ECE586RL: MDPs and Reinforcement Learning **Spring 2020**

Solutions for Homework 1

1.

(a) P1 is irreducible and aperiodic. The chain has a unique stationary distribution that it always converges to.

P2 is not irreducible. P2 is aperiodic. The chain does not have a unique stationary distribution. Hence it is also impossible for the chain to always converge to a unique distribution.

P3 is irreducible but not aperiodic. The chain has a unique stationary distribution. But it does not always converge to this stationary distribution.

(b) Since Ax_k is known, then x_{k+1} will have a Gaussian distribution $\mathcal{N}(Ax_k, W)$. Next, we can compute the mean and covariance of x_{k+1} as

$$\begin{aligned}\mathbb{E}x_{k+1} &= A\mathbb{E}x_k \\ \mathbb{E}x_{k+1}x_{k+1}^{\mathsf{T}} &= A(\mathbb{E}x_k x_k^{\mathsf{T}})A^{\mathsf{T}} + W\end{aligned}$$

2

(a) Denote the policy in the problem statement as μ. We can solve the Bellman equation. The state value function J_μ is just a vector. We can solve J_μ as $J_\mu = (I - \gamma P_\mu)^{-1}\bar{c}_\mu$. Here $\bar{c}_\mu(i) = 0.6c(i, a_1) + 0.4c(i, a_2)$. The (i, j)-th entry of P_μ is defined as $0.6P(j, i, a_1) + 0.4P(j, i, a_2)$ where $P(j, i, a) := P(s_{t+1} = j | s_t = i, a_t = a)$.

We can also solve the Bellman equation for the Q-factor. Specifically, we have

$$Q(i, a) = c(i, a) + \gamma \sum_{j=1}^{n} P(j, i, a)[0.6Q(j, a_1) + 0.4Q(j, a_2)]$$

which is equivalent to another linear equation $Q_\mu = \hat{c}_\mu + \gamma M_\mu Q_\mu$ where the i-th row of M_μ is $\begin{bmatrix}0.6P(1, i, a_1) & 0.4P(1, i, a_2) & 0.6P(2, i, a_1) & 0.4P(2, i, a_2) \dots 0.6P(n, i, a_1) & 0.4P(n, i, a_2)\end{bmatrix}$, and $(Q_\mu, \hat{c}_\mu)$ can be calculated as

$$Q_\mu = \begin{bmatrix} Q(1, a_1) \\ Q(1, a_2) \\ Q(2, a_1) \\ Q(2, a_2) \\ \vdots \\ Q(n, a_1) \\ Q(n, a_2) \end{bmatrix} \quad , \quad \hat{c}_\mu = \begin{bmatrix} c(1, a_1) \\ c(1, a_2) \\ c(2, a_1) \\ c(2, a_2) \\ \vdots \\ c(n, a_1) \\ c(n, a_2) \end{bmatrix}$$

Then Q_μ can be calculated as $Q_\mu = (I - \gamma M_\mu)^{-1}\hat{c}_\mu$. When the transition model is unknown, one can apply Monte Carlo simulation or temporal difference learning to learn value functions directly.

(b) At every t, the action a_t is generated using the policy μ given in Problem 2(a). Next apply a_t and measure s_{t+1} and $c(s_t, a_t)$. Then update the Q-factor as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left(c(s_t, a_t) + \gamma \max[Q_t(s_{t+1}, a_1), Q_t(s_{t+1}, a_2)] - Q_t(s_t, a_t)\right)$$

The size of the Q-table is $2n$. If $s_t = i$ and $a_t = a_j$, then we only update the $(2i + j - 2)$-th entry of the Q-table at step t.

(c) For SARSA, we need to specify an initial action a_0 (which can be generated arbitrarily). At every step t, apply the action a_t, and measure s_{t+1} and $c(s_t, a_t)$. Use Q_t to generate an ε-greedy policy and then use this policy to sample an action a_{t+1}. Then update the Q-factor as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left(c(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\right)$$

So at step $t \geq 1$, the action a_t is already generated using the ε-greedy policy based on Q_{t-1}.

We can see that Q-learning is off-policy in the sense that the choice of behavior policy can be independent of Q_t. In contrast, SARSA is on policy since the behavior policy is directly related to Q_t. Another difference is that in the update rules, Q-learning requires calculating $\max_{a'} Q_t(s_{t+1}, a')$ (which is equal to $\max[Q_t(s_{t+1}, a_1), Q_t(s_{t+1}, a_2)]$ in this problem) and SARSA directly applies $Q_t(s_{t+1}, a_{t+1})$.

(d) Check Pages 3-5 of the pdf file at the following link:
https://uofi.app.box.com/s/sniit2g18p41rgdbmb2ccvkouy7z9hkv

3

(a) Given a linear policy K, it is straightforward to use induction to show

$$V(x) = r_K + \quad x^\mathsf{T} \left(\sum_{t=0}^{\infty} \gamma^t ((A - BK)^\mathsf{T})^t (Q + K^\mathsf{T} R K)(A - BK)^t \right) x \tag{1}$$

where r_K is some extra term introduced by the noise w_t. Therefore, we can parameterize the value function as $x^\mathsf{T} P_K x + r_K$. Therefore, we have

$$V(x) = x^\mathsf{T}(Q + K^\mathsf{T} R K)x + \gamma \left(\mathbb{E}((A - BK)x + w)^\mathsf{T} P_K ((A - BK)x + w) + r_K \right) \tag{2}$$

Notice w is independent from x and has a zero mean, we have

$$\mathbb{E}((A - BK)x + w)^\mathsf{T} P_K ((A - BK)x + w) = x^\mathsf{T}(A - BK)^\mathsf{T} P_K (A - BK)x + \mathbb{E}(w^\mathsf{T} P_K w)$$

Notice that the left side of (2) is just $x^\mathsf{T} P_K x + r_K$. Hence (2) can be rewritten as

$$x^\mathsf{T} P_K x + r_K = x^\mathsf{T}(Q + K^\mathsf{T} R K)x + \gamma x^\mathsf{T}(A - BK)^\mathsf{T} P_K (A - BK)x + \gamma \mathbb{E}(w^\mathsf{T} P_K w) + \gamma r_K$$

To ensure that the quadratic functions on the left and right sides of the above equation are the same, the following have to be true:

$$\begin{aligned} x^\mathsf{T} P_K x &= x^\mathsf{T}(Q + K^\mathsf{T} R K)x + \gamma x^\mathsf{T}(A - BK)^\mathsf{T} P_K (A - BK)x \\ r_K &= \gamma \mathbb{E}(w^\mathsf{T} P_K w) + \gamma r_K \end{aligned}$$

Hence, the Bellman equation becomes

$$P_K = Q + K^\mathsf{T} R K + \gamma (A - BK)^\mathsf{T} P_K (A - BK)$$

and $r_K = \frac{\gamma}{1-\gamma}\mathbb{E}(w^\mathsf{T} P_K w) = \frac{\gamma}{1-\gamma}\operatorname{trace}(PW)$ where W is the covariance matrix of w_t.

For the $\mathcal{Q}$-function, we have

$$\begin{aligned} \mathcal{Q}(x,u) &= x^\top Q x + u^\top R u + \gamma \mathbb{E} V(Ax + Bu + w) \\ &= x^\top Q x + u^\top R u + \gamma \mathbb{E}(Ax + Bu + w)^\mathsf{T} P_K (Ax + Bu + w) + \gamma r_K \\ &= x^\top Q x + u^\top R u + \gamma (Ax + Bu)^\mathsf{T} P_K (Ax + Bu) + \gamma(\mathbb{E}(w^\mathsf{T} P_K w) + r_K) \\ &= \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q + \gamma A^\mathsf{T} P_K A & \gamma A^\mathsf{T} P_K B \\ \gamma B^\mathsf{T} P_K A & R + \gamma B^\mathsf{T} P_K B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + r_K \end{aligned}$$

We can also directly parameterize $\mathcal{Q}(x,u)$ as

$$\mathcal{Q}(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + r_K$$

Notice $V(x) = \mathcal{Q}(x, -Kx)$. Therefore, we can substitute this into the above equation to obtain the Bellman equation for $\mathcal{Q}$:

$$\mathcal{Q}(x,u) = x^\top Q x + u^\top R u + \gamma \mathbb{E}\mathcal{Q}\left(Ax + Bu + w, -K(Ax + Bu + w)\right)$$

which is equivalent to

$$\begin{aligned} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} &= \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} + \gamma \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix} \\ r_K &= \gamma \mathbb{E} \begin{bmatrix} w \\ -Kw \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} w \\ -Kw \end{bmatrix} + \gamma r_K \end{aligned}$$

(b) Optimal Bellman equation: Suppose the optimal state value function is $x^\mathsf{T} P x + r$. We have

$$\begin{aligned} x^\mathsf{T} P x + r &= \min_u (x^\mathsf{T} Q x + u^\mathsf{T} R u + \gamma \mathbb{E}(Ax + Bu + w)^\mathsf{T} P (Ax + Bu + w) + \gamma r) \\ &= \min_u (x^\mathsf{T} Q x + u^\mathsf{T} R u + \gamma (Ax + Bu)^\mathsf{T} P (Ax + Bu) + \gamma \mathbb{E} w^\mathsf{T} P w + \gamma r) \end{aligned}$$

Taking gradient of the function on the right side with respect to u leads to

$$u = -\gamma(R + \gamma B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A x.$$

which can be substituted back to get the following optimal Bellman equation:

$$P = Q + \gamma A^\mathsf{T} P A - \gamma^2 A^\mathsf{T} P B (R + \gamma B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A$$

Then the optimal state-action value function can be calculated as

$$\mathcal{Q}^*(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q + \gamma A^\mathsf{T} P A & \gamma A^\mathsf{T} P B \\ \gamma B^\mathsf{T} P A & R + \gamma B^\mathsf{T} P B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \frac{\gamma}{1-\gamma} \operatorname{trace}(PW)$$

(c) Policy iteration: The PI algorithm iterates as $K^{n+1} = \gamma(\gamma B^\mathsf{T} P^n B + R)^{-1} B^\mathsf{T} P^n A$ where P_n solves the Bellman equation $\gamma(A - BK^n)^\mathsf{T} P^n (A - BK^n) + Q + (K^n)^\mathsf{T} R K^n = P^n$. Another option is to evaluate $\mathcal{Q}$ for every step and then design a policy which is the greedy policy for $\mathcal{Q}$. Specifically, at every step n, we first solve the $\mathcal{Q}$ Bellman equation to obtain $(\mathcal{Q}^n_{11}, \mathcal{Q}^n_{12}, \mathcal{Q}^n_{22})$ (the policy evaluation step), and then update the policy as $K_{n+1} = (\mathcal{Q}^n_{22})^{-1}(\mathcal{Q}^n_{12})^\mathsf{T}$ (the policy improvement step).

To estimate $\mathcal{Q}$-Factor from data, one can either use Monte Carlo simulation or LSTDQ (see the LQR note for the details).

(d) For SARSA, the initial action u_0 can be arbitrary. At every step n, apply the control action u_n and measure x_{n+1} and $c(x_n, u_n) = x_n^\mathsf{T} Q x_n + u_n^\mathsf{T} R u_n$. Choose u_{n+1} using the ε-greedy policy generated by $\mathcal{Q}_n(x,u) = \theta_n^\mathsf{T} \phi(x,u)$ where ϕ is the feature. Then update the weight vector as $\theta_{n+1} = \theta_n + \alpha_n \phi(x_n, u_n)(c(x_n, u_n) + \gamma \theta_n^\mathsf{T} \phi(x_{n+1}, u_{n+1}) - \theta_n^\mathsf{T} \phi(x_n, u_n))$. For every $n \geq 1$, u_n was sampled using the ε-greedy policy generated by $\theta_{n-1}^\mathsf{T} \phi(x,u)$. For Q-learning, we can choose any behavior policy that provides sufficient exploration. At every step n, sample u_n using the behavior policy and measure x_{n+1} and $c(x_n, u_n)$. Then update the weight vector as $\theta_{n+1} = \theta_n + \alpha_k \phi(x_n, u_n)(c(x_n, u_n) + \gamma \min_u \theta_n^\mathsf{T} \phi(x_{n+1}, u) - \theta_n^\mathsf{T} \phi(x_n, u_n))$. Again, $\mathcal{Q}$-learning is off-policy and the sampling can be done using any behavior policy providing sufficient exploration. SARSA is on-policy and the action sampling is done using the ε-greedy policy given by $\theta_n^\mathsf{T} \phi(x,u)$. Finally, it is worth mentioning that a naive implementation of Q-learning may fail for many continuous control problems due to stability issues.

4

(a) Popular options for Ψ_t:

- Monte Carlo estimation: $\sum_{t'=t}^{\infty} \gamma^{t'-t} c_{t'}$
- Baselined versions of Monte Carlo estimation: $\sum_{t'=t}^{\infty} (\gamma^{t'-t} c_{t'} - b(x_t))$
- State-action value function: $Q^\pi(x_t, u_t)$

- Advantage function: $A^\pi(x_t, u_t)$
- TD residual: $c_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t)$
- Generalized advantage estimation

To calculate the gradient, first notice K is a matrix. Hence $\nabla_\theta \log \pi_\theta$ is also a matrix. The (i,j)-th entry of this matrix is just

$$\frac{\partial \log \pi_\theta}{\partial K_{ij}} = -\sigma^{-1}(u_t^{(i)} + \sum_{p=1}^{n_x} K_{ip} x_t^{(p)}) x_t^{(j)}$$

where the superscript (i) denotes the i-th entry of the vector. More compactly, we can write $\nabla_\theta \log \pi_\theta(u_t|x_t) = -(\sigma I)^{-1}(u_t + Kx_t)x_t^\mathsf{T}$.

Now consider the case $u_t \sim \mathcal{N}(W^1 h(W^0 x_t), \sigma I)$. The derivative with respect to W^1 can be directly calculated as

$$\frac{\partial}{\partial W^1} \log \pi_\theta(u_t|x_t) = \sigma^{-1}(u_t - W^1 h(W^0 x_t))(h(W^0 x_t))^\mathsf{T}$$

The derivative with respect to W^0 requires a backpropagation step and can be calculated as

$$\frac{\partial}{\partial W^0} \log \pi_\theta(u_t|x_t) = \sigma^{-1}(W^1 \operatorname{diag}(h'(W^0 x_t)))^\mathsf{T}(u_t - W^1 h(W^0 x_t))x_t^\mathsf{T}$$

where $\operatorname{diag}(h'(W^0 x_t))$ is a diagonal matrix whose (i,i)-th entry is equal to the i-th entry of the vector $h'(W^0 x_t)$. See Section 3.1 of the following survey paper for a detailed treatment of backpropogation:

https://arxiv.org/pdf/1912.08957.pdf

(b)

$$\begin{aligned}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left(\lim_{\sigma\to 0}\frac{\mathcal{C}(K+\sigma\varepsilon)-\mathcal{C}(K)}{\sigma}\right)\varepsilon &= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}(\varepsilon^\mathsf{T}\nabla\mathcal{C}(K))\varepsilon \\ &= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\varepsilon(\varepsilon^\mathsf{T}\nabla C(K)) \\ &= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}(\varepsilon\varepsilon^\mathsf{T})\nabla C(K) \\ &= \nabla C(K)\end{aligned}$$

5

A code provided on the course website. LSTD-Q works efficiently for (b), and Approximate PI works efficiently for (c). Based on the simulation, the approximate PI method can converge to the optimal control gain within 10 iterations. See the code for how to setup behavior policy for efficient exploration.