```
function [] = persuit_method(nB,nA,nP,sigmaReward)
%
% Demonstrate the learning technique of persuit learning
% We compare three methods
%
% M1) action-value method with 1/k update and epsilon=0.1 greedy
% M2) the reinforment comparison method (with zero initial reward)
% M3) the persuit method
%
% Inputs:
%   nB: the number of bandits
%   nA: the number of arms
%   nP: the number of plays (times we will pull a arm)
%   sigmaReward: the standard deviation of the return from each of the arms
%
% Written by:
% --
% John L. Weatherwax                2007-11-13
%
% email: wax@alum.mit.edu
%
% Please send comments and especially bug reports to the
% above email address.
%
%-----

%close all;
%clc;
%clear;

if( nargin<1 ) % the number of bandits:
  nB = 2000;
end
if( nargin<2 ) % the number of arms:
  nA = 10;
end
if( nargin<3 ) % the number of plays (times we will pull a arm):
  nP = 2000;
end
if( nargin<4 ) % the standard deviation of the return from each of the arms:
  sigmaReward = 1.0;
end

%randn('seed',0);

%-----
% Compare three methods:
% M1) action-value method with 1/k update and epsilon=0.1 greedy
% M2) the reinforment comparison method (with zero initial reward)
% M3) the persuit method
%-----
tEps  = 0.1;
alpha = 0.1;

avgReward    = zeros(2,nP);
perOptAction = zeros(2,nP);

allRewards_M1      = zeros(nB,nP);
pickedMaxAction_M1 = zeros(nB,nP);
```

```
allRewards_M2      = zeros(nB,nP);
pickedMaxAction_M2 = zeros(nB,nP);
allRewards_M3      = zeros(nB,nP);
pickedMaxAction_M3 = zeros(nB,nP);
for bi=1:nB, % pick a bandit
  % generate the TRUE reward Q^{\star}:
  qStarMeans = randn(1,nA);

  % for the action-value (\epsilon,1/k) method:
  %
  qT_M1 = zeros(1,nA); % <- initialize action value estimates
  qN_M1 = zeros(1,nA); % <- initialize number of draws on this arm

  % reinforcement comparison method:
  %
  pT = zeros(1,nA); % <- initialize play preference
  rT = 0;           % <- initialize a reference reward (one for all rewards recieved) but
"too" negative

  % action persuit method:
  %
  qT_M3  = zeros(1,nA);
  qN_M3  = zeros(1,nA);
  piT_M3 = ones(1,nA)/nA;

  for pi=1:nP, % make a play ... one for each algorithm

    % EXPLORITORY v.s. GREEDY MOVES (first methods):
    %
    if( rand(1) <= tEps ) % pick a RANDOM arm ... explore:
      [dum,arm_M1] = histc(rand(1),linspace(0,1+eps,nA+1));
    else                  % pick the GREEDY arm:
      [dum,arm_M1] = max( qT_M1 );
    end; clear dum;
    % EXPLORITORY v.s. GREEDY MOVES (reinforcement comparison):
    %
    piT = exp(pT) ./ sum( exp(pT) );
    % draw an arm from the distribution piT
    arm_M2 = sample_discrete( piT, 1, 1 );
    % EXPLORITORY v.s. GREEDY MOVES (action persuit):
    %
    beta = 0.01;
    [dum,arm_M3] = max( qT_M3 ); % <- pick the greedy choice
    piT_M3(arm_M3) = piT_M3(arm_M3) + beta * ( 1 - piT_M3(arm_M3) ); % <- increment the
greedy probability
    for ar=1:nA                  % <- decrement all the others
      if( ar==arm_M3 ) continue; end
      piT_M3(ar) = piT_M3(ar) + beta * ( 0 - piT_M3(ar) );
    end
    arm_M3 = sample_discrete( piT_M3, 1, 1 ); % <- sample from this distribution

    % determine if the arm selected is the best possible:
    [dum,bestArm] = max( qStarMeans );
    if( arm_M1==bestArm ) pickedMaxAction_M1(bi,pi) = 1; end
    if( arm_M2==bestArm ) pickedMaxAction_M2(bi,pi) = 1; end
    if( arm_M3==bestArm ) pickedMaxAction_M3(bi,pi) = 1; end
    % get the reward from drawing on that arm:
    reward_M1 = qStarMeans(arm_M1) + sigmaReward*randn(1);
    reward_M2 = qStarMeans(arm_M2) + sigmaReward*randn(1);
    reward_M3 = qStarMeans(arm_M3) + sigmaReward*randn(1);
```

```
    allRewards_M1(bi,pi) = reward_M1;
    allRewards_M2(bi,pi) = reward_M2;
    allRewards_M3(bi,pi) = reward_M3;
    % update qT:
    qT_M1(arm_M1) = qT_M1(arm_M1) + ( reward_M1-qT_M1(arm_M1) )/(qN_M1(arm_M1)+1);
    qN_M1(arm_M1) = qN_M1(arm_M1) + 1;
    % The reinforcment comparison update (play preference pT and average reward)
    beta = 0.1;
    pT(arm_M2) = pT(arm_M2) + beta*( reward_M2 - rT ); % <- with no \pi correction
    rT         = rT + alpha*( reward_M2 - rT );
    % the action persuit update method:
    qT_M3(arm_M3) = qT_M3(arm_M3) + ( reward_M3-qT_M3(arm_M3) )/(qN_M3(arm_M3)+1);
    qN_M3(arm_M3) = qN_M3(arm_M3) + 1;
  end
end

avgRew         = mean(allRewards_M1,1);
avgReward(1,:) = avgRew(:).';
avgRew         = mean(allRewards_M2,1);
avgReward(2,:) = avgRew(:).';
avgRew         = mean(allRewards_M3,1);
avgReward(3,:) = avgRew(:).';

percentOptAction  = mean(pickedMaxAction_M1,1);
perOptAction(1,:) = percentOptAction(:).';
percentOptAction  = mean(pickedMaxAction_M2,1);
perOptAction(2,:) = percentOptAction(:).';
percentOptAction  = mean(pickedMaxAction_M3,1);
perOptAction(3,:) = percentOptAction(:).';

% produce the average rewards plot:
%
figure; hold on;
all_hnds = plot( 1:nP, avgReward );
legend( all_hnds, { 'act-val', 'rein\_comp', 'action\_persuit' }, 'Location', 'SouthEast'
);
axis tight; grid on;
xlabel( 'plays' ); ylabel( 'Average Reward' );

% produce the percent optimal action plot:
%
figure; hold on;
all_hnds = plot( 1:nP, perOptAction );
legend( all_hnds, { 'act-val', 'rein\_comp', 'action\_persuit' }, 'Location', 'SouthEast'
);
axis( [ 0, nP, 0, 1 ] ); axis tight; grid on;
xlabel( 'plays' ); ylabel( '% Optimal Action' );
```