ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for
Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme
und Regelungstechnik

Dynamic Programming and Optimal Control

Fall 2009

Problem Set:
Infinite Horizon Problems, Value Iteration, Policy Iteration

Notes:

- Problems marked with BERTSEKAS are taken from the book *Dynamic Programming and Optimal Control by Dimitri P. Bertsekas, Vol. I, 3rd edition, 2005, 558 pages, hardcover.*
- The solutions were derived by the teaching assistants. Please report any error that you may find to `strimpe@ethz.ch` or `aschoellig@ethz.ch`.
- The solutions in this document are handwritten. A typed version will be available in future.

Problem Set

Problem 1 (BERTSEKAS, p. 445, exercise 7.1)

Problem 2 (BERTSEKAS, p. 446, exercise 7.3)

Problem 3 (BERTSEKAS, p. 448, exercise 7.12)

Problem 4 (BERTSEKAS, p. 60, exercise 1.23)

Angela Schöllig

Exercise 7.1

- First, recist the tennis rules... ☺

For a stochastic shortest path problem, we need to define
(a) set of states
(b) transition probabilities
(c) control input
(d) cost per stage / step

(a) state space

- $x_k = (u_k, v_k, w_k) \in \{0,1,2,3\} \times \{0,1,2,3\} \times \{1,2\}$

with u_k are points of player 1
v_k " " " " 2
w_k denotes first or second serve of player 1

Note $u_k = 1$ corresponds to score of player 1 is 15
$u_k = 2$ " " " " " 1 " 30
$u_k = 3$ " " " " " 1 " 40.

Note $(3,3,w_k)$ corresponds also to "deuce" situation.
$(3,2,w_k)$ corresponds also to "advantage" for player 1.
$(2,3,w_k)$ corresponds also to "advantage" for player 2.

- We have <u>two</u> termination states
$x_k = g_1$ player 1 wins the game corresponds to $(4,2,w_k)$
$x_k = g_2$ " 2 " " " " " $(2,4,w_k)$

Angela Schöllig

(c) control input

- $u_k \in \{F, S\}$ Fast serve or Slow serve

(b) transition probabilities

Let's consider which transitions are possible and assign probabilities p and q - not yet specifying which input was applied.

conditional probabilities

- $(u_k, v_k, 1) \xrightarrow{qp} (u_k+1, v_k, 1)$ for $u_k \neq 3$, $v_k \neq 3$
  $\xrightarrow{(1-q)p} (u_k, v_k+1, 1)$
  $\xrightarrow{(1-p)} (u_k, v_k, 2)$
- $(u_k, v_k, 2) \xrightarrow{qp} (u_k+1, v_k, 1)$ for $u_k \neq 3$, $v_k \neq 3$
  $\xrightarrow{(1-q)p+(1-p)} (u_k, v_k+1, 1)$
- $(3, v_k, 1) \xrightarrow{qp} g_1$ for $v_k \neq 3$
  $\xrightarrow{(1-q)p} (3, v_k+1, 1)$
  $\xrightarrow{(1-p)} (3, v_k, 2)$
- $(3, v_k, 2) \xrightarrow{qp} g_1$ for $v_k \neq 3$
  $\xrightarrow{(1-q)p+(1-p)} (3, v_k+1, 1)$
- $(u_k, 3, 1) \xrightarrow{qp} (u_k+1, 3, 1)$ for $u_k \neq 3$
  $\xrightarrow{(1-q)p} g_2$
  $\xrightarrow{(1-p)} (u_k, 3, 2)$
- $(u_k, 3, 2) \xrightarrow{qp} (u_k+1, 3, 1)$ for $u_k \neq 3$
  $\xrightarrow{(1-q)p+(1-p)} g_2$
- $(3, 3, 1) \xrightarrow{qp} (3, 2, 1)$ "advantage"
  $\xrightarrow{(1-q)p} (2, 3, 1)$ "
  $\xrightarrow{(1-p)} (3, 3, 2)$
- $(3, 3, 2) \xrightarrow{qp} (3, 2, 1)$ "
  $\xrightarrow{(1-q)p+(1-p)} (2, 3, 1)$

Angela Schöllig

(d) cost per stage

- The objective is to maximize the probability of winning the game, i.e. we do not have stage costs but rather can express this goal by defining the optimal cost

$$J^*(g_1) = 1$$
$$J^*(g_2) = 0$$

In all equations, we always can replace "min" by "max".

- ⇒ Assumption 7.2.1. is satisfied since from each state defined above there is a positive probability to end the game.

⇒ Bellman's Equation:

$$J^*(3,3,2) = \max_{u \in \{S,F\}} \left(q_u p_u J^*(3,2,1) + (1 - q_u p_u) J^*(2,3,1) \right)$$

$$J^*(3,3,1) = \max_{u \in \{S,F\}} \left(q_u p_u J^*(3,2,1) + (1-q_u) p_u J^*(2,3,1) + (1-p_u) J^*(3,3,2) \right)$$

$$J^*(u_k,3,2) = \max_{u \in \{S,F\}} \left(q_u p_u J^*(u_k+1,3,1) \right)$$

for $u_k \neq 3$

$$J^*(u_k,3,1) = \max_{u \in \{S,F\}} \left(q_u p_u J^*(u_k+1,3,1) + (1-p_u) J^*(u_k,3,2) \right)$$

for $u_k \neq 3$

$$J^*(3,v_k,2) = \max_{u \in \{S,F\}} \left(q_u p_u + (1 - q_u p_u) J^*(3,v_k+1,1) \right)$$

for $v_k \neq 3$

$$J^*(3,v_k,1) = \max_{u \in \{S,F\}} \left(q_u p_u + (1-q_u) p_u J^*(3,v_k+1,1) + (1-p_u) J^*(3,v_k,2) \right)$$

Angela Schöllig

for $v_k \neq 3$

$$J^*(u_k, v_k, 2) = \max_{u \in \{S, F\}} \left(q_u p_u J^*(u_{k+1}, v_k, 1) + (1 - q_u p_u) J^*(u_k, v_{k+1}, 1) \right)$$

for $u_k \neq 3, v_k \neq 3$

$$J^*(u_k, v_k, 1) = \max_{u \in \{S, F\}} \left(q_u p_u J^*(u_{k+1}, v_k, 1) + (1 - q_u p_u) J^*(u_k, v_{k+1}, 1) + (1 - p_u) J^*(u_k, v_k, 2) \right)$$

for $u_k \neq 3, v_k \neq 3$

b) see Code

Plot shows three different strategies

(1) always slow serve
(2) slow serve in first serve,
fast " " 2nd "
(3) always fast serve

```
% Solution to Exercise 7.1 b)
% in Bertsekas - "Dynamic Programming and Optimal Control" Vol. 1 p. 445
%
% --
% ETH Zurich
% Institute for Dynamic Systems and Control
% Angela Schöllig
% aschoellig@ethz.ch
%
% --
% Revision history
% [12.11.09, AS]    first version
%
%


% clear workspace and command window
clear;
clc;

%% PARAMETERS
% Landing probability
p(2) = 0.95; % Slow serve

% Winning probability
q(1) = 0.6; % Fast serve
q(2) = 0.4; % Slow serve

% Define value iteration error bound
err = 1e-100;


% Specifiy if you want to test one value for p(1) - landing probability for
% fast serve: use test = 1, or a row of probabilities: use test = 2
test =1;

if test ==1
    % Define p(1)
    p(1) = 0.2;
    % Number of runs
    mend = 1;
else
    % Define increment for p(1)
    incr = 0.01;
    % Number of runs
    mend = ((0.95-incr)/incr+2);
end



%% VALUE ITERATION
for m=1:mend
        %% PARAMETER
        % Landing probability
        if test == 2
```

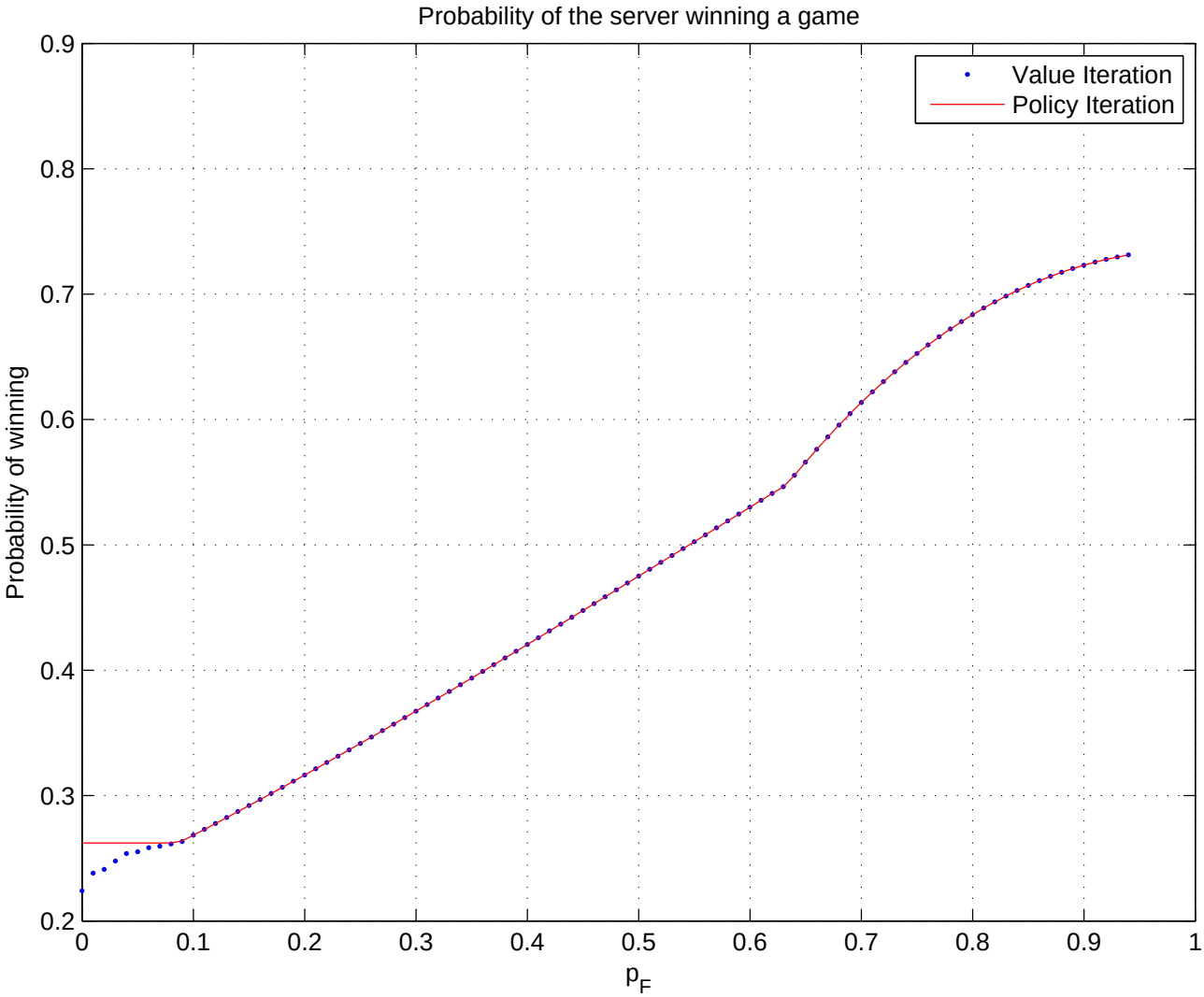
```
            p(1) = (m-1)*incr; % Fast serve, check for 0...0.9
        end


        %% INITIALIZE PROBLEM
        % Our state space is S={0,1,2,3}x{0,1,2,3}x{1,2},
        % i.e. x_k = [score player 1, score player 2, serve]
        % ATTENTION: indices are shifted in code

        % Set the initial costs to 0, although any value will do (except for the
        % termination cost, which must be equal to 0).
        J = (p(1)+0.1).*ones(4, 4, 2);

        % Initialize the optimal control policy: 1 represents Fast serve, 2
        % represents Slow serve
        FVal = ones(4, 4, 2);

        %% POLICY ITERATION

        % Initialize cost-to-go
        costToGo = zeros(4, 4, 2);

        iter = 0;
        while (1)
            iter = iter+1;
            if (mod(iter,1e3)==0)
                disp(['Value Iteration Number ',num2str(iter)]);
            end

            % Update the value
            for i = 1:3
                    [costToGo(4,i,1),FVal(4,i,1)] = max(q.*p + (1-q).*p.*J(4,i+1,1)+↙
(1-p).*J(4,i,2));
                    [costToGo(4,i,2),FVal(4,i,2)] = max(q.*p + (1-q.*p).*J(4,i+1,1));
                    [costToGo(i,4,1),FVal(i,4,1)] = max(q.*p.*J(i+1,4,1) + (1-p).*J↙
(i,4,2));
                    [costToGo(i,4,2),FVal(i,4,2)] = max(q.*p.*J(i+1,4,1));
                for j = 1:3
                    % Maximize over input 1 and 2
                    [costToGo(i,j,1),FVal(i,j,1)] = max(q.*p.*J(i+1,j,1) + (1-q).*p.↙
*J(i,j+1,1)+(1-p).*J(i,j,2));
                    [costToGo(i,j,2),FVal(i,j,2)] = max(q.*p.*J(i+1,j,1) + (1-q.*p).↙
*J(i,j+1,1));
                end
            end

            [costToGo(4,4,1),FVal(4,4,1)] = max(q.*p.*J(4,3,1) + (1-q).*p.*J(3,4,1)+↙
(1-p).*J(4,4,2));
            [costToGo(4,4,2),FVal(4,4,2)] = max(q.*p.*J(4,3,1) + (1-q.*p).*J(3,4,1));

           % max(max(max(J-costToGo)))/max(max(max(costToGo)))
            if (max(max(max(J-costToGo)))/max(max(max(costToGo))) < err)
                % update cost
                J = costToGo;
                break;
```

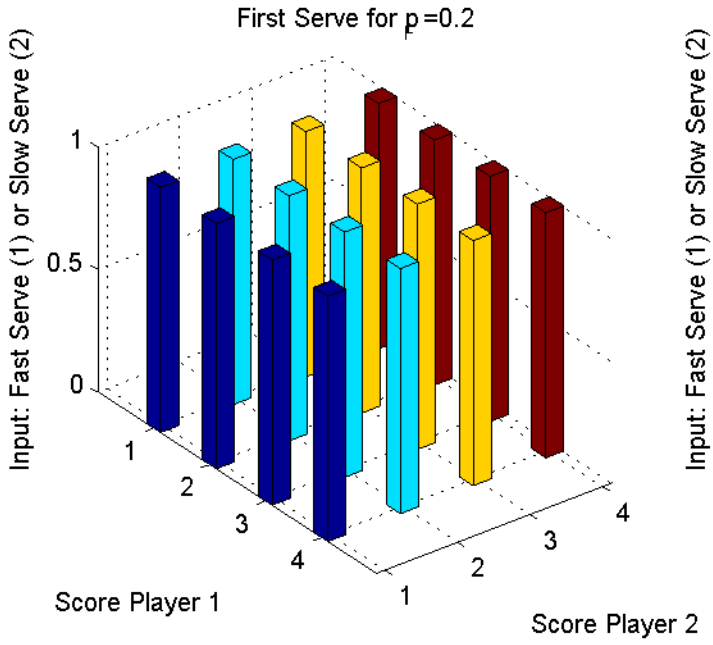
```
            else
                J = costToGo;
            end
        end


        % Probability of player 1 wining the game
        JValsave(m)=J(1, 1, 1);
end;



%% POLICY ITERATION
for m=1:mend
        %% PARAMETER
        % Landing probability
        if test == 2
            p(1) = (m-1)*incr; % Fast serve, check for 0...0.9
        end


        %% INITIALIZE PROBLEM
        % Our state space is S={0,1,2,3}x{0,1,2,3}x{1,2},
        % i.e. x_k = [score player 1, score player 2, serve]
        % ATTENTION: indices are shifted in code


        % Initialize the optimal control policy: 1 represents Fast serve, 2
        % represents Slow serve - in vector form
        FPol = 2.*ones(32,1);

        %% VALUE ITERATION

        iter = 0;
        while (1)
            iter = iter+1;
            disp(['Policy Iteration Number ',num2str(iter)]);
            if (mod(iter,1e3)==0)
                disp(['Policy Iteration Number ',num2str(iter)]);
            end

            % Update the value
            % Initialize matrices
            APol = zeros(32,32);
            bPol = zeros(32,1);
            for i = 1:3
                    % [costToGo(4,i,1),FVal(4,i,1)] = max(q.*p + (1-q).*p.*J(4,i+1,1)↙
+(1-p).*J(4,i,2));
                    f = FPol(12+i);
                    APol(12+i,12+i)=-1;
                    bPol(12+i)= -q(f)*p(f);
                    APol(12+i,13+i)= (1-q(f))*p(f);
                    APol(12+i,28+i)= (1-p(f));
                    % [costToGo(4,i,2),FPol(4,i,2)] = max(q.*p + (1-q.*p).*J(4,i+1,↙
1));
```

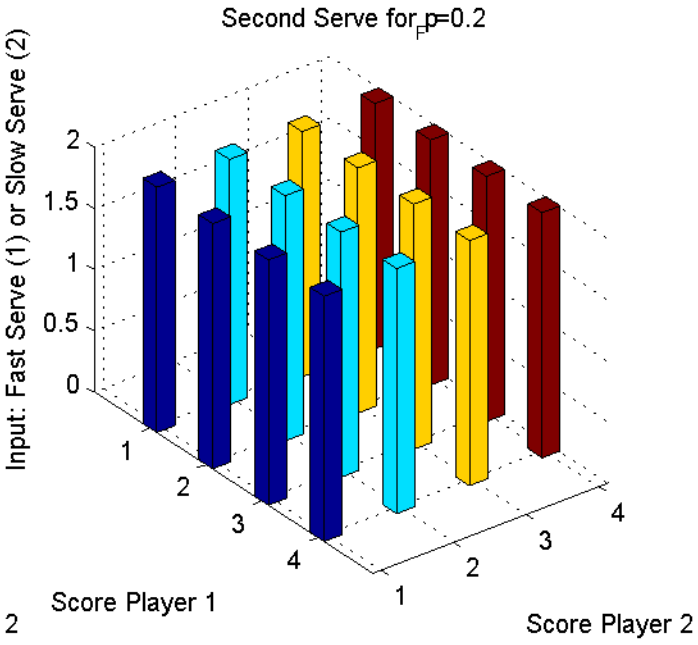
```
                    f = FPol(28+i);
                    APol(28+i,28+i)=-1;
                    bPol(28+i)= -q(f)*p(f);
                    APol(28+i,13+i)= 1-p(f)*q(f);
                    % [costToGo(i,4,1),FPol(i,4,1)] = max(q.*p.*J(i+1,4,1) + (1-p).*J↙
(i,4,2));
                    f = FPol(4*i);
                    APol(4*i,4*i)=-1;
                    APol(4*i,4*i+4)= q(f)*p(f);
                    APol(4*i,20+4*(i-1))= 1-p(f);
                    % [costToGo(i,4,2),FPol(i,4,2)] = max(q.*p.*J(i+1,4,1));
                    f = FPol(4*i+16);
                    APol(4*i+16,4*i+16)=-1;
                    APol(4*i+16,4*i+4)= q(f)*p(f);
                for j = 1:3
                    % Maximize over input 1 and 2
                    % [costToGo(i,j,1),FPol(i,j,1)] = max(q.*p.*J(i+1,j,1) + (1-q).↙
*p.*J(i,j+1,1)+(1-p).*J(i,j,2));
                    f = FPol(4*(i-1)+j);
                    APol(4*(i-1)+j,4*(i-1)+j)=-1;
                    APol(4*(i-1)+j,4*i+j)= q(f)*p(f);
                    APol(4*(i-1)+j,4*(i-1)+j+1)= (1-q(f))*p(f);
                    APol(4*(i-1)+j,4*(i-1)+j+16)= (1-p(f));
                    % [costToGo(i,j,2),FPol(i,j,2)] = max(q.*p.*J(i+1,j,1) + (1-q.↙
*p).*J(i,j+1,1));
                    f = FPol(4*(i-1)+j+16);
                    APol(4*(i-1)+j+16,4*(i-1)+j+16)=-1;
                    APol(4*(i-1)+j+16,4*i+j)= q(f)*p(f);
                    APol(4*(i-1)+j+16,4*(i-1)+j+1)= 1-q(f)*p(f);
                end
            end

            % [costToGo(4,4,1),FPol(4,4,1)] = max(q.*p.*J(4,3,1) + (1-q).*p.*J(3,4,1)↙
+(1-p).*J(4,4,2));
            f = FPol(16);
            APol(16,16)=-1;
            APol(16,15)= q(f)*p(f);
            APol(16,12)= (1-q(f))*p(f);
            APol(16,32)= (1-p(f));
            % [costToGo(4,4,2),FPol(4,4,2)] = max(q.*p.*J(4,3,1) + (1-q.*p).*J↙
(3,4,1));
            f = FPol(32);
            APol(32,32)=-1;
            APol(32,15)= q(f)*p(f);
            APol(32,12)= 1-q(f)*p(f);

            % Calculate cost
            JPol = zeros(32,1);
            JPol = APol\bPol;

            % Can now update the policy.
            FPolNew = FPol;
            JPolNew = JPol;

            JPol_re = reshape(JPol,4,4,2);
```

```
            JPol_re(:,:,1)= JPol_re(:,:,1)';
            JPol_re(:,:,2)= JPol_re(:,:,2)';

            % Initialize cost-to-go
            costToGo = zeros(4, 4, 2);
            FPolNew_re = zeros(4, 4, 2);

            % Update the optimal policy
            for i = 1:3
                    [costToGo(4,i,1),FPolNew_re(4,i,1)] = max(q.*p + (1-q).*p.↙
*JPol_re(4,i+1,1)+(1-p).*JPol_re(4,i,2));
                    [costToGo(4,i,2),FPolNew_re(4,i,2)] = max(q.*p + (1-q.*p).↙
*JPol_re(4,i+1,1));
                    [costToGo(i,4,1),FPolNew_re(i,4,1)] = max(q.*p.*JPol_re(i+1,4,1)↙
+ (1-p).*JPol_re(i,4,2));
                    [costToGo(i,4,2),FPolNew_re(i,4,2)] = max(q.*p.*JPol_re(i+1,↙
4,1));
                for j = 1:3
                    % Maximize over input 1 and 2
                    [costToGo(i,j,1),FPolNew_re(i,j,1)] = max(q.*p.*JPol_re(i+1,j,1)↙
+ (1-q).*p.*JPol_re(i,j+1,1)+(1-p).*JPol_re(i,j,2));
                    [costToGo(i,j,2),FPolNew_re(i,j,2)] = max(q.*p.*JPol_re(i+1,j,1)↙
+ (1-q.*p).*JPol_re(i,j+1,1));
                end
            end

            [costToGo(4,4,1),FPolNew_re(4,4,1)] = max(q.*p.*JPol_re(4,3,1) + (1-q).↙
*p.*JPol_re(3,4,1)+(1-p).*JPol_re(4,4,2));
            [costToGo(4,4,2),FPolNew_re(4,4,2)] = max(q.*p.*JPol_re(4,3,1) + (1-q.↙
*p).*JPol_re(3,4,1));

            JPolNew(1:16) = reshape(costToGo(:,:,1)', 16,1);
            JPolNew(17:32) = reshape(costToGo(:,:,2)', 16,1);
            FPolNew(1:16) = reshape(FPolNew_re(:,:,1)', 16,1);
            FPolNew(17:32) = reshape(FPolNew_re(:,:,2)', 16,1);

            % Quit if the policy is the same, or if the costs are essentially the
            % same.
            if ((norm(JPol - JPolNew)/norm(JPol) < 1e-10) )
                break;
            else
                FPol = FPolNew;
            end
        end


        % Probability of player 1 wining the game
        JPolsave(m)=JPolNew(1,1);


end;
```

```
if test ==2
   plot(0:incr:(0.95-incr), JValsave,'.');
   hold on
   plot(0:incr:(0.95-incr), JPolsave,'r');
   grid on
   title('Probability of the server winning a game')
   xlabel('p_F')
   ylabel('Probability of winning')
   legend('Value Iteration', 'Policy Iteration')
else
    disp(['Probability of player 1 winning ',num2str(JValsave(1))]);

    subplot(1,2,1)
    bar3(FVal(:,:,1),0.25,'detached')
    title(['First Serve for p_F =', num2str(p(1))])
    ylabel('Score Player 1')
    xlabel('Score Player 2')
    zlabel('Input: Fast Serve (1) or Slow Serve (2)')


    subplot(1,2,2)
    bar3(FVal(:,:,2),0.25,'detached')
    title(['Second Serve for p_F =', num2str(p(1))])
    ylabel('Score Player 1')
    xlabel('Score Player 2')
    zlabel('Input: Fast Serve (1) or Slow Serve (2)')
end
```


Probability of the server winning a game
Value Iteration
Policy Iteration
Probability of winning
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
p_F


First Serve for p =0.2
Input: Fast Serve (1) or Slow Serve (2)
1
0.5
0
1
2
3
4
Score Player 1
1
2
3
4
Score Player 2


Second Serve for p=0.2
Input: Fast Serve (1) or Slow Serve (2)
2
1.5
1
0.5
0
1
2
3
4
Score Player 1
1
2
3
4
Score Player 2

11.11.09

Problem 7.3

- states: $i \in S = \{1, 2\}$, 1: sells well, 2: doesn't sell well
- decision / control: $\mu(1) \in U(1) = \{A, \bar{A}\}$
  A: advertise
  $\bar{A}$: don't advertise

  $\mu(2) \in U(2) = \{R, \bar{R}\}$
  R: do research
  $\bar{R}$: don't do research
- rewards and transition probabilities

$i=1, \mu(1)=A$:	$g(1,A) = 4$	$p_{11}(A) = 0.8$	$p_{12}(A) = 0.2$
$i=1, \mu(1)=\bar{A}$:	$g(1,\bar{A}) = 6$	$p_{11}(\bar{A}) = 0.5$	$p_{12}(\bar{A}) = 0.5$
$i=2, \mu(2)=R$:	$g(2,R) = -5$	$p_{21}(R) = 0.7$	$p_{22}(R) = 0.3$
$i=2, \mu(2)=\bar{R}$:	$g(2,\bar{R}) = -3$	$p_{21}(\bar{R}) = 0.4$	$p_{22}(\bar{R}) = 0.6$

a) • Approach: We consider all possible stationary policies μ^i and compute the cost J^i. Then, by Proposition 7.3.1, the policy with the largest reward is an optimal policy since it attains the maximum in Bellman's equation.

Define policy $\mu^i := (\mu(1), \mu(2))$.
→ 4 different stationary policies: (A,R), $(A,\bar{R})$, $(\bar{A},R)$, $(\bar{A},\bar{R})$

$\underline{\mu^1 = (A,R)}$:

$$J(1) = g(1,A) + \alpha\left(p_{11}(A)\cdot J(1) + p_{12}(A)\cdot J(2)\right)$$
$$J(1) = 4 + \alpha\left(0.8\cdot J(1) + 0.2\, J(2)\right)$$

$$J(2) = -5 + \alpha\left(0.7\, J(1) + 0.3\, J(2)\right)$$

Combine into single vector equation:

$$\underbrace{\begin{bmatrix} J(1) \\ J(2) \end{bmatrix}}_{=: J^1} = \begin{bmatrix} 4 \\ -5 \end{bmatrix} + \alpha \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \begin{bmatrix} J(1) \\ J(2) \end{bmatrix}$$

$$\Leftrightarrow \vec{J}^1 = \left(I - \alpha \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}\right)^{-1} \begin{bmatrix} 4 \\ -5 \end{bmatrix}$$

Similarly, obtain the costs for other policies.

$\underline{\mu^2 = (A, R)}$:

$$\vec{J}_4^2 = \left(I - \alpha \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}\right)^{-1} \begin{bmatrix} 4 \\ -3 \end{bmatrix}$$

$\underline{\mu^3 = (\bar{A}, R)}$:

$$\vec{J}^3 = \left(I - \alpha \begin{bmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{bmatrix}\right)^{-1} \begin{bmatrix} 6 \\ -5 \end{bmatrix}$$

$\underline{\mu^4 = (\bar{A}, \bar{R})}$:

$$\vec{J}^4 = \left(I - \alpha \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix}\right)^{-1} \begin{bmatrix} 6 \\ -3 \end{bmatrix}$$

Reformulate inverse $(I - \alpha M)^{-1}$ such that we can then take limits. M has a special structure:

$$M = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}$$

$$\rightarrow (I - \alpha M) = \begin{bmatrix} 1-\alpha+\alpha p & -\alpha p \\ -\alpha q & 1-\alpha+\alpha q \end{bmatrix}$$

$$(I - \alpha M)^{-1} = \frac{1}{\underset{-\alpha^2 pq}{(1-\alpha+\alpha p)(1-\alpha+\alpha q)}} \begin{bmatrix} 1-\alpha+\alpha q & \alpha p \\ \alpha q & 1-\alpha+\alpha p \end{bmatrix}$$

Denominator: $1 - \alpha + \alpha q - \alpha + \alpha^2 - \alpha^2 q + \alpha p - \alpha^2 p + \alpha^2 pq - \alpha^2 pq$

$$= 1 - 2\alpha + \alpha^2 + \alpha q + \alpha p - \alpha^2 q - \alpha^2 p$$
$$= (1-\alpha)^2 + (1-\alpha)\alpha(q+p)$$
$$= (1-\alpha)(1 - \alpha + \alpha(q+p))$$

$$\Rightarrow \lim_{\alpha \to 0} (I - \alpha M)^{-1} = I \quad , \quad \lim_{\alpha \to 1} (I - \alpha M)^{-1} = \frac{1}{(1-\alpha)(p+q)} \begin{bmatrix} q & p \\ q & p \end{bmatrix}$$

Case 1: α sufficiently small: $\alpha \to 0$

$$\lim_{\alpha\to 0} \bar{J}^1 = \begin{bmatrix} 4 \\ -5 \end{bmatrix}, \quad \lim_{\alpha\to 0} \bar{J}^2 = \begin{bmatrix} 4 \\ -3 \end{bmatrix}$$

$$\lim_{\alpha\to 0} \bar{J}^3 = \begin{bmatrix} 6 \\ -5 \end{bmatrix}, \quad \lim_{\alpha\to 0} \bar{J}^4 = \begin{bmatrix} 6 \\ -3 \end{bmatrix}$$

"element-wise"

$$\Rightarrow \lim_{\alpha\to 0} \bar{J}^4 \geq \lim_{\alpha\to 0} \bar{J}^j, \quad j = 1, \dots, 3$$

$\Rightarrow$ "shortsighted" policy $\mu^4 = (\bar{A}, \bar{R})$ is optimal

Case 2: α sufficiently close to 1: $\alpha \to 1$

$$\lim_{\alpha\to 1} \bar{J}^1 = \frac{1}{(1-\alpha)0.9} \begin{bmatrix} 0.7 & 0.2 \\ 0.7 & 0.2 \end{bmatrix} \begin{bmatrix} 4 \\ -5 \end{bmatrix} = \frac{1.8}{0.9} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= 2 \cdot \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lim_{\alpha\to 1} \bar{J}^2 = \frac{1.0}{0.6} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{5}{3} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lim_{\alpha\to 1} \bar{J}^3 = \frac{4.2 - 2.5}{1.2} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{17}{12} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lim_{\alpha\to 1} \bar{J}^4 = \frac{0.9}{0.9} \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \frac{1}{1-\alpha} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\Rightarrow \lim_{\alpha\to 1} \bar{J}^1 \geq \lim_{\alpha\to 1} \bar{J}^j, \quad j = 2, 3, 4$$

$\Rightarrow$ "farsighted" policy $\mu^1 = (\bar{A}, R)$ is optimal

b) Policy Iteration with $\alpha = 0.9$

- Start iteration with stationary policy $\mu^0 = (\mu^0(1), \mu^0(2)) = (\bar{A}, \bar{R})$

- Step 1: Policy Evaluation Step

$$J(1) = g(1, \mu^0(1)) + \alpha \left(p_{11}(\mu^0(1)) \cdot J(1) + p_{12}(\mu^0(1)) \cdot J(2) \right)$$
$$J(2) = g(2, \mu^0(2)) + \alpha \left(p_{21}(\mu^0(2)) \cdot J(1) + p_{22}(\mu^0(2)) \, J(2) \right)$$

$$J_{\mu_0} = \begin{bmatrix} J(1) \\ J(2) \end{bmatrix} = \begin{bmatrix} g(1, \mu^0(1)) \\ g(2, \mu^0(2)) \end{bmatrix} + \alpha \begin{bmatrix} p_{11}(\mu^0(1)) & p_{12}(\mu^0(1)) \\ p_{21}(\mu^0(2)) & p_{22}(\mu^0(2)) \end{bmatrix} J_{\mu_0}$$

$$\Rightarrow J_{\mu_0} = \left(I - \alpha \begin{bmatrix} p_{11}(\mu_0(1)) & p_{12}(\mu^0(1)) \\ p_{21}(\mu_0(2)) & p_{22}(\mu^0(2)) \end{bmatrix} \right)^{-1} \begin{bmatrix} g(1, \mu^0(1)) \\ g(2, \mu^0(2)) \end{bmatrix}$$

$$\underline{J_{\mu_0}} = \left(I - \alpha \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} 6 \\ -3 \end{bmatrix} \approx \underline{\begin{bmatrix} 15.49 \\ 5.60 \end{bmatrix}}$$

- Step 2: Policy Improvement Step

$$\mu^{k+1}(i) = \arg\max_{u \in U(i)} \left(g(i,u) + \alpha \sum_{j=1}^{2} p_{ij}(u) \, J_{\mu^k}(j) \right) \qquad i = 1, 2$$

$$\mu^1(1) = \arg\max \left\{ 4 + 0.9 \left(0.8 \cdot J_{\mu_0}(1) + 0.2 \, J_{\mu_0}(2) \right), \right.$$
$$\left. 6 + 0.9 \left(0.5 \, J_{\mu_0}(1) + 0.5 \, J_{\mu_0}(2) \right) \right\}$$

$$= \arg\max \{ 16.2, 15.5 \} = A$$

$$\mu^1(2) = \arg\max \left\{ -5 + 0.9 \left(0.7 \cdot J_{\mu_0}(1) + 0.3 \cdot J_{\mu_0}(2) \right), \right.$$
$$\left. -3 + 0.9 \left(0.4 \, J_{\mu_0}(1) + 0.6 \, J_{\mu_0}(2) \right) \right\}$$

$$= \arg\max \{ 6.27, 5.6 \} = R$$

$\rightarrow \underline{\mu^1 = (A, R)}$

- Step 1: Policy Evaluation Step #2

$$\underline{J_{\mu^1}} = \left(I - 0.9 \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}\right)^{-1} \begin{bmatrix} 4 \\ -5 \end{bmatrix} = \underline{\begin{bmatrix} 22.20 \\ 12.31 \end{bmatrix}}$$

- Step 2: Policy improvement Step #2

$$\mu^2(1) = \arg\max \{ 4 + 0.9 (0.8\, J_{\mu^1}(1) + 0.2\, J_{\mu^1}(2)),$$
$$6 + 0.9 (0.5\, J_{\mu^1}(1) + 0.5\, J_{\mu^1}(2)) \}$$
$$= \arg\max \{ 22.2, 21.53 \} = A$$

$$\mu^2(2) = \arg\max \{ -5 + 0.9 (0.7\, J_{\mu^1}(1) + 0.3\, J_{\mu^1}(2)),$$
$$-3 + 0.9 (0.4\, J_{\mu^1}(1) + 0.6\, J_{\mu^1}(2)) \}$$
$$= \arg\max \{ 12.31, 11.64 \} = R$$

$\rightarrow \underline{\mu^2 = (A, R)}$

- Next policy evaluation step would yild the same result, i.e.

$$J_{\mu^2}(i) = J_{\mu^1}(i) \quad \forall\, i = 1, 2$$

Therefore, we can terminate. The optimal policy is (A, R).

c) Implementation in Matlab: see printout

Result for $\alpha = 0.99$:

$J^*(1) = 202.22$, $J^*(2) = 192.23$

$\mu^* = (A, R)$.

```
% script_P73c.m
%
% Matlab Script with implementation of a solution to Problem 7.3c from the
% class textbook.
%
% Dynamic Programming and Optimal Control
% Fall 2009
% Problem Set, Chapter 7
% Problem 7.3 c
%
% --
% ETH Zurich
% Institute for Dynamic Systems and Control
% Sebastian Trimpe
% strimpe@ethz.ch
%
% --
% Revision history
% [11.11.09, ST]    first version
%
%


% clear workspace and command window
clear;
%clc;


%% Program control
% Flag defining whether we run the value iteration with or without error
% bounds.
flag_errbnds = true;
% False: No error bounds; fixed number of iterations MAX_ITER.
% True: Use error bounds.  Terminate when result within tolerance ERR_TOL.

MAX_ITER = 1000;
ERR_TOL = 1e-6;


%% Setup problem data
% Stage costs g.  g is a 2x2 matrix where the first index correponds to the
% state i=1,2 and the second index correponds to the admissible control
% input u=1 (=advertising/do research), or 2 (=don't advertise/don't do
% research).
g = [4 6; -5 -3];

% Transition probabilities.  P is a 2x2x2 matrix, where the first index
% corresponds to the origin state, the second index corresponds to the
% destination state and the third input corresponds to the applied control
% input where (1,2) maps to (advertise/do research, don't advertise/don't
% do research).  For example, the probability of transition from node 1 to
% node 2 given that we do not advertice is P(1,2,2).
P = zeros(2,2,2);

% Advertise/do research (u=1):
```

```
P(:,:,1) = [0.8 0.2; 0.7 0.3];

% Don't advertise/don't do research (u=2):
P(:,:,2) = [0.5 0.5; 0.4 0.6];

% Discount factor.
%alpha = 0.9;
alpha = 0.99;


%% Value Iteration
% Initialize variables that we update during value iteration.
% Cost (here it really is the reward):
costJ = [0,0];
costJnew = [0,0];

% Policy
policy = [0,0];

% Control variable for breaking the loop
doBreak = false;


% Loop over value iterations k.
for k=1:MAX_ITER

    for i=1:2   % loop over two states
        % One value iteration step for each state.
        [costJnew(i),policy(i)] = max( g(i,:) + alpha*costJ*squeeze(P(i,:,:)) );
    end;

    % Save results for plotting later.
    res_J(k,:) = costJnew;

    % Construct string to be displayed later:
    dispstr = ['k=',num2str(k,'%5d'), ...
        '   J(1)=',num2str(costJnew(1),'%6.4f'), ...
        '   J(2)=',num2str(costJnew(2),'%6.4f'), ...
        '   mu(1)=',num2str(policy(1),'%3d'), ...
        '   mu(2)=',num2str(policy(2),'%3d')];

    % Use error bounds if flag has been set.
    if flag_errbnds

        % Compute error bounds.
        lower = costJnew + alpha/(1-alpha)*min(costJnew-costJ); % lower bound on J*
        upper = costJnew + alpha/(1-alpha)*max(costJnew-costJ); % upper bound on J*

        % display string:
        dispstr = [dispstr,'   lower=[',num2str(lower,'%9.4f'), ...
            ']   upper=[',num2str(upper,'%9.4f'),']'];

        % If desired tolerance reached: can terminate the algorithm.
        if all((upper-lower)<=ERR_TOL)
            % Difference between upper and lower bound less than specified
```

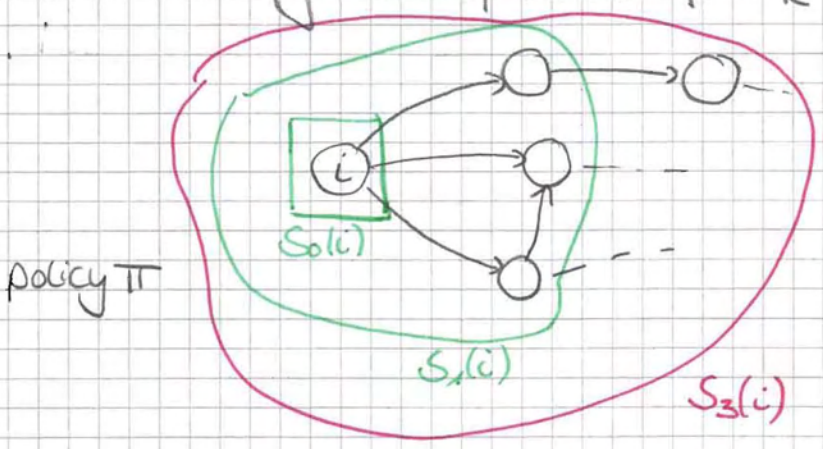
```
            % tolerance for all states: set cost to average of upper and
            % lower bound and terminate.
            % Note that using the bounds, we can get a better result than
            % just using J_{k+1}.
            costJnew = mean([lower; upper],1);

            dispstr = [dispstr, ...
                sprintf('\nResult within desired tolerance. Terminate.\n')];
            doBreak = true;
        end;

        % Save results for plotting later.
        res_lower(k,:) = lower;
        res_upper(k,:) = upper;
    end;

    % Update the cost.  One could also implement an immidiate update of the
    % cost as Gauss-Seidel type update, which would yield faster
    % convergence.
    costJ = costJnew;

    % Display:
    disp(dispstr);

    if doBreak
        break;
    end;
end;


% display the optained costs
disp('Result:');
disp(['  J*(1) = ',num2str(costJ(1),'%9.6f')]);
disp(['  J*(2) = ',num2str(costJ(2),'%9.6f')]);
disp(['  mu*(1) = ',num2str(policy(1))]);
disp(['  mu*(2) = ',num2str(policy(2))]);



%% Plots
% Plot costs and bounds over iterations.
kk = 1:size(res_J,1);

figure;
subplot(2,1,1);
if(flag_errbnds)
    plot(kk,[res_J(:,1),res_lower(:,1), res_upper(:,1)], ...
        [kk(1),kk(end)],[costJ(1), costJ(1)],'k');
else
    plot(kk,[res_J(:,1)],[kk(1),kk(end)],[costJ(1), costJ(1)],'k');
end;
grid;
%legend('lower bound','upper bound','J_k','J*');
%xlabel('iteration');
%
```

```
subplot(2,1,2);
if(flag_errbnds)
    plot(kk,[res_J(:,2),res_lower(:,2), res_upper(:,2)], ...
        [kk(1),kk(end)],[costJ(2),costJ(2)],'k');
else
    plot(kk,[res_J(:,2)],[kk(1),kk(end)],[costJ(2), costJ(2)],'k');
end;
grid;
if(flag_errbnds)
    legend('J_k','lower bound','upper bound','J*');
else
    legend('J_k','J*');
end;
xlabel('iteration');
```

Angela Schöllig

Exercise 7.12

Consider the given definition of $S_k(i)$ for any given π and i.



- Note:
  - $S_0(i) = \{i\}$
  - $S_k(i) \subseteq S_{k+1}(i)$ subset

- Assumption 7.2.1:
  For any π and i, $t \in S_m(i)$. (*)

- To be shown:
  For any π and i, $t \in S_n(i)$ where n is the number of nodes (not including the termination state) (**)

Proof 1

Assume for some $k<n$, $t \notin S_k(i)$ [for any π and i].
Because of (*), we know that $\exists\ j \in S_k(i)$ for which we end up at t after $(n-k)$ steps. That is,

$$S_k(i) \subset S_{k+1}(i)$$ (strict or proper subset)

as long as $t \notin S_k(i)$.

However, that means after at most n steps, we have "explored" the whole state space and already reached t with positive probability.

Angela Schiller

Proof 2 - by contradiction

- Assume (**) is not true.

  ⇕ equivalent to

  For some k, $k < n$, $S_k(i) = S_{k+1}(i) \not\ni t$ [for some π and i]

  ⇕

  $P_{jL}(\mu) = 0$, $\forall j \in S_k(i)$ and $L \notin S_k(i)$

  This is, the termination state is not reachable from any $j \in S_k(i)$ what is a contradiction to (*)

Problem 1.23

[the monotonicity property of DP is used in proof of policy iteration on page 415.]

- Given $J_{N-1}(x) \leq J_N(x) \quad \forall x \in S$.
- To be proven: $J_k(x) \leq J_{k+1}(x) \quad \forall x \in S \quad \forall k$

Proof by Induction

- Start: $J_{N-1}(x) \leq J_N(x) \quad \forall x \in S$ given
- Hypothesis: Assume $J_k(x) \leq J_{k+1}(x) \quad \forall x \in S$ for some k
- Induction step:
  - Let $C_k(x,u) := \underset{w}{E}\left(g(x,u,w) + J_{k+1}\left(f(x,u,w)\right)\right)$.

    Then $C_k(x,u) = \underset{w}{E}\left(g(x,u,w)\right) + \underset{w}{E}\left(J_{k+1}\left(f(x,u,w)\right)\right)$

    $\geq \underset{w}{E}\left(g(x,u,w)\right) + \underset{w}{E}\left(J_k\left(f(x,u,w)\right)\right)$ (by induction hypothesis)

    $= C_{k-1}(x,u)$

    $\Rightarrow \quad C_{k-1}(x,u) \leq C_k(x,u) \quad \forall x,u$

  - Therefore,

    $J_{k-1}(x) = \min_{u \in U(x)} \underset{w}{E}\left(g(x,u,w) + J_k\left(f(x,u,w)\right)\right)$

    $= \min_{u \in U(x)} C_{k-1}(x,u)$

    $\leq \min_{u \in U(x)} C_k(x,u)$

    ($C_{k-1}(x,u) \leq C_k(x,u) \ \forall u$, therefore taking the minimum over some range $u \in U(x)$ preserves the order: $\min_{u \in U(x)} C_{k-1}(x,u) \leq \min_{u \in U(x)} C_k(x,u)$)

    $= J_k(x)$

$\Rightarrow J_{k-1}(x) \leq J_k(x) \quad \forall x$ q.e.d.

The proof for the reversed order, i.e. given $f_{N-1}(x) \geq f_N(x) \quad \forall x$, prove $f_k(x) \geq f_{k+1}(x) \quad \forall x, k$ goes the same way; just reverse $\geq / \leq$ everywhere.