ECE59500RL Homework 4

Robert (Cars) Chandler — chandl71@purdue.edu

Problem 1

1.1

We can evaluate the value functions by obtaining the expected value of the transition function and reward function over each of the policies according to π^t, π^b, and then we can use the analytical solution

$$\mathbf{V}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{R}$$

```
import itertools

import numpy as np
import xarray as xr
from IPython.display import Markdown

state_space = [1, 2, 3]
action_space = list("gh")

# Define a multidimensional array containing probabilities of transitions from
# one state-action pair to another. This is just like a regular numpy array but
# with labeled "coordinates" and axes which make it easy for us to index it for
# a given state and/or action.
trans_fn = xr.DataArray(
    data=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]],
        ]
    ),
```

```
    coords=dict(
        current_state=state_space, action=action_space, next_state=state_space
    ),
)

# For any current state-action pair, the sum of transition probabilities to all
# next states should sum to 1
assert np.allclose(1, trans_fn.sum(dim="next_state"))

reward_fn = xr.DataArray(
    np.zeros([3, 2]),
    coords=dict(state=state_space, action=action_space),
)
reward_fn.loc[dict(state=3, action="h")] = 1

gamma = 0.95

policy = dict(
    target=xr.DataArray(
        data=np.array([[0.9, 0.1], [0.9, 0.1], [0.1, 0.9]]),
        coords=dict(state=state_space, action=action_space),
    ),
    behavior=xr.DataArray(
        data=np.array([[0.85, 0.15], [0.88, 0.12], [0.1, 0.9]]),
        coords=dict(state=state_space, action=action_space),
    ),
)

for pol in policy.values():
    assert np.allclose(1, pol.sum(dim="action"))

expected_trans_fn = {
    key: np.reshape(
        [
            np.dot(
                trans_fn.sel(current_state=s, next_state=sn),
                pol.sel(state=s),
            )
            for s, sn in itertools.product(state_space, state_space)
        ],
        (3, 3),
    )
```

```
    for key, pol in policy.items()
}

expected_reward_fn = {
    key: (reward_fn * pol).sum(dim="action") for key, pol in policy.items()
}

value_fn = {
    key: np.linalg.inv(
        np.identity(len(state_space)) - gamma * expected_trans_fn[key]
    )
    @ expected_reward_fn[key].to_numpy()
    for key in expected_trans_fn
}

value_latex = {
    key: Markdown(
        r"\\".join(
            [
                rf"V^{{\pi^{key[0]}}}({s}) = {v:.5f}"
                for s, v in zip(list("123"), v)
            ]
        )
    )
    for key, v in value_fn.items()
}
```

The value functions are:

$$
\begin{aligned}
V^{\pi^t}(1) &= 10.36761 \\
V^{\pi^t}(2) &= 10.92103 \\
V^{\pi^t}(3) &= 11.78423 \\
V^{\pi^b}(1) &= 10.24052 \\
V^{\pi^b}(2) &= 10.80855 \\
V^{\pi^b}(3) &= 11.68240
\end{aligned}
$$

1.2

Using the formula

$$T = \frac{\frac{\log(R_{max})}{\varepsilon(1-\gamma)}}{\log\left(\frac{1}{\gamma}\right)}$$

we can calculate the number of timesteps required for an effective horizon with errors less than 0.1:

```
import numpy as np
from p1_1 import gamma, reward_fn

epsilon = 0.1

t_effective = int(
    np.ceil(
        np.log(reward_fn.max() / (epsilon * (1 - gamma))) / np.log(1 / gamma)
    )
)
```

$$T = 104$$

1.3

We generate 50 trajectories using the behavior policy and the effective horizon length and then use a Monte Carlo evaluation for the target policy with importance sampling.

```
import numpy as np
import xarray as xr
from IPython.display import Markdown
from p1_1 import action_space, gamma, policy, reward_fn, state_space, trans_fn
from p1_2 import t_effective

NUM_TRAJECTORIES = 50

rng = np.random.default_rng(seed=42)


def sample_trajectory(
    trans_fn: xr.DataArray, policy: xr.DataArray, reward_fn: xr.DataArray
):
    states = []
```

```
    actions = []
    rewards = []

    state = 1

    for _ in range(t_effective):
        action = str(rng.choice(action_space, p=policy.sel(state=state)))
        reward = float(reward_fn.sel(state=state, action=action))

        states.append(state)
        actions.append(action)
        rewards.append(reward)

        state = int(
            rng.choice(
                state_space,
                p=trans_fn.sel(current_state=state, action=action),
            )
        )

    return (np.array(vals) for vals in [states, actions, rewards])


trajectories = []
for _ in range(NUM_TRAJECTORIES):
    states, actions, rewards = sample_trajectory(
        trans_fn, policy["behavior"], reward_fn
    )
    trajectories.append(dict(states=states, actions=actions, rewards=rewards))


gammas = gamma ** np.arange(t_effective)

vhat = {state: 0.0 for state in state_space}
num_visits = {state: 0.0 for state in state_space}
returns = {state: [] for state in state_space}


def return_at_t(t, traj, gammas):
    return float(np.sum(traj["rewards"][t:] * gammas[: len(gammas) - t]))
```

```
def weighted_return_at_t(t, traj, gammas):
    weight = np.prod(
        [
            (
                policy["target"].sel(state=s, action=a)
                / policy["behavior"].sel(state=s, action=a)
            )
            for s, a in zip(traj["states"][t:], traj["actions"][t:])
        ]
    )

    return weight * return_at_t(t, traj, gammas)


def mc_evaluation(vhat, returns, num_visits, return_function):
    state = 1
    for traj in trajectories:
        # For each step in the trajectory...
        for t, s in enumerate(traj["states"]):
            # We only want to evaluate the time where state is visited
            if s != state:
                continue

            ret = return_function(t, traj, gammas)
            num_visits[state] += 1

            returns[state].append(ret)

            # We only want the first visit
            break

        vhat[state] = float((1 / num_visits[state]) * np.sum(returns[state]))

    return vhat, returns, num_visits


mc_evaluation(vhat, returns, num_visits, return_function=weighted_return_at_t)


vhat_latex = Markdown(rf"V^{{\pi^t}}(1) \approx {vhat[1]:.5f} \\")
```

The resulting estimate is:

$$V^{\pi^t}(1) \approx 10.40336$$

1.4

```
from IPython.display import Markdown
from p1_1 import value_fn
from p1_3 import vhat

# NOTE: vhat is a dict; these indices aren't mismatched
value_error = vhat[1] - value_fn["target"][0]

value_error_latex = Markdown(
    rf"\hat{{V}}^{{\pi^t}}(1) - V^{{\pi^t}}(1) = {vhat[1]:.5f} -"
    rf" {value_fn['target'][0]:.5f} = {value_error:.5f}"
)
```

The error between the estimated and true value for $s = 1$ is:

$$\hat{V}^{\pi^t}(1) - V^{\pi^t}(1) = 10.40336 - 10.36761 = 0.03575$$

Problem 2

2.1

We evaluate the policy by using it to form an expected transition and reward function across all actions by choosing the values from each state according to the deterministic policy's action at that state:

```
import itertools

import numpy as np
from IPython.display import Markdown
from p1_1 import gamma, reward_fn, state_space, trans_fn

policy = {1: "g", 2: "g", 3: "h"}

expected_trans_fn = np.reshape(
    [
```

```
        trans_fn.sel(current_state=s, next_state=sn, action=policy[s])
        for s, sn in itertools.product(state_space, state_space)
    ],
    (3, 3),
)

expected_reward_fn = np.array(
    [reward_fn.sel(state=s, action=policy[s]) for s in state_space]
)

value_fn = (
    np.linalg.inv(np.identity(len(state_space)) - gamma * expected_trans_fn)
    @ expected_reward_fn
)

value_latex = Markdown(
    r"\\".join(
        [rf"V^{{\pi}}({s}) = {v:.5f}" for s, v in zip(list("123"), value_fn)]
    )
)
```

The value function is:

$$V^{\pi}(1) = 13.27150$$
$$V^{\pi}(2) = 13.93650$$
$$V^{\pi}(3) = 14.93650$$

2.2

Assuming oracle access, we sample each state-action pair 100 times and use the frequency at which transitions occur to estimate the true transition function:

```
import itertools

import numpy as np
import xarray as xr
from IPython.display import Markdown
from p1_1 import action_space, state_space, trans_fn

NUM_SAMPLES = 100
```

```
phat = xr.zeros_like(trans_fn)

rng = np.random.default_rng(seed=42)

for s, a in itertools.product(state_space, action_space):
    distribution = trans_fn.sel(current_state=s, action=a)

    samples = rng.choice(state_space, p=distribution, size=NUM_SAMPLES)

    phat.loc[dict(current_state=s, action=a)] = np.array(
        [np.sum(samples == s) / NUM_SAMPLES for s in state_space]
    )

phat_latex = Markdown(
    r"\begin{matrix}"
    + r"\\".join(
        [
            ",&".join(
                rf"\hat{{P}}({v.next_state.item()} |"
                rf" {v.current_state.item()},"
                f" {v.action.item()}) = {v.item():.2f}"
                for v in v_sn
            )
            for v_sn in phat.stack(
                {"sa": ["current_state", "action"], "sn": ["next_state"]}
            )
        ]
    )
    + r"\end{matrix}"
)
```

The estimated transition function is:

$$\begin{matrix}
\hat{P}(1|1,g) = 0.09, & \hat{P}(2|1,g) = 0.86, & \hat{P}(3|1,g) = 0.05 \\
\hat{P}(1|1,h) = 0.79, & \hat{P}(2|1,h) = 0.10, & \hat{P}(3|1,h) = 0.11 \\
\hat{P}(1|2,g) = 0.11, & \hat{P}(2|2,g) = 0.18, & \hat{P}(3|2,g) = 0.71 \\
\hat{P}(1|2,h) = 0.05, & \hat{P}(2|2,h) = 0.85, & \hat{P}(3|2,h) = 0.10 \\
\hat{P}(1|3,g) = 0.82, & \hat{P}(2|3,g) = 0.09, & \hat{P}(3|3,g) = 0.09 \\
\hat{P}(1|3,h) = 0.10, & \hat{P}(2|3,h) = 0.12, & \hat{P}(3|3,h) = 0.78
\end{matrix}$$

2.3

We use the given formula to calculate the L1-difference for each state (`numpy` is able to calculate norms of arbitrary order natively).

```
import numpy as np
from IPython.display import Markdown
from p1_1 import state_space, trans_fn
from p2_1 import policy
from p2_2 import phat

l1_error = np.array(
    [
        np.linalg.norm(
            (phat - trans_fn).sel(current_state=s, action=policy[s]), ord=1
        )
        for s in state_space
    ]
)

l1_error_latex = Markdown(
    r"\begin{matrix}"
    + ",&".join([f"{v:.5f}" for v in l1_error])
    + r"\end{matrix}"
)
```

The L1-difference for each state is:

$$0.12000, \quad 0.18000, \quad 0.04000$$

2.4

We use the analytical solution to calculate $\bar{\rho}^{\pi}_{\mu_0}$. Note that μ_0 is simply $[1, 0, 0]^T$ since we know the initial state to be $s = 1$ deterministically.

```
import numpy as np
from IPython.display import Markdown
from p1_1 import gamma
from p2_1 import expected_trans_fn

norm_state_occupancy = (
```

```
    (1 - gamma)
    * np.linalg.inv(np.identity(3) - gamma * expected_trans_fn.T)
    @ [1, 0, 0]
)

assert np.allclose(1, norm_state_occupancy.sum())

norm_state_occupancy_latex = Markdown(
    r"\begin{matrix}"
    + ",&".join([f"{v:.5f}" for v in norm_state_occupancy])
    + r"\end{matrix}"
)
```

The normalized state occupancy measure for each state is:

$$0.14500, \quad 0.19143, \quad 0.66358$$

2.5

We calculate the simulation lemma bound according to

$$|\hat{V}^{\pi}(S_0) - V^{\pi}(S_0)| \leq \frac{\gamma R_{max}}{(1-\gamma)^2} \mathbb{E}_{s \sim \bar{\rho}^{\pi}_{S_0}} \left[\|\hat{P}(\cdot|s, \pi(s)) - P(\cdot|s, \pi(s))\|_1 \right]$$

```
import numpy as np
from IPython.display import Markdown
from p1_1 import gamma, reward_fn
from p2_3 import l1_error
from p2_4 import norm_state_occupancy

simulation_lemma_bound = (
    (gamma * np.max(reward_fn))
    / (1 - gamma) ** 2
    * np.sum(norm_state_occupancy * l1_error)
)


simulation_lemma_bound_latex = Markdown(f"{simulation_lemma_bound:.5f}")
```

The simulation lemma bound on the difference in value functions for policy π in the initial state is:

$$|\hat{V}^{\pi}(1) - V^{\pi}(1)| \leq 29.79181$$

2.6

We can evaluate the policy using the analytical solution by obtaining an expected transition matrix using the deterministic policy.

```
import itertools

import numpy as np
from IPython.display import Markdown
from p1_1 import gamma, state_space
from p2_1 import expected_reward_fn, policy
from p2_2 import phat

expected_phat = np.reshape(
    [
        phat.sel(current_state=s, next_state=sn, action=policy[s])
        for s, sn in itertools.product(state_space, state_space)
    ],
    (3, 3),
)


vhat = (
    np.linalg.inv(np.identity(len(state_space)) - gamma * expected_phat)
    @ expected_reward_fn
)

vhat_latex = Markdown(
    r"\\".join(
        [
            rf"\hat{{V}}^{{\pi}}({s}) = {v:.5f}"
            for s, v in zip(list("123"), vhat)
        ]
    )
)
```

The estimated value function is:

$$V^{\pi}(1) = 13.27150$$
$$V^{\pi}(2) = 13.93650$$
$$V^{\pi}(3) = 14.93650$$

2.7

```
import numpy as np
from IPython.display import Markdown
from p2_1 import value_fn
from p2_6 import vhat

value_error = np.abs(vhat - value_fn)[0]

value_error_latex = Markdown(f"{value_error:.5f}")
```

The absolute difference in value functions at $s = 1$ is:

$$|\hat{V}^{\pi}(1) - V^{\pi}(1)| = 0.77903$$

Problem 3

3.1

We can formulate the tabular representation with a linear combination where each θ_l in $\boldsymbol{\theta}$ is simply the value $Q(s, a)$ for the corresponding index of that state-action pair and each $\boldsymbol{\phi}(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$ is a function returning a vector where each element in the vector is the indicator function supported where $S = s, A = a$:

$$\phi_l(s, a) = \mathbf{1}_{S=s, A=a}(S, A)$$

So, $\boldsymbol{\phi}(s, a)$ is just a vector with 1 at the index corresponding to the pair (s, a) in the argument and 0 everywhere else. The order of the indices in each vector must match in terms of which state-action pair they represent.

So, our class of linear functions $\mathcal{Q}$ is made up of functions that look like this:

$$Q_\theta(s,a) = \begin{bmatrix} Q(S_1,A_1) & Q(S_2,A_2) & \dots & Q(S_{|\mathcal{S}|},A_{|\mathcal{A}|}) \end{bmatrix} \begin{bmatrix} \mathbf{1}_{S=S_1,A=A_1}(S,A) \\ \mathbf{1}_{S=S_2,A=A_1}(S,A) \\ \dots \\ \mathbf{1}_{S=S_{|\mathcal{S}|},A=A_{|\mathcal{A}|}}(S,A) \end{bmatrix}$$

So we have $|\mathcal{S}||\mathcal{A}|$ features and weights.

For example, if we have $\mathcal{S} = \{S_1, S_2\}$ and $\mathcal{A} = \{A_1, A_2\}$, then at (S_1, A_1), the linear function looks like this:

$$Q_\theta(S_1,A_1) = \begin{bmatrix} Q(S_1,A_1) & Q(S_2,A_1) & Q(S_1,A_2) & Q(S_2,A_2) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and we have $|\mathcal{S}||\mathcal{A}| = 2 \cdot 2 = 4$ features/weights in each vector.

3.2

This is similar to the previous problem, but now that the value functions are grouped together into subsets of the state-action space, we can group those together in the formula, reducing the number of features and weights.

Now, instead of each weight being $Q(s,a)$ for a single state-action combination, it will be the value of $Q(s,a)$ which is common to all of the states and actions $(s,a) \in \mathcal{S}_i \times \mathcal{A}_i$. Let

$$Q_{ij} = Q(s,a), \quad (s,a) \in \mathcal{S}_i \times \mathcal{A}_i$$

be this common value, for simplification of notation. Each of these should be multiplied by a feature function which is an indicator function supported at all points where $S \in \mathcal{S}_i, A \in \mathcal{A}_j$:

$$\phi_l(s,a) = \mathbf{1}_{S \in \mathcal{S}_i, A \in \mathcal{A}_j}(S,A)$$

So, $\mathcal{Q}$ is comprised of functions that look like:

$$Q_\theta(s,a) = \begin{bmatrix} Q_{11} & Q_{21} & \dots & Q_{nm} \end{bmatrix} \begin{bmatrix} \mathbf{1}_{S \in \mathcal{S}_1, A \in \mathcal{A}_1}(S,A) \\ \mathbf{1}_{S \in \mathcal{S}_2, A \in \mathcal{A}_1}(S,A) \\ \dots \\ \mathbf{1}_{S \in \mathcal{S}_n, A \in \mathcal{A}_m}(S,A) \end{bmatrix}$$

So we have nm features and weights.

For example, if we have $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$ and $\mathcal{A} = \{A_1\}$, but we can partition $\mathcal{S}$ into $\mathcal{S}_1 = \{S_1, S_2\}$ and $\mathcal{S}_2 = \{S_3, S_4\}$ (and $\mathcal{A}_1$ is a trivial partition of $\mathcal{A}$) due to the fact that $Q(S_1, A_1) = Q(S_2, A_1) = Q_{11}$ and $Q(S_3, A_1) = Q(S_4, A_1) = Q_{21}$, then we can represent Q by Q_θ, which for $s \in \mathcal{S}_1 = \{S_1, S_2\}, a = A_1$ looks like this:

$$Q_\theta(s \in \mathcal{S}_1, a = A_1) = \begin{bmatrix} Q_{11} & Q_{21} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

So even though we have $|\mathcal{S}||\mathcal{A}| = 4$ combinations of states and actions, we have reduced the number of features to $nm = 2 \cdot 1 = 2$ since we could group together the ones with like values.

3.3

If we take the minimization goal and rewrite Q in terms of its parameterized form

$$\hat{Q}^\pi(s, a) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^k} \sum_{i=1}^{N} \left(\boldsymbol{\theta}^T \boldsymbol{\phi}(s^i, a^i) - y^i \right)^2$$

then this is an ordinary least-squares problem of the form

$$\mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\theta} + \boldsymbol{\varepsilon}$$

where $\mathbf{y}$ is an $N \times 1$ column vector of all the observations y^i, $\boldsymbol{\Phi}$ is an $N \times k$ design matrix comprised of a concatenation of row vectors obtained from transposing the column vector returned by each $\boldsymbol{\phi}(x^i) = \boldsymbol{\phi}(s^i, a^i)$ from the observed state-action pairs

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(S_1, A_1)^T \\ \boldsymbol{\phi}(S_2, A_2)^T \\ \cdots \\ \boldsymbol{\phi}(S_N, A_N)^T \end{bmatrix}$$

and $\boldsymbol{\theta}$ is a $k \times 1$ column vector comprised of the weights that we learn in order to minimize the error $\boldsymbol{\varepsilon}$.

It is well known for this form that the solution for the weight vector that yields the least squares error is

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^T \mathbf{y}$$

Note: we can safely assume that $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ has an inverse because we are told that

$$\sum_{i=1}^{N} \boldsymbol{\phi}(s^i, a^i)\boldsymbol{\phi}(s^i, a^i)^T$$

forms an invertible matrix, and this is equivalent to $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$.

From which we can calculate $\hat{Q}^\pi(s, a)$ as the dot product of the learned weight vector and the feature vector for (s, a):

$$\hat{Q}^\pi(s, a) = \boldsymbol{\phi}(s, a)^T\hat{\boldsymbol{\theta}} = \boldsymbol{\phi}(s, a)^T \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T\mathbf{y}$$

Problem 4

$$\begin{aligned}
Q^\pi(s, a) &= \mathbb{E}_{A_t\sim\pi, S_{t+1}\sim P(\cdot|S_t,A_t), T\sim\text{Geo}(1-\gamma)} \left[\sum_{t=0}^{T} R(S_t, A_t)|S_0 = s, A_0 = a\right] \\
&= \mathbb{E}_{A_t\sim\pi, S_{t+1}\sim P(\cdot|S_t,A_t)} \sum_{k=0}^{\infty} \left[(1 - (1 - \gamma))^k (1 - \gamma) \left[\sum_{t=0}^{k} R(S_t, A_t)|S_0 = s, A_0 = a\right]\right] \\
&= \mathbb{E}_{A_t\sim\pi, S_{t+1}\sim P(\cdot|S_t,A_t)} \left[\sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{k} (1 - \gamma)R(S_t, A_t)|S_0 = s, A_0 = a\right]
\end{aligned}$$

If we expand out the double summation, we observe a pattern in the terms:

$$\sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{k} (1 - \gamma)R(S_t, A_t)$$

For $k = 0$, we just have

$$R(S_0, A_0) - \gamma R(S_0, A_0)$$

For $k = 1$, we have

$$\begin{aligned}
&\gamma R(S_0, A_0) - \gamma^2 R(S_0, A_0) \\
+&\gamma R(S_1, A_1) - \gamma^2 R(S_1, A_1)
\end{aligned}$$

such that the first term cancels with the last term of the previous iteration of k...

For $k = 2$, we have

$$
\begin{aligned}
&\gamma^2 R(S_0, A_0) - \gamma^3 R(S_0, A_0) \\
+&\gamma^2 R(S_1, A_1) - \gamma^3 R(S_1, A_1) \\
+&\gamma^2 R(S_2, A_2) - \gamma^3 R(S_2, A_2)
\end{aligned}
$$

such that the first two positive terms cancel with the two negative terms of the previous iteration of k...

For $k = 3$, we have

$$
\begin{aligned}
&\gamma^3 R(S_0, A_0) - \gamma^4 R(S_0, A_0) \\
+&\gamma^3 R(S_1, A_1) - \gamma^4 R(S_1, A_1) \\
+&\gamma^3 R(S_2, A_2) - \gamma^4 R(S_2, A_2) \\
+&\gamma^3 R(S_3, A_3) - \gamma^4 R(S_3, A_3)
\end{aligned}
$$

such that the first three positive terms cancel with the three negative terms of the previous iteration of k...

This pattern continues infinitely as $k \to \infty$ such that all the terms except for the first one in each iteration of k cancel with each other. That is:

$$
\begin{aligned}
&\sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{k} (1-\gamma) R(S_t, A_t) \\
&= R(S_0, A_0) + \gamma R(S_1, A_1) + \gamma^2 R(S_2, A_2) + \dots \\
&= \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)
\end{aligned}
$$

Substituting this into the equation where we left off before expanding the summation, we are left with:

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_{A_t \sim \pi, S_{t+1} \sim P(\cdot|S_t, A_t), T \sim \text{Geo}(1-\gamma)} \left[\sum_{t=0}^{T} R(S_t, A_t) | S_0 = s, A_0 = a \right] \\
&= \mathbb{E}_{A_t \sim \pi, S_{t+1} \sim P(\cdot|S_t, A_t)} \left[\sum_{k=0}^{\infty} \gamma^k \sum_{t=0}^{k} (1-\gamma) R(S_t, A_t) | S_0 = s, A_0 = a \right] \\
&= \mathbb{E}_{A_t \sim \pi, S_{t+1} \sim P(\cdot|S_t, A_t)} \left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) | S_0 = s, A_0 = a \right] \\
&= Q^\pi(s,a) \qquad \blacksquare
\end{aligned}
$$

Problem 5

5.1

We calculate the value function using the given values for $Q^{\pi'}$ by taking the expectation of Q across the action distribution according to π'.

```
import numpy as np
import xarray as xr
from IPython.display import Markdown
from p1_1 import action_space, state_space

policy = xr.DataArray(
    np.array(
        [
            [1.0, 0.0],
            [1.0, 0.0],
            [0.0, 1.0],
        ]
    ),
    coords=dict(state=state_space, action=action_space),
)


policy_prime = xr.DataArray(
    np.array(
        [
            [0.9, 0.1],
            [0.9, 0.1],
            [0.1, 0.9],
        ]
    ),
    coords=dict(state=state_space, action=action_space),
)

q_prime = xr.DataArray(
    np.array(
        [
            [13.0, 12.0],
            [14.0, 13.5],
            [10.0, 15.0],
        ]
```

```
    ),
    coords=dict(state=state_space, action=action_space),
)

value_prime = (q_prime * policy_prime).sum(dim="action")

value_prime_latex = Markdown(
    r"\\".join(
        [
            rf"V^{{\pi}}({s}) = {v:.5f}"
            for s, v in zip(list("123"), value_prime)
        ]
    )
)
```

$$V^{\pi}(1) = 12.90000$$
$$V^{\pi}(2) = 13.95000$$
$$V^{\pi}(3) = 14.50000$$

5.2

Using the performance difference lemma

$$V^{\pi}(S_0) - V^{\pi'}(S_0) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim\bar{\rho}^{\pi}_{S_0}}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^{\pi'}(s,a) - V^{\pi'}(s)\right]\right]$$

```
from IPython.display import Markdown
from p1_1 import gamma
from p5_1 import policy, q_prime, value_prime
from p2_4 import norm_state_occupancy
import numpy as np

expected_q = (policy * q_prime).sum(dim="action")

performance_difference = (1 / (1 - gamma)) * np.sum(
    norm_state_occupancy * (expected_q - value_prime)
)

performance_difference_latex = Markdown(f"{performance_difference:.5f}")
```

$$V^{\pi}(1) - V^{\pi'}(1) = 7.11718$$