Stochastic Optimization homework (project 2)

Tommaso Monopoli
s278727

Please refer to the appendix A at the end of this document for all the Matlab codes of this project

1 Problem introduction

The problem focuses on studying an epidemic model in the context of a company's workplace, and the possible emergency strategies that the company can adopt in order to mantain a high daily income during the epidemic.

The company has $N = 6$ clerks. In the city where the company is located, a light disease is circulating. Each day, every working clerk has a constant probability $p_i = 1/4$ of getting infected. Infected clerks are required not to work until their recovery. Every day each infected clerk has a probability $p_r = 1/5$ of recovering.

We introduce the following notation:

- W_n: the number of effectively working clerks during day n;
- I_n: the number of clerks which get infected on day n (they are easily spotted and cannot work on day n);
- R_n: the number of clerks which recover on day n (they immediately go back to work on day n);

The number of clerks that are out of service on day n is therefore $N - W_n$.

This epidemic model evolves according to the following stochastic law:

$$W_{n+1} = W_n - I_{n+1} + R_{n+1}$$

where $I_{n+1} \sim Bin(W_n, p_i)$ and $R_{n+1} \sim Bin(N - W_n, p_r)$

The transition probabilities can be computed with the following formula

$$p(i, j) = \sum_{k=0}^{N} f_B(k, i, p_i) \cdot f_B(j - i + k, N - i, p_r) \tag{1}$$

where $f_B(k, n, p)$ is the probability mass function for a binomial random variable $Bin(n, p)$.

The daily income gained by the company, when no safety measures are taken to contrast the spread of the disease in the workplace ("*standard safety protocol*"), is:

$$r(i, 1, j) = 50 \cdot (\exp(i/7) - 1) \tag{2}$$

Notice that the income of day n depends only on the number i of clerks which effectively worked on that day.

At any time the company is capable of setting up an "*emergency safety protocol*" with stronger safety measures that decrease the probability of new infections to $p_i = 1/20$. These measures have the side effect of reducing the daily income to the following values:

$$r(i, 2, j) = \begin{cases} 0 & \text{if } i = 0 \\ 35 \cdot (\exp((i-1)/7) - 1) & \text{else} \end{cases} \tag{3}$$

This stochastic process can be clearly seen as a **discrete-time Markov decision process**, characterized by:

- state variable W_n and seven possible states $\mathcal{S} = \{0, 1, \ldots, 6\}$;
- a set of actions $\mathcal{A} = \{1, 2\}$, where 1: standard protocol and 2: emergency protocol;
- transition probabilities given by formula (1) (with p_i equal to $1/4$ in case of standard protocol and $1/20$ in case of emergency protocol);
- avg. total discounted reward with $\lambda = 0.995$ as performance metric:

$$v_\mu(i) = \lim_{n\to\infty} \mathbb{E}\left(\sum_{m=1}^{n} \lambda^{m-1} \cdot r(W_{m-1},\, \mu(W_{m-1}),\, W_m) \;\middle|\; W_0 = i \right)$$

  where μ is a policy.

The goal of the project is to find the optimal policy $\mu^* \in \{1, 2\}^7$ which maximizes all the components of the avg. total discounted reward vector v_μ.

We will solve this problem with two dynamic optimization methods: the Q-factor value iteration algorithm (sect. 2) and the Q-learning algorithm (sect. 3).

2 Dynamic Programming

In the context of dynamic optimization, Dynamic Programming is an approach that transforms a complex dynamic optimization problem into a sequence of simpler problems; the optimal policy is then found by finding the optimal solution at each step recursively.

2.1 Q-factor value iteration algorithm

The Q-factor value iteration algorithm is a dynamic programming method for finding an optimal policy for a MDP. Here it is presented (and later used) in the version for the discounted reward performance metric.

It can be shown that an optimal policy μ^* is so that

$$\mu^*(i) = \arg\max_{a \in \mathcal{A}} Q(i, a)$$

where $Q(i, a)$ is the so-called Q-factor associated to the state-action couple (i, a). There are $|\mathcal{S}| \cdot |\mathcal{A}|$ Q-factors in total.

Q-factors are given by the Q-factor Bellman optimality equation:

$$Q(i,a) = \sum_{j \in \mathcal{S}} p(i,a,j) \left[r(i,a,j) + \lambda \max_{b \in \mathcal{A}} Q(j,b) \right] \tag{4}$$

Note that this equation is nonlinear (because of the max operator), so it cannot be solved using linear algebra techniques. Nonetheless, the Q-factors can be computed iteratively with the following **Q-factor value iteration algorithm**:

1. Set $k = 1$ (iteration number). Select any arbitrary vector for $Q^1(i,a)$, for any $i \in \mathcal{S}$ and any $a \in \mathcal{A}$. Specify a small value ε
2. For each $i \in \mathcal{S}$ and $a \in \mathcal{A}$ compute

$$Q^{k+1}(i,a) = \sum_{j \in \mathcal{S}} p(i,a,j) \left[r(i,a,j) + \lambda \max_{b \in \mathcal{A}} Q^k(j,b) \right]$$

3. Calculate

$$J^{k+1}(i) = \max_{a \in \mathcal{A}} Q^{k+1}(i,a) \text{ and } J^k(i) = \max_{a \in \mathcal{A}} Q^k(i,a)$$

   if $\|J^{k+1} - J^k\|_\infty < \varepsilon \frac{1-\lambda}{2\lambda}$ increment k by 1 and go to the next step. Otherwise increment k by 1 and go back to step 2.
4. For each $i \in \mathcal{S}$ set the ε-optimal policy μ^* to

$$\mu^*(i) = \arg\max_{a \in \mathcal{A}} Q^k(i,a)$$

It can be proven that an ε-optimal policy tends to an actual optimal policy for a sufficiently small ε; therefore, the choice of ε is critical.

2.2 Experiment and results

In our setting there are a total of $|\mathcal{S}| \cdot |\mathcal{A}| = 7 \cdot 2 = 14$ Q-factors to be computed.

Note that P_1 and P_2, the transition matrices associated to the standard protocol and emergency protocol respectively, contain $7 \times 7 = 49$ elements each, so they can be easily stored in memory. Therefore, it is convenient to compute them beforehand (using formula (1)) and read from them when performing step 2 of the value iteration algorithm, instead of computing transition probabilities on-the-fly without caching them. This becomes particularly important when the tolerance ε of the algorithm is set to be very small: in this scenario, the algorithm would need a relatively high number of iterations to converge, say K, resulting in a total of $7K$ on-the-fly calls to function (1).

The same arguments apply to the reward matrices R_1 and R_2, computed with formulas (2) and (3). Moreover, it is particularly convenient to compute these two matrices beforehand, because the rewards depend only on the initial state i and the action a, and not on the final state j of the transition, so each of these two matrices is composed of 7 identical columns (therefore, only 7 elements have to be computed and stored).

I have run the algorithm initializing $Q^1(i,a) = 0 \ \forall i, a$, and experimented with different values of ε. Results are reported in table 1.

ε	#iterations	ε-optimal policy
0.0001	3680	$(1,1,1,2,2,2,2)$
0.001	3221	$(1,1,1,2,2,2,2)$
0.01	2761	$(1,1,1,2,2,2,2)$
0.1	2302	$(1,1,1,2,2,2,2)$

Table 1: ε-optimal policies found with Q-factor value iteration algorithm

The optimal policy found is $(1,1,1,2,2,2,2)$. The found Q-factors (when $\varepsilon = 0.0001$, i.e. the best approximation of them derived from this experiment) are:

$$Q = \begin{pmatrix} 5043.23 & 5043.23 \\ 5063.19 & 5059.97 \\ 5085.19 & 5083.70 \\ 5109.58 & 5110.27 \\ 5136.66 & 5140.00 \\ 5166.60 & 5172.50 \\ 5199.60 & 5207.89 \end{pmatrix} \tag{5}$$

(values are rounded to the 2nd decimal digit)

We can interpret this result as follows:

- when 3 or more clerks are currently working, it is optimal to safeguard them with the emergency safety protocol;
- when 1 or 2 clerks are currently working, it is optimal to follow the standard safety protocol;
- when there are no clerks working, the two protocols are equivalent (since the income on that day would be 0 either way); in fact, notice that $Q(1,1) = Q(1,2)$

Since the stopping criterion is a tolerance on the quantity $\|J^{k+1} - J^k\|_\infty$, it is interesting to plot the evolution of this quantity for each iteration k (see fig. (1)). The sequence seems to decrease with an exponential decay.

3 Reinforcement Learning

Reinforcement Learning can be seen as a form of simulation-based dynamic programming. Contrary to DP, RL has the advantage of not requiring the transition probabilities $p(i,a,j)$, and despite this it can generate near-optimal solution (which converge to the optimal solution with a sufficiently high number of iterations).

3.1 Q-learning algorithm

It can be shown that every Q-factor can be expressed as an average of a random variable:

$$Q(i,a) = \mathbb{E}\left(r(i,a,j) + \lambda \max_{b \in \mathcal{A}} Q(j,b)\right) = \mathbb{E}\,(\text{SAMPLE})$$

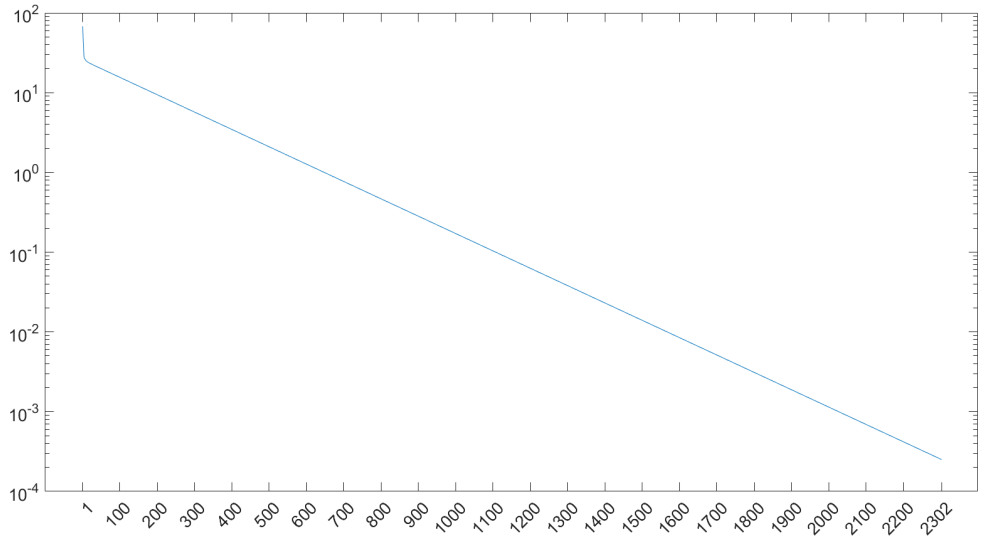

Figure 1: Evolution of $\|J^{k+1} - J^k\|_\infty$ (y axis is in logarithmic scale)

Therefore, instead of using eq. (4) to estimate the Q-factors (as shown in the Q-factor value iteration algorithm), we could instead compute incrementally this average (in a Robbins-Monro algorithm, with step size α_k) within a simulator for the stochastic process.

This results in the following **Q-learning algorithm**:

1. Set k=0. Select any arbitrary value for $Q^0(i, a)$, for any $i \in \mathcal{S}$ and any $a \in \mathcal{A}$. Specify a (large) maximum number of iterations $k_{\max}$. Set the initial state X_0 uniformly at random. Decide a sequence α_k
2. Select an action a uniformly at random
3. Simulate the next state X_{k+1}. Read off the values α_{k+1} and $r(X_k, a, X_{k+1})$
4. Update $Q(X_k, a)$ with the following rule

$$Q^{k+1}(X_k, a) \longleftarrow (1 - \alpha_{k+1})Q^k(X_k, a) + \alpha_{k+1}\left[r(X_k, a, X_{k+1}) + \lambda \max_{b \in \mathcal{A}} Q^k(X_{k+1}, b)\right]$$

5. Increment k by 1. If $k < k_{\max}$ go to step 2, otherwise go to step 6
6. For each $i \in \mathcal{S}$ set the optimal policy μ^* to

$$\mu^*(i) = \arg\max_{b \in \mathcal{A}} Q^k(i, b)$$

It can be proven that the learned policy tends to an actual optimal policy for a sufficiently large n. of iterations; therefore, the choice of $k_{\max}$ is critical.

Moreover, also the choice of the sequence α_k is important. α_k is called learning rate, or step size, and determines to what extent newly acquired information overrides old information. Values close to 0 will result in an update of $Q(i, a)$ which exploits prior knowledge almost exclusively, whereas values close to 1 will result in updates which consider almost only the most recent information.

In order for the Robbins-Monro algorithm to converge, two technical conditions needs to be fulfilled: $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$. A well-known choice in literature is $\alpha_k = \frac{A}{B+k}$, typically with $B = 2A$. The higher A and B, the slower the sequence decreases to 0, and this is a good choice whenever $k_{\max}$ is high (because then the Q-factors can be updated in a somewhat relevant manner even in very late iterations).

3.2 Experiment and results

I have run the algorithm initializing $Q^0(i, a) = 0 \ \forall i, a$, and experimented with different values of $k_{\max}$. For each value $k_{\max}$, many runs/simulations were performed, to check whether the algorithm learns the same policy every time, independently from the particular starting state W_0. I have chosen $\alpha_k = \frac{3000}{6000+k}$. The graph of this sequence is reported in figure 2 as a reference.

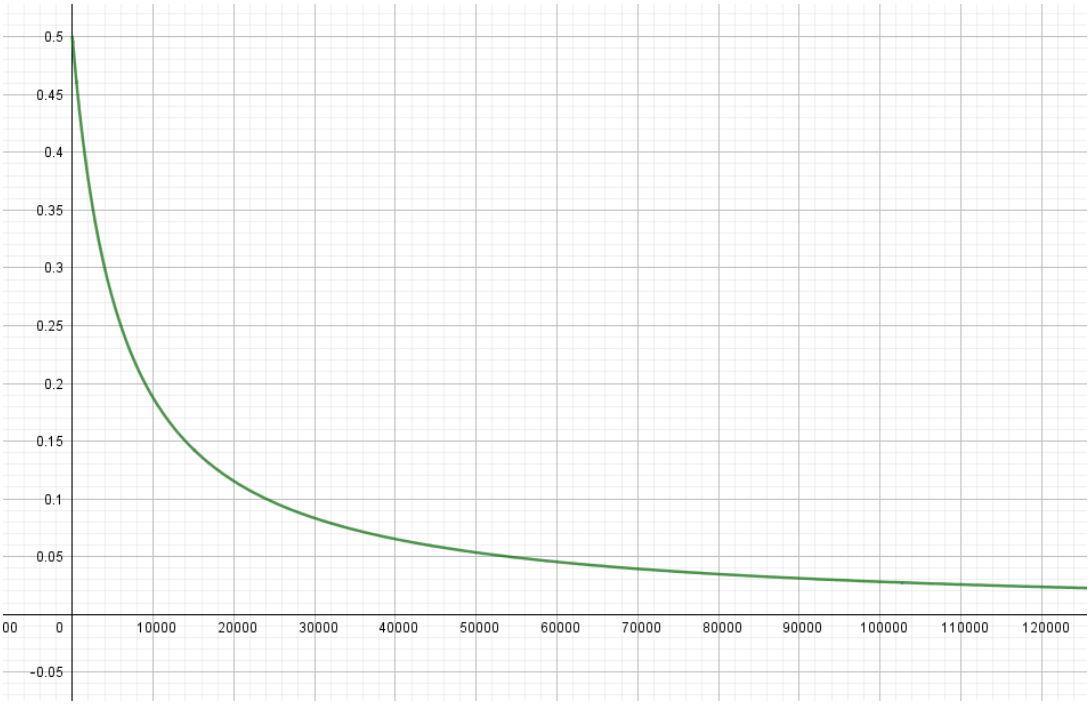

Figure 2: Graph of the sequence $\alpha_k = \frac{3000}{6000+k}$

Results are reported in table 2.

$k_{\max}$	Elapsed time	Learned policy
10^7	$\sim$ 150s	$(1, 1, 1, 2, 2, 2, 2)$
10^6	$\sim$ 15s	$(1, 1, 1, 2, 2, 2, 2)$, $(1, 1, 1, 1, 2, 2, 2)$
10^5	$\sim$ 1.5s	very different at every run

Table 2: Learned policies found with Q-learning algorithm

Some comments on these results:

- When $k_{\max} = 10^7$, the learned policy is always $(1, 1, 1, 2, 2, 2, 2)$, which is exactly the optimal policy found with the value iteration algorithm.
- When $k_{\max} = 10^6$, the learned policy is $(1, 1, 1, 1, 2, 2, 2)$ on some runs, $(1, 1, 1, 1, 2, 2, 2)$ on others.

- When $k_{\max} = 10^5$, the learned policy is very different at each run.

The peculiar behavior of the algorithm for $k_{\max} = 10^5, 10^6$ might be explained by looking at the Q-factors found with the value iteration algorithm (matrix (5)): for each state $i \in \mathcal{S}$, $Q(i,1)$ and $Q(i,2)$ are very close, so the Q-learning algorithm might need to simulate a large number of iterations in order to approximate this small distance well enough $\forall i$, i.e. in order to converge to the actual optimal policy. This is especially true for $\mu^*(3)$: $Q(3,2) - Q(3,1) = 0.69$ (hence the oscillation in the learned $\mu(3)$ when $k_{\max} = 10^6$).

The found Q-factors when $k_{\max} = 10^7$, i.e. their best approximation derived from this experiment, are:

$$Q_{\text{learned}} = \begin{pmatrix} 5052.76 & 5052.62 \\ 5072.38 & 5068.50 \\ 5093.41 & 5091.63 \\ 5118.18 & 5118.41 \\ 5145.03 & 5148.83 \\ 5175.32 & 5181.24 \\ 5207.88 & 5215.86 \end{pmatrix}$$

(values are rounded to the 2nd decimal digit)

Despite $Q_{\text{learned}}(i,a)$ being a bit higher than its corresponding $Q(i,a)$ $\forall i, a$, the differences between Q-factors of same state and different action is approximated well, and this proves essential in order to find the actual optimal policy μ^*.

4 Conclusion

In conclusion, an optimal policy has been found both with dynamic programming and reinforcement learning: $\mu^* = (1,1,1,2,2,2,2)$. This is the policy which maximizes the expected total discounted reward with $\lambda = 0.995$.

To validate my findings, I computed $\overline{v}_{\mu^*}(i)$, the sample mean of the total discounted reward over 100 simulations, for each of the 7 states taken as starting state. Each simulation consisted of 1000 iterations. The result is the following:

$$\overline{v}_{\mu^*} = (5008.5, 5023.8, 5027.5, 5081.9, 5107.4, 5130.2, 5172.3)$$

In comparison, when the chosen policy is $\mu = (1,1,1,1,1,1,1)$ (only the standard safety protocol is applied) is the following:

$$\overline{v}_{\mu} = (4732.9, 4792.8, 4818.3, 4832.7, 4840.9, 4902.5, 4920.1)$$

Therefore, it is clear that adopting the emergency protocol can prove more convenient for the company than applying only standard safety measures, and in particular when there are many clerks in the workplace (3 or more).

A Matlab code

In this section is reported the Matlab code produced for this project

```
%% Preliminar objects
% There are
% - 7 states (S={0,1,...,6})
% - 2 possible actions (A={1,2} where 1: standard safety protocol ...
    and 2:
%   emergency safety protocol)
% Therefore, there are 2^7=128 possible policies

clear; clc; close all;

N = 6;
p_i = [1/4, 1/20];
p_r = 1/5;
lambda = 0.995;

rng(1234); % fixed random seed, for replicability of the code

% Compute the transition matrix associated to the standard safety ...
    protocol
P1 = zeros(7,7);
for i=0:6
    for j=0:6
        P1(i+1,j+1) = binopdf([0:6], i, p_i(1)) * binopdf([0:6]+j-i, ...
            N-i, p_r)';
    end
end

% Compute the transition matrix associated to the emergency safety ...
    protocol
P2 = zeros(7,7);
for i=0:6
    for j=0:6
        P2(i+1,j+1) = binopdf([0:6], i, p_i(2)) * binopdf([0:6]+j-i, ...
            N-i, p_r)';
    end
end

% Compute the reward matrix associated to the standard safety protocol
R1 = zeros(7,7);
for i=0:6
    for j=0:6
        R1(i+1,j+1) = 50*(exp(i/7)-1);
    end
end

% Compute the reward matrix associated to the emergency safety protocol
R2 = zeros(7,7);
for i=0:6
    for j=0:6
        R2(i+1,j+1) = (i~=0)*35*(exp((i-1)/7)-1);
    end
end
```

```
48
49 % Concatenate the two transition matrices
50 P = cat(3,P1,P2);
51 % Concatenate the two reward matrices
52 R = cat(3,R1,R2);
53
54 % NB: I computed P1, P2, R1, R2 explicitly (because these matrices ...
       are just
55 % 7x7, hence they are not so massive in terms of memory occupation);
56 % nonetheless, one could compute p(i,a,j) and r(i,a,j) on-the-fly ...
       with the
57 % following functions
58 %
59 p = @(i,a,j) binopdf([0:6], i-1, p_i(a)) * binopdf([0:6]+j-i, ...
       N-(i-1), pr)';
60 r_standard = @(i) 50*(exp(i/7)-1);
61 r_emergency = @(i) (i~=0)*35*(exp((i-1)/7)-1);
62 r = @(i,a,j) (a==1)*r_standard(i-1) + (a==2)*r_emergency(i-1);
63 %
64 % NB: in this last formula j (the new state) is not used: all ...
       columns of
65 % R1 and R2 respectively are equal
66
67
68 %-------------------------------------------------------------------------%
69 %% Find mu* with dynamic programming (Q-factor value iteration ...
       algorithm)
70
71 % Step 1
72 k = 1;
73 Q_old = zeros(7,2); % |S|*|A|
74 epsilon = 0.1;
75 J_infty_norm_list = []; % this list will contain the quantities ...
       max(J_new-J_old)
76
77 while true
78
79     % Step 2 - update Q
80     for i=1:7
81         for a=1:2
82             Q_new(i,a) = sum( P(i,:,a) .* ( R(i,:,a) + ...
                   lambda.*max(Q_old,[],2)' ) );
83             % NB: the command max(Q_old,[],2) means "find the maximum
84             % element of each row of Q_old"; it is therefore a column
85             % vector of dimension 7
86         end
87     end
88
89     % Step 3 - calculate J^k+1 and J^k and check stopping criterion
90     J_new = max(Q_new,[],2)';
91     J_old = max(Q_old,[],2)';
92     J_infty_norm_list = [J_infty_norm_list, max(J_new-J_old)];
93     if max(J_new-J_old) < epsilon*(1-lambda)/(2*lambda)
94         % Step 4 - compute the optimal policy
95         [~, optimal_policy_qdp] = max(Q_new,[],2);
96         optimal_policy_qdp = optimal_policy_qdp';
97         break
98     else
```

```
99          k = k+1;
100         Q_old = Q_new;
101     end     %...and back to step 2
102
103 end
104
105 disp(['The optimal policy is:'])
106 disp(optimal_policy_qdp)
107
108 for i=1:7
109     if Q_old(i,1) == Q_old(i,2)
110         disp(['The 2 protocols are equivalent when there are ', ...
                num2str(i-1), ' clerks working.'])
111     end
112 end
113
114 % Plot the evolution of max(J_new-J_old)
115 fig = figure();
116 semilogy(1:k,J_infty_norm_list)
117 xticks([1,100:100:2299,2302])
118 a = get(gca,'XTickLabel');
119 set(gca,'XTickLabel',a,'fontsize',16)
120 xtickangle(45)
121
122
123 %-------------------------------------------------------------------------%
124 %% Find mu* with reinforcement learning (Q-learning algorithm)
125
126 % Step 1
127 Q_old = zeros(7,2); % Q factors (will be updated iteratively)
128 kmax = 1e7;   % max n. of iterations
129 old_state = randi(7);  % initial state
130 compute_alpha = @(k) 3000/(6000+k);
131
132 tic    % start timer
133 for k=1:kmax
134
135     % Step 2 - select an action a uniformly at random
136     a = randi(2);
137
138     % Step 3 - simulate the next state X(k+1), from P(X(k),:,a)
139     new_state = old_state - binornd(old_state-1,p_i(a)) + ...
            binornd(N+1-old_state,p_r);
140
141     % Read off the values alpha_k+1 and R(X(k),X(k+1),a)
142     alpha = compute_alpha(k);
143     %rew = R(old_state,new_state,a);
144
145     % Step 4 - update Q(X(k),a)
146     Q_new = Q_old;
147     Q_new(old_state,a) = (1-alpha) * Q_old(old_state,a) + alpha * ...
            (R(old_state,new_state,a)+lambda*max(Q_old(new_state,:)));
148
149     % Step 5 - if k < kmax, go to next iteration
150     Q_old = Q_new;
151     old_state = new_state;
152 end
153
```

```
[~, optimal_policy_rl] = max(Q_new,[],2);
toc    % stop timer
optimal_policy_rl = optimal_policy_rl';

disp(['The optimal policy is:'])
disp(optimal_policy_rl)


%--------------------------------------------------------------------------%
%% Simulation

kmax = 1e3;     % n. of iterations
max_n_simulations = 100;    % n. of simulations
tot_disc_rew = zeros(1,max_n_simulations);

% Starting state choice
starting_state = 1;     % 2,3,4,5,6,7

% Choose policy
%policy = optimal_policy_dp;
policy = [1,1,1,2,2,2,2];   % [1,1,1,1,1,1,1]

% Begin simulation
for s=1:max_n_simulations
    old_state = starting_state;

    for k=1:kmax
        % simulate the next state
        new_state = old_state - ...
            binornd(old_state-1,p_i(policy(old_state))) + ...
            binornd(N+1-old_state,p_r);

        % update reward
        tot_disc_rew(s) = tot_disc_rew(s) + ...
            (lambda^(k-1))*(R(old_state,new_state,policy(old_state)));

        % go to next iteration
        old_state = new_state;
    end
end

mean_tot_disc_rew = mean(tot_disc_rew)
disp(mean_tot_disc_rew)


%--------------------------------------------------------------------------%
```