



Ben Gurion University of the Negev
Faculty of Engineering Sciences

Intelligent Robotics System

Exercise 3

Nir Dgani 200517670

Lecturer: Dr. Armin Biess

Date: 11/07/2017

1. Define and explain the following terms:

- a. Markov Decision Process (MDP):** a mathematical model of decision processes in which the system's transitions function maintains the Markov attribute (the probability of reaching a situation depends only on the state and on the previous action). For Markov decision processes, the action outcomes depend only on the current state.

$$p(S_{t+1} = s' | S_t = s, A_t = a)$$

- b. State-Value function:** the “world” state in time t is defined: \mathbf{s}_t . The state-value function describes how good it is to be in a particular state. This function is the expected return starting from state \mathbf{s} and then following policy π :

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

Where G_t is the discounted reward from time step t :

$$G_t = \sum_{j=0}^{\infty} \gamma^j R_{t+j+1} = \frac{|R_{max}|}{1 - \gamma}$$

Where γ is the discount factor ($0 < \gamma < 1$).

- c. Action-Value function:** the action the agent chooses in time t is defined: A_t . This action makes the “world” transition from state \mathbf{s}_t to state \mathbf{s}_{t+1} . The action-value function describes how good it is to take a particular action \mathbf{a} from given state \mathbf{s} . The action-value function for policy π is the

expected return starting from state \mathbf{s} , taking action \mathbf{a} and then following policy π :

$$q_{\pi}(s, a) = E_{\pi}[G_t | S_t = s, A_t]$$

- d. Policy:** the core problem of the MDP is to find “policy” for the decision maker. The policy is a function that specifies the action that the decision maker will choose when in state s . The policy mapping from states to actions, that maximizes expected total future rewards.

Deterministic policy is defined by: $a_t = \pi(s_t)$

Stochastic policy is defined by: $\pi(a|s) = P(A_t = a | S_t = s)$

- e. Dynamic programming:** (MDPs can be solved by linear-programming or dynamic programming). Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler sub-problems, solving each of those sub-problems just once and storing their solutions. The next time the same sub-problem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. Example for solution by dynamic programming is calculation of the general factor in a Fibonacci series.

- f. Value iteration:** In value iteration, we combined the value calculation and the reward calculation. In value iteration, the policy function π is not used, instead, the value of $\pi(s)$ is calculating within value $V(s)$ whenever it is needed.

The function is:

$$V(s) = \max \left(\sum_{s'} p(s', s, a) (r(s, s') + \gamma V(s')) \right)$$

g. Policy iteration: the policy-iteration algorithm is composed out of two steps:

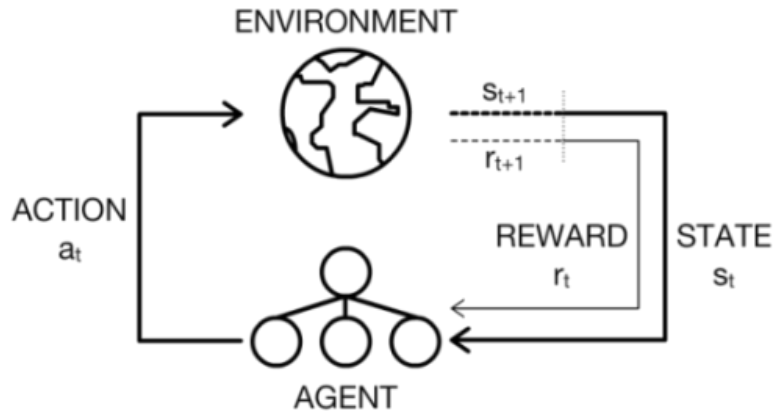
Step 1: policy evaluation: can be solved in two ways – exact/iterative policy evaluation.

Step 2: policy improvement: improve the policy at each state in a greedy way.

The solution repeat two steps until no improvement to $v_{\pi}(s)$ occur. This leads to the optimal policy $\pi^*(s)$ and optimal value function $v_{\pi^*}(s)$:

$$\pi(s) \rightarrow v_{\pi}(s) \rightarrow \pi'(s) \rightarrow v_{\pi'}(s) \rightarrow \dots \pi^*(s) \rightarrow v_{\pi^*}(s)$$



h. Reinforcement learning (RL): is an area of machine learning (is different from the supervised/unsupervised learning methods), concerned with how software agent ought to take actions in an environment so as to maximize some notion of cumulative reward and punishment.



2. Markov decision Process (MDP) in a robot grid-world

The grid-world size is: 3 x 4.

	1	2	3	4
1	(1)	(4)	(7)	(10)
2	(2)	(5)	(8)	(11)
3	(3)	(6)	(9)	(12)

The grid world has one obstacle () and two absorbing (terminal) () state.

In the terminal state, the robot receives rewards of +1 (goal state) and -1.

For all states a reward of $r = -0.04$ is collected.

The robot can perform four actions: north (N), east (E), south (S) and west (W).

$$\mathbf{A} = \{\uparrow, \rightarrow, \downarrow, \leftarrow\} = \{N, E, S, W\}$$

$$|\mathbf{A}| = 4$$

The probability for the commanded direction to occur is 0.8, but with probability 0.2 the robot moves at right angles to the intended direction.

For example:

$$p(s' = (2,4)|s = (2,3), a = E) = 0.8$$

$$p(s' = (1,3)|s = (2,3), a = E) = 0.1$$

$$p(s' = (3,3)|s = (2,3), a = E) = 0.1$$

Find the optimal policy for the robot in this grid-world by solving the corresponding MDP.

	1	2	3	4
1	(1)	(4)	(7)	(10) +1
2	(2)	(5)	(8)	(11) -1
3	(3) start	(6)	(9)	(12)

a. The transition model $p(s'|s,a)$:

$$p(s'|s,a) = \begin{cases} 0.8; & \text{to the commanded direction} \\ 0.2; & 90^\circ \text{ to the intended direction} \end{cases}$$

```

clc; close all; clear all;
%% ***** a
%% *****
myworld = cWorld3();

myworld.R = [-0.04;-0.04;-0.04;-0.04;0;-0.04;-0.04;-0.04;-0.04;1;-1;-0.04];

SO = 5; % stateObstacles
ST = [10;11]; % stateTerminals
PR(:,:,,:) = zeros(12,12,4); % PR(s',s,a)

% a=North;
for s = 1:12
    PR(:,s,1) = 0;
    if s==1
        PR(1,s,1) = 0.9; PR(4,s,1) = 0.1;
    elseif s==2
        PR(1,s,1) = 0.8; PR(2,s,1) = 0.2;
    elseif s==3
        PR(2,s,1) = 0.8; PR(s,s,1) = 0.1; PR(6,s,1) = 0.1;
    elseif s==4 || s==6 || s==7
        PR(s,s,1) = 0.8; PR(s-3,s,1) = 0.1; PR(s+3,s,1) = 0.1;
    elseif s==8
        PR(7,s,1) = 0.8; PR(s,s,1) = 0.1; PR(11,s,1) = 0.1;
    elseif s==9
        PR(8,s,1) = 0.8; PR(6,s,1) = 0.1; PR(12,s,1) = 0.1;
    elseif s==12
        PR(s-1,s,1) = 0.8; PR(s-3,s,1) = 0.1; PR(s,s,1) = 0.1;
    end
end
end

```

```

% a=East;
for s = 1:12
    PR(:,s,2) = 0;
    if s==1
        PR(4,s,2) = 0.8; PR(s,s,2) = 0.1; PR(2,s,2) = 0.1;
    elseif s==2
        PR(s-1,s,2) = 0.1; PR(s,s,2) = 0.8; PR(s+1,s,2) = 0.1;
    elseif s==3 || s==9
        PR(s+3,s,2) = 0.8; PR(s,s,2) = 0.1; PR(s-1,s,2) = 0.1;
    elseif s==4 || s==6
        PR(s+3,s,2) = 0.8; PR(s,s,2) = 0.2;
    elseif s==7
        PR(10,s,2) = 0.8; PR(s,s,2) = 0.1; PR(8,s,2) = 0.1;
    elseif s==8
        PR(11,s,2) = 0.8; PR(7,s,2) = 0.1; PR(9,s,2) = 0.1;
    elseif s==12
        PR(11,s,2) = 0.1; PR(s,s,2) = 0.9;
    end
end
% a=South;
for s = 1:12
    PR(:,s,3) = 0;
    if s==1
        PR(2,s,3) = 0.8; PR(s,s,3) = 0.1; PR(4,s,3) = 0.1;
    elseif s==2
        PR(s,s,3) = 0.2; PR(3,s,3) = 0.8;
    elseif s==3
        PR(s,s,3) = 0.9; PR(6,s,3) = 0.1;
    elseif s==4 || s==6 || s==9
        PR(s+3,s,3) = 0.1; PR(s,s,3) = 0.8; PR(s-3,s,3) = 0.1;
    elseif s==7
        PR(8,s,3) = 0.8; PR(4,s,3) = 0.1; PR(10,s,3) = 0.1;
    elseif s==8
        PR(s,s,3) = 0.1; PR(9,s,3) = 0.8; PR(11,s,3) = 0.1;
    elseif s==12
        PR(9,s,3) = 0.1; PR(s,s,3) = 0.9;
    end
end
% a=West;
for s = 1:12
    PR(:,s,4) = 0;
    if s==1
        PR(s,s,4) = 0.9; PR(2,s,4) = 0.1;
    elseif s==2 || s==8
        PR(s-1,s,4) = 0.1; PR(s,s,4) = 0.8; PR(s+1,s,4) = 0.1;
    elseif s==3
        PR(s,s,4) = 0.9; PR(2,s,4) = 0.1;
    elseif s==4 || s==6
        PR(s-3,s,4) = 0.8; PR(s,s,4) = 0.2;
    elseif s==7
        PR(s-3,s,4) = 0.8; PR(s,s,4) = 0.1; PR(s+1,s,4) = 0.1;
    elseif s==9 || s==12
        PR(s,s,4) = 0.1; PR(s-3,s,4) = 0.8; PR(s-1,s,4) = 0.1;
    end
end
myworld.Pr = PR(:,:,:);

```


Reward function:

$$r(s) = \begin{cases} 1; & \text{goal state} \\ -1; & \text{"chasm"} \\ -0.04; & \text{all other states} \end{cases}$$

```
R = -0.04; % reward
Reward = R*ones(12,1); Reward(5)=0; Reward(10) = 1; Reward(11) = -1;
myworld.R = Reward;
```

b. Solving the MDF using *value iteration*:

The discount factor is: γ .

The termination threshold is: $\theta = 10^{-4}$.

The reward function: $\mathbf{r}_{1 \times 12} = \{r, r, r, r, 0, r, r, r, r, r, r, +1, -1, r\}$

The initial value: $v_0 = \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, +1, -1, 0\}$

$$V(s) = r(s) + \gamma \max \left(\sum_{s'} p(s', s, a) V(s') \right)$$

First case:

$\gamma = 1$; $r = -0.04$

	1	2	3	4
1	(1) 0	(4) 0	(7) 0	(10) +1
2	(2) 0	(5) 0	(8) 0	(11) -1
3	(3) 0	(6) 0	(9) 0	(12) 0

One iteration for example:

Finding $V(7)$:

(1) **a=North:**

$$\begin{aligned} & p(4|7, N) \cdot V(4) + p(7|7, N) \cdot V(7) + p(10|7, N) \cdot V(10) = \\ & = 0.1 \cdot 0 + 0.8 \cdot 0 + 0.1 \cdot 1 = 0.1 \end{aligned}$$

(2) **a=East:**

$$\begin{aligned} & p(10|7, E) \cdot V(10) + p(7|7, E) \cdot V(7) + p(8|7, E) \cdot V(8) = \\ & = 0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0.8 \end{aligned}$$

(3) a=South:

$$p(4|7, S) \cdot V(4) + p(8|7, S) \cdot V(8) + p(10|7, S) \cdot V(10) = \\ = 0.1 \cdot 0 + 0.8 \cdot 0 + 0.1 \cdot 1 = 0.1$$

(4) a=West:

$$p(4|7, N) \cdot V(4) + p(7|7, E) \cdot V(7) + p(8|7, W) \cdot V(8) = \\ = 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0$$

$$V(7) = r(7) + \gamma \max[0.1, 0.8, 0.1, 0] = -0.04 + 1 \cdot 0.8$$

$$V(7) = 0.76$$

	1	2	3	4
1	(1) 0	(4) 0	(7) 0.76	(10) +1
2	(2) 0	(5) 0	(8) 0	(11) -1
3	(3) 0	(6) 0	(9) 0	(12) 0

Finding $V(8)$:

(1) a=North:

$$p(8|8, N) \cdot V(8) + p(7|8, N) \cdot V(7) + p(11|8, N) \cdot V(11) = \\ = 0.1 \cdot 0 + 0.8 \cdot 0.76 + 0.1 \cdot (-1) = 0.508$$

(2) a=East:

$$p(11|8, E) \cdot V(11) + p(7|8, E) \cdot V(7) + p(9|8, E) \cdot V(9) = \\ = 0.8 \cdot (-1) + 0.1 \cdot 0.76 + 0.1 \cdot 0 = -0.724$$

(3) a=South:

$$p(8|8, S) \cdot V(8) + p(9|8, S) \cdot V(9) + p(11|8, S) \cdot V(11) = \\ = 0.1 \cdot 0 + 0.8 \cdot 0 + 0.1 \cdot (-1) = -0.1$$

(4) a=West:

$$p(8|8, N) \cdot V(8) + p(7|8, E) \cdot V(7) + p(9|8, W) \cdot V(9) = \\ = 0.8 \cdot 0 + 0.1 \cdot 0.76 + 0.1 \cdot 0 = 0.076$$

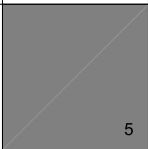
$$V(8) = r(8) + \gamma \max[0.508, -0.724, -0.1, 0.076] = -0.04 + 1 \cdot 0.508$$

$$V(8) = 0.468$$

	1	2	3	4
1	(1) 0	(4) 0	(7) 0.76	(10) +1
2	(2) 0	(5) 0	(8) 0.468	(11) -1
3	(3) 0	(6) 0	(9) 0	(12) 0

And after convergence ($\Delta < \theta$), we get:

MDP gridworld

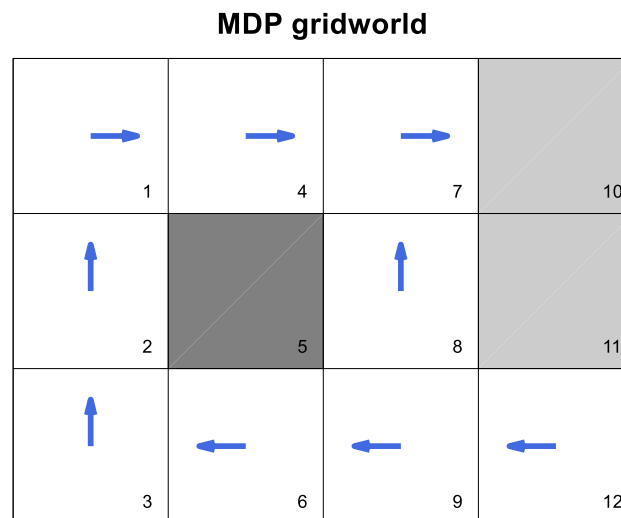
0.812 1	0.868 4	0.918 7	1.000 10
0.762 2	 5	0.660 8	-1.000 11
0.705 3	0.655 6	0.611 9	0.388 12

The policy is:

$$\pi^*(s) = \operatorname{argmax} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_*(s') \right)$$

The solution find policy, which return the action with the highest reward.

The optimal policy:



Second case:

$$\gamma = 0.9 \ ; \ r = -0.04$$

The optimal value function:

MDP gridworld

0.509 1	0.650 4	0.795 7	1.000 10
0.398 2	5	0.486 8	-1.000 11
0.296 3	0.254 6	0.345 9	0.130 12

The optimal policy:

MDP gridworld

→ 1	→ 4	→ 7	10
↑ 2	5	↑ 8	11
↑ 3	→ 6	↑ 9	← 12

In this case, we can see the program is more “adventurer” and willing to take more risk in order to get the short route to the goal. This occur due to the discount factor is smaller than the one in case 1. (Each step is “more expensive”).

Third case:

$$\gamma = 1 ; r = -0.02$$

The optimal value function:

MDP gridworld

0.899 1	0.928 4	0.953 7	1.000 10
0.874 2	5	0.773 8	-1.000 11
0.846 3	0.821 6	0.794 9	0.593 12

The optimal policy:

MDP gridworld

→ 1	→ 4	→ 7	10
↑ 2	5	← 8	11
↑ 3	← 6	← 9	↓ 12

In this case, we can see the program don't take risks and the agent refer to remains in the same grid cell and don't reach cell num. 11. (Even if that would be more longer way). This occur due to the reward is smaller than the one in case 1.

The Matlab code:

```
%% ***** b *****%
Tresh = 10^-4; % Termination threshold
gamma = 0.9; % discount factor
R = -0.04; % reward
% initialize
delta = Tresh;
Reward = R*ones(12,1); Reward(5)=0; Reward(10) = 1; Reward(11) = -1;
V = zeros(12,1); V(10) = 1; V(11) = -1;
Policy = zeros(12,1);

myworld.plot;
myworld.plot_value(V);
myworld.plot;
myworld.plot_policy(Policy);

while delta >= Tresh
    v = V; % v = value of the next state; V = value of the current state;
    delta = 0; % initialize delta
    for state = [1:4, 6:9, 12]
        v0(state) = V(state);
        for Act = 1:4 % Act = N-E-S-W
            val(Act) = 0;
            for nxt_state = [1:4, 6:12]
                val(Act) = val(Act) + PR(nxt_state,state,Act)*v(nxt_state);
                % val(a) = sum(PR(s',s,a)*V(s'))
            end
        end
        V(state) = Reward(state)+gamma*max(val);
        dV = abs(v0(state)-V(state));
        delta = max(delta,dV);
    end
    myworld.plot;
    myworld.plot_value(V);
end

Policy = zeros(12,1);
    for state = [1:4, 6:9, 12]
        for Act = 1:4 % Act = N-E-S-W
            val_p(Act) = 0;
            for nxt_state = [1:4, 6:12]
                val_p(Act) = val_p(Act) +
PR(nxt_state,state,Act)*V(nxt_state);
                % val(a) = sum(PR(s',s,a)*V(s'))
            end
        end
        VP = Reward(state)+gamma*val_p;
        [~,Policy(state)] = max(VP);
    end
myworld.plot;
myworld.plot_value(V);
myworld.plot;
myworld.plot_policy(Policy);
```


c. Solving the MDF using *policy iteration*:

$\gamma = 0.9$; $r = -0.04$ (Same like the second case in section 2.b)

The Matlab code:

```
%% ***** c *****%
Tresh = 10^-4; % Termination threshold
gamma = 0.9;
R = -0.04;

% initialize
delta = Tresh;
Reward = R*ones(12,1); Reward(5)=0; Reward(10) = 1; Reward(11) = -1;
V = zeros(12,1); V(10) = 1; V(11) = -1;
Policy_C = randi(4,12,1); % random policy

PS = 0;
while PS == 0

    %% step one : policy evaluation
    while delta >= Tresh
        delta = 0;
        v = V; % v = value of the next state; V = value of the
current state;
        for state = [1:4, 6:9, 12]
            v0(state) = V(state);
            val(state) = 0;
            for nxt_state = [1:4, 6:12]
                val(state) = val(state) +
PR(nxt_state,state,Policy_C(state))*v(nxt_state);
                % val(a) = sum(PR(s',s,policy)*v(s'))
            end
            V(state) = Reward(state)+gamma*val(state);
            dV = abs(v0(state)-V(state));
            delta = max(delta,dV);
        end
    end

    %% step two : policy improvement
    PS = 1;
    for state = [1:4, 6:9, 12]
        Prev_Act = Policy_C(state);
        for Act = 1:4 % Act = N-E-S-W
            val_act(Act) = 0;
            for nxt_state = [1:4, 6:12]
                val_act(Act) = val_act(Act) +
PR(nxt_state,state,Act)*V(nxt_state);
                % val(a) = sum(PR(s',s,a)*V(s'))
            end
        end
        VPC = Reward(state)+gamma*val_act;
```

```
        [~,Policy_C(state)] = max(VPc);  
        if Prev_Act == Policy_C(state)  
            PS = 1;  
        else  
            PS = 0;  
        end  
    end  
  
    delta = Tresh;  
    myworld.plot;  
    myworld.plot_value(V);  
    myworld.plot;  
    myworld.plot_policy(Policy_C);  
end
```

The Matlab code was run several times and every time, after few iterations (between about 1 and 4 iterations, depend on the initial random policy) the solution converged to the optimal result. Below shown example of the solving the MDP using policy-iteration (3 iterations to get the optimal result):

Iteration one:

MDP gridworld


-0.365 1	-0.327 4	-0.209 7	1.000 10
-0.369 2		-0.319 8	-1.000 11
-0.373 3	-0.372 6	-0.340 9	-0.412 12

MDP gridworld




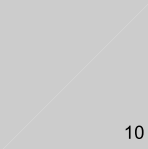



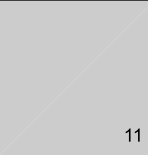




→ 1	→ 4	→ 7	
↑ 2		↑ 8	
↑ 3	→ 6	↑ 9	↓ 12

Iteration two:

MDP gridworld

0.509 1	0.650 4	0.795 7	1.000 10
0.399 2	 5	0.486 8	-1.000 11
0.295 3	0.238 6	0.327 9	-0.056 12

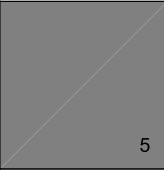
MDP gridworld

 1	 4	 7	 10
 2	 5	 8	 11
 3	 6	 9	 12

Iteration three:




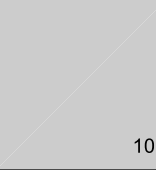








The optimal value function:

MDP gridworld

0.509 1	0.650 4	0.795 7	1.000 10
0.399 2	 5	0.486 8	-1.000 11
0.296 3	0.254 6	0.345 9	0.130 12

The optimal policy:

MDP gridworld

 1	 4	 7	 10
 2	 5	 8	 11
 3	 6	 9	 12

As can see in the optimal results (optimal value function and optimal policy), the solution obtained using value iteration (section b – in the second case) is same likes the results obtained using policy iteration (shown above).

The Solving of the MDF using *policy iteration* or using *value iteration* for same problem (similar discount factor γ and similar reward function R) is obtained similar optimal results for value function and policy.

During the solution, I noticed that sometimes the solution is not converged to the optimal result but is converged to other optimal result (“sub-optimal”).

This may occur due to the initial random policy.

In conclusion, we prefer to use *policy iteration* when we have many actions or when we start from a “fair” policy. In the other side, we prefer to use *value iteration* when we have few actions and transition is acyclic.