Hasan Emre Erdemoglu - 21401462

Due: July 26, 2019

EEE 446 – Control & Optimization of Stochastic Systems

Homework 5 – MATLAB Assignment Report

Solution to Problem 1:

Linear system is given by the following set of equations:

$$x_{t+1} = Ax_t + \omega_t$$

$$y_t = Cx_t + v_t$$

Matrices A and C are given. Additionally, noise processes are known to be Gaussian and known to be independent and identically distributed. Finally, $\sum_{0|-1} = I$.

Define these relations:

$$m_t := E[x_t | y_{[0,t-1]}]$$

$$\sum\nolimits_{t|t-1} = E[(x_t - E[x_t|y_{[0,t-1]}])(x_t - E[x_t|y_{[0,t-1]}])^T]$$

After some calculations through the set of equations and relations listed above, the following update rule is discovered:

$$\sum\nolimits_{t+1|t} = A\sum\nolimits_{t|t-1} A^T + W - \left(A\sum\nolimits_{t|t-1} C^T\right)\left(C\sum\nolimits_{t|t-1} C^T + V\right)^{-1}(C\sum\nolimits_{t|t-1} A^T)$$

Iteration 1:

$$\sum\nolimits_{1|0} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\sum\nolimits_{0|-1}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^T + \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} - \left(\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\sum\nolimits_{0|-1}[2 \quad 1]^T\right)\left([2 \quad 1]\sum\nolimits_{0|-1}[2 \quad 1]^T + 1\right)^{-1}\left(C\sum\nolimits_{0|-1}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^T\right) = \begin{bmatrix}1.5 & 0.5\\ 0.5 & 1.83\end{bmatrix}$$

Iteration 2:

$$\sum\nolimits_{2|1} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\sum\nolimits_{1|0}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^T + \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} - \left(\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\sum\nolimits_{1|0}[2 \quad 1]^T\right)\left([2 \quad 1]\sum\nolimits_{1|0}[2 \quad 1]^T + 1\right)^{-1}\left(C\sum\nolimits_{1|0}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^T\right) = \begin{bmatrix}1.63 & 0.67\\ 0.67 & 2.09\end{bmatrix}$$

Both iteration results are also confirmed by MATLAB. Additionally, for a error rate of 0.00001 this matrix converges to [1.65, 0.70; 0.70 2.13] within 7 iterations. The code directly uses the Ricatti recursion equation above.

The code below will do all iterations automatically. It is confirmed that the calculations are correct for first and second iterations. The code will report how long does it take for the recursions to converge as well as the final point it goes to.

```
% ************************************************************************
% EEE 446 - MATLAB Assignment - Problem 1
% Hasan Emre Erdemoglu - 21401462 - Summer 2019
% References: Lecture Notes
% ************************************************************************
clear; clc; close all; % For debugging purposes
% Given entities:
A = [1 1; 0 1]; C = [2 1]; w = eye(2); v = 1; s0 = eye(2);
% Start from initial cond. Recurse until consecutive steps are eps away
% from each other:
s_next = s0; eps = 0.00001;
max_it = 1000;% If not converged, stop after this limit.
for k=1:max_it
    % Keep current state in temp, calculate next recursion, look at
    % absolute difference:
    s_cur = s_next;
    s_next = A*s_next*A'+w-(A*s_next*C')/(C*s_next*C'+v)*(C*s_next*A');
    if abs(s_cur - s_next) <= eps
        fprintf('Convergence achieved in %d steps. \n',k);
        fprintf('Unique fixed point given as follows: \n');
        disp(s_next);
        break;
    end
end
```

For the sake of completeness, MATLAB's output is given below:

```
Convergence achieved in 7 steps.
Unique fixed point given as follows:
    1.6542    0.7071
    0.7071    2.1322
```

Problem 2:

In this problem I extensively used lecture notes and the book Reinforcement Learning by Sutton & Barto. Meanwhile the lecture notes provide necessary theoretical background, the supplementary book provides pseudocode algorithms to efficiently and rapidly deploy the algorithms in any programming language.

Part A:

Part A deals with discounted cost criterion for the fading channel problem. Three different algorithms will be implemented. First Value Iteration was implemented as it felt more natural and easier to start with. Secondly, Policy Iteration and lastly Q-Learning was attempted.

As all three methods solve the same problem, it is expected for all of them to converge to the same values for states 1 and 2, perhaps taking different iterations to converge.

For completeness the notations on the pseudocodes mean the following:

- $P_{ss\prime}^{a}$: Probability of state transition from state s to state s' given action a
- $R_{ss\prime}^{a}$: Reward/Cost gathered from the state transition from state s to state s' given action a
- γ: Discount Factor
- s: Current State, s': Next State, a: Action

Section ii: - Value Iteration

The pseudocode for the Value Iteration algorithm is given below:

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s')\right]$

In this pseudocode, first value functions for all states are set to zero. Then, until convergence occurs (which is monitored by delta); current value function for each state is kept in memory. After this, new value function is implemented.

The unique property of Value Iteration is that, for each update on the value function a deterministic policy is decided upon for each state (even though it is not explicitly used, it only outputs a deterministic policy at the end). It also is better than Policy iteration as it converges faster than Policy iteration. This happens because, for a given policy, policy iteration has to generate a stable value for each state. After that policy improvement is used along with value function to provide a better policy. At the end, for each provided policy, the entire value function has to be re-calculated. This means there are fewer iterations in value iteration.

The following MATLAB code was developed to solve the discounted cost criterion problem:

```
% ************************************************************************
% EEE 446 - MATLAB Assignment - Problem 2 Part A-ii:
% Hasan Emre Erdemoglu - 21401462 - Summer 2019
% References: Lecture Notes & RL by Sutton & Barto
% This is almost the same with Policy Evaluation in the previous section.
% Here we do not update by iteratively improving policy but the action.
% ************************************************************************
clear; clc; close all; % For debugging purposes
% Fix arbitrary discount factor btwn (0,1):
beta = 0.62; % Fixed for debugging purposes
eps = 0.00001; % Difference between two consec. iterations to stop VI
% Look-up table for this question is already built in:
% Probability of state transition using a particular action.
P1 = [0.1 0.9; 0.8 0.2];  P2 = [0.5 0.5; 0.2 0.8];
c = [0 0.5; 0 -0.5]; % Transition costs ([1->1 1->2 2->1 2->2])
% Implement Value Iteration Algorithm:
V = [0; 0]; % Assign initial value.
max_it = 1000; % Max iter. keeps loop terminate at some point
for it = 1:max_it
    Vtemp = V;
    delta = 0;
    % Keep the function in matrix form so that we can find the minimum of
    % elements (1 or 2):
    % Update rule: V(s) <- (P_ss')^a * ((R_ss')^a)+beta*V(s'))
    % Take the minimum of either a (or u) = 1 or a = 2.
    % Do it for state 1:
    [V1(it), pol1] = min( ...
        [sum(P1(1,:)'.*(c(:,1)+beta*Vtemp)), ...
         sum(P2(1,:)'.*(c(:,2)+beta*Vtemp))]);
    % Do it for state 2:
    [V2(it), pol2] = min( ...
        [sum(P1(2,:)'.*(c(:,1)+beta*Vtemp)), ...
         sum(P2(2,:)'.*(c(:,2)+beta*Vtemp))]);
    % Check if converged:
    delta = max(delta,max(abs(V(1)-V1(it)),abs(V(2)-V2(it))));
```

```
    V = [V1(it); V2(it)];
    pol(:,it) = [pol1; pol2]; clear pol1; clear pol2;
    % Break out of the loop if convergence condition is reached
    if (delta < eps)
        break;
    end
end
% Print out the results:
figure; title('Value Functions with respect to Iterations');
xlabel('Iterations'); ylabel('Value');
hold on; axis tight; grid on;
plot(V1); plot(V2); legend('V1', 'V2');
disp('Policies selected (First Row - State 1, Second Row - State 2):')
disp(pol)
fprintf('Convergence occurred after %d iterations. \n', it);
```

Using the code given above it is observed that our value function converges at [-0.4148 -0.6973] with a discount factor of 0.62 (comes after last two digits of my school number, chosen greater than 0.5 on purpose – for more refined results). Convergence is reached after 23 iterations and policy became deterministic [1, 2].

Following results are found. Note that in the figure below, graphs V1 and V2 does not start from zero, since initial condition 0 was not added on top of the measurements.

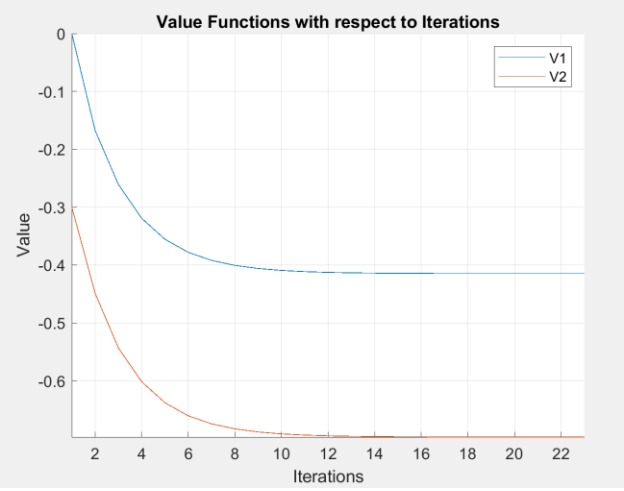
```
Policies selected (First Row - State 1, Second Row - State 2):
  Columns 1 through 20

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2

  Columns 21 through 23

     1     1     1
     2     2     2

Convergence occurred after 23 iterations.
```



Section i: - Policy Iteration

The pseudocode for the Policy Iteration is given below:

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[r(s, \pi(s), s') + \gamma v(s') \Big]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \gamma v(s') \Big]$
      If $temp \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If *policy-stable*, then stop and return v and π; else go to 2

Here first, random values are set for all states and a random policy has been established from the set of available policies.

First, a policy evaluation on the random state-values is applied. After the state-value converges; the algorithm proceeds to policy improvement stage. Here, the algorithm is able to use converged state-value to find an optimal policy, which is in fact better than the policy that we initially started with (as proved in class).

Then, policy evaluation process starts again. It converges and triggers policy improvement. Until the policy stabilizes the process continues. The pro of using such approach is that at each policy iteration policy does not have to be stationary, like in value iteration. However, this process takes significantly longer to finish compared to value iteration as you iterate over policy evaluation multiple times to reach convergence.

The MATLAB code for this algorithm can be found below. After that there are some documentation of the findings using policy iteration.

```
% ************************************************************************
% EEE 446 - MATLAB Assignment - Problem 2 Part A-i:
% Hasan Emre Erdemoglu - 21401462 - Summer 2019
% References: Lecture Notes & RL by Sutton & Barto
% Note: I did Value Iteration first as I it felt easier to deal with.
% ************************************************************************
clear; clc; close all; % For debugging purposes
% Fix arbitrary discount factor btwn (0,1):
beta = 0.62; % Fixed for debugging purposes
eps = 0.00001; % Difference between two consec. iterations to stop VI
% Look-up table for this question is already built in:
% Probability of state transition using a particular action.
P1 = [0.1 0.9; 0.8 0.2];  P2 = [0.5 0.5; 0.2 0.8];
c = [0 0.5; 0 -0.5]; % Transition costs ([1->1 1->2 2->1 2->2])
% 1. Initialization:
V = [rand; rand]; % Assign initial value.
% Run the algorithm
max_it = 1000; % Max iter. keeps loop terminate at some point
% At any case less than max_it policies will be used. - Remaining garbage.
pols = randi([1 2],2,1); % init policies decided randomly
count = 1;
Vgraph(:,count) = V;
polsGraph(:,count) = pols; itpast = 1; it = 1;
while it <= max_it
    Vtemp = V;
    delta = 0;
    [V] = polEval(P1, P2, pols, beta, c, Vtemp)
    delta = max(delta,max((abs(V(1)-Vtemp(1))),abs(V(2)-Vtemp(2))));
    % for graphing over all policy improvements
    Vgg(:,itpast) = V;
    itpast = itpast + 1;
    it = it + 1;
    % Break out of the loop if convergence condition is reached
    if (delta < eps)
        % for scorekeeping:
        count = count + 1;
        Vgraph(:,count) = V;
        iterationState(count-1) = it;
        % now continue to policy improvement
        stableflag = 1;
        cur_pol = pols; % we may truncate unnecessary policies
        % calculate new optimal policy, if stable get out
        [pol_next] = polImp(P1, P2, beta, c, V);
        % if not stable then it = 1, return to policy evaluation with new
        % policy pi.
        if cur_pol(1) ~= pol_next(1) || cur_pol(2) ~= pol_next(2) % x stbl
            stableflag = 0;
```

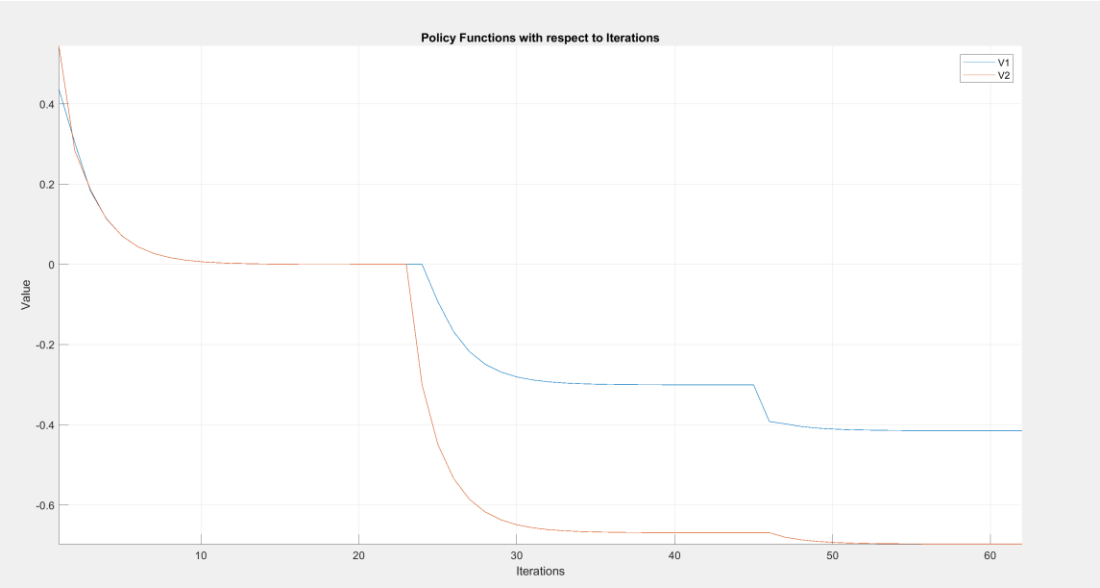
```
            it = 1;
            pols = pol_next;
            polsGraph(:,count) = pols;
        else % stable
            break;
        end
    end
end

% Documentation of results:
fprintf('%d Policy Improvement steps were made.');
disp('After each policy improvement this many iterations were taken to reach
to stable point ');
disp(iterationState);
disp('Stabilized state values at each policy evaluation:')
disp(Vgraph);
disp('At the end of the algorithm under optimal policy, we reach to optimal
state-value ');
disp(V);
disp('Chosen policies at each policy improvements:')
disp(polsGraph);
% Print out the results:
figure; title('Policy Functions with respect to Iterations');
xlabel('Iterations'); ylabel('Value');
hold on; axis tight; grid on;
plot(Vgg(1,:)); plot(Vgg(2,:)); legend('V1', 'V2');
fprintf('Convergence occurred after %d iterations. \n', itpast);

% 2. Policy Evaluation & Improvement Chunklets:
% prob1, prob2 2x2 state transition array with action 1 and 2 respectively
% policy is the policy that we want to find value of
% beta is the discount factor
% cost is 2x2 cost metric
% Vprev is the initial value that we give, Vnext is the output
function [Vnext] = polEval(prob1, prob2, policy, beta, cost, Vprev)
% Do it for state 1:
if policy(1) == 1 % picked action 1
    V1 = sum(prob1(1,:)'.*(cost(:,1)+beta*Vprev));
else % picked action 2
    V1 = sum(prob2(1,:)'.*(cost(:,2)+beta*Vprev));
end
% Do it for state 2:
if policy(2) == 1 % picked action 1
    V2 = sum(prob1(2,:)'.*(cost(:,1)+beta*Vprev));
else % picked action 2
    V2 = sum(prob2(2,:)'.*(cost(:,2)+beta*Vprev));
end
Vnext = [V1; V2];
end
function [pol_next] = polImp(prob1, prob2, beta, cost, Vprev)
% Keep the function in matrix form so that we can find the minimum of
% elements (1 or 2):
```

```
% Update rule: V(s) <- (P_ss')^a * ((R_ss')^a)+beta*V(s'))
% Take the minimum of either a (or u) = 1 or a = 2.
% Do it for state 1:
[~, pol1] = min( ...
    [sum(prob1(1,:)'.*(cost(:,1)+beta*Vprev)), ...
    sum(prob2(1,:)'.*(cost(:,2)+beta*Vprev))]);
% Do it for state 2:
[~, pol2] = min( ...
    [sum(prob1(2,:)'.*(cost(:,1)+beta*Vprev)), ...
    sum(prob2(2,:)'.*(cost(:,2)+beta*Vprev))]);
pol_next = [pol1; pol2];
end
```

The findings are listed below. These listings can be also be found in MATLAB. The crucial fact is that Policy Iteration and Value Iteration indeed gives the exact-same stabilized state-value [-0.4148 -0.6973].




```
After each policy improvement this many iterations were taken to reach to stable point
    24    23    18

Stabilized state values at each policy evaluation:
    0.9958    0.0000   -0.3006   -0.4148
    0.4124    0.0000   -0.6692   -0.6973

At the end of the algorithm under optimal policy, we reach to optimal state-value
   -0.4148
   -0.6973

Chosen policies at each policy improvements:
     2     2     1
     1     2     2

Convergence occurred after 63 iterations.
```

Section iii: - Q-Learning Implementation

Similar to parts (i) and (ii), the pseudo-algorithm is given as follows. It is expected that also for Q-Learning, state-values should converge to the same value observed in previous parts:

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Here it is hinted that one may pick coefficient alpha from:

$$\alpha = \frac{1}{1 + \sum_{k=0}^{t} 1_{\{x_k = x, u_k = u\}}}$$

Q should be selected as 2x2, since State-Action can be mapped by a 2x2 relation.

I did not continue the rest of the question, due to my final grade.