MATH4406 (Control Theory), HW4

Trent Skorka - s42897282

September 26, 2014

1. **Think of how this may fit in some applied situation, where your controller is trying to regulate a system at a "set-point".**

   Consider a situation in which there are two equally powerful groups to which you may pledge your support. This could represent an alliance of companies, an investment of money, time or energy, or an alliance of kingdoms. Let us consider the example of kingdoms. At each time step (say, each month), you must choose either to increase your support, or withdraw it. Naturally, your proposal for support (or withdrawal) may be rejected due to poor actions on your behalf (or very commendable actions - and who can deny the decisions of support from kings?). The more you pledge support to one group, the greater your reward (aid, protection, financial support), but if you fully support one group, the other will percieve you as an enemy, and attack (represented by the high cost incurred at state ± 5).

2. **What would optimal policies look like? What role does λ play? What do you think the optimal policy when $\lambda \approx 0$ (but > 0). What do you think is the optimal policy when $\lambda \approx 1$ (but < 1)? State your reasons clearly.**

   Optimal policies for this problem will depend on the value of λ - this represents the discounting of future rewards. It would be preferable not to be in one of the end states, but in the case where $\lambda = 0$, future states do not affect the reward at all, so it would not matter if you actually made it to an end state, as it would not affect the expected value of the MDP. Thus, for $\lambda \approx 0$, we would expect the optimal policy to be $d_n = 1$ for $n = \{1, 2, 3, 4\}$ and $d_n = -1$ for $n = \{-4, -3, -2, -1, 0\}$. Note that it does not actually matter what decision is made at state $s = \{0\}$. We arbitrarily set it to be -1.

   If $\lambda = 1$, all future rewards are not discounted at all, and so it is important to consider the long-term outcome of your policy. It would thus make sense to move away from the end states as much as possible. However, if you always attempt to move towards the center state, you will always incur a cost, rather than a reward (as $s < 0, a > 0 \Rightarrow r < 0$, and visa versa). Thus, I would expect the optimal policy for $\lambda \approx 1$ to be $d_n = 1$ for $n = \{-4, -3, -2, 1\}$ and $d_n = -1$ for $n = \{-1, 0, 2, 3, 4\}$, where again d_0 was arbitrarily chosen.

   Regardless of the choice of λ, the optimal policy should be symmetric about d_0, with of course $d_5 = \{-3\}$ and $d_{-5} = \{3\}$.

3. **Write a function that finds the optimal policy by brute force of all** (512) **policies and solution of the policy evaluation equations.**

For this question, we require the construction of all policies. There are $2^9 = 512$ policies, as for each state in $\{-4, \ldots, 4\}$, we have two potential actions. For each of these policies, we wish to solve the equation

$$v_\lambda^d = r_d + \lambda P_d v_\lambda^d \tag{1}$$

This has the solution

$$v_\lambda = (I - \lambda P_d)^{-1} r_d \tag{2}$$

We then wish to find the policy which gives the maximum value for each state. Each state should produce the same result, with the exception of the action at $s = \{0\}$ which is arbitrary as it does gives a reward of 0, and the problem is symmetric.
The code for this question is given in *Appendix A* .

4. **Write a function that finds the optimal policy using value iteration with specified $\epsilon > 0$ and a stopping criterion adapted to λ.**

The value iteration algorithm requires an initial $v^0 \in V$ to be chosen. For simplicity, we arbitrarily set $v^0 = \{0, \ldots, 0\}$. The algorithm is then run until the condition below is met:

$$||v^{n+1} - v^n|| < \frac{\epsilon(1-\lambda)}{2\lambda} \tag{3}$$

Here, $\epsilon > 0$ is some small constant. Once this condition is met, the policy used to find v^{n+1} is the optimal policy.
The code for this question is given in *Appendix B*, where we chose $\epsilon = 0.01$.

5. **Write a function that finds the optimal policy using the policy iteration algorithm.**

The policy iteration algorithm obtains v^n for each decision rule, d by solving equation **(1)**. It requires an inital arbitrary decision rule $d_0 \in D$ to be chosen. For this question, we set $d_0 = \{-3, -1, ..., -1, 3\}$. The algorithm is then run until $d_{n+1} = d_n$, at which point we set $d^* = d_n$. At each step, we recalculate the transition matrix P, and the value of the policy is calculated using equation **(2)**.
The code for this question is given in *Appendix C*.

6. **(Optional) Write a function that finds the optimal policy using linear programming.**
– Not Attempted –

7. **Use 3-6 to find the optimal policy for a range of $\lambda \in (0, 1)$ in steps of 0.01 (ie. run the algorithm 99 times). For the value iteration use a "small enough" epsilon. Present the optimal policies (for each λ) and briefly comment on the results.**

The optimal policy changes at $\lambda = 0.64$, 0.86, 0.94 and 0.96. The policies are summarised below, and the code used is presented in *Appendix D*. The value iteration algorithm, with $\epsilon = 0.01$ was used, and no discrepancies were noted.

State	$0.01 \leq \lambda \leq 0.63$	$0.64 \leq \lambda \leq 0.86$	$0.87 \leq \lambda \leq 0.93$	$0.94 \leq \lambda \leq 0.96$	$0.97 \leq \lambda \leq 0.99$
-5	3	3	3	3	3
-4	-1	1	1	1	1
-3	-1	-1	1	1	1
-2	-1	-1	-1	1	1
-1	-1	-1	-1	-1	1
0	-1	-1	-1	-1	-1
1	1	1	1	1	-1
2	1	1	1	-1	-1
3	1	1	-1	-1	-1
4	1	-1	-1	-1	-1
5	-3	-3	-3	-3	-3

The optimal policy for $\lambda \approx 0$ was as expected (refer to **Q2**), but it is interesting to not how large λ must be before this policy of always striving for the end states is no longer optimal. Also of interest is $\lambda \approx 1$, namely $0.97 \leq \lambda \leq 0.99$. This is not the optimal policy that was predicted, as the policy ensures one will never gain a positive reward. Thus, for these larger values of λ, the objective would perhaps be better represented as "minimise the cost", rather than "maximise the reward".

Note that the decisions rules are, indeed, symmetric about $s = \{0\}$, as expected. As λ increases, it becomes more preferable to move away from the end states before reaching them, a result of future rewards having more of an influence.

APPENDIX A

Matlab Code for Question 3

```
clear all
clc

S=[-5:5];
lambda = 0.1;
epsilon=0.01;


% Question 3 - Brute Force
Dec=zeros(512,11); %Generate all decisions
k=1;
for j1=1:2
    for j2=1:2
        for j3=1:2
            for j4=1:2
                for j5=1:2
                    for j6=1:2
                        for j7=1:2
                            for j8=1:2
                                for j9=1:2
                                    Dec(k,:)=[3 j1 j2 j3 j4 j5 j6 j7 j8 j9 -3];
                                    Dec(Dec==2)=-1;
                                    k=k+1;
                                end
                            end
                        end
                    end
                end
            end
        end
     end
end
P=zeros(length(S));%Create matrix P
P(1,1)=0.5;
P(1,2)=0.5;
P(end,end-1)=0.5;
P(end,end)=0.5;
I=eye(length(S)); %Construct Identity matrix
for k=1:512
   for j=2:length(S)-1 %Construct P_d for each policy
        if  Dec(k,j)==-1
            P(j,j-1)=0.75;
            P(j,j+1)=0.25;
        else
            P(j,j-1)=0.25;
            P(j,j+1)=0.75;
        end
    end
    A=I-lambda.*P; %Find (I-lambda*P)^-1
    reward(k,:)=S.*Dec(k,:);
    v(k,:)=inv(A)*reward(k,:)'; %Policy Evaluation
end
for s=1:length(S)
    [v(s), d(s)]=max([v(:,s)]);
end
Dec(d(3),:) %Optimal decision rule

```

APPENDIX B

Matlab Code for Question 4

```
clear all
clc

S=[-5:5];
lambda = 0.1;
epsilon=0.01;

% Question 4 - Value Iteration Policy
d=zeros(size(S))'; %Create decision vector
d(1)=3;
d(end)=-3;
oldv=zeros(size(S))'; %Old value
newv=zeros(size(S))'+1; %New value
while norm(newv-oldv)>epsilon*(1-lambda)/(2*lambda) %Stopping criteria
    oldv=newv;
    newv(1)=-15+(lambda*0.5*(oldv(1)+oldv(2)));
    newv(end)=-15+(lambda*0.5*(oldv(end)+oldv(end-1)));
    for s=2:length(S)-1 %Maximising value over all policies
        psminus=zeros(1,length(S));
        psminus(s-1)=0.75;
        psminus(s+1)=0.25;
        psplus=zeros(1,length(S));
        psplus(s-1)=0.25;
        psplus(s+1)=0.75;
        [newv(s),d(s)]=max([-S(s)+lambda*psminus*oldv,S(s)+lambda*psplus*oldv]);
    end
d(d==1)=-1; %Fix decision vector to have correct values
d(d==2)=1;
end
d %Output result
```

APPENDIX C

Matlab Code for Question 5

```
clear all
clc

S=[-5:5];
lambda = 0.1;
epsilon=0.01;


% Question 5 - Policy Iteration Algorithm
old_d=zeros(size(S))'-1; %Old decision rule
old_d(1)=3;
old_d(end)=-3;
new_d=zeros(size(S))'+1; %New decision rule - ensuring different to old_d
new_d(1)=3;
new_d(end)=-3;
P=zeros(length(S)); %Create matrix P
P(1,1)=0.5;
P(1,2)=0.5;
P(end,end-1)=0.5;
P(end,end)=0.5;
I=eye(length(S)); %Construct Identity matrix
while max(new_d~=old_d)>0
    old_d=new_d;
    for j=2:length(S)-1 %Construct P_d
         if  old_d(j)==-1
             P(j,j-1)=0.75;
             P(j,j+1)=0.25;
         else
             P(j,j-1)=0.25;
             P(j,j+1)=0.75;
         end
     end
     A=I-lambda.*P; %Find (I-lambda*P)^-1
     reward=S'.*old_d;
     v=inv(A)*reward; %Policy Evaluation
     for s=2:length(S)-1 %Policy Improvement
         psminus=zeros(1,length(S));
         psminus(s-1)=0.75;
         psminus(s+1)=0.25;
         psplus=zeros(1,length(S));
         psplus(s-1)=0.25;
         psplus(s+1)=0.75;
         [v(s),new_d(s)]=max([-S(s)+lambda*psminus*v,S(s)+lambda*psplus*v]);
     end
new_d(new_d==1)=-1; %Fix decision vector to have correct values
new_d(new_d==2)=1;
end
new_d %Output optimal policy
```

APPENDIX D

Matlab Code for Question 7

```
clear all
clc

S=[-5:5];
lambda = 0.1;
epsilon=0.01;

% Question 7 - Examination of all policies
for Lambda=1:99
    lambda=Lambda/100; %0.01 < lambda < 0.99
    d=zeros(size(S))';
    d(1)=3;
    d(end)=-3;
    oldv=zeros(size(S))';
    newv=zeros(size(S))'+1;
    while norm(newv-oldv)>epsilon*(1-lambda)/(2*lambda)
        oldv=newv;
        newv(1)=-15+(lambda*0.5*(oldv(1)+oldv(2)));
        newv(end)=-15+(lambda*0.5*(oldv(end)+oldv(end-1)));
        for s=2:length(S)-1
            psminus=zeros(1,length(S));
            psminus(s-1)=0.75;
            psminus(s+1)=0.25;
            psplus=zeros(1,length(S));
            psplus(s-1)=0.25;
            psplus(s+1)=0.75;
            [newv(s),d(s)]=max([-S(s)+lambda*psminus*oldv,S(s)+lambda*psplus*oldv
        end
    d(d==1)=-1; %Fix decision vector to have correct values
    d(d==2)=1;
    end
    for s=1:length(S)
        D(s,Lambda+1)=d(s); %Construct matrix of optimal policies
    end
end
changes=zeros(1,99); %Look for changes in policies
for j=1:98
    if max(D(:,j)~=D(:,j+1))==1
        changes(j)=j; %Record where the change occurs
    else changes(j)=0;
    end
end
```