Question 1

My CID is 00821349 therefore I have $X = 4$ and $Y = 9$.
$p = 0.6 + 0.4 \times 4 \times \frac{1}{10} = 0.76$
$\gamma = 0.1 + 0.9 \times 9 \times \frac{1}{10} = 0.91$

Question 2

By initializing the transition probabilities and number of actions as well as the reward for each state, an unbiased policy can be obtained from these initial conditions. The MATLAB code shown in Appendix B firstly defined the state transition probability matrix and reward matrix for this staring climbing MDP. An unbiased policy was then obtained and the value function $V^{\pi^u}(s)$ was also evaluated for each state. The result is shown in Table 1. The values are symmetric with respect to the starting state 4 because for an unbiased policy and only two actions (left and right), the probability of going to either direction are equally likely of 50%.

State	$V^{\pi^u}(s)$
1	0
2	-6.6487
3	-2.9411
4	0
5	2.9411
6	6.6487
7	0

Table 1: Value function for S_1 to S_7

Question 3

3. a

From Question 1 and 2 the probabilities of move or stay and the probability of an unbiased policy to choose move to either direction (left or right) were shown as following:
p(stay)=0.24
p(move)=0.76
p(Left_{π^u})=0.5
p(Right_{π^u})=0.5, as there are only two actions (right or left) and the policy is unbiased, the probability of moving in either direction is 50% for each state.

The three state transitions observed from the above MDP are: {s4,s5,s6,s7}, {s4,s5,s6,s7}, {s4,s3,s4,s5,s6,s7} respectively. To identify the probability of each sequence, following statements were written up:
Trace 1: $s4 \to s5 \to s6 \to s7$
$p(T1) = 0.5 \times 0.76 \times 0.5 \times 0.76 \times 0.5 \times 0.76 = 0.055$
Trace 2: $s4 \to s5 \to s6 \to s7$
$p(T2) = 0.5 \times 0.76 \times 0.5 \times 0.76 \times 0.5 \times 0.76 = 0.055$
Trace 3: $s4 \to s3 \to s4 \to s5 \to s6 \to s7$
$p(T3) = 0.5 \times 0.76 \times 0.5 \times 0.76 \times 0.5 \times 0.76 \times 0.5 \times 0.76 \times 0.5 \times 0.76 = 0.0079$

Likelihood: L(π_u)=$p(T1) \times p(T2) \times p(T3) = 0.055 \times 0.055 \times 0.0079 = 2.39 \times 10^{-9}$

3. b

The unbiased policy is shown in Table 2 below. State 1 and State 7 are absorbing state which means once access to these two states there is no chance to move out, therefore the policy for them are 0. The rest of states all have unbiased probability of moving to the right or left. To find a policy with higher likelihood than the unbiased policy π^u does for the observed three sequences, only policies of visited states need to be altered. As all three sequences have not reached any state of 1 and 2, these two states can be ignored in this case. Except from State 4, at all the other states took an action of moving to the right. The policy for State 3, 5 and 6 to move to the right can be set to 1 in order to get a maximum likelihood. At State 4, it moved to the right and once to the left.

To calculate the best policy for State 4:
$\pi(S_3, R) = \pi(S_5, R) = \pi(S_6, R) = 1$
Assume the policy for State 4 to move to the right $\pi(S_4, R) = k$

$$\begin{aligned} L(\pi_M) &= p(T1) \times p(T2) \times p(T3) \\ &= (kp^3)^2 \times k(1-k) \times p^4 \\ &= k^3(1-k)p^{10} \end{aligned}$$

Write down the first derivative of the likelihood as $\frac{dL}{dk} = 3k^2(1-k) \times -1 \times 0.76^{10}$. Set $\frac{dL}{dk} = 0$ to calculate k when L is maximum.

$$\begin{aligned} 3k^2(1-k) &= 0 \\ k &= \frac{3}{4} \end{aligned}$$

The new π^M which has higher likelihood to observe the three sequences is shown in Table 3.

State	Left	Right
1	0	0
2	0.5	0.5
3	0.5	0.5
4	0.5	0.5
5	0.5	0.5
6	0.5	0.5
7	0	0

Table 2: Unbiased Policy

State	Left	Right
3	0	1
4	0.25	0.75
5	0	1
6	0	1
7	0	0

Table 3: Policy with higher likelihood

4. a

The random traces generated by MATLAB by using Algorithm 1. See full code in Appendix A.

Algorithm 1 Random traces generation

---

for $i = 1 \rightarrow 10$ **do** ▷ number of traces required to generate
Require: Initialise starting state, current and prior state, vector to store trace, reward and path has been visited
  while $current > 1\&\& < 7$ **do** ▷ Not reach the absorbing state
    if $randomNumber > p$ **then** ▷ Obtained a possibility to move right
      Update current and prior state
      Update visited path
      Update reward vector
    else ▷ Else move left
      Update current and prior state
      Update visited path
      Update reward vector
    end if
  end while
end for

One of the example of 10 random traces generated are shown as following:
Trace 1: 's04,R,0,s04,R,-1,s05,L,1,s04,L,1,s03,R,-1,s04,R,0,s04,L,1,s03,R,-1,s04,L,1,s03,L,1,s02,R,-1,s03,L,1,s02,L,-10'
Trace 2: 's04,L,1,s03,L,1,s02,L,-10'
Trace 3: 's04,R,-1,s05,R,-1,s06,R,0,s06,L,1,s05,R,-1,s06,R,10'
Trace 4: 's04,R,-1,s05,R,-1,s06,L,0,s06,R,0,s06,L,0,s06,R,10'
Trace 5: 's04,R,-1,s05,L,1,s04,L,0,s04,R,0,s04,R,-1,s05,R,0,s05,R,-1,s06,R,10'
Trace 6: 's04,R,-1,s05,L,1,s04,L,0,s04,L,1,s03,R,-1,s04,L,1,s03,R,-1,s04,R,-1,s05,L,1,s04,L,1,s03,L,0,s03,L,0,s03,
1,s03,L,1,s02,L,-10'
Trace 7: 's04,R,-1,s05,L,1,s04,L,1,s03,R,-1,s04,R,-1,s05,L,0,s05,R,0,s05,L,1,s04,R,-1,s05,L,1,s04,L,1,s03,L,1,s02,
1,s03,L,1,s02,L,0,s02,R,0,s02,L,-10'
Trace 8: 's04,L,1,s03,R,-1,s04,R,-1,s05,R,-1,s06,R,0,s06,R,10'
Trace 9: 's04,R,-1,s05,R,0,s05,L,1,s04,L,0,s04,L,0,s04,R,-1,s05,R,-1,s06,R,0,s06,L,1,s05,R,-1,s06,L,1,s05,R,-1,s06,R,10'
Trace 10: 's04,L,1,s03,R,-1,s04,L,1,s03,L,1,s02,L,-10'

The following tables show the policy evaluations of unbiased policy generated by dynamic programming and an example policy generated by Monte-Carlo method respectively. From the values in Table 5 the convergence to the expected value can be observed. There was still small difference but as more returns are observed from the new episodes, the average should converge more closely to the expected values.

State	$V^{\pi^u}(s)$
1	0
2	-6.6487
3	-2.9411
4	0
5	2.9411
6	6.6487
7	0

Table 4: Unbiased Policy Evaluation

State	$V^{\pi^u}(s)$
1	0
2	-6.6510
3	-3.0790
4	-1.4023
5	3.9268
6	6.0405
7	0

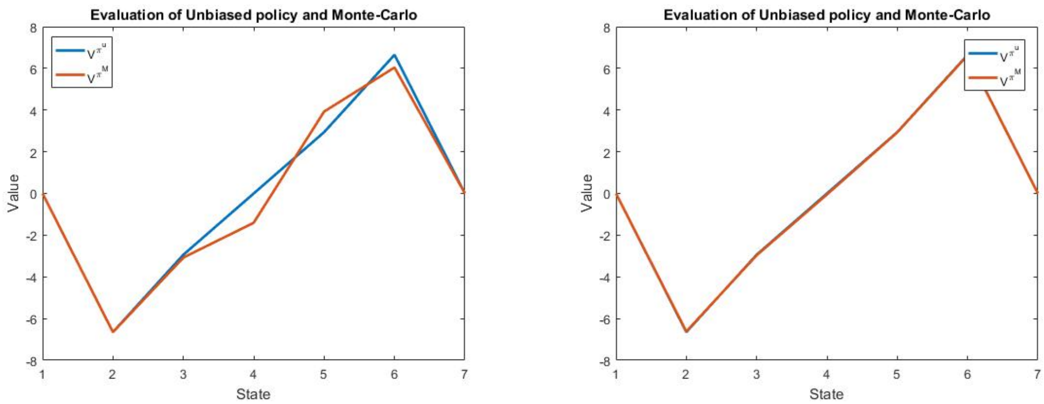
Table 5: Policy Evaluation by Monte-Carlo

Question 5

5. a

V^{π^M} obtained from Question 4.b is evaluated from a Monte-Carlo method, which made an assumption on the next state would be visited according to a probability. Inversely, V^{π^u} obtained from Question 2.a with a known probability distribution on all the possible visited states in the next step. Therefore, V^{π^u} can be regarded as an actual observed data whereas V^{π^M} is the estimated data by a probability. To measure how similar these two sets of values are, mean square error (MSE) could be a reasonable parameter used to evaluate. MSE squares the difference on each data point and accumulate them together to give one single number as the measure.

Figure 1(a) provided an example of graphic comparison of how these two sets of values are similar. From this example it can be observed that V^{π^M} does converges to V^{π^u}. The MSE in this example is 0.4753. This relatively high error could be due to the small number of episodes generated in this case. If 10000 traces were generated and were used to calculate V^{π^M}, the error could be reduce to below 10^{-3}. An 10000 episodes example shown in Figure 1(b) which has $MSE = 5.63 \times 10^{-4}$ looks much more converged than 10 episodes.



(a) 10 Episodes (b) 10k Episodes

Figure 1: Similarity of V^{π^u} and V^{π^M}

5. b

Randomly selecting actions without reference to an estimated probability distribution could lead to give very poor performance. However if a policy is deterministic, some state-action pairs will only stick with one possibility and some of the others will be never visited which is not ideal in terms of finding greedy policy. Therefore it is necessary to implement an ϵ-greedy policy by setting ϵ as the probability for choosing an action randomly, and $1 - \epsilon$ as the probability for choosing greedy action, to help increase the chance of state-action exploration to get possible maximum Q and thus to carry out an effective policy improvement.

At the beginning of learning, choosing a high ϵ to give high randomness in choosing an action would be potentially beneficial to explore the possible maximum Q which is used to improve the policy. During the learning process, it also helps realise that the current policy is bad. However, if the current policy is already good enough to approach a greedy policy, a high ϵ with too much randomness could lead to a worse performance and therefore take a long time to converge.

Appendices

A Main.m

```
%% Question 2
% clear all
mygamma = 0.91;
tol = 0.001;
[S, A, T, R,StateNames, ActionNames, Absorbing] = chain1(); % define transition prob and reward
[UnbiasedPolicy] = GetUnbiasedPolicy(Absorbing, A); % get unbiased policy
[V,i,del] = PolicyEvaluation(UnbiasedPolicy, T, R, Absorbing, mygamma, tol); % value function for unbiased policy

%% Question 3
% a.
p_stay=0.24;
p_move=0.76;
p_lu=0.5;
p_ru=0.5;
% Trace 1: s4 -> s5 -> s6 -> s7
p_t1= p_ru * p_move * p_ru * p_move *p_ru * p_move;
% Trace 2: s4 -> s5 -> s6 -> s7
p_t2= p_ru * p_move * p_ru * p_move *p_ru * p_move;
% Trace 3: s4 -> s3 -> s4 -> s5 -> s6 -> s7
p_t3= p_lu * p_move * p_ru * p_move *p_ru * p_move * p_ru * p_move * p_ru * p_move;
% Likelihood
L_u=p_t1 * p_t2 * p_t3;

% b.


%% Question 4
% a.
trace={};seq={};R={};V_mc={};V_mcall={};
R_mat=zeros(length(trace),7); % first visited return matrix
                                % row: trace number
                                % col: state

for i=1:10
    % initialise every trace
    temp=[4];
    current=4;
    t=[];
    prior=[];
    priorStep=[];
    rew=[];
```

```
    while current>1 && current<7
        if rand(1)>=0.5 % move to right
            if rand(1)>=0.24 %move
                prior=current; % store the last step
                current=current+1; % increment current step
                if current==7
                   t=[t,StateNames(prior,:),',R',',10']; % move to the absorbing state
                   rew=[rew 10];
                else
                   t=[t,StateNames(prior,:),',R',',-1,']; % move to next transition state
                   rew=[rew -1];
                end
            else
                prior=current; % store the last step but not increment current step
                t=[t,StateNames(prior,:),',R',',0,']; % stay
                rew=[rew 0];
            end
        else % move to the left
            if rand(1)>=0.24 % move
                prior=current;
                current=current-1;
                 if current==1
                   t=[t,StateNames(prior,:),',L',',-10'];
                   rew=[rew -10];
                else
                   t=[t,StateNames(prior,:),',L',',1,'];
                   rew=[rew 1];
                end
            else
                prior=current;
                t=[t,StateNames(prior,:),',L',',0,'];
                rew=[rew 0];
            end
        end
        temp=[temp current]; % update all the states visited
        priorStep=[priorStep prior]; % update all the prior states visited
     end
trace{i}=temp; % store the finished trace
seq{i,1}=t; % store the finished path
R{i}=rew; % store the reward for this trace (for monte-carlo policy evaluation aim)

[unitrace, uni_i]=unique(trace{i},'stable'); % get the first visited state and their idx
v_mc=[];
for c=1:length(temp)-1 %
    v_mc(c) = mc(temp,c,mygamma,rew);
end
```

```
% V_mcall{i}=v_mc; % Return for all visited states. A check reference for
% % return of first visted states
V_mc{i}=[v_mc(uni_i(1:end-1)) v_mc(end)]; % return for first visited states




for g=1:length(unitrace)-1 % ignore absorbing state (return for absorbing state
    is 0)
    for m=1:7
            if unitrace(g)==m
                if g<length(unitrace)-1 % return has one less value than trace
                    does
                    R_mat(i,m)=V_mc{i}(g);
                else
                    R_mat(i,m)=V_mc{i}(g);
                end
            end
    end
end

end

avg_R=sum(R_mat,1)./sum(R_mat~=0,1) % calculate the mean for each state over ten
    traces


%% Question 5
avg_r=[0 avg_R(2:end-1) 0]; % reformat the evaluated value from Monte-carlo
MSE=immse(avg_r,V')

plot(V,'LineWidth',2);hold on;plot(avg_r,'LineWidth',2)
title('Evaluation of Unbiased policy and Monte-Carlo')
xlabel('State')
ylabel('Value')
legend('V^{\pi^u}','V^{\pi^M}')
```

B Markov Chain

```
function [S, A, T, R, StateNames, ActionNames, Absorbing] = chain1()

% Number of states
S = 7;
StateNames =  ['s01'; 's02'; 's03'; 's04'; 's05'; 's06'; 's07'];
% States are laid out as follows:
%  index             name
% 1 2 3 4 5 6 7      S1 S2 S3 S4 S5 S6 S7
%    -->


```

```
% Number of actions
A = 2;
ActionNames = ['L'; 'R'];
% Actions are as follows:
%index       name
% 1    -->   L
% 2    -->   R


% Matrix indicating absorbing states
Absorbing = [
%1  2   3   4   5   6   7  STATE
1   0   0   0   0   0   1
];

% load transition
T = transition_matrix();

% load reward matrix
R = reward_matrix(S,A);

%--------------------------------------------------------------------------

% get the transition matrix (defined as local function)
function T = transition_matrix()
%
% 1 2 3 4       S1 S2 S3  S4
% 5 # 6 7   --> S5 #  S6  S7
% 8 9 10 11     S8 S9 S10 S11
%
TR = [
%1    2     3     4     5     6     7   FROM STATE
1     0     0     0     0     0     0  ; % 1 TO STATE
0   0.24    0     0     0     0     0  ; % 2
0   0.76  0.24    0     0     0     0  ; % 3
0     0   0.76   0.24   0     0     0  ; % 4
0     0     0    0.76  0.24   0     0  ; % 5
0     0     0     0    0.76  0.24   0  ; % 6
0     0     0     0     0    0.76   1  ; % 7
];
%
TL = [
%1    2     3     4     5     6     7   FROM STATE
1   0.76    0     0     0     0     0  ; % 1 TO STATE
0   0.24  0.76    0     0     0     0  ; % 2
0     0   0.24   0.76   0     0     0  ; % 3
0     0     0    0.24  0.76   0     0  ; % 4
0     0     0     0    0.24  0.76   0  ; % 5
0     0     0     0     0    0.24   0  ; % 6
0     0     0     0     0     0     1  ; % 7
];
```

```
%

%
% T(2,2,1) = 0.24 % i.e. 1th matrix tW, 2rd row, 2th column (postState,priorState, action)
T = cat(3, TL, TR); % transition probabilities for each action

%--------------------------------------------------------------------------

% the reward function (defined as a local function - subfunction)
function rew = reward_function(priorState, a, postState) % reward function (defined locally)
if ((priorState == 2) && (postState == 1))
    rew = -10;
elseif ((priorState == 6) && (postState == 7))
    rew = 10;
elseif priorState > postState
    rew = 1;
elseif priorState < postState
    rew = -1;
else
    rew=0;
end

% get the reward matrix (defined as a local function - subfunction)
function R = reward_matrix(S, A)
% i.e. 11x11 matrix of rewards for being in state s, performing action a and ending in state s'
R = zeros(S, S, A);
for i = 1:S
   for j = 1:A
      for k = 1:S
         R(k, i, j) = reward_function(i, j, k);
      end
   end
end
```

C Monte-Carlo Policy Evaluation

```
function v=mc(trace,s,gamma,r)
v=0;
if (s+1)~= length(trace) % not to the last state in a trace
    v=r(s)+gamma*mc(trace,s+1,gamma,r);
else
    v=v+r(end);
end

end
```

D Get Unbiased Policy

```
function [UnbiasedPolicy] = GetUnbiasedPolicy(Absorbing, A)
UnbiasedPolicy = 1./A * ~Absorbing'*ones(1,A);
```

E Dynamic Programming Policy Evaluation

```
function [V,i,del] = PolicyEvaluation(Policy, T, R, Absorbing, gamma, tol)
% Dynamic Programming: Policy Evaluation. Estimates V(s) for each state s.
% Using 2 vectors for keeping track of Value Function, V(s)
S = length(Policy); % number of states - introspecting transition matrix
A = length(Policy(1,:)); % number of actions - introspecting policy matrix
V = zeros(S, 1); % optimal value function vector 11x1 (V at step i)
newV = V; % (V at step i+1)
Delta = 2*tol; % ensure initial Delta is greater than tolerance
i=0;
del=[];
while Delta > tol % keep approximating while not met the tolerane level
    for priorState = 1 : S
        if Absorbing(priorState) % do not update absorbing states
            continue;
        end
        tmpV = 0;
        for action = 1 : A
            tmpQ = 0;
            for postState = 1 : S
                tmpQ = tmpQ + T(postState,priorState,action)*(R(postState,
                    priorState,action) + gamma*V(postState));
            end
            tmpV = tmpV + Policy(priorState,action)*tmpQ;
        end
        newV(priorState) = tmpV;
    end
    diffVec = abs(newV - V);
    Delta = max(diffVec);
    V = newV;
    i=i+1;
    del=[del Delta];
end
% plot(del) % for step 4
end
```