EEE598 Fall 2021
HW4
Jacob Sindorf

Homework 4

Assistant Professor Nicolò Michelusi

Office: GWC 330

In your submission, please include:

- printout of Matlab scripts (pdf) that you created and .m files
- printout of figures
- discussion and steps as requested
- all printouts (solutions, coding parts, figures and discussions) should be included in a SINGLE pdf file.

Please, be clear, concise and organized, and make sure your hand-writing is readable. Make sure that your Matlab code is clearly organized, and provide sufficient descriptions to help me follow your reasoning.

10 points total

I. Inventory control problem with infinite horizon discounted cost

Consider the same inventory management problem of HW1 and HW3:

A company needs to manage its inventory levels of a certain product in a warehouse. Assume that the control problem operates at discrete stages $k = 0, 1, 2, \ldots$. For instance, each stage may represent a month, or a year (depends on the application).

At stage k, let $S_k \in \{0, \ldots, M\}$ be the current inventory level at the warehouse, and M be the maximum amount of products that can be stocked at the warehouse. The company then refills $U_k \in \{0, 1, \ldots, M - S_k\}$ units, with U_k part of a policy to be optimized in Part 1.

~~The company refills its inventory only when its inventory level reaches 0.~~

During stage k, D_k items are purchased from customers (demand). It follows a probability distribution $P(\cdot)$, independent and identically distributed (i.i.d.) over stages, so that $P(d)$ is the probability that $D_k = d$ items are purchased by costumers in stage k.

The cost for the company to purchase each unit is c. The revenue to the company for each unit sold is r. However, due to finite inventory, some of the requests may not be met. In this case, each unit of unsatisfied requests incur a penalty of p. Finally, the cost of maintaining the inventory is m per unit per stage.

Assumptions on the sequence of events:

1 - The stock level is S_k at the beginning of stage k

2 - Then, the maintenance cost is incurred

3 - Then, U_k new units are purchased, if necessary (based on the policy outlined earlier)

4 - Then, D_k units of demand occur. We assume that D_k is uniform in $\{0, \ldots, M\}$, so that $P(D_k = d) = 1/(M + 1), \ \forall d = 0, 1, \ldots, M$.

States: $\{0, 1, \dots, M\}$

Action: $U(i) = \{0 \dots M-i\} \; \forall i$ ~ action cannot exceed M

$S_{k+1} = f(S_k, U_k, D_k) = \max(S_k + U_k - D_k, 0)$

- if $D_k > S_k + U_k$, $S_{k+1} = 0$

Profit

Cost ⇒ 1) $S_k = i$ at beginning

2) maintenance $-m(i)$

3) purchase $\mu_k(i) = u$ $-c(u)$

4) demand

$$\text{new state} = (i+u) \begin{cases} +r(\underbrace{i+u}_{\text{available}}) - p(\underbrace{d-(i+u)}_{\text{unmet}}) & d > i+u \text{ goes to } 0 \\ +r(d) & d \leq i+u \end{cases}$$

Consider a scenario with $M = 20$, $c = 1$, $r = 2$, $p = 1$, $m = 0.5$, and discount factor $\gamma = 0.9$ (same as HW1 and HW3). You will need the optimal policy that maximizes the infinite horizon discounted expected profit (you should have already done this in HW3):

$$V(i) \triangleq \mathbb{E}\left[\Big(\sum_{k=0}^{\infty} \gamma^k p_k\Big) | S_0 = i\right].$$

1) **[1 point]** Implement a function that takes as input the current state S_k, current action U_k, current demand D_k, (and the system parameters) and outputs the profit p_k and next state S_{k+1}:

$$[S_{k+1}, p_k] = \text{nextstate\_profit}(S_k, U_k, D_k, \text{parameters})$$

Note that p_k is the realization of the demand (function of S_k, U_k, D_k), not to be confused with the expected profit. Note, you will need this for the Monte Carlo simulation; you should have already done something similar in HW1 and HW3.

[Sk+1 , pk] = nextstate_profit (Sk , Uk , Dk

```
function [Snext, profit] = nextstate_profit(S,U,d,c,r,p,m)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Generate next State
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Snext = max((U + S) - d, 0);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Generate profit
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if d > (S + U) %sequence is greater, get penalty
        profit= -S*m  -c*U + ...
              (S+U)*r - ...
               p*(d - (S+U));
    else
        profit = -S*m - c*U +...
               d*r;
    end

end
```

Published with MATLAB® R2020b

2) **[3 point]** Implement a Q-learning algorithm with ϵ-greedy action selection. The algorithm should run over $K = 10^4$ stages, starting from an empty inventory ($S_0 = 0$), with discount factor $\gamma = 0.9$. I suggest to use the learning rate parameter $\alpha = 0.1$ and the exploration parameter $\epsilon = 0.4$ (but you are welcome to try different values and explore what works best). Note that you will need to use the function "nextstate_profit" implemented in the previous item, to generate the sequence of states and profits, where D_k is uniform in $\{0, \ldots, M\}$ and U_k is chosen based on ϵ-greedy action selection. Since it will be helpful later, I recommend to generate the sequence of demands beforehand, i.e, generate DD $= [D_0, D_1, \ldots, D_{K-1}]$ based on the uniform distribution. This will be helpful to make the comparison of various algorithms as fair as possible. I suggest to initialize the state-value function as $Q(i, u) = 0, \forall i, u$, and the base policy as the lazy one ($v(0) = M$ and $v(i) = 0, \forall i > 0$).

Q learning

$\gamma = .9$; $\alpha = 0.1$; $\varepsilon = 0.4$; $K = 10^4$

- Initialize : $Q(i,u) = 0 \quad \forall i \in \{0, \ldots, M\}$

$\forall u \in \{0 \ldots M-i\}$

base policy $\begin{cases} v(0) = M \\ v(i) = 0 \quad \forall i > 0 \end{cases}$

- Initialize state, $S_0 = 0$

for $k = 0, 1, 2 \ldots\ldots K$

- Execute action U_k (ε-greedy)

$$U_k = \begin{cases} V(S_k) & \text{wp. } (1-\varepsilon) \\ \text{Uniform}(0, \ldots M-S_k) & \text{wp. } \varepsilon \end{cases}$$

- get next state / profit

$$[P_k, S_{k+1}] = \text{nextstate\_profit}(S_k, U_k, D_k)$$

↳ generated before

- Update state action value

$$Q(S_k, U_k) := Q(S_k, U_k) + \alpha \left[C_k + \gamma \max_{u \in (0 \ldots M-S_k)} Q(S_{k+1}, u) - Q(S_k, U_k) \right]$$

- update policy

$$V(S_k) = \underset{u \in \{0, \ldots M-S_k\}}{\text{argmax}} \; Q(S_k, u)$$

- $S_k = S_{k+1}$;

- next k

$0 \ldots M-S$

end

```
function [bestpolicy, q_sampleavg, q_tot] =
 q_learn(D,gamma,alpha,eps,K)
%need sequence of demands,gamma,alpha,eps,K
%returns sequence of profits and best policy
    %constants
    M = 20;
    c = 1; r = 2; p = 1; m = .5;

    %Initialize
    s = 0;
    %Base policy
    mubase = zeros(1,M+1); mubase(1) = M;
    %Q
    %Note, Action is at most M - i, but easier to have zeros in unused
 parts
    %M+1 to account for 0 to M
    Q = zeros(M+1,M+1);

    %Loop steps
    for k = 1:K
        %eps greedy, get action
        if rand < (1 - eps)
            u = mubase(s+1);
        else
            u = randi([0 M-s]);
        end
        %get next state and profit
        [Snext, profit] = nextstate_profit(s,u,D(k),c,r,p,m);

        %update Q
        Q(s+1,u+1) = Q(s+1,u+1) + alpha*(profit + ...
                     gamma*max(Q(Snext+1,1:(M-s+1))) - Q(s+1,u+1));
        %update policy
        [val, mubase(s+1)] = max(Q(s+1,(1:M-s+1)));
        mubase(s+1) = mubase(s+1) - 1; %fix 1 index
        %Store the profit
        q_profits(k) = profit;
        q_sampleavg(k) = 1/k*sum(q_profits(1:k)); %%%%%1/k or 1/(k+1)?
        q_tot(k) = sum(q_profits(1:k));
        s = Snext;

    end
    bestpolicy = mubase;

end
```

Published with MATLAB® R2020b

3) **[3 point]** Implement am expected Sarsa algorithm with ϵ-greedy action selection. Same as above, $K = 10^4$ stages, starting from an empty inventory ($S_0 = 0$), discount factor $\gamma = 0.9$, learning rate parameter $\alpha = 0.1$, exploration parameter $\epsilon = 0.4$; state-value function initialized as $Q(i, u) = 0, \forall i, u$, and base policy initialized as the lazy one.

Expected SARSA

$\gamma = .9$; $\alpha = 0.1$; $\varepsilon = 0.4$; $K = 10^4$

- Initialize : $Q(i,u) = 0 \quad \forall i \in \{0, \ldots, M\}$
$\forall u \in \{0 \ldots M-i\}$

base policy $\begin{cases} V(0) = M \\ V(i) = 0 \quad \forall i > 0 \end{cases}$

- Initialize state, $S_0 = 0$

for $k = 0, 1, 2, \ldots, K$

- Execute action U_k (ε-greedy)

$$U_k = \begin{cases} V(S_k) & \text{w.p. } (1-\varepsilon) \\ \text{Uniform}(0, \ldots, M - S_k) & \text{w.p. } \varepsilon \end{cases}$$

- get next state / profit

$$[P_k, S_{k+1}] = \text{nextstate\_profit}(S_k, U_k, D_k)$$

(D_k: generated before)

- Compute V_{next}

$$V_{next} = (1-\varepsilon) Q(S_{k+1}, V(S_{k+1})) + \frac{\varepsilon}{(M - S_{k+1}) + 1} \sum_{u=0}^{M - S_{k+1}} Q(S_{k+1}, u)$$

- Update state action value

$$Q(S_k, U_k) := Q(S_k, U_k) + \alpha \left[C_k + \gamma \cdot V_{next} - Q(S_k, U_k) \right]$$

- update policy

$$V(S_k) = \underset{u \in \{0, \ldots M - S_k\}}{\text{argmax}} \; Q(S_k, u)$$

$S_k = S_{k+1}$

next k

end

```
function [bestpolicy, es_sampleavg, es_tot] =
 exp_sarsa(D,gamma,alpha,eps,K)
%need sequence of demands,gamma,alpha,eps,K
%returns sequence of profits and best policy
    %constants
    M = 20;
    c = 1; r = 2; p = 1; m = .5;

    %Initialize
    s = 0;
    %Base policy
    mubase = zeros(1,M+1); mubase(1) = M;
    %Q
    %Note, Action is at most M - i, but easier to have zeros in unused
 parts
    %M+1 to account for 0 to M
    Q = zeros(M+1,M+1);

    %Loop steps
    for k = 1:K
        %eps greedy, get action
        if rand < (1- eps)
            u = mubase(s+1);
        else
            u = randi([0 M-s]);
        end
        %get next state and profit
        [Snext, profit] = nextstate_profit(s,u,D(k),c,r,p,m);
        %calc Vnext
        Vnext = (1-eps)*Q(Snext+1,(mubase(Snext+1)) + 1) + ... %1
 index
                (eps/(M - Snext + 1))*sum(Q(Snext+1,(1:(M-Snext+1))));
        %update Q
        Q(s+1,u+1) = Q(s+1,u+1) + alpha*(profit + gamma*Vnext - Q(s
+1,u+1));
        %update policy
        [val, mubase(s+1)] = max(Q(s+1,(1:M-s+1)));
        mubase(s+1) = mubase(s+1) - 1; %fix 1 index
        %Store the profit
        es_profits(k) = profit;
        es_sampleavg(k) = 1/k*sum(es_profits(1:k)); %%%%%1/k or 1/(k
+1)?
        es_tot(k) = sum(es_profits(1:k));
        s = Snext;
    end
    bestpolicy = mubase;
end
```

4) **[3 point]** At this point, with the Q-learning and expected Sarsa algorithms set up, we are ready to do some simulations. You are going to do 100 independent runs, and average out the results. In each run:

a) Generate the sequence of demands DD $= [D_0, D_1, \ldots, D_{K-1}]$ based on the uniform distribution.

b) Simulate the following policies, to generate a sequence of states, actions, and profits for each, starting from an empty inventory ($S_k = 0$): Q-learning algorithm, expected Sarsa algorithm, optimal policy (from HW3), lazy inventory management policy (from HW1). Note: you need to use the same sequence of demands (generated in the previous step) to simulate these policies; also, remember to initialize the state-value functions and base policies.

c) For each of the 4 policies, compute the following metrics:

$$\text{SAMPLEAVG}_k = \frac{1}{k}\sum_{t=0}^{k-1} p_t : \text{ sample average profit up to time } k, \text{ for } k = 1, 2, \ldots, K$$

$$\text{TOT}_k = \sum_{t=0}^{k-1} p_t : \text{ total profit up to time } k, \text{ for } k = 1, 2, \ldots, K$$

(note that SAMPLEAVG$_k$ can be simply computed as TOT$_k/k$)

d) In addition, for the Q-learning and expected Sarsa algorithms, return the base policy υ at the end of the last stage, let's call them υ_{QL} and υ_{Sarsa}.

With the above steps, you have found the sequence of total profits TOT$_{1:K}$, sample average profits SAMPLEAVG$_{1:K}$ for each of the 4 policies, and the base policies υ_{QL} and υ_{Sarsa}. This constitutes a single run. Now, repeat this process for 100 independent runs (this is helpful to average out the randomness inherent in Monte Carlo simulation), and compute the following:

a) average of the TOT$_{1:K}$ over the 100 independent runs, let's call it AVGTOT$_{1:K}$, for each policy

b) average of the SAMPLEAVG$_{1:K}$ over the 100 independent runs, let's call it AVGSAMPLEAVG$_{1:K}$, for each policy

c) average of the base policies υ_{QL} and υ_{Sarsa} over the 100 independent runs, let's call them $\bar{\upsilon}_{QL}$ and $\bar{\upsilon}_{Sarsa}$

Finally, it is time to make the following plots:

- Regrets of Q-learning, expected Sarsa and lazy policies vs time, that is AVGTOT$_{1:K}-$AVGTOT$^*_{1:K}$ vs $(1:K)$, where AVGTOT$^*_{1:K}$ is the sequence of total profits of the optimal policy.
- Avg profits of the 4 policies vs time, that is AVGSAMPLEAVG$_{1:K}$ vs $(1:K)$ for each of the 4 policies; for this figure, I suggest to use a log scale for time (using semilogx Matlab function)
- Plot the optimal policy, lazy policy, base policies $\bar{\upsilon}_{QL}$ and $\bar{\upsilon}_{Sarsa}$ as a function of the states.

Finally, comment on the following. What do you observe and why? How does expected Sarsa compare to Q-learning, and how do you interpret these results? How do the base policies compare to the optimal and lazy policies?

Discussion

In all graphs, we can see the optimal policy is the gold standard for this problem set as it performs the best and gives a good comparison for regret. Looking at the plot for regret we can see that over time all methods have a linearly decreasing trend. Lazy decreases the most over time, and Qlearning and sarsa decrease around the same rate. However, sarsa has a slightly higher value. Ideally, we would want to see our RL method stay close to zero and not increase much at all. This could potentially be achieved by messing around with the problems hyperparameters but overall Sarsa and Qlearning are not perfect.

The average profit also allows us to look at the performance. Here the optimal policy settles around 6, and reaches it quickly. The lazy policy spikes to 2 and settles without moving. The two RL algorithms both start to increase quickly but hit a dip. This is due to them exploring and losing some of the overall rewards early on. We can see this is adjusted and the trend begins to increase again but settles around 3.5 or 4. Given more time, these algorithms could potentially converge on 6, but with the limited step size, it settles short. Again Sarsa has a slight advantage.

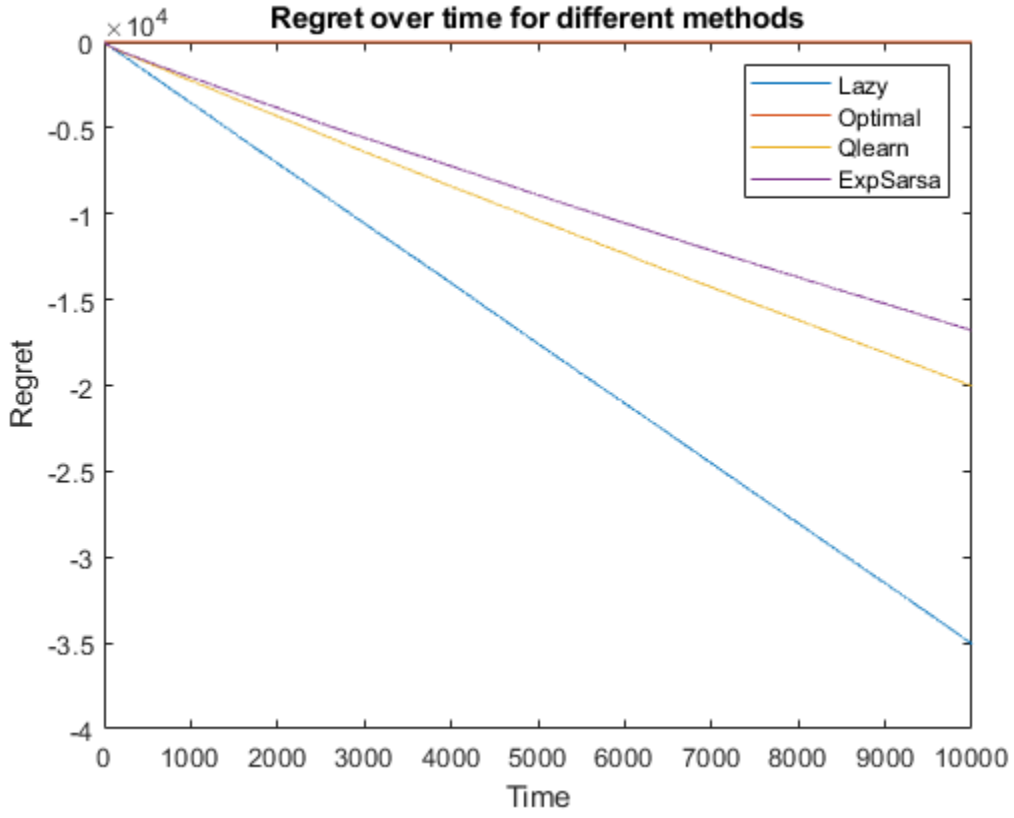
Looking at the policy plot, we see that the optimal policy almost sets a boundary for what the other policies should follow. So policies that closely follow the optimal one will perform better overall. This is reflected with Sarsa as it just barely follows the optimal one more closely than Q learning. That is the same trend seen in the other graphs.

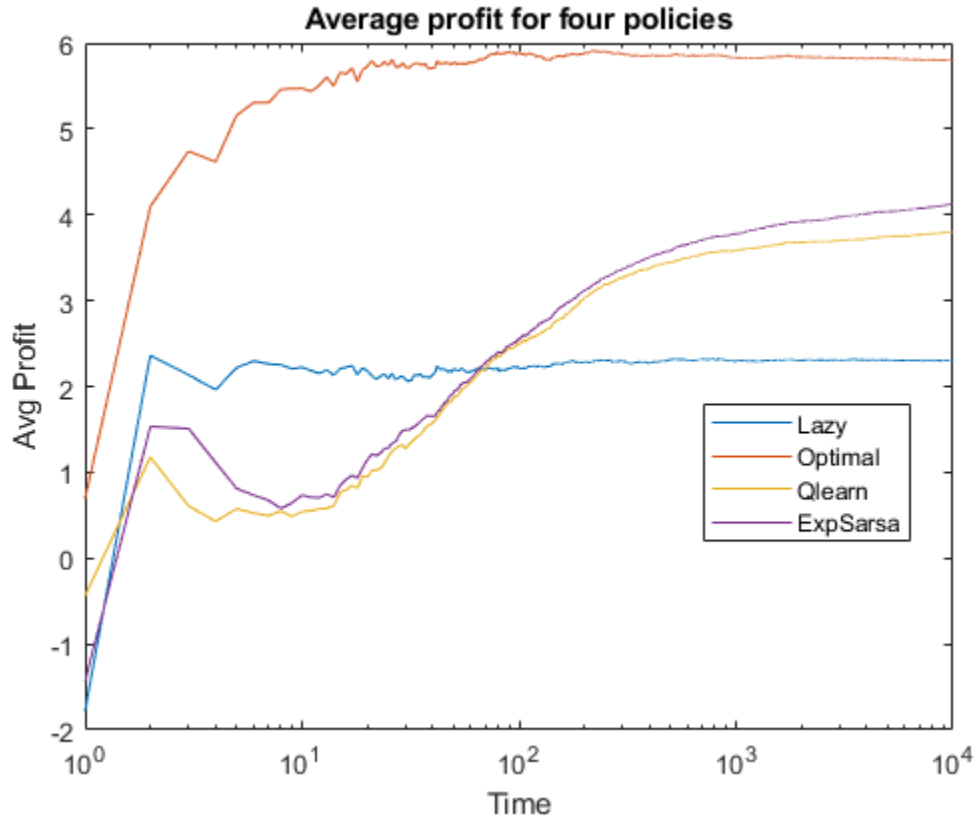
Overall, with some adjustments to the hyperparameters and step sizes, we could potentially get better results from our Q and Sarsa algorithms. It would depend on the explore vs exploit, but more runs do mean more computation time. Using K = 10^4 and 100 realizations took around a minute to compute everything, which is still considered fast. But, increasing M, and in turn states and actions, would greatly increase the computation time of this problem.

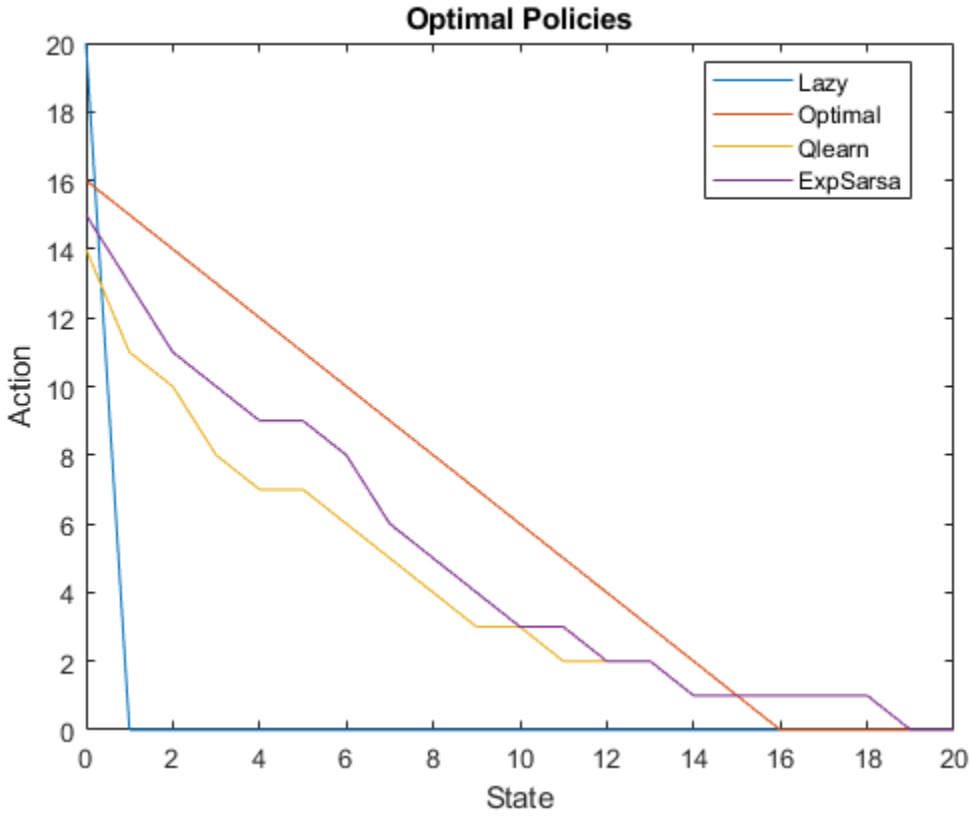
```
%Set constants
episodes = 100;
M = 20;
c = 1; r = 2; p = 1; m = .5;
%Set parameters
gamma = .9;
alpha = .1;
eps = .4;
K = 10^4;
%From HW3, optimal policy
muoptimal = [16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0  0  0
 0  0];
%Lazy policy
mulazy = zeros(1,M+1); mulazy(1) = M;


lazy_samples = zeros(1,K);
lazy_tots = zeros(1,K);
opt_samples = zeros(1,K);
opt_tots = zeros(1,K);
q_samples = zeros(1,K);
q_tots = zeros(1,K);
es_samples = zeros(1,K);
es_tots = zeros(1,K);
baseQLs = zeros(1,M+1);
baseSarsas = zeros(1,M+1);

for episode = 1:episodes
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % %Generate demand sequence
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    d = zeros(1,K+1);  %initialize vector for demand sequence
    for i = 1:K
        d(i) = randi([0 M]); %random
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%Get lazy policy states and profits
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    S = 0;
    for k = 1:K
        U = mulazy(S+1);
        [Snext, profit] = nextstate_profit(S,U,d(k),c,r,p,m);
        lazy_profits(k) = profit;
        lazy_sampleavg(k) = 1/k*sum(lazy_profits(1:k)); %%%%%1/k or 1/
(k+1)?
        lazy_tot(k) = sum(lazy_profits(1:k));
        S = Snext;
    end
    lazy_samples = lazy_samples + lazy_sampleavg;
    lazy_tots = lazy_tots + lazy_tot;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %%Get Opt policy states and profits
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    S = 0;
    for k = 1:K
        U = muoptimal(S+1);
        [Snext, profit] = nextstate_profit(S,U,d(k),c,r,p,m);
        opt_profits(k) = profit;
        opt_sampleavg(k) = 1/k*sum(opt_profits(1:k)); %%%%%1/k or 1/(k
+1)?
        opt_tot(k) = sum(opt_profits(1:k));
        S = Snext;
    end
    opt_samples = opt_samples + opt_sampleavg;
    opt_tots = opt_tots + opt_tot;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Get Sarsa and Q learn
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [baseQL, q_sampleavg, q_tot] = q_learn(d,gamma,alpha,eps,K);
    [baseSarsa, es_sampleavg, es_tot] =
 exp_sarsa(d,gamma,alpha,eps,K);

    q_tots = q_tots + q_tot;
    q_samples = q_sampleavg + q_samples;
    baseQLs = baseQLs + baseQL;

    es_tots = es_tots + es_tot;
    es_samples = es_sampleavg + es_samples;
    baseSarsas = baseSarsas + baseSarsa;

end

avgtot_lazy = (1/episodes).*lazy_tots;
avgtot_opt = (1/episodes).*opt_tots;
avgtot_q = (1/episodes).*q_tots;
avgtot_es = (1/episodes).*es_tots;

avgsamavg_lazy = (1/episodes).*lazy_samples;
avgsamavg_opt= (1/episodes).*opt_samples;
avgsamavg_q= (1/episodes).*q_samples;
avgsamavg_es= (1/episodes).*es_samples;

baseQLstar = round((1/episodes).*baseQLs);
baseSarsastar = round((1/episodes).*baseSarsas);

%Plots

%Regrets
figure(1)
time = [1:K];
plot(time,avgtot_lazy - avgtot_opt,time,avgtot_opt - avgtot_opt,...
     time,avgtot_q - avgtot_opt,time,avgtot_es - avgtot_opt);
xlabel('Time')
ylabel('Regret')
title('Regret over time for different methods');
```

```
legend('Lazy','Optimal','Qlearn','ExpSarsa');

figure(2)
%Avg profits
semilogx(time,avgsamavg_lazy,time,avgsamavg_opt,time,avgsamavg_q,time,avgsamavg_es
title('Average profit for four policies');
xlabel('Time')
ylabel('Avg Profit');
legend('Lazy','Optimal','Qlearn','ExpSarsa','Location','best');

figure(3)
%Policies
states = [0:M];
plot(states,mulazy,states,muoptimal,states,baseQLstar,states,baseSarsastar);
title('Optimal Policies');
xlabel('State')
ylabel('Action');
legend('Lazy','Optimal','Qlearn','ExpSarsa','Location','best');
```




Average profit for four policies
Avg Profit
Time
Lazy
Optimal
Qlearn
ExpSarsa


Optimal Policies
Action
State
Lazy
Optimal
Qlearn
ExpSarsa

Published with MATLAB® R2020b