ECE59500RL Exam

Robert (Cars) Chandler — chandl71@purdue.edu

Problem 1

1.1

The set of trajectories described is the set of all trajectories such that we start at $X_0 = 1$ and all following states are either 5, 6, or 7. Because the transitions between 5, 6, and 7 are periodic, there is no way to return back to 1 once 5 is reached. As a result, we only have to analyze the first step in the trajectory. If the first step takes us to 5, then the trajectory must belong to the set described. The probability of transitioning to state 5 from the initial state, 1, is labeled as 0.5. If we let $\mathcal{T}$ be the set of trajectories described in the problem statement, then:

$$\mathbb{P}(X_1 = 5|X_0 = 1) = \mathbb{P}(\tau \in \mathcal{T}|X_0 = 1) = 0.5$$

Where τ is any trajectory resulting from $X_0 = 1$.

Final answer: 0.5

1.2

We can solve this by finding the left eigenvectors of the $\mathbf{P}$ matrix:

```
import numpy as np

p = np.array(
    [
        [0, 0.1, 0, 0, 0.5, 0, 0, 0.4, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0.6, 0.4, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0.75, 0.25, 0, 0, 0, 0],
```

```
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0.2, 0.8, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0.9, 0, 0.1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    ]
)

# Using p.T gives us left eigenvectors instead of right
eigvals, eigvecs = np.linalg.eig(p.T)

# Get the indices where the eigenvalues are 1 (accounting for floating point
# math error)
i_ones = np.argwhere(np.abs(eigvals - 1) < 1e-10).squeeze()

# Get the eigenvectors with eigenvalues of 1 (each column of vecs1 is an eigenvector)
vecs1 = eigvecs[:, i_ones]

# The vectors are normalized by default to have a magnitude of 1, but we want
# them to sum to 1
vecs1 /= np.sum(vecs1, axis=0)

# Make each row an eigenvector
# vecs1 = vecs1.T

print("Stationary distributions:")

# Ensure theese eigenvectors display the desired properties
with np.printoptions(precision=4):
    for vec in vecs1.T:
        # Should sum to 1
        assert np.allclose(np.sum(vec), 1)

        # Should yield the same state after applying P
        assert np.allclose(vec @ p, vec)

        print(vec, "\n")
```

```
Stationary distributions:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

[0.    0.    0.375 0.625 0.    0.    0.    0.    0.    0.   ]
```

```
[0.     0.     0.     0.     0.2857 0.3571 0.3571 0.     0.     0.    ]
```

The vectors above represeent the stationary distributions of the Markov chain. We have proven that they are indeed stationary distributions by proving that they all sum to 1 and that $\bar{\boldsymbol{\mu}}\mathbf{P} = \bar{\boldsymbol{\mu}}$ for each vector.

1.3

If $\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2$ is a stationary distribution, then it will satisfy the following properties:

$$\alpha\bar{\mu}^1(s) + (1-\alpha)\bar{\mu}^2(s) \geq 0, \quad \forall s \in \mathcal{S}$$
$$\sum_{s\in\mathcal{S}} \alpha\bar{\mu}^1(s) + (1-\alpha)\bar{\mu}^2(s) = 1$$
$$\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2 = \left(\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2\right)\mathbf{P}$$

Note that if the quantity above is a convex combination, then both α and $(1-\alpha)$ must be nonnegative, and therefore:

$$0 \leq \alpha \leq 1$$

First we show that for every element $\bar{\mu}_i$ of any stationary distribution $\bar{\boldsymbol{\mu}}$:

$$0 \leq \bar{\mu}_i \leq 1$$

The left side of the inequality holds by the definition of a stationary distribution, and the right side holds because $\sum_{s\in\mathcal{S}} \bar{\mu}_i = 1$ and since each $\bar{\mu}_i$ is positive, the maximum value for any one $\bar{\mu}_i$ is 1.

With this in mind, it becomes apparent that the first of the three properties above holds, because if each element of μ^1, μ^2 is between 0 and 1 and α and $(1-\alpha)$ are also between 0 and 1, then we just have a sum of two products of positive values, the result of which will always be positive.

Next:

$$
\begin{aligned}
&\sum_{s\in\mathcal{S}} \alpha\bar{\mu}^1(s) + (1-\alpha)\bar{\mu}^2(s) \\
&= \alpha \sum_{s\in\mathcal{S}} \bar{\mu}^1(s) + (1-\alpha)\sum_{s\in\mathcal{S}} \bar{\mu}^2(s) && \text{by linearity of summation} \\
&= \alpha \cdot 1 + (1-\alpha)\cdot 1 && \text{since } \sum_{s\in\mathcal{S}} \bar{\mu} = 1 \text{ by definition} \\
&= 1
\end{aligned}
$$

And finally:

$$
\begin{aligned}
\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2 &= \left(\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2\right)\mathbf{P} \\
\alpha\bar{\boldsymbol{\mu}}^1 + (1-\alpha)\bar{\boldsymbol{\mu}}^2 &= \alpha\bar{\boldsymbol{\mu}}^1\mathbf{P} + (1-\alpha)\bar{\boldsymbol{\mu}}^2\mathbf{P} \\
\alpha\bar{\boldsymbol{\mu}}^1 - \alpha\bar{\boldsymbol{\mu}}^1\mathbf{P} &= (1-\alpha)\bar{\boldsymbol{\mu}}^2\mathbf{P} - (1-\alpha)\bar{\boldsymbol{\mu}}^2 \\
\alpha\left(\bar{\boldsymbol{\mu}}^1 - \bar{\boldsymbol{\mu}}^1\mathbf{P}\right) &= (1-\alpha)\left(\bar{\boldsymbol{\mu}}^2\mathbf{P} - \bar{\boldsymbol{\mu}}^2\right) \\
\alpha\left(\bar{\boldsymbol{\mu}}^1 - \bar{\boldsymbol{\mu}}^1\right) &= (1-\alpha)\left(\bar{\boldsymbol{\mu}}^2 - \bar{\boldsymbol{\mu}}^2\right) && \text{since } \bar{\boldsymbol{\mu}}\mathbf{P} = \bar{\boldsymbol{\mu}} \text{ by definition} \\
\alpha\cdot 0 &= (1-\alpha)\cdot 0 \\
0 &= 0
\end{aligned}
$$

■

Having proven that the convex combination satisfies all the properties of a stationary distribution, we have proven that it is itself a stationary distribution

1.4

TODO: what else can we do with this?

Initializing our chain at $\alpha\mu_0^1 + (1-\alpha)\mu_0^2$ will cause μ_t to converge to

$$\alpha\bar{\mu}^1 + (1-\alpha)\bar{\mu}^2$$

as $t \to \infty$. Intuitively, the pieces of the initial state distribution that were carried to their respective stationary distribution will still be carried to the same distribution, but in proportion based on the fractions α and $(1-\alpha)$.

Problem 3

3.1

The objective of the primal linear program is:

$$\min_{V \in \mathbb{R}^{|\mathcal{S}|}} \sum_{s \in \mathcal{S}} \mu_0(s) V(s) = \min_{V \in \mathbb{R}^{|\mathcal{S}|}} \left[0.4V(b) + 0.6V(c) \right]$$

and this objective is subject to the following constraint:

$$V(s) \geq R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[V(s') \right], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

We will first define this for $s = b, a = x$:

$$\begin{aligned}
V(b) &\geq R(b,x) + \gamma \mathbb{E}_{s' \sim P(\cdot|b,x)} \left[V(s') \right] \\
V(b) &\geq R(b,x) + \gamma \sum_{s' \in \mathcal{S}} P(s'|b,x) V(s') \\
V(b) &\geq R(b,x) + \gamma \left[P(b|b,x)V(b) + P(c|b,x)V(c) \right] \\
V(b) &\geq R(b,x) + \gamma \left[P(b|b,x)V(b) + P(c|b,x)V(c) \right]
\end{aligned}$$

We now show this final line for each of the four state-action combinations and substitute in our known values to yield four simultaneous constraints that must be met:

$$
\begin{aligned}
V(b) &\geq R(b,x) + \gamma\left[P(b|b,x)V(b) + P(c|b,x)V(c)\right] \\
V(b) &\geq 0.5 + 0.9\left[1 \cdot V(b) + 0 \cdot V(c)\right] \\
V(b) &\geq 0.5 + 0.9V(b) \\
0.1V(b) &\geq 0.5 \\
V(b) &\geq 5
\end{aligned}
$$

$$
\begin{aligned}
V(b) &\geq R(b,y) + \gamma\left[P(b|b,y)V(b) + P(c|b,y)V(c)\right] \\
V(b) &\geq 0.2 + 0.9\left[0.2V(b) + 0.8V(c)\right] \\
V(b) &\geq 0.2 + 0.18V(b) + 0.72V(c) \\
0.82V(b) &\geq 0.2 + 0.72V(c) \\
V(b) &\geq 0.2439 + 0.878V(c)
\end{aligned}
$$

$$
\begin{aligned}
V(c) &\geq R(c,x) + \gamma\left[P(b|c,x)V(b) + P(c|c,x)V(c)\right] \\
V(c) &\geq 0.5 + 0.9\left[0.1V(b) + 0.9V(c)\right] \\
V(c) &\geq 0.5 + 0.09V(b) + 0.81V(c) \\
0.19V(c) &\geq 0.5 + 0.09V(b) \\
V(c) &\geq 2.6316 + 0.47368V(b)
\end{aligned}
$$

$$
\begin{aligned}
V(c) &\geq R(c,y) + \gamma\left[P(b|c,y)V(b) + P(c|c,y)V(c)\right] \\
V(c) &\geq 1 + 0.9\left[0.6V(b) + 0.4V(c)\right] \\
V(c) &\geq 1 + 0.54V(b) + 0.36V(c) \\
0.64V(c) &\geq 1 + 0.54V(b) \\
V(c) &\geq 1.5625 + 0.84375V(b)
\end{aligned}
$$

3.2

The objective of the dual linear program is:

$$\max_{\nu \geq 0} \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \nu(s,a) R(s,a) = \max_{\nu \geq 0} \left[0.5\nu(b,x) + 0.2\nu(b,y) + 0.5\nu(c,x) + \nu(c,y)\right]$$

and this objective is subject to the following constraint:

$$\sum_{a \in \mathcal{A}} \nu(s,a) = \mu_0(s) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} P(s|s',a')\nu(s',a'), \quad \forall s \in \mathcal{S}$$

Beginning with $s = b$:

$$\begin{aligned}
\sum_{a \in \mathcal{A}} \nu(b, a) &= \mu_0(b) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} P(b|s', a')\nu(s', a') \\
\nu(b, x) + \nu(b, y) &= 0.4 + 0.9\left[\nu(b, x) + 0.2\nu(b, y) + 0.1\nu(c, x) + 0.6\nu(c, y)\right] \\
\nu(b, x) + \nu(b, y) &= 0.4 + 0.9\nu(b, x) + 0.18\nu(b, y) + 0.09\nu(c, x) + 0.54\nu(c, y) \\
0.1\nu(b, x) + 0.82\nu(b, y) - 0.09\nu(c, x) - 0.54\nu(c, y) &= 0.4
\end{aligned}$$

and for $s = c$:

$$\begin{aligned}
\sum_{a \in \mathcal{A}} \nu(c, a) &= \mu_0(c) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} P(c|s', a')\nu(s', a') \\
\nu(c, x) + \nu(c, y) &= 0.6 + 0.9\left[0 \cdot \nu(b, x) + 0.8\nu(b, y) + 0.9\nu(c, x) + 0.4\nu(c, y)\right] \\
\nu(c, x) + \nu(c, y) &= 0.6 + 0.72\nu(b, y) + 0.81\nu(c, x) + 0.36\nu(c, y) \\
0.19\nu(c, x) + 0.64\nu(c, y) - 0.72\nu(b, y) &= 0.6
\end{aligned}$$

So in conclusion, the dual LP is subject to the two following constraints:

$$\begin{aligned}
0.1\nu(b, x) + 0.82\nu(b, y) - 0.09\nu(c, x) - 0.54\nu(c, y) &= 0.4 \\
0.19\nu(c, x) + 0.64\nu(c, y) - 0.72\nu(b, y) &= 0.6
\end{aligned}$$

Problem 4

4.1

The return for each trajectory is calculated as:

$$\sum_{t=0}^{T} \gamma^t R(S_t, A_t)$$

We can calculate these in Python:

```
import numpy as np
from IPython.display import Markdown

trajectories = [
    dict(
        states=np.array(list("cbdeec")),
        actions=np.array(list("zxyxx")),
```

```
        rewards=np.array([1, 0, 1, 2, 2]),
    ),
    dict(
        states=np.array(list("cdecee")),
        actions=np.array(list("yxzzx")),
        rewards=np.array([0, 1, -1, 1, 2]),
    ),
    dict(
        states=np.array(list("cbcebe")),
        actions=np.array(list("yzyxy")),
        rewards=np.array([0, 0.5, 0, 2, 0]),
    ),
    dict(
        states=np.array(list("ccebed")),
        actions=np.array(list("zxxyy")),
        rewards=np.array([1, 0.5, 2, 0, -1]),
    ),
]

gamma = 0.9

gammas = gamma ** np.arange(5)

returns = [np.sum(traj["rewards"] * gammas) for traj in trajectories]

returns_latex = Markdown(
    r"\\".join([f"G_{i+1} &= {ret:0.4f}" for i, ret in enumerate(returns)])
)
```

The returns for each trajectory are:

$$
\begin{aligned}
G_1 &= 4.5802 \\
G_2 &= 2.1312 \\
G_3 &= 1.9080 \\
G_4 &= 2.4139
\end{aligned}
$$

4.2

We can sum the probabilities of taking each action in τ_2 at the corresponding states using the policy tables defined in the problem statement. We can recreate these tables in Python and

then determine the probability of the trajectory by multiplying out the probability of taking each individual step and compare the result for each policy. We will go ahead and do this for τ_4 as well to obtain the answer for problem 4.3.

```
from copy import deepcopy

import numpy as np
import xarray as xr

# Define a multidimensional array containing probabilities of state-action
# pairs. This is just like a regular numpy array but with labeled "coordinates"
# and axes.
policy_t = xr.DataArray(
    data=np.array(
        [
            [0.1, 0.1, 0.8],
            [0.05, 0.2, 0.75],
            [0.25, 0.5, 0.25],
            [0.9, 0.02, 0.08],
        ]
    ),
    coords=dict(state=list("bcde"), action=list("xyz")),
)

policy_b = xr.DataArray(
    data=np.array(
        [
            [0.5, 0.1, 0.4],
            [0.05, 0.2, 0.75],
            [0.5, 0.2, 0.3],
            [0.9, 0.02, 0.08],
        ]
    ),
    coords=dict(state=list("bcde"), action=list("xyz")),
)


# Double-check that our probabilities for a given state all sum to one
assert np.allclose(policy_t.sum(dim="action"), 1)
assert np.allclose(policy_b.sum(dim="action"), 1)

from p4_1 import trajectories
```

```
p_target = {2: 0.0, 4: 0.0}
p_behavior = {2: 0.0, 4: 0.0}

for prob, policy in zip([p_target, p_behavior], [policy_t, policy_b]):
    for traj, i_traj in zip([trajectories[1], trajectories[3]], [2, 4]):

        # Walk each step of the trajectory, determine the probability of taking an
        # action at a given state according to the policy of interest, and take the
        # product of all the probabilities to get the total probability of the
        # trajectory.
        # We included the final state in the definition, but we don't need it here,
        # so we skip it in our iteration (i.e. traj["states"[:-1]])
        prob[i_traj] = float(
            np.prod(
                [
                    policy.sel(state=s, action=a)
                    for s, a in zip(traj["states"][:-1], traj["actions"])
                ]
            )
        )
```

The probability of taking τ_2 according to π^t is 0.0027 while the probability according to π^b is 0.0054, so τ_2 **is more probable under** π^b **than under** π^t.

4.3

As described above, we already calculated the probabilities for τ_4 in 4.2.

The probability of taking τ_4 according to π^t is $6.75e-05$ while the probability according to π^b is $6.75e-05$, so τ_4 **is equally probable under** π^b **or** π^t.

4.4

We start with the (on-policy) evaluation of $V^{\pi^b}(c)$ of the behavior polic using an every-visit Monte Carlo method:

```
import numpy as np
from IPython.display import Markdown
from p4_1 import gammas, trajectories

vhat = {state: 0.0 for state in "bcde"}
```

```
num_visits = {state: 0.0 for state in "bcde"}
returns = {state: [] for state in "bcde"}


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


def mc_evaluation(vhat, returns, num_visits, return_function):
    for traj in trajectories:
        # For each state-action pair... (excluding state c since we end at that state)
        for state in np.unique(traj["states"][:-1]):
            # For each step in the trajectory...
            for t, (s, a, r) in enumerate(
                zip(traj["states"][:-1], traj["actions"], traj["rewards"])
            ):
                # We only want to evaluate each time where state is visited
                if s != state:
                    continue

                ret = return_function(t, traj, gammas)

                returns[state].append(ret)

                num_visits[state] += 1

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

    return vhat, returns, num_visits


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

vhat_latex = Markdown(
    r"\\".join(
        [rf"V^{{\pi^b}}({s}) &\approx {v:.5f}" for s, v in vhat.items()]
    )
)
```

The resulting value function for each state is:

$$V^{\pi^b}(b) \approx 1.20803$$
$$V^{\pi^b}(c) \approx 2.27806$$
$$V^{\pi^b}(d) \approx 2.85570$$
$$V^{\pi^b}(e) \approx 1.19880$$

4.5

We repeat the same every-visit Monte Carlo estimation of the value function, but this time we perform an off-policy estimate for the target policy using the behavior policy:

```
import numpy as np
from IPython.display import Markdown
from p4_1 import gammas, trajectories
from p4_2 import policy_b, policy_t
from p4_4 import mc_evaluation, return_at_t

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


def weighted_return_at_t(t, traj, gammas):
    weight = np.prod(
        [
            (
                policy_t.sel(state=s, action=a)
                / policy_b.sel(state=s, action=a)
            )
            for s, a in zip(traj["states"][:-1][t:], traj["actions"][t:])
        ]
    )

    return weight * return_at_t(t, traj, gammas)


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

vhat_latex = Markdown(
    r"\\".join(
        [rf"V^{{\pi^b}}({s}) &\approx {v:.5f}" for s, v in vhat.items()]
```

```
    )
)
```

The resulting value function for each state is:

$$V^{\pi^b}(b) \approx 1.23750$$
$$V^{\pi^b}(c) \approx 2.07124$$
$$V^{\pi^b}(d) \approx 5.00805$$
$$V^{\pi^b}(e) \approx 1.19880$$