

```

%=====
% ME 652, Spring 2020
% Course instructor: Jinwhan Kim
%=====
function MDP_ValueIteration_skeleton(RValue, GammaValue)

    if nargin ~ = 2
        RValue = -0.04;
        GammaValue = 1;
    end

    env = init(RValue, GammaValue); % initialization
    max_iterations = 100; % maximum iteration limit
    err_threshold = 1e-5; % error tolerance
    V = env.R; % initialize value function with reward values
    i_count = 0; % iteration counter

    while (1) % iteration loop
        i_count = i_count + 1;
        newV = value_iteration(V, env); % value iteration
        if (sum(abs(newV - V)) < err_threshold) % break if converged
            fprintf('\nConverged after %i iterations...\n\n', i_count);
            break;
        elseif (i_count >= max_iterations)
            fprintf('\nMaximum iteration limit reached...\n\n');
            break;
        end
        V = newV;
    end

    optimal_V = newV; % final value function
    disp(reshape(optimal_V', env.maxY, env.maxX)); % show value function

    strP = extract_policy(optimal_V, env); % policy extraction
    disp(reshape(strP, env.maxY, env.maxX)); % show policy
end

%=====
% initialize MDP parameters
%=====
function env = init(RValue, GammaValue)
    env.maxX = 4; % maximum X
    env.maxY = 3; % maximum Y
    env.nS = env.maxX * env.maxY; % number of states (grid cells)
    env.nA = 4; % number of actions (up, down, left, right)
    env.nT = 3; % number of potential transitions
    env.prob = [0.8, 0.1, 0.1]; % transition probability

    env.obst = XY2S(env, 2, 2); % obstacle state
    env.tms1 = XY2S(env, 4, 1); % terminal state 1
    env.tms2 = XY2S(env, 4, 2); % terminal state 2

    % prompt = 'Type in R value : ';
    % R = input(prompt);
    R = RValue;

    env.R = R * ones(env.nS, 1);
    env.R(env.obst) = 0; % reward at obstacle state
    env.R(env.tms1) = 1; % reward at terminal state 1
    env.R(env.tms2) = -1; % reward at terminal state 2
end

```

```

% env.gamma = 1; % discount factor
% prompt = "Type in gamma value : ";
env.gamma = GammaValue;
end

%-----
% value iteration (complete this)
%-----
function [newV,act] = value_iteration(V, env)
    newV = zeros(env.nS,1);
    act = zeros(env.nS,1);
    % for each state
    for i = 1 : env.nS
        if ((i == env.obst) || (i == env.tms1) || (i == env.tms2))
            continue;
        end

        vals = zeros(env.nA,1);

        % for each action
        for j = 1 : env.nA
            nextS = next_state(env, i, j);
            value = env.prob * V(nextS);
            vals(j) = value;
        end

        [M, I] = max(vals);
        newV(i) = env.gamma * M;
        act(i) = I;
    end

    newV = newV + env.R;
end

%-----
% extract policy (complete this)
%-----
function strP = extract_policy(V, env)

    strP = repmat("X",env.nS,1);
    [~,act] = value_iteration(V, env);

    for i = 1:env.nS
        if act(i) == 1
            strP(i) = "U";
        elseif act(i) == 2
            strP(i) = "R";
        elseif act(i) == 3
            strP(i) = "D";
        elseif act(i) == 4
            strP(i) = "L";
        end
    end

end

%-----
% probabilistic state propagation (s -> s')
```

```

%-----
function nextS = next_state(env, S, action)
    nextS = zeros(env.nT, 1);
    if action == 1 % move up (intended)
        action_list = [1, 2, 4]; % up or right or left (actual)
    elseif action == 2 % move right (intended)
        action_list = [2, 1, 3]; % right or up or down (actual)
    elseif action == 3 % move down (intended)
        action_list = [3, 2, 4]; % down or right or left (actual)
    elseif action == 4 % move left (intended)
        action_list = [4, 1, 3]; % left or up or down (actual)
    end
    for i_action = 1:env.nT
        Sprime = S2Sprime(env,S, action_list(i_action));
        nextS(i_action) = Sprime;
    end
end

```

```

%-----
% non-probabilistic state propagation (s -> s')
%-----
function newS = S2Sprime(env, S, action)
    [X,Y] = S2XY(env, S); % index conversion
    if action == 1 % move up
        Y = max([Y-1, 1]);
    elseif action == 2 % move right
        X = min([X+1, env.maxX]);
    elseif action == 3 % move down
        Y = min([Y+1, env.maxY]);
    elseif action == 4 % move left
        X = max([X-1, 1]);
    end
    newS = XY2S(env, X, Y); % index conversion
    if (newS == env.obst)
        newS = S;
    end
end

```

```

%-----
% state id to matrix xy coordinates
%-----
function [X,Y] = S2XY(env,S)
    X = fix((S-1)/env.maxY) + 1;
    Y = rem(S, env.maxY);
    if Y == 0
        Y = env.maxY;
    end
end

```

```

%-----
% matrix xy coordinates to state id
%-----
function S = XY2S(env, X, Y)
    S = (X-1)*env.maxY + Y;
end

```