Peter Ferguson

10/18/2018

EE209AS Problem Set 2

0(a). https://github.com/Sarnlest/EE209AS-Problem-Set-2-Take2 I am not sure how you "comment" github. I have commented each of the matlab files. HW2Base is the main file that branches to the rest of the files.

0(b). I did not collaborate with anyone beyond discussing ideas for how to interpret the instructions for the initial policy and comparing final answers.

0(c). 100% done by me.

0(d). 100% done by me.

```
%Peter Ferguson, 10/11/2018, EE209AS, Problem Set 2
%Completed alone
clear
close all
clc

%1(a).
disp('1(a)')
disp('States are a 3d matrix where [x,y] cell corresponds to positions in grid world, [z] refers to direction on the clock the robot is facing')
disp('Eg [1,0,4] corresponds to state with x=1, y=0, and h=4 (facing 4 oclock)')
disp('The number of states, N_s=36*12=432')

%1(b)
disp('1(b)')
disp('Possible actions are ["FL","F","FR","N","BL","B","BR"] where F=Forward, B=Back, L=Left, R=Right, N=no movement.')
disp('The number of possible actions, N_A=7')


%1(c)
disp('1(c)')
disp('See TransProb2')

%1(d)
disp('1(d)')
disp('See NextS')

%2(a)
disp('2(a)')
disp('See Reward')


%3(a)
disp('3(a)')
disp('See InitPoli3 and PiNot')

%3(b)
disp('3(b)')
disp('See GenTraj')

%3(c)
disp('3(c)')
PiNotTrajectory=GenTraj(PiNot,[1,4,6],0);
disp('The trajectory of the robot prescribed by Pi_0 with Pe=0 is:')
disp(PiNotTrajectory)

%3(d)
disp('3(d)')
disp('See PolicyEval')

%3(e)
disp('3(e)')
PiNotValues=PolicyEval(PiNot(),0.9,0);
disp(['The value of the robot prescribed by Pi_0 at position [1,4,6] is ',num2str(PiNotValues(2,5,7)),'.'])

%3(f)
disp('3(f)')
disp('See NextPi2')

%3(g)
disp('3(g)')
disp('See PolicyIteration')

%3(h)
disp('3(h)')
tic
PiStar=PolicyIteration(PiNot(),0.9,0);
OptimalTrajectory=GenTraj(PiStar,[1,4,6],0);
disp('The trajectory of the robot prescribed by Pi* with Pe=0 is:')
disp(OptimalTrajectory)
Values=PolicyEval(PiStar,0.9,0);
disp(['The value of the robot prescribed by Pi* with Pe=0 at position [1,4,6] is ',num2str(Values(2,5,7)),'.'])
%3(i)
disp('3(i)')
toc

%4(a)
disp('4(a)')
disp('See ValueIteration')

%4(b)
disp('4(b)')
tic
[VIPiStar,VIVStar]=ValueIteration(0.9,0);
VITrajectory=GenTraj(VIPiStar,[1,4,6],0);
disp('It obtains the same Policy and Values for Value Iteration and Policy Iteration')
differencePi=432-sum(sum(sum(PiStar==VIPiStar)))
differenceValue=sum(sum(sum(Values-VIVstar)))
%4(c)
disp('4(c)')
toc

%5(a)
disp('5(a)')
tic
PiStar2=PolicyIteration(PiNot(),0.9,0.25);
OptimalTrajectory2=GenTraj(PiStar2,[1,4,6],0.25);
disp('The trajectory of the robot prescribed by Pi* with Pe=0.25 is:')
disp(OptimalTrajectory2)
Values2=PolicyEval(PiStar2,0.9,0.25);
disp(['The value of the robot prescribed by Pi* with Pe=0.25 at position [1,4,6] is ',num2str(Values2(2,5,7)),'.'])
toc
```

```
%5(b)
disp('5(b)')
tic
PiStar3=PolicyIteration2(PiNot(),0.9,0.25);
OptimalTrajectory3=GenTraj(PiStar3,[1,4,6],0.25);
disp('The trajectory of the robot prescribed by Pi* with Pe=0.25 is:')
disp(OptimalTrajectory3)
Values3=PolicyEval(PiStar3,0.9,0.25);
disp(['The value of the robot prescribed by Pi* with Pe=0.25 with goal pointing downward for starting position [1,4,6] is ',num2str(Values3(2,5,7)),'.'])
toc

%5(c)
disp('5(c)')
disp('When possibility for error is introduced, there is a chance that the robot must take a significantly longer path, or retrace its steps when an error occurs.')
disp('If the goal is made to only be facing downward, the robot must take a much longer path, and with the probability of error introduced, the path can become extremely long.')
disp('Furthermore, when the goal is facing downward, the robot tends to try to back into the goal as opposed to move forward into it')
```

```
1(a)
States are a 3d matrix where [x,y] cell corresponds to positions in grid world, [z] refers to direction on the clock the robot is facing
Eg [1,0,4] corresponds to state with x=1, y=0, and h=4 (facing 4 oclock)
The number of states, N_s=36*12=432
1(b)
Possible actions are ["FL","F","FR","N","BL","B","BR"] where F=Forward, B=Back, L=Left, R=Right, N=no movement.
The number of possible actions, N_A=7
1(c)
See TransProb2
1(d)
See NextS
2(a)
See Reward
3(a)
See InitPoli3 and PiNot
3(b)
See GenTraj
3(c)
The trajectory of the robot prescribed by Pi_0 with Pe=0 is:
     1     4     6
     1     3     7
     1     4     8
     2     4     9
     3     4     9

3(d)
See PolicyEval
3(e)
The value of the robot prescribed by Pi_0 at position [1,4,6] is -0.72901.
3(f)
See NextPi2
3(g)
See PolicyIteration
3(h)
The trajectory of the robot prescribed by Pi* with Pe=0 is:
     1     4     6
     1     3     5
     1     2     5
     1     1     4
     2     1     4
     3     1     5
     3     2     5
     3     3     5
     3     4     4

The value of the robot prescribed by Pi* with Pe=0 at position [1,4,6] is 4.3038.
3(i)
Elapsed time is 379.626169 seconds.
4(a)
See ValueIteration
4(b)
It obtains the same Policy and Values for Value Iteration and Policy Iteration

differencePi =

     0



Undefined function or variable 'VIVstar'.

Error in HW2Base (line 87)
differenceValue=sum(sum(sum(Values-VIVstar)))
```

Published with MATLAB® R2017b

```
function [Trajectory] = GenTraj(Pi,S0,pe)
%Generate and plots a trajectory of a robot given policy matrix/array Pi,
%initial state S0, and error probability pe
%   Detailed explanation goes here
Trajectory=S0;
s=S0;
while (abs(s(1)-3)+abs(s(2)-4))~=0;
    x=s(1); y=s(2); h=s(3);
    action=Pi(x+1,y+1,h+1);
    s=NextS(pe,s,action);
    Trajectory=cat(1,Trajectory,s);
end
Length=size(Trajectory,1);
figure
plot3(Trajectory(:,1),Trajectory(:,2),Trajectory(:,3))
xlabel('X')
ylabel('Y')
zlabel('H')
axis([0,5,0,5,0,11])
xticks([0:5])
yticks([0:5])
zticks([0:11])
grid on
hold on
scatter3(Trajectory(1,1),Trajectory(1,2),Trajectory(1,3),'r')
text(Trajectory(1,1),Trajectory(1,2),Trajectory(1,3),'Start')
scatter3(Trajectory(Length,1),Trajectory(Length,2),Trajectory(Length,3),'g')
text(Trajectory(Length,1),Trajectory(Length,2),Trajectory(Length,3),'Goal')
hold off


end
```

```
Not enough input arguments.

Error in GenTraj (line 5)
Trajectory=S0;
```

Published with MATLAB® R2017b

```
function [a] = InitPoli3(s)
%Given a current state, s, provide the action, a, that most directly goes
%towards the goal at state [3,4,h] for 0<=h<12
%   Splits the plane in half as seen by the robot.  If the goal is on the robot don't move; if in front
%   of or directly to the side of the robot, move forward; otherwise move
%   backwards.  If the robot moves forward or backward, update predicted
%   position with Pe=0, and calculate the error between the current and
%   desired heaading.  Turn left/right so that forward or backward heading
%   is most closely points towards the goal.  If there is a tie between
%   making the forward or backward heading closer to the goal, favor the
%   forward heading.
h=0; %completely arbitrary, randomly picked 0, but could be anything.  Simply used to keep the form of goal and s the same.
goal=[3,4,h];
xerror=goal(1)-s(1);
yerror=goal(2)-s(2);
angled=atan2d(yerror,xerror); %desired heading in degrees
hangle=90-s(3)*30; %convert current heading into angle in degrees
herror=mod(angled-hangle,360); %find error between current heading and desired heading and make between 0 and 360 degrees

%Take the appropriate action for the given policy based on the heading error
%If the heading error is 15 degrees from heading 0 or 6, don't change
%heading (chosen arbitrarily to reduce turning)
if (abs(xerror)+abs(yerror))==0;
    a="N"; %Output next action is to stay still.
elseif herror<=90 || herror>=270 %Move forward
    s=NextS(0,s,"F");
    xerror=goal(1)-s(1);
    yerror=goal(2)-s(2);
    angled=atan2d(yerror,xerror); %desired heading in degrees
    hangle=90-s(3)*30; %convert current heading into angle in degrees
    herror=mod(angled-hangle,360); %find error between current heading and desired heading
    if (((herror<=15) || (herror>=345))||((herror>=165)&&(herror<=195)))
        a="F"; %Output next action should be to move forward
    elseif (((herror>15) && (herror<=90))||((herror>195)&&(herror<270)))
        a="FL"; %Output next action should be to move forward left
    elseif (((herror>=270) && (herror<345))||((herror>90)&&(herror<165)))
        a="FR"; %Output next action should be to move forward right
    end
else %Move Backward
    s=NextS(0,s,"B");
    xerror=goal(1)-s(1);
    yerror=goal(2)-s(2);
    angled=atan2d(yerror,xerror); %desired heading in degrees
    hangle=90-s(3)*30; %convert current heading into angle in degrees
    herror=mod(angled-hangle,360); %find error between current heading and desired heading
    if (((herror<=15) || (herror>=345))||((herror>=165)&&(herror<=195)))
        a="B"; %Output next action should be to move backward
    elseif (((herror>15) && (herror<=90))||((herror>195)&&(herror<270)))
        a="BL"; %Output next action should be to move backward left
    elseif (((herror>=270) && (herror<345))||((herror>90)&&(herror<165)))
        a="BR"; %Output next action should be to move backward right
    end
end

end
```

```
Not enough input arguments.

Error in InitPoli3 (line 14)
xerror=goal(1)-s(1);
```

Published with MATLAB® R2017b

```
function [Pi] = NextPi2(CurrentEval,pe)
%Creates a new policy Pi based on the Policy Evaluation CurrentEval of an existing
%policy and error probability pe
%   Calculates the sum of the transition proability from s to sprime times the value of the
%  CurrentEval at sprime for each every action of every state.  Selects as
%  policy the action at each state with the highest sum.
PiValues=zeros(6,6,12);
PiValues=PiValues-10^100; %Initialize the values to compare calculated values against to negative infinity
for i=0:5 %X
    for j=0:5 %Y
        for k=0:11 %H
            for a=["FL","F","FR","BL","B","BR"]; %do not consider staying still
                b=0;
                for l=0:5 %X'
                    for m=0:5 %Y'
                        for n=0:11 %H'
                            %if (sum([i,j,k]==[l,m,n])~=3) || (sum([i,j]==[3,4])==2) %prevents no movem action outside of the goal
                                b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*CurrentEval(l+1,m+1,n+1); %dummy variable to reduce calculation repetition
                            %end
                        end
                    end
                end
                if PiValues(i+1,j+1,k+1)<b
                    PiValues(i+1,j+1,k+1)=b;
                    Pi(i+1,j+1,k+1)=a;
                end
            end
        end
    end
end
Pi(4,5,:)="N";
end
```

```
Not enough input arguments.

Error in NextPi2 (line 18)
                            b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*CurrentEval(l+1,m+1,n+1); %dummy variable to reduce calculation repetition
```

Published with MATLAB® R2017b

```
function [Pi] = NextPi3(CurrentEval,pe)
%Creates a new policy Pi based on the Policy Evaluation CurrentEval of an existing
%policy and error probability pe
%   Calculates the sum of the transition proability from s to sprime times the value of the
%  CurrentEval at sprime for each every action of every state.  Selects as
%  policy the action at each state with the highest sum.  Goal is at the
%  specified spot pointing downward.
Pi=strings(6,6,12);
PiValues=zeros(6,6,12);
PiValues=PiValues-10^100; %Initialize the values to compare calculated values against to negative infinity
for i=0:5 %X
    for j=0:5 %Y
        for k=0:11 %H
            for a=["FL","F","FR","BL","B","BR"]; %do not consider staying still
                b=0;
                for l=0:5 %X'
                    for m=0:5 %Y'
                        for n=0:11 %H'
                                b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*CurrentEval(l+1,m+1,n+1); %dummy variable to reduce calculation repetition
                        end
                    end
                end
                if PiValues(i+1,j+1,k+1)<b
                    PiValues(i+1,j+1,k+1)=b;
                    Pi(i+1,j+1,k+1)=a;
                end
            end
        end
    end
end
for c=6:8; %dummy variable to specify down directions
    Pi(4,5,c)="N";
end
end
```

```
Not enough input arguments.

Error in NextPi3 (line 19)
                                b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*CurrentEval(l+1,m+1,n+1); %dummy variable to reduce calculation repetition
```

Published with MATLAB® R2017b

```
function [SPrime] = NextS(pe,s,a)
%Generates the next state given an error probability Pe, current state S,
%and action a
%   Uses TransProb2
b=rand(); %Generate a number between 0 and 1
for i=0:5; %x'
    for j=0:5; %y'
        for k=0:11; %h'
            b=b-TransProb2(pe,s,a,[i,j,k]); %Subtract the transition probability from the number
            if b<=0
                SPrime=[i,j,k]; %If the number goes negative, the checked next state is the actual next state
                b=b+100; %prevent the number from going negative again, ensures only the one action was taken.
            end
        end
    end
end


end
```

```
Not enough input arguments.

Error in NextS (line 9)
            b=b-TransProb2(pe,s,a,[i,j,k]); %Subtract the transition probability from the number
```

Published with MATLAB® R2017b

```
function [AStars] = PiNot()
%Call function InitPoli for every state, output the action prescribed by
%the policy at all states in a matrix of strings
%   self explanatory
for i=0:5; %x
    for j=0:5; %y
        for k=0:11; %h
            AStars(i+1,j+1,k+1)=InitPoli3([i,j,k]); %follow the initial policy
        end
    end
end
end
```

```
  6×6×12 string array

ans(:,:,1) =

    "FR"    "FR"    "FR"    "FR"    "FL"    "BR"
    "FR"    "FR"    "FR"    "FR"    "FL"    "BR"
    "FR"    "FR"    "FR"    "FR"    "FL"    "BR"
    "F"     "F"     "F"     "FR"    "N"     "BR"
    "FL"    "FL"    "FL"    "FL"    "FR"    "BL"
    "FL"    "FL"    "FL"    "FL"    "FR"    "BL"


ans(:,:,2) =

    "F"     "FR"    "FR"    "FR"    "FR"    "FR"
    "F"     "F"     "FR"    "FR"    "FR"    "FR"
    "F"     "F"     "F"     "FR"    "FL"    "BR"
    "FL"    "FL"    "FL"    "FR"    "N"     "BR"
    "FL"    "FL"    "FL"    "FR"    "BL"    "BR"
    "FL"    "FL"    "FR"    "BL"    "BR"    "BR"


ans(:,:,3) =

    "FL"    "FL"    "F"     "F"     "FR"    "FR"
    "FL"    "FL"    "FL"    "F"     "FR"    "FR"
    "FL"    "FL"    "FL"    "FL"    "FR"    "FL"
    "FL"    "FL"    "FL"    "FR"    "N"     "BR"
    "FL"    "FR"    "FR"    "BL"    "BR"    "BL"
    "FL"    "BL"    "BL"    "BR"    "BR"    "B"


ans(:,:,4) =

    "FL"    "FL"    "FL"    "FL"    "F"    "FR"
    "FL"    "FL"    "FL"    "FL"    "F"    "FR"
    "FL"    "FL"    "FL"    "FL"    "F"    "FR"
    "FR"    "FR"    "FR"    "FR"    "N"    "FL"
    "BL"    "BL"    "BL"    "BL"    "B"    "BR"
    "BR"    "BR"    "BR"    "BR"    "B"    "BL"
```

```
ans(:,:,5) =

    "FR"    "FL"    "FL"    "FL"    "FL"    "F"
    "BL"    "FR"    "FR"    "FL"    "FL"    "F"
    "BR"    "BL"    "BL"    "FR"    "FL"    "FR"
    "BR"    "BR"    "BR"    "BL"    "N"     "FL"
    "BR"    "BR"    "BR"    "BR"    "BL"    "BR"
    "BR"    "BR"    "BR"    "B"     "BL"    "BL"


ans(:,:,6) =

    "BR"    "BR"    "BL"    "FR"    "FL"    "FL"
    "BR"    "BR"    "BL"    "FR"    "FL"    "FL"
    "BR"    "BR"    "BR"    "BL"    "FR"    "FL"
    "BR"    "BR"    "BR"    "BL"    "N"     "FL"
    "B"     "B"     "B"     "BL"    "BR"    "FL"
    "B"     "B"     "BL"    "BL"    "BL"    "BL"


ans(:,:,7) =

    "BR"    "BR"    "BR"    "BL"    "FR"    "FL"
    "BR"    "BR"    "BR"    "BL"    "FR"    "FL"
    "BR"    "BR"    "BR"    "BL"    "FR"    "FL"
    "B"     "B"     "B"     "BL"    "N"     "FL"
    "BL"    "BL"    "BL"    "BR"    "FL"    "FR"
    "BL"    "BL"    "BL"    "BR"    "FL"    "FR"


ans(:,:,8) =

    "B"     "BR"    "BR"    "BR"    "BR"    "BR"
    "B"     "B"     "BR"    "BR"    "BR"    "BR"
    "B"     "B"     "B"     "BR"    "BL"    "FR"
    "BL"    "BL"    "BL"    "BR"    "N"     "FR"
    "BL"    "BL"    "BL"    "BR"    "FL"    "FR"
    "BL"    "BL"    "BR"    "FL"    "FR"    "FR"


ans(:,:,9) =

    "BL"    "BL"    "B"     "B"     "BR"    "BR"
    "BL"    "BL"    "BL"    "B"     "BR"    "BR"
    "BL"    "BL"    "BL"    "BL"    "BR"    "BL"
    "BL"    "BL"    "BL"    "BR"    "N"     "FR"
    "BL"    "BR"    "BR"    "FL"    "FR"    "FL"
    "BL"    "FL"    "FL"    "FR"    "FR"    "F"


ans(:,:,10) =

    "BL"    "BL"    "BL"    "BL"    "B"    "BR"
    "BL"    "BL"    "BL"    "BL"    "B"    "BR"
```

```
    "BR"    "BR"    "BR"    "BR"    "B"    "BL"
    "FL"    "FL"    "FL"    "FL"    "N"    "FR"
    "FR"    "FR"    "FR"    "FR"    "F"    "FL"
    "FR"    "FR"    "FR"    "FR"    "F"    "FL"


ans(:,:,11) =

    "BR"    "BL"    "BL"    "BL"    "BL"    "B"
    "FL"    "BR"    "BR"    "BL"    "BL"    "B"
    "FR"    "FL"    "FL"    "BR"    "BL"    "BR"
    "FR"    "FR"    "FR"    "FL"    "N"     "BL"
    "FR"    "FR"    "FR"    "FR"    "FL"    "FR"
    "FR"    "FR"    "FR"    "F"     "FL"    "FL"


ans(:,:,12) =

    "FR"    "FR"    "FL"    "BR"    "BL"    "BL"
    "FR"    "FR"    "FL"    "BR"    "BL"    "BL"
    "FR"    "FR"    "FR"    "FL"    "BR"    "BL"
    "FR"    "FR"    "FR"    "FL"    "N"     "BL"
    "F"     "F"     "F"     "FL"    "FR"    "BL"
    "F"     "F"     "FL"    "FL"    "FL"    "FL"
```

Published with MATLAB® R2017b

```
function [Values] = PolicyEval(Pi,Lambda,pe)
%Evaluates the policy Pi given a Lambda and error probability pe.
%   Initialize values of all states to zero, then given the reward of each
%   state: Repeatedly Update the values of each state with the sum of the
%   transfer probability times (reward(state)+lambda*value(next state)).
%   Continue until the largest change in value of any state is less than
%   0.0001. Uses rewards for problem 3.
Values=zeros(6,6,12);
OldValues=Values+1;
%For maximum accuracy, uncomment the next line and comment the following.
% while sum(sum(sum(OldValues==Values)))~=432 %Run until no values change
while sum(sum(sum(abs((OldValues-Values))<.0001)))~=432 %run until none of the values change by more than 0.0001
    OldValues=Values;
    Values=zeros(6,6,12);
    for i=0:5 %X
        for j=0:5 %Y
            for k=0:11 %H
                for l=0:5 %X'
                    for m=0:5 %Y'
                        for n=0:11 %H'
                            Values(i+1,j+1,k+1)=Values(i+1,j+1,k+1)+TransProb2(pe,[i,j,k],Pi(i+1,j+1,k+1),[l,m,n])*(Reward([i,j,k])+Lambda*OldValues(l+1,m+1,n+1));
                        end
                    end
                end
            end
        end
    end
%sum(sum(sum(abs((OldValues-Values)<.0001))))
end
end
```

```
Not enough input arguments.

Error in PolicyEval (line 21)
                            Values(i+1,j+1,k+1)=Values(i+1,j+1,k+1)+TransProb2(pe,[i,j,k],Pi(i+1,j+1,k+1),[l,m,n])*(Reward([i,j,k])+Lambda*OldValues(l+1,m+1,n+1));
```

Published with MATLAB® R2017b

```
function [Values] = PolicyEval(Pi,Lambda,pe)
%Evaluates the policy Pi given a Lambda and error probability pe.
%   Initialize values of all states to zero, then given the reward of each
%   state: Repeatedly Update the values of each state with the sum of the
%   transfer probability times (reward(state)+lambda*value(next state)).
%   Continue until the largest change in value of any state is less than
%   0.0001. Uses rewards for problem 5b.
Values=zeros(6,6,12);
OldValues=Values+1;
%For maximum accuracy, uncomment the next line and comment the following.
% while sum(sum(sum(OldValues==Values)))~=432 %Run until no values change
while sum(sum(sum(abs((OldValues-Values))<.0001)))~=432 %run until none of the values change by more than 0.0001
    OldValues=Values;
    Values=zeros(6,6,12);
    for i=0:5 %X
        for j=0:5 %Y
            for k=0:11 %H
                for l=0:5 %X'
                    for m=0:5 %Y'
                        for n=0:11 %H'
                            Values(i+1,j+1,k+1)=Values(i+1,j+1,k+1)+TransProb2(pe,[i,j,k],Pi(i+1,j+1,k+1),[l,m,n])*(Reward2([i,j,k])+Lambda*OldValues(l+1,m+1,n+1));
                        end
                    end
                end
            end
        end
    end
end
end
```

```
Not enough input arguments.

Error in PolicyEval2 (line 21)
                            Values(i+1,j+1,k+1)=Values(i+1,j+1,k+1)+TransProb2(pe,[i,j,k],Pi(i+1,j+1,k+1),[l,m,n])*(Reward2([i,j,k])+Lambda*OldValues(l+1,m+1,n+1));
```

Published with MATLAB® R2017b

```
function [PiOptimal] = PolicyIteration(PiNot,Lambda,Pe)
%Policy iteration algorithm Outputs the optimal policy PiOptimal given an
%initial policy PiNot, Lambda, and error probability Pe
%   Repeatedly does policy evaluation and generates an improved policy
%   based upon the policy evaluation.  Loop continues until the policy does
%   not change in an iteration
Pi=PiNot;
Loop=true;
while Loop==true;
    CurrentEval=PolicyEval(Pi,Lambda,Pe);
    PiNew=NextPi2(CurrentEval,Pe);
    if sum(sum(sum(PiNew==Pi)))==432;
        Loop=false;
    end
    %sum(sum(sum(PiNew==Pi))) %lets user check how fast conversion is occuring
    Pi=PiNew;
end
PiOptimal=Pi;
end
```

```
Not enough input arguments.

Error in PolicyIteration (line 7)
Pi=PiNot;
```

Published with MATLAB® R2017b

```
function [PiOptimal] = PolicyIteration2(PiNot,Lambda,Pe)
%Policy iteration algorithm Outputs the optimal policy PiOptimal given an
%initial policy PiNot, Lambda, and error probability Pe
%   Repeatedly does policy evaluation and generates an improved policy
%   based upon the policy evaluation.  Loop continues until the policy does
%   not change in an iteration.  Goal is at (3,4) but pointing downward
Pi=PiNot;
Loop=true;
while Loop==true;
    CurrentEval=PolicyEval(Pi,Lambda,Pe);
    PiNew=NextPi3(CurrentEval,Pe);
    if sum(sum(sum(PiNew==Pi)))==432;
        Loop=false;
    end
    %sum(sum(sum(PiNew==Pi))) %lets user check how fast conversion is occuring
    Pi=PiNew;
end
PiOptimal=Pi;
end
```

```
Not enough input arguments.

Error in PolicyIteration2 (line 7)
Pi=PiNot;
```

Published with MATLAB® R2017b

```
function [value] = Reward(s)
%Outputs a reward given the prompt grid world and an input state s (x,y,h)
%   Edge reward=-100.  Lane marker reward=-10.  Goal reward=+1.  All other
%   rewards=0.
if s(1)==0 || s(1)==5 || s(2)==0 || s(2)==5;
    value=-100;
elseif s(1)==1 || s(2)==1;
    value=0;
elseif s(1)==2 || s(1)==4;
    value=-10;
elseif s(1)==3 && s(2)==4;
    value=1;
else
    value=0;
end
end
```

```
Not enough input arguments.

Error in Reward (line 5)
if s(1)==0 || s(1)==5 || s(2)==0 || s(2)==5;
```

Published with MATLAB® R2017b

```
function [value] = Reward(s)
%Outputs a reward given the prompt grid world and an input state s (x,y,h)
%   Edge reward=-100.  Lane marker reward=-10.  Goal reward pointing down=+1.  All other
%   rewards=0.
if s(1)==0 || s(1)==5 || s(2)==0 || s(2)==5;
    value=-100;
elseif s(1)==1 || s(2)==1;
    value=0;
elseif s(1)==2 || s(1)==4;
    value=-10;
elseif s(1)==3 && s(2)==4
    if s(3)==5 || s(3)==6 || s(3)==7;
        value=1;
    end
else
    value=0;
end
end
```

```
Not enough input arguments.

Error in Reward2 (line 5)
if s(1)==0 || s(1)==5 || s(2)==0 || s(2)==5;
```

Published with MATLAB® R2017b

```
function [P_saSPrime] = TransProb2(pe,s,a,sprime)
```

```
%Provides the transition probability of state S' given an error
%probability, previous state, previous action, and S'
%   Returns the probability of a state given 12 possible actions
%   representing.  Functions by generating all possible outcomes (x#,y#,h#, and there
%   given probability p# for any combination of inputs.

%a=convertStringsToChars(a);
a=lower(a); %allows ignoring of upper vs lower case
x=s(1); y=s(2); h=s(3); %dummy variables for x, y, and h states before move
xnew=[x,x,x]; ynew=[y,y,y]; hnew=[h,h,h]; p=[0,0,0]; %initialize variables for possible outcomes
P_saSPrime=0; %Unless one of the following conditions is true, the transition probability is zero
% a1=a(1); %break up the actions into forward, neutral, back, and left, neutral, right
% if length(a)==2;
%     a2=a(2);
% else
%     a2='n';
% end


%%account for the option not to move
if a=="n"
    hnew=[mod(h,12),0,0]; p=[1,0,0]; %don't move with certainty.  Only outcome
end

%%account for left/right/no turn after moving
if ((a=="fl") || (a=="bl")) %try to turn left after moving
    hnew=[mod(h-1,12),mod(h,12),mod(h-2,12)]; %No prerotation error, right error, left error
    p=[1-2*pe,pe,pe]; %Corresponding frequencies
elseif ((a=="fr")||(a=="br")) %try to turn right after moving
    hnew=[mod(h+1,12),mod(h+2,12),mod(h,12)]; %No prerotation error, right error, left error
    p=[1-2*pe,pe,pe]; %Corresponding frequencies
elseif ((a=="f") || (a=="b")) %move but don't turn
    hnew=[mod(h,12),mod(h+1,12),mod(h-1,12)]; %No prerotation error, right error, left error
    p=[1-2*pe,pe,pe]; %Corresponding frequencies
end

%%Determine position based on forward/backward and heading
if ((a=="f")||(a=="fl")||(a=="fr")) %try to move forward;
    switch h
        case 0
            xnew=[x,x,x];
            ynew=[y+1,y+1,y+1];
        case 1
            xnew=[x,x+1,x];
            ynew=[y+1,y,y+1];
        case 2
            xnew=[x+1,x+1,x];
            ynew=[y,y,y+1];
        case 3
            xnew=[x+1,x+1,x+1];
            ynew=[y,y,y];
        case 4
```

```
            xnew=[x+1,x,x+1];
            ynew=[y,y-1,y];
        case 5
            xnew=[x,x,x+1];
            ynew=[y-1,y-1,y];
        case 6
            xnew=[x,x,x];
            ynew=[y-1,y-1,y-1];
        case 7
            xnew=[x,x-1,x];
            ynew=[y-1,y,y-1];
        case 8
            xnew=[x-1,x-1,x];
            ynew=[y,y,y-1];
        case 9
            xnew=[x-1,x-1,x-1];
            ynew=[y,y,y];
        case 10
            xnew=[x-1,x,x-1];
            ynew=[y,y+1,y];
        case 11
            xnew=[x,x,x-1];
            ynew=[y+1,y+1,y];
    end
elseif ((a=="b")||(a=="bl")||(a=="br")) %move backward
    switch h
        case 0
            xnew=[x,x,x];
            ynew=[y-1,y-1,y-1];
        case 1
            xnew=[x,x-1,x];
            ynew=[y-1,y,y-1];
        case 2
            xnew=[x-1,x-1,x];
            ynew=[y,y,y-1];
        case 3
            xnew=[x-1,x-1,x-1];
            ynew=[y,y,y];
        case 4
            xnew=[x-1,x,x-1];
            ynew=[y,y+1,y];
        case 5
            xnew=[x,x,x-1];
            ynew=[y+1,y+1,y];
        case 6
            xnew=[x,x,x];
            ynew=[y+1,y+1,y+1];
        case 7
            xnew=[x,x+1,x];
            ynew=[y+1,y,y+1];
        case 8
            xnew=[x+1,x+1,x];
            ynew=[y,y,y+1];
        case 9
            xnew=[x+1,x+1,x+1];
            ynew=[y,y,y];
        case 10
```

```
            xnew=[x+1,x,x+1];
            ynew=[y,y-1,y];
        case 11
            xnew=[x,x,x+1];
            ynew=[y-1,y-1,y];
    end
end

%%Prevent going out of bounds;
xnew=xnew+(xnew<0)-(xnew>5);
ynew=ynew+(ynew<0)-(ynew>5);
```

```
Not enough input arguments.

Error in TransProb2 (line 9)
a=lower(a); %allows ignoring of upper vs lower case
```

```
%Sum the probability of outcomes giving sprime
if sprime(1)==xnew(1) && sprime(2)==ynew(1) && sprime(3)==hnew(1);
    P_saSPrime=P_saSPrime+p(1);
end
if sprime(1)==xnew(2) && sprime(2)==ynew(2) && sprime(3)==hnew(2);
    P_saSPrime=P_saSPrime+p(2);
end
if sprime(1)==xnew(3) && sprime(2)==ynew(3) && sprime(3)==hnew(3);
    P_saSPrime=P_saSPrime+p(3);
end
```

```
end
```

Published with MATLAB® R2017b

```
function [PiStar,VStar] = ValueIteration(Lambda,pe)
%Calculates the optimal policy PiStar and correspond values VStar for a
%given Lambda and error probability pe.  Uses initial values of 0.
%   Detailed explanation goes here
VOld=zeros(6,6,12); %Initialize all values to 0
check=true; %loop until condition
Policy=strings(6,6,12);
while check
    Policy=strings(6,6,12);
    for i=0:5 %x
        for j=0:5 %y
            for k=0:11 %h
                c=-10^100; %dummy value of negative infinity to compare against
                for a=["FL","F","FR","BL","B","BR"] %consider all moves
                    b=0;
                    for l=0:5 %x'
                        for m=0:5 %y'
                            for n=0:11 %h'
                                b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*(Reward([i,j,k])+Lambda*VOld(l+1,m+1,n+1));
                            end
                        end
                    end
                    if b>c
                        c=b;
                        VNew(i+1,j+1,k+1)=c;
                        Policy(i+1,j+1,k+1)=a;
                    end
                    if sum([i,j]==[3,4])==2 %Set the action for the goal to be no move, and update value
                        VNew(i+1,j+1,k+1)=TransProb2(pe,[i,j,k],"N",[i,j,k])*(Reward([i,j,k])+Lambda*VOld(i+1,j+1,k+1));
                        Policy(i+1,j+1,k+1)="N";
                    end
                end
            end
        end
    end
    if sum(sum(sum(abs(VOld-VNew)<0.0001)))==432 %if V has converged
        check=false; %stop looping
        PiStar=Policy;
        VStar=VNew;
    end
    %CheckConverge=sum(sum(sum(abs(VOld-VNew)<0.0001))) %lets user see convergence
    VOld=VNew;
end

end
```

```
Not enough input arguments.

Error in ValueIteration (line 19)
                                b=b+TransProb2(pe,[i,j,k],a,[l,m,n])*(Reward([i,j,k])+Lambda*VOld(l+1,m+1,n+1));
```

Published with MATLAB® R2017b