```
% Implements a soft policy exploration scheem Monte Carlo estimation algorihtm
% using first visit estimation to compute the optimal action-value function for
% the black jack example.
%
% Mods to try:
% 0) Implement recursive averaging update of Q
%
%    Q_{k+1} <- Q_k + (1/(k+1)) (r_{k+1} - Q_{k})
%
% Result: seemed to change the final policy but only in a few places which might be due
to
%         the specific random draws i.e. the differences could be due to randomness.
%
% 1) Add a fixed step size learning algorithm,
%
%    Q_{k+1} <- Q_k + \alpha (r_{k+1} - Q_{k})
%
%  which should be better for problems where the action value function may
%  change over time which is the case when we are performing value iteration.
%
% Result: results in very spotty policies that don't seem correct.
%
% 2) Set Q initially very large to encourage exploration.  A value ~ +5
% should be large enough.
%
% Results:
%
% With geometric update (involving alpha):is quite poor very spoty policy.
%
% With recursive update: seems too look quite good with similar results to the
% exploring starts algorithm
%
% Written by:
% --
% John L. Weatherwax                2007-12-07
%
% email: wax@alum.mit.edu
%
% Please send comments and especially bug reports to the
% above email address.
%
%-----

close all;
clc;

% the epsilon in our epsilon-soft policy: (eps=0 => greedy; eps=1 => random)
%eps = 0.03;
eps = 0.1;
%eps = 0.5;
%eps = 0.85;

% the fixed step size learning parameter:
%alpha = 0.1;
%alpha = 0.05;

%N_HANDS_TO_PLAY=10;     % a numerical approximation of +Inf
N_HANDS_TO_PLAY=2*1e4;
%N_HANDS_TO_PLAY=5e5;
```

```
N_HANDS_TO_PLAY=4.4e7;  % <- should take 12 hours ...

% sample_discrete.m
addpath( genpath( '../../../FullBNT-1.0.4/KPMstats/' ) );

rand('seed',0); randn('seed',0);

%--
% implement hands of bj
%--

nStates       = prod([21-12+1,13,2]);
posHandSums   = 12:21;
nActions      = 2; % 0=>stick; 1=>hit
%Q             = zeros(nStates,nActions);      % the initial action-value function
Q             = +5*ones(nStates,nActions);      % the initial action-value function taken
initially large to encourage exploration
pol_pi        = 0.5*ones(nStates,nActions);   % our initial policy is random
firstSARewSum = zeros(nStates,nActions);
firstSARewCnt = zeros(nStates,nActions);
%timePerPlay   = zeros(1,N_HANDS_TO_PLAY);

tic
for hi=1:N_HANDS_TO_PLAY,

  %tic;
  stateseen = []; pol_taken = [];
  deck = shufflecards();

  % the player gets the first two cards:
  p = deck(1:2); deck = deck(3:end); phv = handValue(p);
  % the dealer gets the next two cards (and shows his first card):
  d = deck(1:2); deck = deck(3:end); dhv = handValue(d); cardShowing = d(1);

  % disgard states who's initial sum is less than 12 (the decision is always to hit):
  while( phv < 12 )
    p = [ p, deck(1) ]; deck = deck(2:end); phv = handValue(p); % HIT
  end

  % accumulate/store the first state seen:
  stateseen(1,:) = stateFromHand( p, cardShowing );

  % implement the policy specified by pol_pi (keep hitting till we should "stick"):
  si = 1;
  polInd       = sub2ind( [21-12+1,13,2], stateseen(si,1)-12+1, stateseen(si,2),
stateseen(si,3)+1 );
  pol_to_take  = sample_discrete( pol_pi(polInd,:), 1, 1 )-1; % value in (0,1)
"stick/hit"
  pol_taken(1) = pol_to_take;
  while( pol_to_take && (phv < 22) )
    p = [ p, deck(1) ]; deck = deck(2:end); phv = handValue(p); % HIT
    stateseen(end+1,:) = stateFromHand( p, cardShowing );

    if( phv <= 21 ) % only then do we need to querry the next policy action when we have
not gone bust
      si = si+1;
      %[ stateseen(si,1), stateseen(si,2), stateseen(si,3) ]
      polInd           = sub2ind( [21-12+1,13,2], stateseen(si,1)-12+1, stateseen(si,2),
stateseen(si,3)+1 );
      pol_to_take      = sample_discrete( pol_pi(polInd,:), 1, 1 )-1;
```

```
      pol_taken(end+1) = pol_to_take;
    end
  end
  % implement the fixed deterministic policy of the dealer (hit until we have a hand
value of 17):
  while( dhv < 17 )
    d = [ d, deck(1) ]; deck = deck(2:end); dhv = handValue(d); % HIT
  end
  % determine the reward for playing this game:
  rew = determineReward(phv,dhv);
  %fprintf( '[phv, dhv, rew] = \n' ); [ phv, dhv, rew ]

  % accumulate these values used in computing statistics on this action value function
Q^{\pi}:
  % Note 1) in this game there is no difference between first occurance and each
occurance ...
  % Note 2) since each state is entered only once there is no need to "unjam" this for
loop into
  %         separate for loops like "(b)" and "(c)" as seen in the algorihtm given in
Fig~5.6 of the book
  for si=1:size(stateseen,1),
    if( (stateseen(si,1)>=12) && (stateseen(si,1)<=21) ) % we don't count "initial" and
terminal states
      %[stateseen(si,1)]
      %[stateseen(si,1)-12+1, stateseen(si,2), stateseen(si,3)+1]
      staInd = sub2ind( [21-12+1,13,2], stateseen(si,1)-12+1, stateseen(si,2),
stateseen(si,3)+1 );
      actInd = pol_taken(si)+1;
      firstSARewCnt(staInd,actInd) = firstSARewCnt(staInd,actInd)+1;
      firstSARewSum(staInd,actInd) = firstSARewSum(staInd,actInd)+rew;
      %Q(staInd,actInd)             =
firstSARewSum(staInd,actInd)/firstSARewCnt(staInd,actInd); % <-take the direct average
      Q(staInd,actInd)             = Q(staInd,actInd) +
(1/firstSARewCnt(staInd,actInd))*(rew - Q(staInd,actInd)); % <-use incremental averaging
      %Q(staInd,actInd)             = Q(staInd,actInd) + alpha*(rew - Q(staInd,actInd));
% <-take a geometric average
      [dum,greedyChoice]           = max( Q(staInd,:) );
      notGreedyChoice              = 3-greedyChoice; % linear function that maps 1=>2 and
2=>1
      % perform the eps-soft on-policy MC update:
      pol_pi(staInd,greedyChoice)    = 1 - eps + eps/nActions;
      pol_pi(staInd,notGreedyChoice) = eps/nActions;
    end
  end
  %timePerPlay(hi) = toc;

end % end number of hands loop
toc

%fprintf('timePerPlay = %f\n',mean(timePerPlay));

% plot the optimal state-value function V^{*}:
%
mc_value_fn = max( Q, [], 2 );
mc_value_fn = reshape( mc_value_fn, [21-12+1,13,2]);
if( 1 )
  figure; mesh( 1:13, 12:21, mc_value_fn(:,:,1) );
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy; %view([67,5]);
  title( 'no usable ace' ); drawnow;
  fn=sprintf('state_value_fn_nua_%d_mesh.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2'
```

```
);
  figure; mesh( 1:13, 12:21,  mc_value_fn(:,:,2) );
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy; %view([67,5]);
  title( 'a usable ace' ); drawnow;
  fn=sprintf('state_value_fn_ua_%d_mesh.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2' );

  figure;imagesc( 1:13, 12:21, mc_value_fn(:,:,1) ); caxis( [-1,+1] ); colorbar;
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy;
  title( 'no usable ace' ); drawnow;
  fn=sprintf('state_value_fn_nua_%d_img.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2' );
  figure;imagesc( 1:13, 12:21, mc_value_fn(:,:,2) ); caxis( [-1,+1] ); colorbar;
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy;
  title( 'a usable ace' ); drawnow;
  fn=sprintf('state_value_fn_ua_%d_img.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2' );
end

% plot the optimal policy:
%
pol_pi = pol_pi(:,2);
pol_pi = reshape( pol_pi, [21-12+1,13,2] );
if( 1 )
  figure; imagesc( 1:13, 12:21, pol_pi(:,:,1) ); colorbar;
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy; %view([67,5]);
  title( 'no usable ace' ); drawnow;
  fn=sprintf('bj_opt_pol_nua_%d_image.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2' );
  figure; imagesc( 1:13, 12:21,  pol_pi(:,:,2) ); colorbar;
  xlabel( 'dealer shows' ); ylabel( 'sum of cards in hand' ); axis xy; %view([67,5]);
  title( 'a usable ace' ); drawnow;
  fn=sprintf('bj_opt_pol_ua_%d_image.eps',N_HANDS_TO_PLAY); saveas( gcf, fn, 'eps2' );
end

return;
```