-1,S14,S,-1,S14,N,-1,S10,W,-1,S10,S,-1,S14,N,-1,S10,N,-1,S10,N,-1,S8,E,-1,S8,W,-1,S10,W,-1,S10,E,-1,S10,E,-1,S10,S,-1,S14,E,-1,S14,S,-1,S14,W,-1,S14,E,-1,S14,S,-1,S14,N,-1,S14,E,-1,S14,S,-1,S14,N,-1,S10,S,-1,S14,N,-1,S10,W,-1,S10,E,-1,S14,W,-1,S10,S,-1,S14,W,-1,S14,N,-1,S13,N,-1,S13,N,-1,S13,S,-1,S13,W,-1,S13,S,-1,S13,N,-1,S13,N,-1,S13,N,-1,S12,W,-1,S12,E,-1,S12,S,-1,S11,E,-1,S12,W,-1,S11,S,-1,S11,W,-1,S11,S,-1,S11,W,-1,S11,W,-1,S11,S,-1,S11,E,-1,S12,W,-1,S12,S,-1,S12,W,-1,S11,W,-1,S11,N,-1,S12,S,-1,S12,E,-1,S13,W,-1,S12,S,-1,S12,E,-1,S13,S,-1,S13,W,-1,S12,N,-1,S12,E,-1,S12,S,-1,S12,N,-1,S12,S,-1,S12,E,-1,S13,W,-1,S12,E,-1,S13,W,-1,S13,S,-1,S14,E,-1,S14,E,-1,S14,W,-1,S13,W,-1,S12,W,-1,S11,E,-1,S11,S,-1,S12,N,-1,S13,S,-1,S12,S,-1,S12,N,-1,S11,S,-1,S11,N,-1,S9,N,-1,S5,W,-1,S5,S,-1,S9,N,-1,S5,W,-1,S9,N,-1,S9,E,-1,S9,S,-1,S11,E,-1,S12,E,-1,S12,E,-1,S13,N,-1,S13,W,-1,S13,E,-1,S13,S,-1,S14,S,-1,S13,W,-1,S13,S,-1,S13,N,-1,S12,E,-1,S12,W,-1,S11,S,-1,S11,S,-1,S11,W,-1,S11,E,-1,S9,E,-1,S9,S,-1,S9,W,-1,S9,W,-1,S9,N,-1,S5,S,-1,S9,E,-1,S11,N,-1,S11,S,-1,S12,S,-1,S13,E,-1,S14,W,-1,S10,S,-1,S10,S,-1,S10,W,-1,S10,W,-1,S8,E,-1,S4,N,-1,S4,S,-1,S8,N,-1,S4,W,-1,S4,W,-10

7. S11,W,-1,S11,W,-1,S11,S,-1,S11,W,-1,S9,N,-1,S9,E,-1,S9,S,-1,S11,E,-1,S12,W,-1,S11,W,-1,S11,W,-1,S11,N,-1,S12,W,-1,S12,W,-1,S11,W,-1,S11,S,-1,S11,S,-1,S11,W,-1,S11,N,-1,S9,E,-1,S9,E,-1,S5,W,-1,S9,S,-1,S11,N,-1,S9,N,-1,S5,N,-1,S1,W,-1,S1,W,-1,S1,S,0

8. S14,S,-1,S14,E,-1,S14,S,-1,S14,W,-1,S13,E,-1,S14,W,-1,S14,N,-1,S13,W,-1,S12,W,-1,S11,N,-1,S12,S,-1,S11,N,-1,S9,W,-1,S5,S,-1,S5,E,-1,S6,S,-1,S6,S,-1,S6,E,-1,S6,S,-1,S6,N,0

9. S12,S,-1,S12,S,-1,S12,N,-1,S11,W,-1,S11,W,-1,S11,W,-1,S11,W,-1,S11,W,-1,S11,S,-1,S11,S,-1,S11,E,-1,S9,S,-1,S11,S,-1,S11,N,-1,S11,W,-1,S11,N,-1,S9,N,-1,S9,N,-1,S9,W,-1,S11,E,-1,S11,S,-1,S11,W,-1,S11,E,-1,S12,S,-1,S12,W,-1,S11,E,-1,S12,S,-1,S12,S,-1,S12,W,-1,S12,W,-1,S11,S,-1,S11,E,-1,S12,E,-1,S13,E,-1,S14,W,-1,S13,E,-1,S14,E,-1,S14,E,-1,S14,N,-1,S14,W,-1,S13,E,-1,S14,W,-1,S13,S,-1,S13,S,-1,S13,E,-1,S13,E,-1,S13,S,-1,S12,S,-1,S12,N,-1,S13,N,-1,S13,N,-1,S13,E,-1,S14,S,-1,S14,S,-1,S14,W,-1,S13,W,-1,S13,S,-1,S13,S,-1,S13,N,-1,S13,N,-1,S13,S,-1,S12,W,-1,S11,S,-1,S11,E,-1,S12,E,-1,S13,S,-1,S14,E,-1,S14,E,-1,S14,E,-1,S14,S,-1,S14,S,-1,S14,S,-1,S14,N,-1,S14,S,-1,S14,W,-1,S13,S,-1,S13,W,-1,S12,N,-1,S12,W,-1,S12,E,-1,S13,S,-1,S13,W,-1,S12,N,-1,S11,S,-1,S11,E,-1,S11,S,-1,S12,S,-1,S12,W,-1,S12,N,-1,S13,S,-1,S13,W,-1,S12,N,-1,S13,E,-1,S14,E,-1,S14,E,-1,S10,E,-1,S8,N,-1,S8,N,-1,S4,S,-10

10. S13,S,-1,S12,E,-1,S13,S,-1,S13,E,-1,S14,E,-1,S10,E,-1,S10,E,-1,S10,E,-1,S14,S,-1,S14,E,-1,S10,E,-1,S10,W,-1,S8,E,-1,S10,N,-1,S8,E,-1,S8,W,-1,S7,N,-10

Table of Contents


```
% Coursework in Machine Learning and Neural Computation
% Jonas Tjomsland, CID = 01570830
% To make it easier for the reader to understand I have tried to use
% similar set up as Dr. A. Aldo Faisal used for the first lab.

clc
clear all
close all
RunCoursework();

function RunCoursework()
```

Question 1

```
%Calculating persoonal p and personal gamma:
p = 0.5 + 0.5*(3/10);
gamma = 0.2 + 0.5*(0/10);


% Get system parameters from given grid world function
[NumStates, NumActions, TransitionMatrix, ...
 RewardMatrix, StateNames, ActionNames, AbsorbingStates] ...
 = PersonalisedGridWorld(p);

% Simplifying names:
n = NumStates;
a = NumActions;
T = TransitionMatrix;
R = RewardMatrix;
S = StateNames;
A = ActionNames;
Absorbing = AbsorbingStates;

% Creating policy matrix where the rows represent states and the
 columns
% possible actions: S1: N, E, S, W (14x4) matrix.
% Unbiased policy means equal probability of all actions.(1/4 in this
 case)
Policy = 1/4*ones(14,4);
% Chooosing tolerance for policy evaluation:
tol = 0.01;
```

Question 2

```
% Calling policy evaluation function:
V = policy_evaluation(n, a, T, R, Absorbing, Policy, tol, gamma);
disp("Value function: ")
disp(" ")
% Calling print function for V
format short
print_V_table(V)
```

```
Value function:

     State        S1        S2     S3       S4          S5          S6
     S7          S8         S9         S10         S11         S12
 S13         S14

    _______     ________    __     __     _______     _______     _______
    _______     _______    ______    _______     _______     _______
 _______     _______

    "Value"     -0.90098    0      0      -3.6942     -1.2203     -1.0406
    -3.5546     -1.5003    -1.248    -1.2635     -1.2495     -1.2496
 -1.2496     -1.2503
```

Question 3

```
% a) Likelihood

% Sequence vectors:
seq1 = [14, 10, 8, 4, 3];
seq2 = [11, 9, 5, 6, 6, 2];
seq3 = [12, 11, 11, 9, 5, 9,  5, 1, 2];

% Calling likelihood function:
likelihood1 = likelihood(seq1, T, Policy, a);
likelihood2 = likelihood(seq2, T, Policy, a);
likelihood3 = likelihood(seq3, T, Policy, a);

% b) Optimizing policy for likelihood

%Calling function for policy optimization:
optimal_policy = unbiased_policy(seq1, T, Policy, a);
optimal_policy = unbiased_policy(seq2, T, optimal_policy, a);
optimal_policy = unbiased_policy(seq3, T, optimal_policy, a);

% New likelihoods
likelihood1_improved = likelihood(seq1, T, optimal_policy, a);
likelihood2_improved = likelihood(seq2, T, optimal_policy, a);
likelihood3_improved = likelihood(seq3, T, optimal_policy, a);


% Print as table:
```

```
likelihoods = table(likelihood1, likelihood1_improved, likelihood2...
                    ,likelihood2_improved, likelihood3,
 likelihood3_improved);

disp("Likelihoods before and after policy optimisation: ")
disp(" ")
disp(likelihoods)
```

Likelihoods before and after policy optimisation:

likelihood1	likelihood1_improved	likelihood2	likelihood2_improved	likelihood3	likelihood3_improved
0.00097656	0.044627	0.00024414	0.0021026	7.6294e-06	0.00015546

Question 4

```
% a)
% Generate trace with unbiased policy
% Remove comments here and in trace function to display:
disp("Traces with unbiased policy:")
disp(" ")
% Variable to let the functions know if we want to print:
print = 1;
% Number of traces:
n_traces = 10;
% Generate traces, using a nested function.
[traces, all_rewards, all_actions] = generate_traces(n, a, T, R,
 Absorbing, Policy, n_traces, print);
disp(" ")

% b)

% Generate the returns for every state from a given set of traces and
% and corresponding rewards.
returns = MC_policy_returns(n, traces, all_rewards, Absorbing, gamma);

disp("Value function estimated with MC-First visit method for 10
 traces:")
disp(" ")
V_MC = MC_Value_function(returns);
format short
print_V_table(V_MC)

% c)
% I use Mean Squared Error as measure of similarity between V and V_MC
MSE = [];
% Variable to let the functions know if we want to print traces:
print = 0;
```

```
% Compute the distance for 1 to 10 traces and plot the result.
 Essentially
% repeating the steps in b) but compute the distance every step.
for n_traces = 1:10
    [traces, all_rewards] = generate_traces(n, a, T, R, Absorbing,
 Policy, n_traces,print);
    returns = MC_policy_returns(n, traces, all_rewards, Absorbing,
 gamma);
    V_MC = MC_Value_function(returns);
    MSE = [MSE, mean(sqrt((V-V_MC).^2))];
end

%To see plot for 1 to 10 traces, remove comments below:
n_traces = 1:10;
figure
plot(n_traces,MSE)
grid on
xlabel("Number of traces")
ylabel("Mean Squared Error")
title("MSE for MC First-Visit value function with respect to number of
 traces")
```

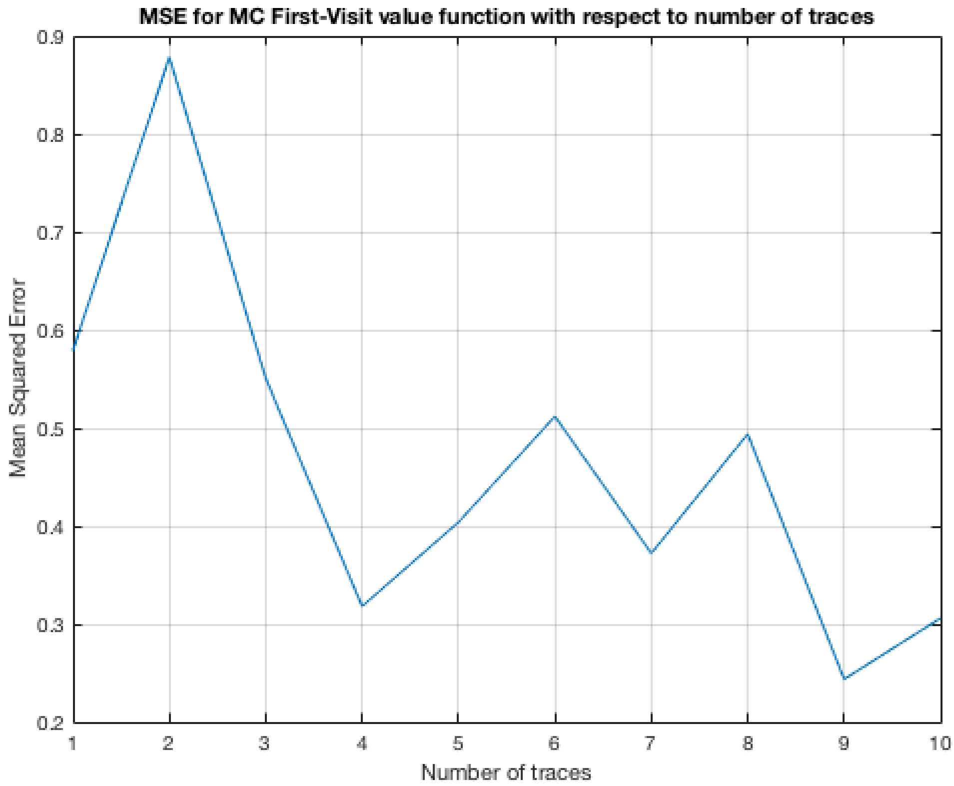
Traces with unbiased policy:

1:
s11,N,-1,s9,E,-1,s11,E,-1,s9,W,-1,s9,N,-1,s9,S,-1,s9,W,-1,s9,S,-1,s11,E,-1,s12,W,
2:
s13,W,-1,s12,E,-1,s13,S,-1,s13,S,-1,s13,E,-1,s14,E,-1,s14,N,-1,s10,N,-1,s8,W,-1,s
3:
s11,W,-1,s11,N,-1,s9,S,-1,s9,S,-1,s9,S,-1,s11,N,-1,s9,S,-1,s11,N,-1,s12,E,-1,s12,
4:
s13,S,-1,s12,E,-1,s13,S,-1,s14,S,-1,s13,N,-1,s13,S,-1,s13,W,-1,s13,E,-1,s14,N,-1,
5:
s14,S,-1,s14,N,-1,s13,S,-1,s14,S,-1,s14,E,-1,s14,W,-1,s13,N,-1,s12,N,-1,s11,S,-1,
6:
s13,E,-1,s14,W,-1,s13,N,-1,s13,S,-1,s13,S,-1,s13,W,-1,s13,W,-1,s13,N,-1,s13,E,-1,
7:
s12,S,-1,s12,E,-1,s13,E,-1,s14,N,-1,s14,N,-1,s10,W,-1,s10,W,-1,s14,W,-1,s13,W,-1,
8:
s11,W,-1,s11,N,-1,s11,W,-1,s11,W,-1,s11,N,-1,s11,S,-1,s11,N,-1,s9,E,-1,s9,S,-1,s9
9:
s11,S,-1,s11,W,-1,s11,N,-1,s9,W,-1,s5,S,-1,s9,S,-1,s11,E,-1,s11,N,-1,s11,S,-1,s11
10:
s13,S,-1,s13,N,-1,s14,S,-1,s14,S,-1,s14,E,-1,s14,E,-1,s14,N,-1,s10,N,-1,s8,E,-1,s

Value function estimated with MC-First visit method for 10 traces:

State	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
________	________	__	__	______	_____	________	________	________	______	________	______	______	______	_______

```
"Value"    -1.2369    0     0     -8.25    -1.3    -1.6001
-5.6257    -2.1857    -1.25    -1.3375    -1.25    -1.25    -1.25
-1.252
```



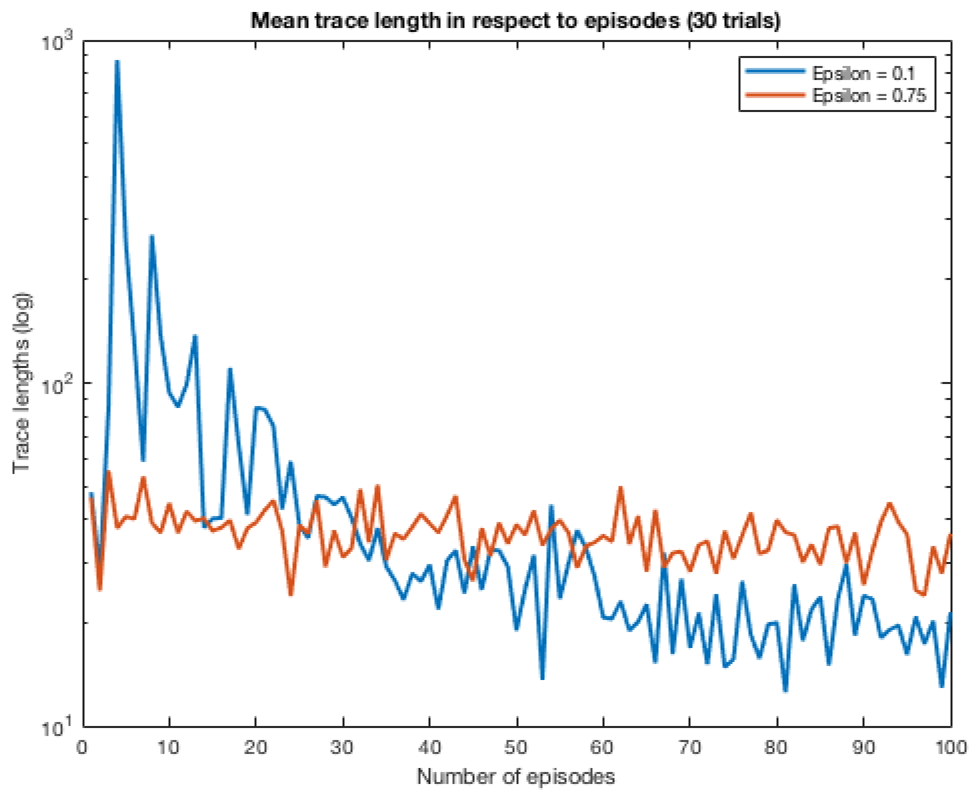
Question 5

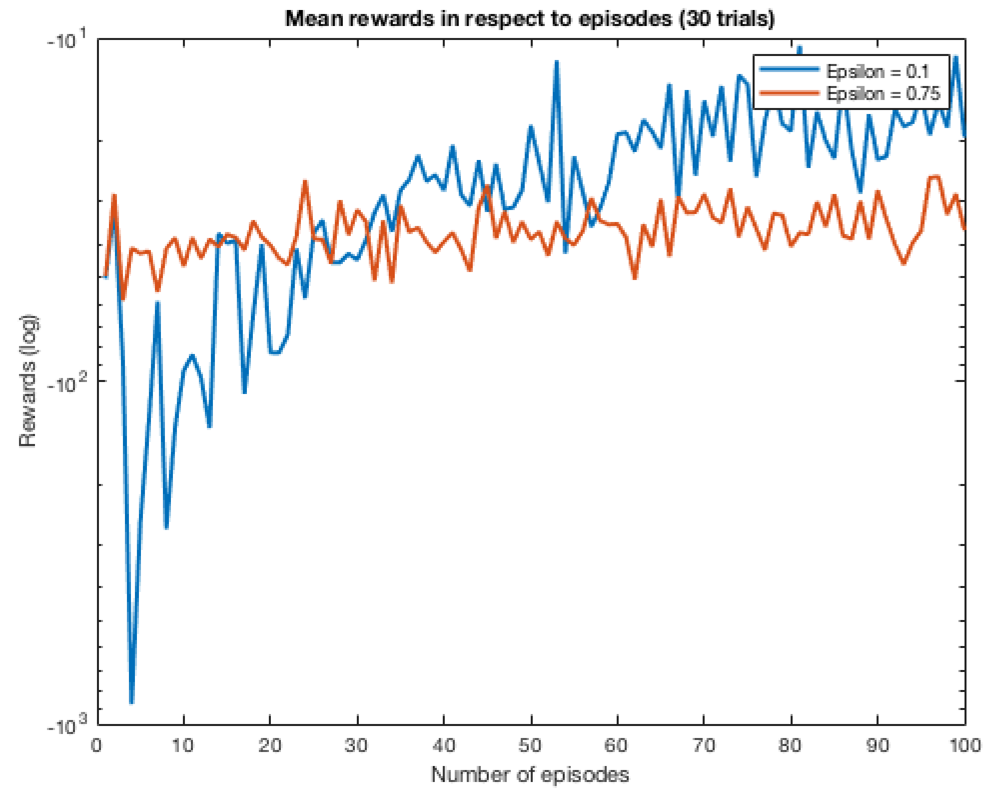
```
% Implementing epsilon-greedy policy for First-Fisit MC Control:
% Initialize epsilon 1 & 2
epsilon_1 = 0.1;
epsilon_2 = 0.75;
epsilon = [epsilon_1, epsilon_2];

% Max number of trials and episodes:
max_trials = 30;
max_episodes = 100;

% Create cell array to store trace length and rewards for every trial.
% Every cell represents 50 trials with a given number of episodes,
 e.g.
% cell ten in "trace_lengths" contains an array of 50 elements where
% every element is the trace length from one trial with ten episodes.
% Double the numbers of cells and place the results for epsilon 2
 after
% those from epsilon 1.
trace_lengths = cell(1, 2*max_episodes);
```

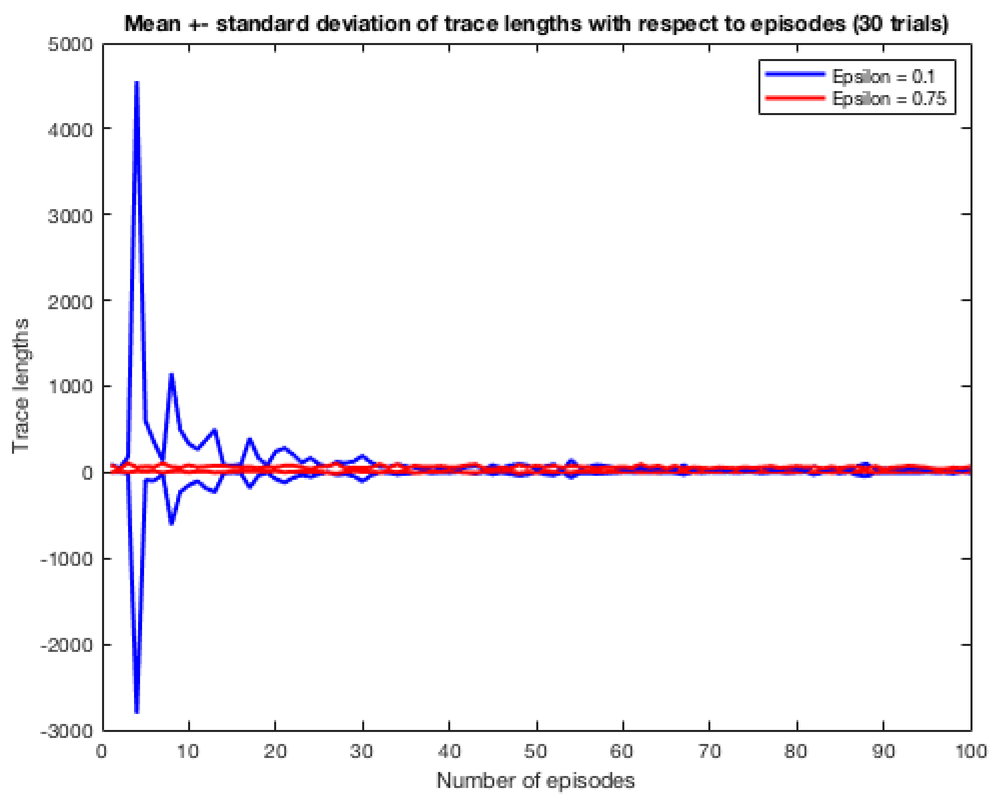
```
trials_rewards = cell(1, 2*max_episodes);

% For both epsilon:
for epsilon = epsilon
    % Do 20 trials for every number of episodes:
    for trials = 1:max_trials
        % Initiate policy as unbiased:
        greedy_policy = Policy;
        % Initiate empty cell matrix for state-action returns:
        total_s_a_returns = cell(n,a);
        % Let the agen operate for 1 to 200 episodes:
        for episodes = 1:max_episodes
            [greedy_policy, states, rewards, actions,
 total_s_a_returns] = MC_control(n, a, T, R, Absorbing, greedy_policy,
 gamma, epsilon, total_s_a_returns);
            % Append trace length and sum of rewards to cell array,
 place
            % results for epsilon 2 at cell 201-400:
            if epsilon == epsilon_2
                trace_lengths{max_episodes+episodes} =
 [trace_lengths{max_episodes+episodes}, length(states)];
                trials_rewards{max_episodes+episodes} =
 [trials_rewards{max_episodes+episodes}, sum(rewards)];
            else
                trace_lengths{episodes} = [trace_lengths{episodes},
 length(states)];
                trials_rewards{episodes} = [trials_rewards{episodes},
 sum(rewards)];
            end
        end
    end
end

% Array for averaged trace lengths and rewards as well as standard
 deviation:
mean_trace_lengths_e1 = [];
mean_rewards_e1 = [];
std_trace_lengths_e1 = [];
std_rewards_e1 = [];

mean_trace_lengths_e2 = [];
mean_rewards_e2 = [];
std_trace_lengths_e2 = [];
std_rewards_e2 = [];

% Compute mean and std for trace lengths and rewards for both espilon:
for i = 1:length(trace_lengths)
    % For epsilon 2:
    if i > length(trace_lengths)/2
        mean_trace_lengths_e2 = [mean_trace_lengths_e2,
 mean(trace_lengths{i})];
        mean_rewards_e2 = [ mean_rewards_e2, mean(trials_rewards{i})];
        std_trace_lengths_e2 = [std_trace_lengths_e2,
 std(trace_lengths{i})];
```

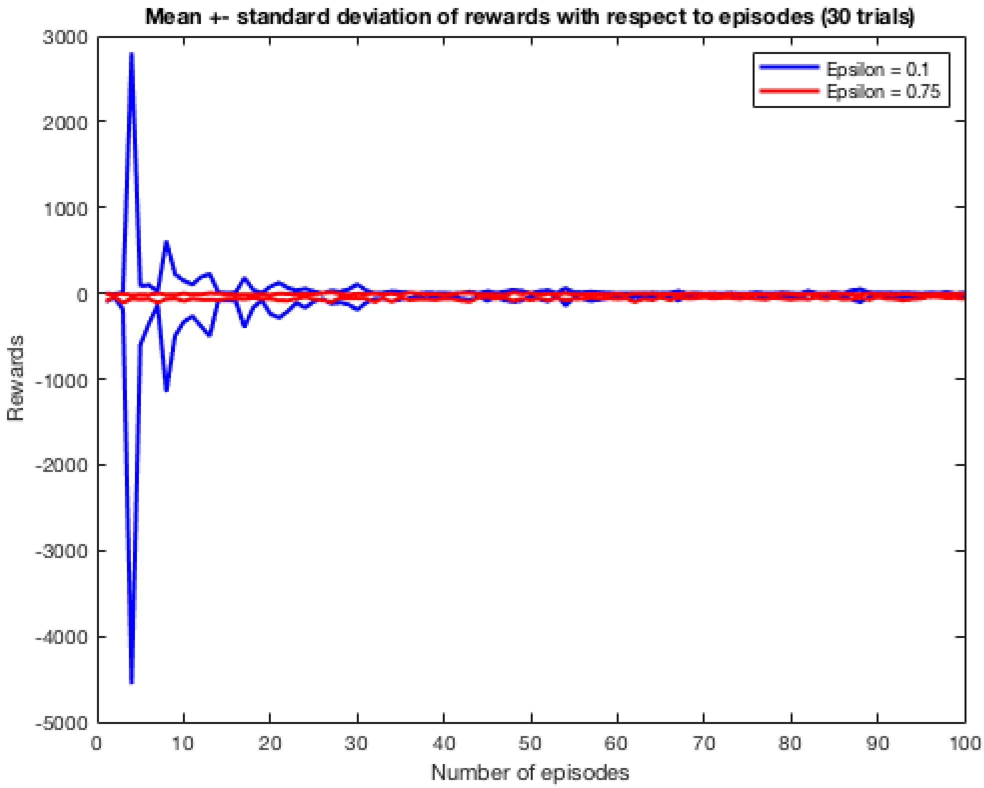
```
        std_rewards_e2 = [std_rewards_e2, std(trials_rewards{i})];
    % For epsilon 1:
    else
        mean_trace_lengths_e1 = [mean_trace_lengths_e1,
 mean(trace_lengths{i})];
        mean_rewards_e1 = [mean_rewards_e1, mean(trials_rewards{i})];
        std_trace_lengths_e1 = [std_trace_lengths_e1,
 std(trace_lengths{i})];
        std_rewards_e1 = [std_rewards_e1, std(trials_rewards{i})];
    end
end

% Plotting results. First only the mean trace lengths adn rewards are
% plotted against episodes. Secondly mean plus/minus std are plotted.
 All
% for both epsilon 1 and epsilon 2.

% Mean trace length against episodes:
figure
semilogy(1:max_episodes,mean_trace_lengths_e1,'LineWidth',2)
hold on
semilogy(1:max_episodes,mean_trace_lengths_e2, 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Trace lengths (log)")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean trace length in respect to episodes (30 trials)")

% Mean rewards against episodes:
figure
semilogy(1:max_episodes,mean_rewards_e1,'LineWidth',2)
hold on
semilogy(1:max_episodes,mean_rewards_e2, 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Rewards (log)")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean rewards in respect to episodes (30 trials)")

% Mean trace lengths plus/minus standard deviation against episodes
figure
plot(1:max_episodes,mean_trace_lengths_e1-
std_trace_lengths_e1, 'b','LineWidth',2)
hold on
plot(1:max_episodes,mean_trace_lengths_e2-
std_trace_lengths_e2, 'r', 'LineWidth',2)
plot(1:max_episodes,mean_trace_lengths_e1+std_trace_lengths_e1,'b', 'LineWidth',2)
plot(1:max_episodes,mean_trace_lengths_e2+std_trace_lengths_e2, 'r', 'LineWidth',2
xlabel("Number of episodes")
ylabel("Trace lengths")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean +- standard deviation of trace lengths with respect to
 episodes (30 trials)")

% Mean rewards plus/minus standard deviation against episodes
figure
```

```
plot(1:max_episodes,mean_rewards_e1-std_rewards_e1, 'b','LineWidth',2)
hold on
plot(1:max_episodes,mean_rewards_e2-
std_rewards_e2, 'r', 'LineWidth',2)
plot(1:max_episodes,mean_rewards_e1+std_rewards_e1,'b', 'LineWidth',2)
plot(1:max_episodes,mean_rewards_e2+std_rewards_e2, 'r', 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Rewards")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean +- standard deviation of rewards with respect to episodes
 (30 trials)")
```




Mean rewards in respect to episodes (30 trials)
Epsilon = 0.1
Epsilon = 0.75
Rewards (log)
-10^1
-10^2
-10^3
0
10
20
30
40
50
60
70
80
90
100
Number of episodes


Mean +- standard deviation of trace lengths with respect to episodes (30 trials)
Epsilon = 0.1
Epsilon = 0.75
Trace lengths
5000
4000
3000
2000
1000
0
-1000
-2000
-3000
0
10
20
30
40
50
60
70
80
90
100
Number of episodes



Functions:

```
% Creating policy evaluation function
% Takes number of states, transition matrix, reward matrix, list of...
% absorbing states, policy and iteration tolerance as input.
% Returns value function V.
function V = policy_evaluation(n, a, T, R, Absorbing, Policy, tol,
 gamma)

    V = zeros(1,n);     % Optimal value function vector
    Vnew = V;        % Value function vector for step i+1.
    delta = 2*tol;   % Used to measure difference in V & Vnew.

    % Value iteration:
    while delta > tol
        for current_state = 1:n          % Iterating over all states
            if Absorbing(current_state) % Skipping terminal states
                continue
            end
            current_V = 0;              % Current value of current
 state
            for action = 1:a            % Iterating over all actions
                current_Q = 0;          % Current state action value
                for next_state = 1:n    % Iterating over all states,
 the transition matrix will cancel oout states out of reach.
```

```
                    current_Q = current_Q + T(next_state,
 current_state, action)*(R(next_state, current_state, action) +
 gamma*V(next_state));
                end
                current_V = current_V + Policy(current_state,
 action)*current_Q;
            end
            Vnew(current_state) = current_V;  % Storing new value for
 current state
        end
    diff = abs(Vnew - V);  % Calculates changes in value function
 vector
    delta = max(diff);  % Compute new delta
    V = Vnew;                % Update value function
    end
end

% Function for printing value function as table.
% Takes value function vector as input and prints the table.
function print_V_table(V)

    % Creating table:
    State = "Value";
    S1 = V(1);
    S2 = V(2);
    S3 = V(3);
    S4 = V(4);
    S5 = V(5);
    S6 = V(6);
    S7 = V(7);
    S8 = V(8);
    S9 = V(9);
    S10 = V(10);
    S11 = V(11);
    S12 = V(12);
    S13 = V(13);
    S14 = V(14);
    Table =
 table(State,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14);
    disp(Table)
end

% Function calculating likelihood of given sequence occuring.
% Takes sequence, transition matrix, policy and number of actions as
 input.
% Returns likelihood.
function likelihood = likelihood(seq, T, Policy, a)

    % Initializing total probability for iteration
    total_p = 1;
    % Iterating over the states of the sequence, stopping at last
 state
    for i = 1:length(seq)-1
        current_state = seq(i);
```

```
        next_state = seq(i+1);
        % Initializing local probability for iteration
        local_p = 0;
        % Iterating over every action
        for action = 1:a
            local_p = local_p + T(next_state, current_state,
 action)*Policy(current_state, action);
        end
        total_p = total_p*local_p;
    end
    likelihood = total_p*(1/4); % Multiply by 1/4 because that is the
 probability of starting in state 14
end

% Function optimizing the policy to increase the likelihood of
 observing a
% given sequence.
% Takes sequence, transition matrix, policy and number of actions as
 input.
% Returns optimal_policy.
function optimal_policy = unbiased_policy(seq, T, Policy, a)

    % Optimal policy matrix, just intend to change the policy of the
 states occuring in the sequence.
    optimal_policy = Policy;

    % Iterating over the states of the sequence, stopping at last
 state
    for i = 1:length(seq)
        current_state = seq(i);
        % Setting no policy for terminal states
        if i == length(seq)
            optimal_policy(current_state, :) = 0;
            continue
        end
        next_state = seq(i+1);
        % Initializing local probability for iteration
        local_p = [];

        % Handle state 6 differently
        if current_state == 6 | current_state == 5
            optimal_action = 2;   % Picked East
            for action = 1:a
                if action == optimal_action
                    optimal_policy(current_state, action) = 1;
                else
                    optimal_policy(current_state, action) = 0;
                end
            end
            continue    % Skipping to next state in the sequence
        end

        % Iterating over every action
        for action = 1:a
```

```
            local_p = [local_p, T(next_state, current_state,
 action)*Policy(current_state, action)];
        end

        [value, optimal_action] = max(local_p);
        for action = 1:a
            if action == optimal_action
                optimal_policy(current_state, action) = 1;
            else
                optimal_policy(current_state, action) = 0;
            end
        end
    end
end

% Function that generates a given number of traces, it takes
 desired...
% number of traces and just calls trace generating function that
 amount of times.
% Returns a cell array of traces and rewards.
function [traces, all_rewards, all_actions] = generate_traces(n, a, T,
 R, Absorbing, Policy, n_traces, print)
% Cell array for storing trace states and rewards, every cell
 represents
% a trace and contains a list of states and rewards.
traces = cell(1,n_traces);
all_rewards = cell(1,n_traces);
all_actions = cell(1,n_traces);
% Create n_traces traces:
for i = 1:n_traces
    if print
        fprintf('%d%s', i, ": ")
    end
    [states, rewards, actions] = generate_trace(n, a, T, R, Absorbing,
 Policy, print);
    traces{i} = states;
    all_rewards{i} = rewards;
    all_actions{i} = actions;
end
end

% Function for trace generation for given MDP
% Takes MDP information like number of states, actions, transistion
 matrix...
% reward matrix and absorbing states. In addition it  takes a policy.
% Returns, length of trace, the states in the trace as well as the
 rewards
function [states, rewards, actions] = generate_trace(n, a, T, R,
 Absorbing, Policy, print)
    % Start out by defining starting state and set that as current
 state:
    starting_states = [11, 12, 13 ,14];
    staring_states_p = (1/4)*ones(1,length(starting_states));
    current_state = randsrc(1, 1, [starting_states;staring_states_p]);
```

```
   % Array of the states visited, actions taken and rewards in the
trace
   states = [current_state];
   action_strs = [];
   actions = [];
   rewards = [];
   % Initialize empyt trace array:
   trace = [];
   % Iterate until terminal state is reached:
   while 1
       % Decide action: (N=1, E=2, S=3, W=4)
       action = randsrc(1,1,[1:a; Policy(current_state,:)]);
       % Iterates over all states to decide next state.
       % Store prob for ending in state s in a array.
       next_state_p = [];
       for next_state = 1:n
           next_state_p = [ next_state_p,  T(next_state,
current_state, action)];
       end
       % Choose next state:
       next_state = randsrc(1,1,[1:n; next_state_p]);
       % Define action as string:
       if action == 1
           action_str = "N";
       elseif action == 2
           action_str = "E";
       elseif action == 3
           action_str = "S";
       else
           action_str = "W";
       end
       reward = R(next_state,current_state,action);
       current_state = next_state;
       states = [states, current_state];  % Append to list of states
       action_strs = [action_strs, action_str];  % Append to list of
action strings
       actions = [actions, action];
       rewards = [rewards, reward];
       if Absorbing(next_state)
           break
       end
   end
   if print
       %Printing trace:
       for state = 1:length(actions)
           if rewards(state) == 0 | rewards(state) == -10
               fprintf('s%d,%s,%d', states(state),
action_strs(state),rewards(state))
           else
               fprintf('s%d,%s,%d,', states(state),
action_strs(state),rewards(state))
           end
       end
       disp(" ")
```

```
    end
end

% Function that generate list of returns for every state for a
 given...
% number of traces. Takes a cell array of traces and a cell array of
% rewards. Calls "trace_returns" function for every trace.
function returns = MC_policy_returns(n, traces, all_rewards,
 Absorbing, gamma)

    % Empty returns cell array:
    returns = cell(1,n);
    % Iterate over all traces:
    for i = 1:length(traces)
        % Call function for Monte Carlo Policy Evaluation:
        trace_returns = MC_policy_evaluation(n, traces{i},
 all_rewards{i}, Absorbing, gamma);
        % Append new returns to cell array:
        for j = 1:length(trace_returns)
            returns{j} = [returns{j} trace_returns{j}];
        end
    end
end

% Function which estimates value function based on observed episodes
% Takes number of states in MDP, an observed trace of states, the
 rewards
% obtained in that trace, information about terminal states and
 discount
% factor gamma.
function trace_returns = MC_policy_evaluation(n, trace, trace_rewards,
 Absorbing, gamma)

        % Array for keeping track of visited states in trace:
        first_visit = ones(1,n);
        % Cell array for returns:
        trace_returns = cell(1,n);

        % Iterate over every state in trace:
        for i = 1:length(trace)
            % Skip terminal states:
            if Absorbing(trace(i))
                continue
            end
            % Only compute return if state hasn't been visited (First
 visit):
            if first_visit(trace(i))
                % Declare variable for discount:
                k = 0:(length(trace)-1-i); % (0 -> number of states
 left in trace)
                % Reward function:
                state_return = @(k)
 (gamma.^k).*trace_rewards(i:length(trace_rewards));
                state_return = sum(state_return(k));
```

```
            else
                continue
            end
            % Store that current state has been visited:
            first_visit(trace(i)) = 0;
            % Save state's return in list of state returns
            trace_returns{trace(i)} = state_return;
        end
end

% Function that estimates value function given a list of discounted
 returns
% for every state.cReturns have previously been observed in a set of
% traces.
function V_MC = MC_Value_function(returns)
    % Arbitrary initial value function:
    V_MC = zeros(1,n);
    % Estimate value function:
    % Iterate over cells in returns
    for i = 1:length(returns)
        % Skip terminal states:
        if Absorbing(i)
            continue
        else
            if isempty(returns{i})
                 V_MC(i) = 0;
            else
                V_MC(i) = mean(returns{i});
            end
        end
    end
end

% Function for epsilon-soft algorithm for on-policy MC Control.
% Takes information about MDP, a policy, epsilon and gamma. Returns an
% improved policy, based on MDP episodes it generates and the updated
 list of returns.
function [greedy_policy, states, rewards, actions, total_s_a_returns]
 = MC_control(n, a, T, R, Absorbing, Policy, gamma, epsilon,
 total_s_a_returns)
    % Variable to let the functions know if we want to print:
    print = 0;
    % Generate trace using current policy:
    [states, rewards, actions] = generate_trace(n, a, T, R, Absorbing,
 Policy, print);
    % Obtain state-action returns for current trace
    state_action_returns = greedy_policy_returns(states, rewards,
 actions, a, n, gamma);
    % Append state-action returns to total returns:
    % Iterate over all states:
    for i = 1:n
        % And every action of every state:
        for j = 1:a
            % Appending:
```

```
            total_s_a_returns{i,j} = [total_s_a_returns{i,j},
 state_action_returns{i,j}];
        end
    end
    % Obtain new Q function:
    Q_MC = Q_MC_function(total_s_a_returns);
    % Update policy:
    for state = 1:length(Q_MC)
        [value, optimal_action] = max(Q_MC(state,:));
        for action = 1:a
            if action == optimal_action
                greedy_policy(state,action) = 1 - epsilon + (epsilon/
a);
            else
                greedy_policy(state,action) = epsilon/a;
            end
        end
    end
end

% Function for state-action returns. Takes a list of states, rewards
 and
% actions from a trace. Also takes number of states and actions in
 MDP,
% as well as information about terminal states and dicount factor.
 Returns
% the discounted reward of every state-action pair.
function state_action_returns = greedy_policy_returns(states, rewards,
 actions, a, n, gamma)
        % Matrix for keeping track of visited state-action pairs in
 trace,
        % rows represent states and columns represent actions
 (1,2,3,4).
        first_visit = ones(n,a);
        % Cell matrix for returns:
        state_action_returns = cell(n,a);

        % Iterate over every state-action pair in the trace:
        for i = 1:length(actions)
            % Skip terminal states:
            if Absorbing(states(i))
                continue
            end
         % Only compute return if state-action pair hasn't been
 visited:
            if first_visit(states(i),actions(i))
                % Declare variable for discount:
                k = 0:(length(actions)-i); % (0 -> number of actions
 left in trace)
                % Reward function:
                state_action_return = @(k)
 (gamma.^k).*rewards(i:length(rewards));
                state_action_return = sum(state_action_return(k));
            else
```

```
                continue
            end
            % Store that current state-action pair has been visited:
            first_visit(states(i),actions(i)) = 0;
            % Store return in cell matrix
            state_action_returns{states(i),actions(i)} =
 state_action_return;
        end
end

% Function that takes a cell matrix of returns obtained from an
 unknown
% number of traces and returns the Q function as the average return of
% every state-action pair.
function Q_MC = Q_MC_function(total_s_a_returns)
    % Arbitrary initial Q function:
    Q_MC = zeros(n ,a);
    % Estimate value function:
    % Iterate over cells in returns
    dim_returns = size(total_s_a_returns);
    % Iterate over all states
    for i = 1:(dim_returns(1))
        % Skip terminal states:
        if Absorbing(i)
            continue
        else
        % Iterate over all actions of current state
        for j = 1:(dim_returns(2))
            if isempty(total_s_a_returns{i,j})
                 Q_MC(i,j) = 0;
            else
                Q_MC(i,j) = mean(total_s_a_returns{i,j});
            end
        end
        end
    end
end

end
```

Published with MATLAB® R2018b

Table of Contents


```
% Coursework in Machine Learning and Neural Computation
% Jonas Tjomsland, CID = 01570830
% To make it easier for the reader to understand I have tried to use
% similar set up as Dr. A. Aldo Faisal used for the first lab.

clc
clear all
close all
RunCoursework();

function RunCoursework()
```

Question 1

```
%Calculating persoonal p and personal gamma:
p = 0.5 + 0.5*(3/10);
gamma = 0.2 + 0.5*(0/10);


% Get system parameters from given grid world function
[NumStates, NumActions, TransitionMatrix, ...
 RewardMatrix, StateNames, ActionNames, AbsorbingStates] ...
 = PersonalisedGridWorld(p);

% Simplifying names:
n = NumStates;
a = NumActions;
T = TransitionMatrix;
R = RewardMatrix;
S = StateNames;
A = ActionNames;
Absorbing = AbsorbingStates;

% Creating policy matrix where the rows represent states and the
 columns
% possible actions: S1: N, E, S, W (14x4) matrix.
% Unbiased policy means equal probability of all actions.(1/4 in this
 case)
Policy = 1/4*ones(14,4);
% Chooosing tolerance for policy evaluation:
tol = 0.01;
```

Question 2

```
% Calling policy evaluation function:
V = policy_evaluation(n, a, T, R, Absorbing, Policy, tol, gamma);
disp("Value function: ")
disp(" ")
% Calling print function for V
format short
print_V_table(V)
```

```
Value function:

     State        S1        S2     S3       S4          S5          S6
     S7          S8          S9         S10         S11         S12
 S13         S14

   _______    ________    __     __    _______     _______     _______
   _______    _______    ______    _______     _______     _______
_______     _______

   "Value"    -0.90098    0      0      -3.6942     -1.2203     -1.0406
   -3.5546    -1.5003    -1.248    -1.2635     -1.2495     -1.2496
-1.2496    -1.2503
```

Question 3

```
% a) Likelihood

% Sequence vectors:
seq1 = [14, 10, 8, 4, 3];
seq2 = [11, 9, 5, 6, 6, 2];
seq3 = [12, 11, 11, 9, 5, 9,  5, 1, 2];

% Calling likelihood function:
likelihood1 = likelihood(seq1, T, Policy, a);
likelihood2 = likelihood(seq2, T, Policy, a);
likelihood3 = likelihood(seq3, T, Policy, a);

% b) Optimizing policy for likelihood

%Calling function for policy optimization:
optimal_policy = unbiased_policy(seq1, T, Policy, a);
optimal_policy = unbiased_policy(seq2, T, optimal_policy, a);
optimal_policy = unbiased_policy(seq3, T, optimal_policy, a);

% New likelihoods
likelihood1_improved = likelihood(seq1, T, optimal_policy, a);
likelihood2_improved = likelihood(seq2, T, optimal_policy, a);
likelihood3_improved = likelihood(seq3, T, optimal_policy, a);


% Print as table:
```

```
likelihoods = table(likelihood1, likelihood1_improved, likelihood2...
                    ,likelihood2_improved, likelihood3,
 likelihood3_improved);

disp("Likelihoods before and after policy optimisation: ")
disp(" ")
disp(likelihoods)
```

```
Likelihoods before and after policy optimisation:

    likelihood1    likelihood1_improved    likelihood2
 likelihood2_improved    likelihood3    likelihood3_improved
    ___________    ____________________    ___________

 ____________________    ___________    ____________________


    0.00097656           0.044627           0.00024414
 0.0021026          7.6294e-06         0.00015546
```

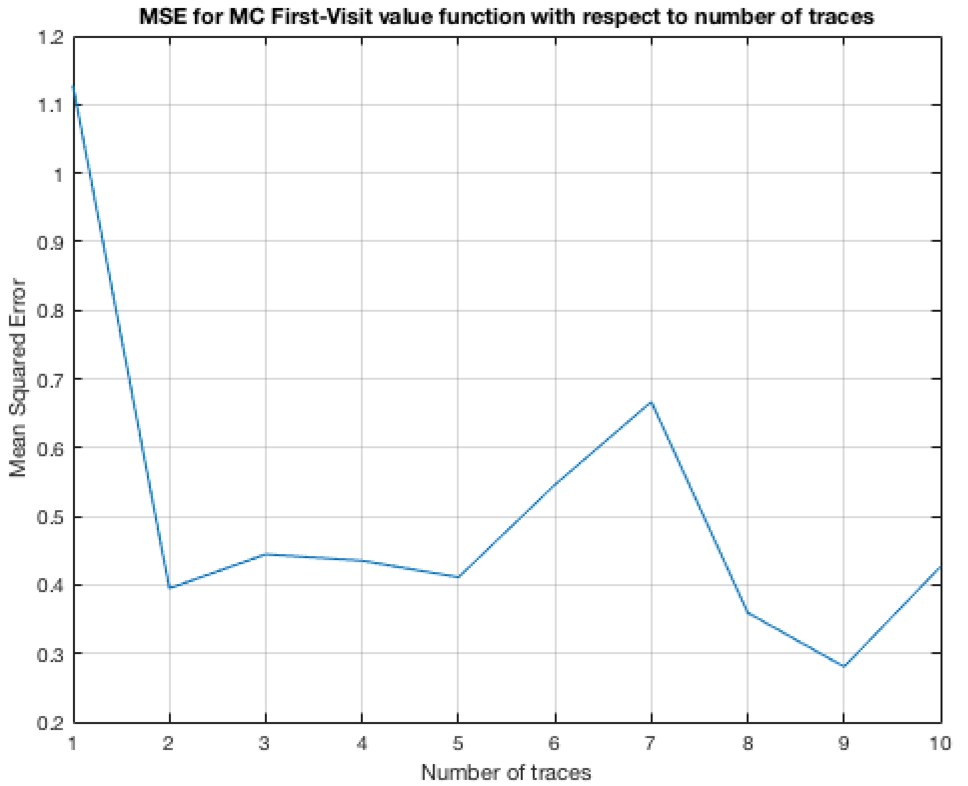
Question 4

```
% a)
% Generate trace with unbiased policy
% Remove comments here and in trace function to display:
disp("Traces with unbiased policy:")
disp(" ")
% Variable to let the functions know if we want to print:
print = 1;
% Number of traces:
n_traces = 10;
% Generate traces, using a nested function.
[traces, all_rewards, all_actions] = generate_traces(n, a, T, R,
 Absorbing, Policy, n_traces, print);
disp(" ")

% b)

% Generate the returns for every state from a given set of traces and
% and corresponding rewards.
returns = MC_policy_returns(n, traces, all_rewards, Absorbing, gamma);

disp("Value function estimated with MC-First visit method for 10
 traces:")
disp(" ")
V_MC = MC_Value_function(returns);
format short
print_V_table(V_MC)

% c)
% I use Mean Squared Error as measure of similarity between V and V_MC
MSE = [];
% Variable to let the functions know if we want to print traces:
print = 0;
```

```
% Compute the distance for 1 to 10 traces and plot the result.
 Essentially
% repeating the steps in b) but compute the distance every step.
for n_traces = 1:10
    [traces, all_rewards] = generate_traces(n, a, T, R, Absorbing,
 Policy, n_traces,print);
    returns = MC_policy_returns(n, traces, all_rewards, Absorbing,
 gamma);
    V_MC = MC_Value_function(returns);
    MSE = [MSE, mean(sqrt((V-V_MC).^2))];
end

%To see plot for 1 to 10 traces, remove comments below:
n_traces = 1:10;
figure
plot(n_traces,MSE)
grid on
xlabel("Number of traces")
ylabel("Mean Squared Error")
title("MSE for MC First-Visit value function with respect to number of
 traces")
```

```
Traces with unbiased policy:

1:
 S11,N,-1,S9,S,-1,S9,S,-1,S11,E,-1,S12,E,-1,S12,E,-1,S13,W,-1,S12,N,-1,S12,S,-1,S1
2:
 S11,N,-1,S11,W,-1,S11,N,-1,S12,E,-1,S12,S,-1,S11,N,-1,S11,S,-1,S11,S,-1,S11,W,-1,
3:
 S13,N,-1,S13,N,-1,S12,W,-1,S11,S,-1,S12,E,-1,S13,W,-1,S12,S,-1,S11,W,-1,S11,E,-1,
4:
 S12,W,-1,S12,S,-1,S12,N,-1,S11,W,-1,S11,S,-1,S11,N,-1,S9,W,-1,S5,E,-1,S6,N,-1,S5,
5:
 S11,W,-1,S11,N,-1,S12,N,-1,S12,W,-1,S11,W,-1,S11,S,-1,S11,N,-1,S9,N,-1,S5,S,-1,S9
6:
 S14,N,-1,S10,S,-1,S14,E,-1,S14,E,-1,S14,S,-1,S14,S,-1,S13,E,-1,S13,S,-1,S13,W,-1,
7:
 S14,E,-1,S10,E,-1,S10,W,-1,S10,S,-1,S14,N,-1,S10,N,-1,S8,S,-1,S10,W,-1,S10,N,-1,S
8:
 S11,S,-1,S12,W,-1,S11,W,-1,S11,S,-1,S11,E,-1,S12,N,-1,S13,N,-1,S13,W,-1,S12,S,-1,
9:
 S11,S,-1,S11,W,-1,S11,W,-1,S11,S,-1,S11,N,-1,S9,S,-1,S11,N,-1,S11,S,-1,S11,W,-1,S
10:
 S12,N,-1,S12,N,-1,S12,N,-1,S12,N,-1,S11,S,-1,S12,W,-1,S11,E,-1,S12,S,-1,S12,N,-1,

Value function estimated with MC-First visit method for 10 traces:

     State      S1      S2     S3      S4          S5          S6
 S7         S8          S9          S10        S11       S12       S13
  S14
    _______    _____    __    __    _______    ______    ________
 _____    _______    _______    _______    _____    _____    _____
 _______
```

```
  "Value"    -0.81    0     0     -2.3006    -1.158    -0.41667
-1.25    -1.2668    -1.2435    -1.2506    -1.25    -1.25    -1.25
-1.2501
```



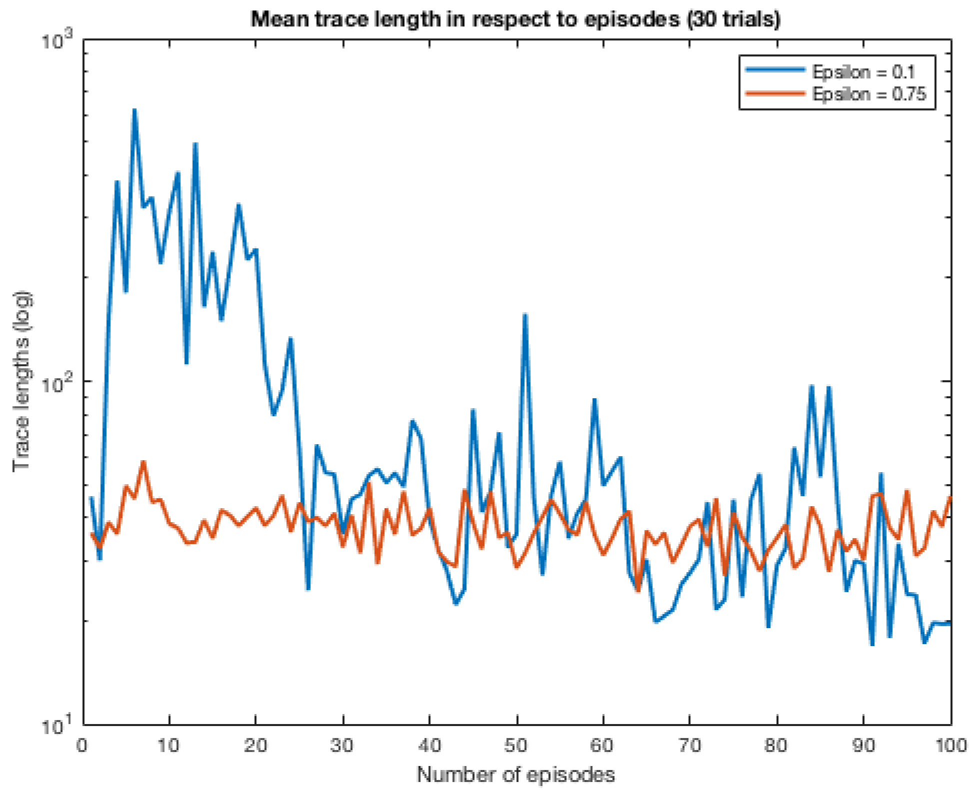
Question 5

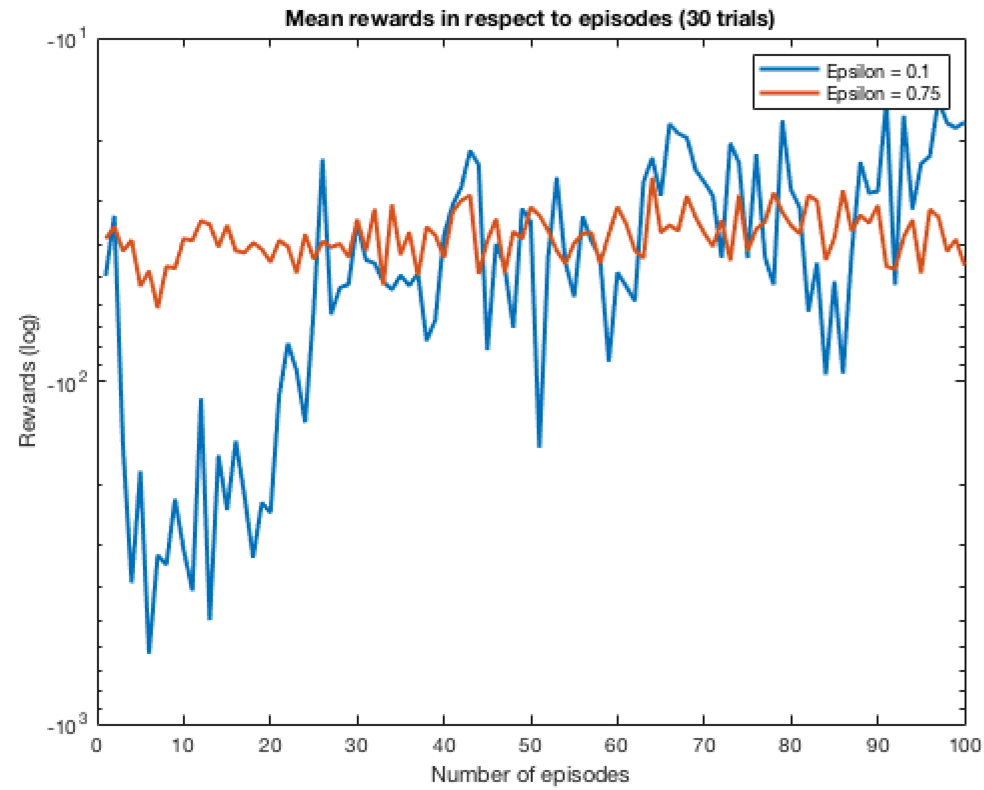
```
% Implementing epsilon-greedy policy for First-Fisit MC Control:
% Initialize epsilon 1 & 2
epsilon_1 = 0.1;
epsilon_2 = 0.75;
epsilon = [epsilon_1, epsilon_2];

% Max number of trials and episodes:
max_trials = 30;
max_episodes = 100;

% Create cell array to store trace length and rewards for every trial.
% Every cell represents 50 trials with a given number of episodes,
 e.g.
% cell ten in "trace_lengths" contains an array of 50 elements where
% every element is the trace length from one trial with ten episodes.
% Double the numbers of cells and place the results for epsilon 2
 after
% those from epsilon 1.
trace_lengths = cell(1, 2*max_episodes);
```

```
trials_rewards = cell(1, 2*max_episodes);

% For both epsilon:
for epsilon = epsilon
    % Do 20 trials for every number of episodes:
    for trials = 1:max_trials
        % Initiate policy as unbiased:
        greedy_policy = Policy;
        % Initiate empty cell matrix for state-action returns:
        total_s_a_returns = cell(n,a);
        % Let the agen operate for 1 to 200 episodes:
        for episodes = 1:max_episodes
            [greedy_policy, states, rewards, actions,
 total_s_a_returns] = MC_control(n, a, T, R, Absorbing, greedy_policy,
 gamma, epsilon, total_s_a_returns);
            % Append trace length and sum of rewards to cell array,
 place
            % results for epsilon 2 at cell 201-400:
            if epsilon == epsilon_2
                trace_lengths{max_episodes+episodes} =
 [trace_lengths{max_episodes+episodes}, length(states)];
                trials_rewards{max_episodes+episodes} =
 [trials_rewards{max_episodes+episodes}, sum(rewards)];
            else
                trace_lengths{episodes} = [trace_lengths{episodes},
 length(states)];
                trials_rewards{episodes} = [trials_rewards{episodes},
 sum(rewards)];
            end
        end
    end
end

% Array for averaged trace lengths and rewards as well as standard
 deviation:
mean_trace_lengths_e1 = [];
mean_rewards_e1 = [];
std_trace_lengths_e1 = [];
std_rewards_e1 = [];

mean_trace_lengths_e2 = [];
mean_rewards_e2 = [];
std_trace_lengths_e2 = [];
std_rewards_e2 = [];

% Compute mean and std for trace lengths and rewards for both espilon:
for i = 1:length(trace_lengths)
    % For epsilon 2:
    if i > length(trace_lengths)/2
        mean_trace_lengths_e2 = [mean_trace_lengths_e2,
 mean(trace_lengths{i})];
        mean_rewards_e2 = [ mean_rewards_e2, mean(trials_rewards{i})];
        std_trace_lengths_e2 = [std_trace_lengths_e2,
 std(trace_lengths{i})];
```

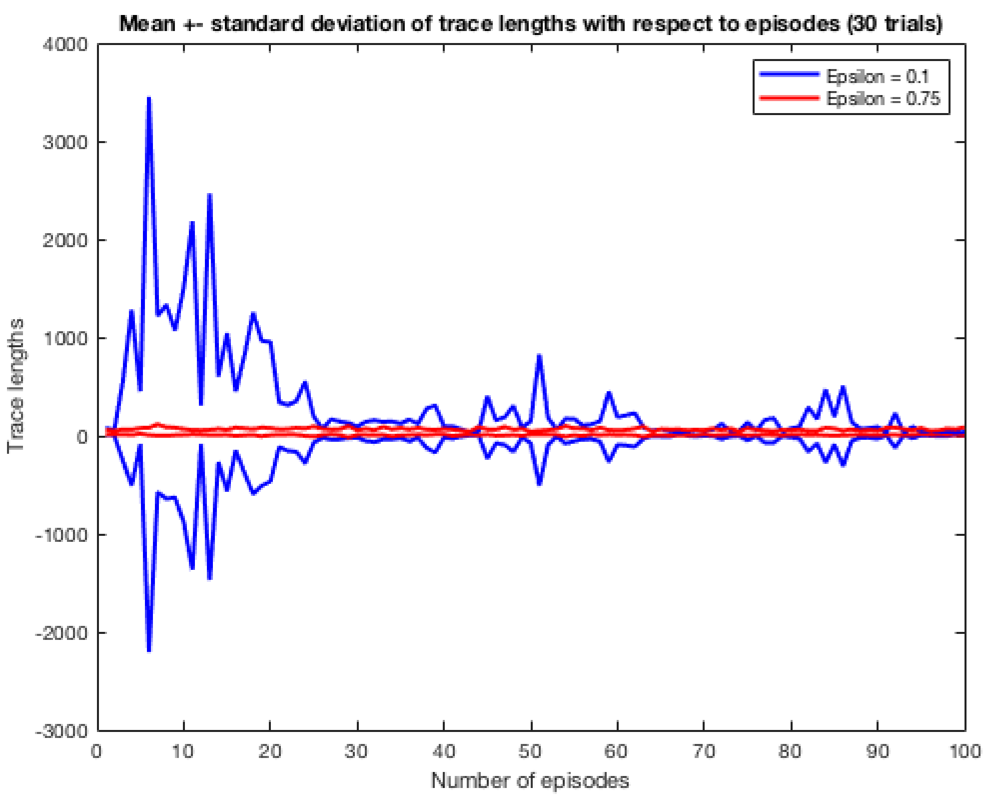
```
        std_rewards_e2 = [std_rewards_e2, std(trials_rewards{i})];
    % For epsilon 1:
    else
        mean_trace_lengths_e1 = [mean_trace_lengths_e1,
 mean(trace_lengths{i})];
        mean_rewards_e1 = [mean_rewards_e1, mean(trials_rewards{i})];
        std_trace_lengths_e1 = [std_trace_lengths_e1,
 std(trace_lengths{i})];
        std_rewards_e1 = [std_rewards_e1, std(trials_rewards{i})];
    end
end

% Plotting results. First only the mean trace lengths adn rewards are
% plotted against episodes. Secondly mean plus/minus std are plotted.
 All
% for both epsilon 1 and epsilon 2.

% Mean trace length against episodes:
figure
semilogy(1:max_episodes,mean_trace_lengths_e1,'LineWidth',2)
hold on
semilogy(1:max_episodes,mean_trace_lengths_e2, 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Trace lengths (log)")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean trace length in respect to episodes (30 trials)")

% Mean rewards against episodes:
figure
semilogy(1:max_episodes,mean_rewards_e1,'LineWidth',2)
hold on
semilogy(1:max_episodes,mean_rewards_e2, 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Rewards (log)")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean rewards in respect to episodes (30 trials)")

% Mean trace lengths plus/minus standard deviation against episodes
figure
plot(1:max_episodes,mean_trace_lengths_e1-
std_trace_lengths_e1, 'b','LineWidth',2)
hold on
plot(1:max_episodes,mean_trace_lengths_e2-
std_trace_lengths_e2, 'r', 'LineWidth',2)
plot(1:max_episodes,mean_trace_lengths_e1+std_trace_lengths_e1,'b', 'LineWidth',2)
plot(1:max_episodes,mean_trace_lengths_e2+std_trace_lengths_e2, 'r', 'LineWidth',2
xlabel("Number of episodes")
ylabel("Trace lengths")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean +- standard deviation of trace lengths with respect to
 episodes (30 trials)")

% Mean rewards plus/minus standard deviation against episodes
figure
```

```
plot(1:max_episodes,mean_rewards_e1-std_rewards_e1, 'b','LineWidth',2)
hold on
plot(1:max_episodes,mean_rewards_e2-
std_rewards_e2, 'r', 'LineWidth',2)
plot(1:max_episodes,mean_rewards_e1+std_rewards_e1,'b', 'LineWidth',2)
plot(1:max_episodes,mean_rewards_e2+std_rewards_e2, 'r', 'LineWidth',2)
xlabel("Number of episodes")
ylabel("Rewards")
legend("Epsilon = 0.1","Epsilon = 0.75")
title("Mean +- standard deviation of rewards with respect to episodes
 (30 trials)")
```

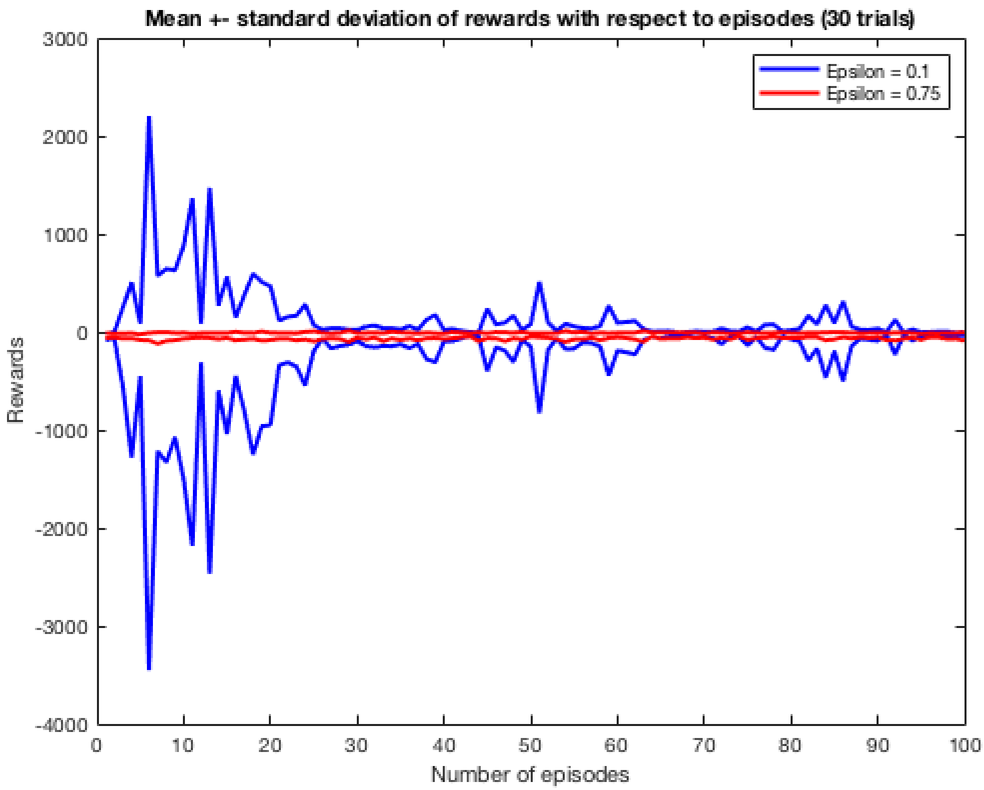




Mean rewards in respect to episodes (30 trials)
Epsilon = 0.1
Epsilon = 0.75
Rewards (log)
Number of episodes



Mean +- standard deviation of trace lengths with respect to episodes (30 trials)
Epsilon = 0.1
Epsilon = 0.75
Trace lengths
Number of episodes



Functions:

```
% Creating policy evaluation function
% Takes number of states, transition matrix, reward matrix, list of...
% absorbing states, policy and iteration tolerance as input.
% Returns value function V.
function V = policy_evaluation(n, a, T, R, Absorbing, Policy, tol,
 gamma)

    V = zeros(1,n);     % Optimal value function vector
    Vnew = V;        % Value function vector for step i+1.
    delta = 2*tol;   % Used to measure difference in V & Vnew.

    % Value iteration:
    while delta > tol
        for current_state = 1:n          % Iterating over all states
            if Absorbing(current_state) % Skipping terminal states
                continue
            end
            current_V = 0;               % Current value of current
 state
            for action = 1:a             % Iterating over all actions
                current_Q = 0;           % Current state action value
                for next_state = 1:n     % Iterating over all states,
 the transition matrix will cancel oout states out of reach.
```

```
                    current_Q = current_Q + T(next_state,
 current_state, action)*(R(next_state, current_state, action) +
 gamma*V(next_state));
                end
                current_V = current_V + Policy(current_state,
 action)*current_Q;
            end
            Vnew(current_state) = current_V;  % Storing new value for
 current state
        end
    diff = abs(Vnew - V);  % Calculates changes in value function
 vector
    delta = max(diff);  % Compute new delta
    V = Vnew;                % Update value function
    end
end

% Function for printing value function as table.
% Takes value function vector as input and prints the table.
function print_V_table(V)

    % Creating table:
    State = "Value";
    S1 = V(1);
    S2 = V(2);
    S3 = V(3);
    S4 = V(4);
    S5 = V(5);
    S6 = V(6);
    S7 = V(7);
    S8 = V(8);
    S9 = V(9);
    S10 = V(10);
    S11 = V(11);
    S12 = V(12);
    S13 = V(13);
    S14 = V(14);
    Table =
 table(State,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14);
    disp(Table)
end

% Function calculating likelihood of given sequence occuring.
% Takes sequence, transition matrix, policy and number of actions as
 input.
% Returns likelihood.
function likelihood = likelihood(seq, T, Policy, a)

    % Initializing total probability for iteration
    total_p = 1;
    % Iterating over the states of the sequence, stopping at last
 state
    for i = 1:length(seq)-1
        current_state = seq(i);
```

```
        next_state = seq(i+1);
        % Initializing local probability for iteration
        local_p = 0;
        % Iterating over every action
        for action = 1:a
            local_p = local_p + T(next_state, current_state,
 action)*Policy(current_state, action);
        end
        total_p = total_p*local_p;
    end
    likelihood = total_p*(1/4); % Multiply by 1/4 because that is the
 probability of starting in state 14
end

% Function optimizing the policy to increase the likelihood of
 observing a
% given sequence.
% Takes sequence, transition matrix, policy and number of actions as
 input.
% Returns optimal_policy.
function optimal_policy = unbiased_policy(seq, T, Policy, a)

    % Optimal policy matrix, just intend to change the policy of the
 states occuring in the sequence.
    optimal_policy = Policy;

    % Iterating over the states of the sequence, stopping at last
 state
    for i = 1:length(seq)
        current_state = seq(i);
        % Setting no policy for terminal states
        if i == length(seq)
            optimal_policy(current_state, :) = 0;
            continue
        end
        next_state = seq(i+1);
        % Initializing local probability for iteration
        local_p = [];

        % Handle state 6 differently
        if current_state == 6 | current_state == 5
            optimal_action = 2;   % Picked East
            for action = 1:a
                if action == optimal_action
                    optimal_policy(current_state, action) = 1;
                else
                    optimal_policy(current_state, action) = 0;
                end
            end
            continue    % Skipping to next state in the sequence
        end

        % Iterating over every action
        for action = 1:a
```

```
            local_p = [local_p, T(next_state, current_state,
 action)*Policy(current_state, action)];
        end

        [value, optimal_action] = max(local_p);
        for action = 1:a
            if action == optimal_action
                optimal_policy(current_state, action) = 1;
            else
                optimal_policy(current_state, action) = 0;
            end
        end
    end
end

% Function that generates a given number of traces, it takes
 desired...
% number of traces and just calls trace generating function that
 amount of times.
% Returns a cell array of traces and rewards.
function [traces, all_rewards, all_actions] = generate_traces(n, a, T,
 R, Absorbing, Policy, n_traces, print)
% Cell array for storing trace states and rewards, every cell
 represents
% a trace and contains a list of states and rewards.
traces = cell(1,n_traces);
all_rewards = cell(1,n_traces);
all_actions = cell(1,n_traces);
% Create n_traces traces:
for i = 1:n_traces
    if print
        fprintf('%d%s', i, ": ")
    end
    [states, rewards, actions] = generate_trace(n, a, T, R, Absorbing,
 Policy, print);
    traces{i} = states;
    all_rewards{i} = rewards;
    all_actions{i} = actions;
end
end

% Function for trace generation for given MDP
% Takes MDP information like number of states, actions, transistion
 matrix...
% reward matrix and absorbing states. In addition it  takes a policy.
% Returns, length of trace, the states in the trace as well as the
 rewards
function [states, rewards, actions] = generate_trace(n, a, T, R,
 Absorbing, Policy, print)
    % Start out by defining starting state and set that as current
 state:
    starting_states = [11, 12, 13 ,14];
    staring_states_p = (1/4)*ones(1,length(starting_states));
    current_state = randsrc(1, 1, [starting_states;staring_states_p]);
```

```
    % Array of the states visited, actions taken and rewards in the
trace
    states = [current_state];
    action_strs = [];
    actions = [];
    rewards = [];
    % Initialize empyt trace array:
    trace = [];
    % Iterate until terminal state is reached:
    while 1
        % Decide action: (N=1, E=2, S=3, W=4)
        action = randsrc(1,1,[1:a; Policy(current_state,:)]);
        % Iterates over all states to decide next state.
        % Store prob for ending in state s in a array.
        next_state_p = [];
        for next_state = 1:n
            next_state_p = [ next_state_p,  T(next_state,
current_state, action)];
        end
        % Choose next state:
        next_state = randsrc(1,1,[1:n; next_state_p]);
        % Define action as string:
        if action == 1
            action_str = "N";
        elseif action == 2
            action_str = "E";
        elseif action == 3
            action_str = "S";
        else
            action_str = "W";
        end
        reward = R(next_state,current_state,action);
        current_state = next_state;
        states = [states, current_state];  % Append to list of states
        action_strs = [action_strs, action_str];  % Append to list of
action strings
        actions = [actions, action];
        rewards = [rewards, reward];
        if Absorbing(next_state)
            break
        end
    end
    if print
        %Printing trace:
        for state = 1:length(actions)
            if rewards(state) == 0 | rewards(state) == -10
                fprintf('S%d,%s,%d', states(state),
action_strs(state),rewards(state))
            else
                fprintf('S%d,%s,%d,', states(state),
action_strs(state),rewards(state))
            end
        end
        disp(" ")
```

```
    end
end

% Function that generate list of returns for every state for a
 given...
% number of traces. Takes a cell array of traces and a cell array of
% rewards. Calls "trace_returns" function for every trace.
function returns = MC_policy_returns(n, traces, all_rewards,
 Absorbing, gamma)

    % Empty returns cell array:
    returns = cell(1,n);
    % Iterate over all traces:
    for i = 1:length(traces)
        % Call function for Monte Carlo Policy Evaluation:
        trace_returns = MC_policy_evaluation(n, traces{i},
 all_rewards{i}, Absorbing, gamma);
        % Append new returns to cell array:
        for j = 1:length(trace_returns)
            returns{j} = [returns{j} trace_returns{j}];
        end
    end
end

% Function which estimates value function based on observed episodes
% Takes number of states in MDP, an observed trace of states, the
 rewards
% obtained in that trace, information about terminal states and
 discount
% factor gamma.
function trace_returns = MC_policy_evaluation(n, trace, trace_rewards,
 Absorbing, gamma)

        % Array for keeping track of visited states in trace:
        first_visit = ones(1,n);
        % Cell array for returns:
        trace_returns = cell(1,n);

        % Iterate over every state in trace:
        for i = 1:length(trace)
            % Skip terminal states:
            if Absorbing(trace(i))
                continue
            end
            % Only compute return if state hasn't been visited (First
 visit):
            if first_visit(trace(i))
                % Declare variable for discount:
                k = 0:(length(trace)-1-i); % (0 -> number of states
 left in trace)
                % Reward function:
                state_return = @(k)
 (gamma.^k).*trace_rewards(i:length(trace_rewards));
                state_return = sum(state_return(k));
```

```
            else
                continue
            end
            % Store that current state has been visited:
            first_visit(trace(i)) = 0;
            % Save state's return in list of state returns
            trace_returns{trace(i)} = state_return;
        end
end

% Function that estimates value function given a list of discounted
 returns
% for every state.cReturns have previously been observed in a set of
% traces.
function V_MC = MC_Value_function(returns)
    % Arbitrary initial value function:
    V_MC = zeros(1,n);
    % Estimate value function:
    % Iterate over cells in returns
    for i = 1:length(returns)
        % Skip terminal states:
        if Absorbing(i)
            continue
        else
            if isempty(returns{i})
                 V_MC(i) = 0;
            else
                V_MC(i) = mean(returns{i});
            end
        end
    end
end

% Function for epsilon-soft algorithm for on-policy MC Control.
% Takes information about MDP, a policy, epsilon and gamma. Returns an
% improved policy, based on MDP episodes it generates and the updated
 list of returns.
function [greedy_policy, states, rewards, actions, total_s_a_returns]
 = MC_control(n, a, T, R, Absorbing, Policy, gamma, epsilon,
 total_s_a_returns)
    % Variable to let the functions know if we want to print:
    print = 0;
    % Generate trace using current policy:
    [states, rewards, actions] = generate_trace(n, a, T, R, Absorbing,
 Policy, print);
    % Obtain state-action returns for current trace
    state_action_returns = greedy_policy_returns(states, rewards,
 actions, a, n, gamma);
    % Append state-action returns to total returns:
    % Iterate over all states:
    for i = 1:n
        % And every action of every state:
        for j = 1:a
            % Appending:
```

```
            total_s_a_returns{i,j} = [total_s_a_returns{i,j},
 state_action_returns{i,j}];
        end
    end
    % Obtain new Q function:
    Q_MC = Q_MC_function(total_s_a_returns);
    % Update policy:
    for state = 1:length(Q_MC)
        [value, optimal_action] = max(Q_MC(state,:));
        for action = 1:a
            if action == optimal_action
                greedy_policy(state,action) = 1 - epsilon + (epsilon/
a);
            else
                greedy_policy(state,action) = epsilon/a;
            end
        end
    end
end

% Function for state-action returns. Takes a list of states, rewards
 and
% actions from a trace. Also takes number of states and actions in
 MDP,
% as well as information about terminal states and dicount factor.
 Returns
% the discounted reward of every state-action pair.
function state_action_returns = greedy_policy_returns(states, rewards,
 actions, a, n, gamma)
        % Matrix for keeping track of visited state-action pairs in
 trace,
        % rows represent states and columns represent actions
 (1,2,3,4).
        first_visit = ones(n,a);
        % Cell matrix for returns:
        state_action_returns = cell(n,a);

        % Iterate over every state-action pair in the trace:
        for i = 1:length(actions)
            % Skip terminal states:
            if Absorbing(states(i))
                continue
            end
         % Only compute return if state-action pair hasn't been
 visited:
            if first_visit(states(i),actions(i))
                % Declare variable for discount:
                k = 0:(length(actions)-i); % (0 -> number of actions
 left in trace)
                % Reward function:
                state_action_return = @(k)
 (gamma.^k).*rewards(i:length(rewards));
                state_action_return = sum(state_action_return(k));
            else
```

```
                continue
            end
            % Store that current state-action pair has been visited:
            first_visit(states(i),actions(i)) = 0;
            % Store return in cell matrix
            state_action_returns{states(i),actions(i)} =
 state_action_return;
        end
end

% Function that takes a cell matrix of returns obtained from an
 unknown
% number of traces and returns the Q function as the average return of
% every state-action pair.
function Q_MC = Q_MC_function(total_s_a_returns)
    % Arbitrary initial Q function:
    Q_MC = zeros(n ,a);
    % Estimate value function:
    % Iterate over cells in returns
    dim_returns = size(total_s_a_returns);
    % Iterate over all states
    for i = 1:(dim_returns(1))
        % Skip terminal states:
        if Absorbing(i)
            continue
        else
        % Iterate over all actions of current state
        for j = 1:(dim_returns(2))
            if isempty(total_s_a_returns{i,j})
                 Q_MC(i,j) = 0;
            else
                Q_MC(i,j) = mean(total_s_a_returns{i,j});
            end
        end
        end
    end
end

end
```

Published with MATLAB® R2018b