Sudarshan Nambiar

ECE 49595-RL : HW 3

Problem 1:

1) $$P^{\pi}(s' | s) = \sum_{a \in A} \pi(a|s) \cdot P(s' | s, a)$$

$\pi(1) = g, \quad \pi(2) = g, \quad \pi(3) = h$

$$\left.\begin{aligned} \therefore P^{\pi}(1|1) &= (1 \times 0.1) + (0 \times 0.8) = 0.1 \\ P^{\pi}(2|1) &= (1 \times 0.8) + (0 \times 0.1) = 0.8 \\ P^{\pi}(3|1) &= (1 \times 0.1) + (0 \times 0.1) = 0 \end{aligned}\right\} \text{row 1}$$

$$\left.\begin{aligned} P^{\pi}(1|2) &= (1 \times 0.1) + (0 \times 0.1) = 0.1 \\ P^{\pi}(2|2) &= (1 \times 0.1) + (0 \times 0.8) = 0.1 \\ P^{\pi}(3|2) &= (1 \times 0.8) + (0 \times 0.1) = 0.8 \end{aligned}\right\} \text{row 2}$$

$$\left.\begin{aligned} P^{\pi}(1|3) &= (0 \times 0.8) + (1 \times 0.1) = 0.1 \\ P^{\pi}(2|3) &= (0 \times 0.1) + (1 \times 0.1) = 0.1 \\ P^{\pi}(3|3) &= (0 \times 0.1) + (1 \times 0.8) = 0.8 \end{aligned}\right\} \text{row 3}$$

$$\therefore P^{\pi} = \begin{bmatrix} 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

2) $\mu_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$\therefore \; \rho^\pi \mu_0 = (I - \gamma P^\pi)^{-1} \mu_0$

```python
import numpy as np

# Define the discount factor, initial state distribution, and probability transition matrix
gamma = 0.95
mu_0 = np.array([[1], [0], [0]])
P_pi = np.array([[0.1, 0.8, 0.1],
                 [0.1, 0.1, 0.8],
                 [0.1, 0.1, 0.8]])

# Compute the transpose of P_pi
P_pi_t = P_pi.T

# Compute (I - gamma * P_pi^T)
I = np.eye(3)
I_minus_gamma_P_pi_t = I - gamma * P_pi_t

# Compute (I - gamma * P_pi^T)^-1
I_minus_gamma_P_pi_t_inv = np.linalg.inv(I_minus_gamma_P_pi_t)

# Compute rho_pi_mu_0 = (I - gamma * P_pi^T)^-1 * mu_0
rho_pi_mu_0 = np.dot(I_minus_gamma_P_pi_t_inv, mu_0)

print("State occupancy measure:")
for i, rho in enumerate(rho_pi_mu_0, 1):
    print(f"rho pi mu_0 ({i}) = {rho[0]:.4f}")
```

$\therefore \; \rho^\pi \mu_0 = \begin{bmatrix} 2.9 \\ 3.829 \\ 13.272 \end{bmatrix}$

3) $$\bar{P}^{\pi} N_0 = \frac{P^{\pi} N_0}{\sum P^{\pi} N_0} = \underline{\begin{bmatrix} 0.1450 \\ 0.1914 \\ 0.6636 \end{bmatrix}}$$

4) To find $\| P'(\cdot \mid s, \pi(s)) - P(\cdot \mid s, \pi(s)) \|_1$

```python
import numpy as np

# original transition probabilities under policy pi
P_pi = np.array([[0.1, 0.8, 0.1], [0.1, 0.1, 0.8], [0.1, 0.1, 0.8]])

# new transition probabilities under policy pi
P_prime_pi = np.array([[0.15, 0.7, 0.15], [0.05, 0.05, 0.9], [0.15, 0.8, 0.05]])

# compute the L1-difference for each state
l1_diff = np.sum(np.abs(P_prime_pi - P_pi), axis=1)

# print the L1-difference for each state
for i, diff in enumerate(l1_diff, 1):
    print(f"s = {i}: |P'(.|{i}, pi({i})) - P(.|{i}, pi({i}))|₁ = {diff:.1f}")

```

$$\therefore \left.\begin{aligned} s &\leftarrow 1 = 0.2 \\ s &\leftarrow 2 = 0.2 \\ s &\leftarrow 3 = 1.5 \end{aligned}\right\} L_s$$

5) For $s = 1$, find $|V'^{\pi}(1) - V^{\pi}(1)|$

We can use the previously calculated $\rho^{\pi} \nu_0$ & $\bar{\rho}^{\pi} \nu_0$

$$\therefore |V'^{\pi}(1) - V^{\pi}(1)| = \frac{1}{1-\gamma} \sum \bar{\rho}^{\pi} \nu_0 \cdot L_s$$

L_1 diff

```python
import numpy as np

# original transition probabilities under policy pi
P_pi = np.array([[0.1, 0.8, 0.1], [0.1, 0.1, 0.8], [0.1, 0.1, 0.8]])

# new transition probabilities under policy pi
P_prime_pi = np.array([[0.15, 0.7, 0.15], [0.05, 0.05, 0.9], [0.15, 0.8, 0.05]])

# compute the L1-difference for each state
l1_diff = np.sum(np.abs(P_prime_pi - P_pi), axis=1)

# print the L1-difference for each state
for i, diff in enumerate(l1_diff, 1):
    print(f"s = {i}: |P’(.|{i}, pi({i})) - P(.|{i}, pi({i}))|₁ = {diff:.1f}")

```

$$\therefore |V'^{\pi}(1) - V^{\pi}(1)| = \underline{\underline{21.2536}}$$

Problem 2:

1) $R(s,a) = \begin{cases} 1 & , (3,h) \\ 0 & \end{cases}$

$\therefore \pi(1) = g \quad , \pi(2) = g \quad , \pi(3) = h$

$$\therefore P = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

$R = [[0,0], [0,0], [0,1]]$

```
import numpy as np

# Transition probabilities and rewards
P = np.array(
    [
        [[0.1, 0.8, 0.1], [0.8, 0.1, 0.1]],
        [[0.1, 0.1, 0.8], [0.1, 0.8, 0.1]],
        [[0.8, 0.1, 0.1], [0.1, 0.1, 0.8]],
    ]
)
R = np.array([[0, 0], [0, 0], [0, 1]])

# Policy
pi = np.array([0, 0, 1])

gamma = 0.95

# set up system of linear equations
A = np.eye(3) - gamma * P[np.arange(3), pi]
b = R[np.arange(3), pi]

# solve system of linear equations
V_pi = np.linalg.solve(A, b)

for i, v in enumerate(V_pi, 1):
    print(f"V π({i}) ≈ {v:.4f}")
```

2) Given oracle access to $P(s'|s,a)$, we can generate 100 samples of s' using each s,a pair, count the number of times each s' appears, & divide by 100 to obtain $\hat{P}(s'|s,a)$

* Assuming oracle access in code → using given transition probabilities to sample

```python
import numpy as np

P_true = np.array([
    [[0.1, 0.8, 0.1], [0.8, 0.1, 0.1]],
    [[0.1, 0.1, 0.8], [0.1, 0.8, 0.1]],
    [[0.8, 0.1, 0.1], [0.1, 0.1, 0.8]]
])

num_samples = 100
states = [1, 2, 3]
actions = ['g', 'h']

# initialize the estimated transition function
P_hat = np.zeros((3, 2, 3))

# sample each state-action pair and estimate the transition probabilities
for s in states:
    for a_idx, a in enumerate(actions):
        # generate samples using the true transition probabilities
        samples = np.random.choice([1, 2, 3], size=num_samples, p=P_true[s - 1, a_idx])

        # count the occurrences of each next state in the samples
        for s_prime in states:
            count = np.sum(samples == s_prime)
            P_hat[s - 1, a_idx, s_prime - 1] = count / num_samples

print("Estimated transition function P^:")
for s in states:
    for a_idx, a in enumerate(actions):
        print(f"P^(.|{s}, {a}) = {P_hat[s - 1, a_idx]}")
```

PROBLEMS OUTPUT TERMINAL GITLENS PORTS

```
PS C:\Users\suddu\Downloads\Code\ECE\495_RL\hw3> & C:/Users/suddu/AppData/Local/Programs/Python/Python312/python.exe c:/Users/suddu/Downloads/Code/ECE/495_RL/hw3/hw3_2.2.py
Estimated transition function P^:
P^(.|1, g) = [0.09 0.84 0.07]
P^(.|1, h) = [0.85 0.09 0.06]
P^(.|2, g) = [0.11 0.05 0.84]
P^(.|2, h) = [0.11 0.79 0.1 ]
P^(.|3, g) = [0.75 0.06 0.19]
P^(.|3, h) = [0.13 0.14 0.73]
PS C:\Users\suddu\Downloads\Code\ECE\495_RL\hw3>
```

Question 2.3

```python
import numpy as np

P_true = np.array([
    [[0.1, 0.8, 0.1], [0.8, 0.1, 0.1]],
    [[0.1, 0.1, 0.8], [0.1, 0.8, 0.1]],
    [[0.8, 0.1, 0.1], [0.1, 0.1, 0.8]]
])

num_samples = 100
states = [1, 2, 3]
actions = ['g', 'h']

# initialize the estimated transition function
P_hat = np.zeros((3, 2, 3))

# sample each s, a pair and estimate the transition probabilities
for s in states:
    for a_idx, a in enumerate(actions):
        # generate samples using the true transition probabilities
        samples = np.random.choice([1, 2, 3], size=num_samples, p=P_true[s - 1, a_idx])

        # count the occurrences of each s' in the samples
        for s_prime in states:
            count = np.sum(samples == s_prime)
            P_hat[s - 1, a_idx, s_prime - 1] = count / num_samples

print("Estimated transition function P^:")
for s in states:
    for a_idx, a in enumerate(actions):
        print(f"P^(.|{s}, {a}) = {P_hat[s - 1, a_idx]}")

pi = ['g', 'g', 'h']

# compute the L1-difference for each state under policy pi
l1_diff = np.zeros(3)
for s in range(3):
    a_idx = 0 if pi[s] == 'g' else 1
    l1_diff[s] = np.sum(np.abs(P_hat[s, a_idx] - P_true[s, a_idx]))

print("\nL1-difference under policy pi:")
for s in range(3):
    print(f"|P^(.|{s+1}, {pi[s]}) - P(.|{s+1}, {pi[s]})|_1 = {l1_diff[s]:.4f}")
```

$\therefore \| \hat{P}(\cdot \mid 1, g) - P(\cdot \mid 1, g) \|_1 = \underline{\underline{0.08}}$

$\| \hat{P}(\cdot \mid 2, g) - P(\cdot \mid 2, g) \|_1 = \underline{\underline{0.1}}$

$\| \hat{P}(\cdot \mid 3, h) - P(\cdot \mid 3, h) \|_1 = \underline{\underline{0.1}}$

Question 2.5

```python
import numpy as np

P_true = np.array([
    [[0.1, 0.8, 0.1], [0.8, 0.1, 0.1]],
    [[0.1, 0.1, 0.8], [0.1, 0.8, 0.1]],
    [[0.8, 0.1, 0.1], [0.1, 0.1, 0.8]]
])

num_samples = 100
states = [1, 2, 3]
actions = ['g', 'h']

# initialize the estimated transition function
P_hat = np.zeros((3, 2, 3))

# sample each s, a pair and estimate the transition probabilities
for s in states:
    for a_idx, a in enumerate(actions):
        # generate samples using the true transition probabilities
        samples = np.random.choice([1, 2, 3], size=num_samples, p=P_true[s - 1, a_idx])

        # count the occurrences of each s' in the samples
        for s_prime in states:
            count = np.sum(samples == s_prime)
            P_hat[s - 1, a_idx, s_prime - 1] = count / num_samples

print("Estimated transition function P^:")
for s in states:
    for a_idx, a in enumerate(actions):
        print(f"P^(.|{s}, {a}) = {P_hat[s - 1, a_idx]}")

pi = ['g', 'g', 'h']

# compute the L1-difference for each state under policy pi
l1_diff = np.zeros(3)
for s in range(3):
    a_idx = 0 if pi[s] == 'g' else 1
    l1_diff[s] = np.sum(np.abs(P_hat[s, a_idx] - P_true[s, a_idx]))

print("\nL1-difference under policy pi:")
for s in range(3):
    print(f"|P^(.|{s+1}, {pi[s]}) - P(.|{s+1}, {pi[s]})|_1 = {l1_diff[s]:.4f}")

R = np.array([[0, 0], [0, 0], [0, 1]])

gamma = 0.95

# set up the system of linear equations
A = np.eye(3)
b = np.zeros(3)

for s in range(3):
    a_idx = 0 if pi[s] == 'g' else 1
    A[s] -= gamma * P_hat[s, a_idx]
    b[s] = R[s, a_idx]

# solve system of linear equations
V_hat_pi = np.linalg.solve(A, b)

print("\nEstimated value function V^ pi:")
for s in range(3):
    print(f"V^ π({s+1}) = {V_hat_pi[s]:.4f}")
```

$\therefore \hat{V}^{\pi}(1) = 12.955$

$\hat{V}^{\pi}(2) = 13.650$

$\hat{V}^{\pi}(3) = 15.395$

6) $|\hat{V}^{\pi}(1) - V^{\pi}(1)| = \underline{12.8841}$

4) Bonus

We can use the following formula

$$|\hat{V}^{\pi}(1) - V^{\pi}(1)| \leq \frac{1}{1-\gamma} \sum \bar{\rho}^{\pi}_{\nu_0}(s) \| \hat{P}(\cdot|s,\pi(s)) - P(\cdot|s,\pi(s)) \|_1$$

```python
rho_bar_pi_mu_0 = np.array([2.9, 3.8285, 13.2715])

# compute the upper bound using the simulation lemma
upper_bound = (1 / (1 - gamma)) * np.sum(rho_bar_pi_mu_0 * l1_diff)

print(f"\nUpper bound on |V^ pi(1) - V pi(1)| using the simulation lemma:")
print(f"|V^ pi(1) - V pi(1)| <= {upper_bound:.4f}")

```

$\therefore \|\hat{V}^{\pi}(1) - V^{\pi}(1)\|_1 \leq \underline{13.3828}$

Problem 3 :

1) To show that the tabular representations of $Q(s,a)$ is a special class of linear functions, we need to define the number of features k & the vector val. function $\phi(s,a)$ s.t the lin. func. is eq. to the tabular representation.

$$\therefore k = |S| \times |A|$$
$$\phi(s,a) : S \times A \rightarrow \mathbb{R}^k$$

$$\therefore Q = \{ Q_\theta : Q_\theta(s,a) = \theta^T \phi(s,a) \text{ for all } s \in S \text{ \& } a \in A, \theta \in \mathbb{R}^k \}$$

For any (s,a), $\phi(s,a)$ is a vector with a 1 in the position corresponding to (s,a) & 0's elsewhere.

$$\therefore Q_\theta(s,a) = \theta^T \phi(s,a) = \underbrace{\theta^*(s,a)}$$

* elements of parameter vector θ corr. to (s,a)

This shows that the linear func. approx. $Q_\theta(s,a)$ with the defined feature function $\phi(s,a)$ & number of features $k = |S| \times |A|$ is eq. to the tabular representation, where (s,a) pairs have their own indep. parameter $\theta^*(s,a)$

$\therefore$ The # of features k should be equal to the total number (s,a) pairs

→ The feature function $\phi(s,a)$ should map each s,a pair to a binary vector with only one 1 of size k, positioned corresponding to (s,a).

With these definitions, the resulting class of linear function Q parametrized by θ is equivalent to the tabular representations of the s,a value function.

2) $k = n \times m$ where n : # of state partitions
m : # of action partitions

$\phi(s,a) = e\{j,i\}$ is a one-hot vector of size k with a 1 in the position corr. to the state partition index j & action partition index i, & 0's elsewhere.
$\therefore$ $s \in S_j$ & $a \in A_i$, $\phi(s,a) = e\{j,i\}$

$\therefore$ $Q_\theta(s,a) = \theta^T \phi(s,a)$ is the lin. func. approx.

$$= \theta^T e\{j,i\} = \underbrace{\theta\{j,i\}}_{\substack{\text{element of the parameter} \\ \text{vector } \theta \text{ corr. to the} \\ \text{state partition index } j \\ \text{\& action index } i}}$$

This func. shows that the linear func. approx. $Q_\theta(s,a)$ with the defined feature function $\phi(s,a)$ & # of features $k = n \times m$ assigns the same val. $\theta\{j,i\}$ to all s,a pairs.

This representation is more efficient than the tabular one because it reduces the parameter size from $|S| \times |A|$ to $\underbrace{n \times m}_{\text{\# of state \& action partitions}}$

By grouping similar states & actions into partitions we can capture the shared structure in Q func. & reduce its parameter size. //

3) Bonus

Since we have a linear func. approx. with $\phi(s,a)$ parametrized by θ :

$$Q^{\pi}(s,a) = \operatorname{argmin}_{Q \in \mathcal{Q}} \sum_{i=1}^{N} \left(y_i - Q(s_i,a_i)\right)^2$$

where Q represents the class of linear functions based on the single feature $\phi(s,a)$

$Q(s,a) = \theta\phi(s,a) \rightarrow$ lin. func. approx.

$\therefore Q^{\pi}(s,a) = \arg\min_{\theta} \sum_{i=1}^{N} (y_i^2 - 2y_i\theta\phi(s_i, a_i) + \theta^2\phi(s_i, a_i)^2$

since $\phi(s_i, a_i)$ is known & indep. of θ, we can treat it as constant

$\therefore \frac{\partial\phi}{\partial\theta} = 0$

$\therefore \sum_{i=1}^{N} (-2y_i\phi(s_i, a_i) + 2\theta\phi(s_i, a_i)^2) = 0$

$\rightarrow -2\sum_{i=1}^{N} y_i\phi(s_i, a_i) + 2\theta\sum_{i=1}^{N}\phi(s_i, a_i)^2 = 0$

$\rightarrow \theta\sum_{i=1}^{N}\phi(s_i, a_i)^2 = \sum_{i=1}^{N} y_i\phi(s_i, a_i)$

$$\therefore \theta = \frac{\sum_{i=1}^{N} y_i\phi(s_i, a_i)}{\sum_{i=1}^{N}\phi(s_i, a_i)^2} \quad //$$