```
function [arms_Err_mc,arms_Err_td] = eg_rw_batch_learn(nEpisodes,alpha)
% EG_RW_BATCH_LEARN - using TD(0) and consntant step size monte carlo learn (in batch)
% the value function for the random walk example.
%
% 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.
%
%-----

PRNT_CONV=1;

if( nargin < 2 ) alpha = 0.1; end;
gamma = 1; % the discount factor

% the number of non terminal states plus two terminal states:
nStates=5+2;

% generate many episodes of our random walk:
%
episodeHistory = {};     % <- save a history of the sequence of states seen
for ei=1:nEpisodes,
  % perform a random walk:
  %
  % --states are [1=X,2,3,4,5,6,7=X] == [X,A,B,C,D,E,X] and we start at C=4 (X is a
terminal code)
  st = 3+1; R=0; stateseen = [st]; rewseen = [];
  while( 1 < st && st < 7 )
    if( rand<0.5 )
      stp1=st+1;
    else
      stp1=st-1;
    end
    % assign a reward for this step:
    if( stp1~=7 ) rew = 0; else rew = +1; end
    R = R+rew;
    rewseen   = [ rewseen; rew ];
    stateseen = [stateseen; stp1];
    st = stp1;
  end
  % store this episode into our history:
  anEp.stateseen = stateseen;
  anEp.rewseen   = rewseen;
  anEp.R         = R;
  episodeHistory{end+1} = anEp;
end

truth = (1:5)/6;

conv_tol = 1e-6;

arms_Err_mc = zeros(1,nEpisodes); arms_Err_td = zeros(1,nEpisodes); tic;
for ei=1:nEpisodes,
  if( 1 || PRNT_CONV ) fprintf('batch learing of V{mc,td} on %d episodes in ~ %f sec
```

```
time...\n',ei,toc); end; tic
  thisBatch = episodeHistory(1:ei);

  %--
  % perform batch updating of Vmc:
  %--
  Vmc = 0.5*ones(1,nStates); Vmc(1)=0; Vmc(end)=0; icnt=0;

  while( 1 )
    % -- accumulate the MC differences for thisBatch
    V_mc_incrs = zeros(1,nStates);
    for bi=1:length(thisBatch),
      anEp = thisBatch{bi};      % get an episode
      stateseen = anEp.stateseen;
      rewseen   = anEp.rewseen;
      R         = anEp.R;
      for si=1:(length(stateseen)-1),         % <- the last state is always the terminal
state
        st = stateseen(si); rt = rewseen(si);
        V_mc_incrs(st) = V_mc_incrs(st) + alpha*(R-Vmc(st));
      end
    end
    icnt=icnt+1;
    if( PRNT_CONV )
      if( (icnt<10) || mod(icnt,100)==0 ) fprintf('MC iter %d
error=%8.5g...\n',icnt,max(abs(V_mc_incrs))); end
    end
    if( max(abs(V_mc_incrs))<conv_tol ) break; end
    if( max(abs(V_mc_incrs))>1e1 ) error( 'MC algorithm is diverging...decrease
alpha...\n' ); end

    % accumulate all updates and apply them to Vmc/Vtd:
    Vmc = Vmc + V_mc_incrs;
  end
  Vmc = Vmc(2:end-1); % remove terminal states
  arms_Err_mc(ei) = mean( (Vmc-truth).^2 );

  %--
  % perform batch updating of Vtd:
  %--
  Vtd = 0.5*ones(1,nStates); Vtd(1)=0; Vtd(end)=0; icnt=0;

  while( 1 )
    % -- accumulate the TD differences for thisBatch
    V_td_incrs = zeros(1,nStates);
    for bi=1:length(thisBatch),
      anEp = thisBatch{bi};      % get an episode
      stateseen = anEp.stateseen;
      rewseen   = anEp.rewseen;
      R         = anEp.R;
      for si=1:(length(stateseen)-1),         % <- the last state is always the terminal
state
        st = stateseen(si); rt = rewseen(si);
        %if( stp1~=1 && stp1~=7 ) % don't update terminal states
          stp1 = stateseen(si+1);
          V_td_incrs(st) = V_td_incrs(st) + alpha*(rt + gamma*Vtd(stp1) - Vtd(st));
        %end
      end
    end
    icnt=icnt+1;
```

```
    if( PRNT_CONV )
      if( (icnt<10) || mod(icnt,100)==0 ) fprintf('TD iter %d
error=%8.5g...\n',icnt,max(abs(V_td_incrs))); end
    end
    if( max(abs(V_td_incrs))<conv_tol ) break; end
    if( max(abs(V_mc_incrs))>1e1 ) error( 'TD algorithm is diverging...decrease
alpha...\n' ); end

    % accumulate all updates and apply them to Vtd/Vtd:
    Vtd = Vtd + V_td_incrs;
  end
  Vtd = Vtd(2:end-1); % remove terminal states
  arms_Err_td(ei) = mean( (Vtd-truth).^2 );

end % end episode loop
```