ECE59500RL HW2

Robert (Cars) Chandler — chandl71@purdue.edu

Problem 1

Problem 2

Problem 3

Problem 4

First, we need to assign labels to the states, which are the different spaces on the board. We will use a zero-indexed x-y coordinate system to refer to the different states $s_{xy} \in \mathcal{S}$, with the origin at the bottom left square, $s_{0,0}$. Moving horizontally will increase the x-component and vertically the y-component, so that our state space is

$$\mathcal{S} = \{s_{ij} : i, j \in \mathbb{N}_0, \quad i, j \leq 5\}$$

> **Note**
>
> Although we are using two "dimensions" to identify each state, we still treat it as a one-dimensional vector, so that we have one row for each $s \in \mathcal{S}$ in $\vec{v}$, P^π, and so forth. The order of the states for this vector will always be in row-major order:
>
> $$(0,0),(1,0),(2,0),(3,0),(4,0),(0,1),(1,1),...,(3,4),(4,4)$$

4.1.a

The policy can be evaluated analytically using the following equation:

$$\vec{v}^\pi = (I - \gamma P^\pi)^{-1} \vec{R}^\pi \tag{1}$$

We have $\gamma = 0.95$ from the problem statement. P^π and $\vec{R}^\pi$ each need to be evaluated by going over each state in $\mathcal{S}$ and using the information given to us to evaluate them. Beginning with P^π:

$$P^\pi_{ij} = P(s_j|s_i, a) = P(s_j|s_i, \pi(s_i))$$

We can use Python to encode the logic described in the problem statement to programatically calculate P^π for each state transition:

```
from enum import Enum
import numpy as np
import itertools
from IPython.display import Markdown


class Space(Enum):
    LIGHTNING = -1
    NORMAL = 0
    MOUNTAIN = 1
    TREASURE = 2


width = 5

board = np.full([width, width], Space.NORMAL)

board[2, 1] = Space.MOUNTAIN
board[3, 1] = Space.MOUNTAIN
board[1, 3] = Space.MOUNTAIN
board[2, 3] = Space.LIGHTNING
board[4, 4] = Space.TREASURE

policy = np.array(
    [
        list("URRUU"),
        list("UDDDU"),
        list("UURRR"),
        list("LULLU"),
        list("RRRRU"),
    ]
).T
```

```python
p = np.zeros([25, 25])


def i_1d(x, y):
    return np.ravel_multi_index([y, x], dims=[width, width])


def is_blocked(x, y):
    return (
        x < 0
        or y < 0
        or x >= width
        or y >= width
        or board[x, y] == Space.MOUNTAIN
    )


for x, y in itertools.product(range(5), range(5)):
    i_cur_1d = i_1d(x, y)

    if board[x, y] != Space.NORMAL:
        p[i_cur_1d, i_cur_1d] = 1
        continue

    for a, (i2, j2) in zip(
        list("LRUD"), [[x - 1, y], [x + 1, y], [x, y + 1], [x, y - 1]]
    ):
        prob = 0.85 if a == policy[x, y] else 0.05
        if is_blocked(i2, j2):
            p[i_cur_1d, i_cur_1d] += prob
        else:
            p[i_cur_1d, i_1d(i2, j2)] += prob

state_text = []
for i in range(width**2):
    y1, x1 = np.unravel_index(i, [width, width])
    a = policy[x1, y1]
    for j in np.argwhere(p[i]).flatten():
        y2, x2 = np.unravel_index(j, [width, width])
        state_text.append(
            rf"P^{{\pi}}(s_{{ {x2}, {y2} }} | s_{{ {x1}, {y1} }}, \pi(s_{{"
            rf" {x1},"
```

```
            rf" {y1} }}) = \text{{{a}}}) &= {p[i, j]:g} \\"
        )
    state_text.append(r"\\")

state_text = "\n".join(state_text)


def vecfmt(v):
    return Markdown(", ".join([f"{e:g}" for e in v]))
```

The resulting state transition probabilities are listed as follows:

$$
\begin{aligned}
P^\pi(s_{0,0}|s_{0,0}, \pi(s_{0,0}) = \text{U}) &= 0.1 \\
P^\pi(s_{1,0}|s_{0,0}, \pi(s_{0,0}) = \text{U}) &= 0.05 \\
P^\pi(s_{0,1}|s_{0,0}, \pi(s_{0,0}) = \text{U}) &= 0.85 \\
\\
P^\pi(s_{0,0}|s_{1,0}, \pi(s_{1,0}) = \text{R}) &= 0.05 \\
P^\pi(s_{1,0}|s_{1,0}, \pi(s_{1,0}) = \text{R}) &= 0.05 \\
P^\pi(s_{2,0}|s_{1,0}, \pi(s_{1,0}) = \text{R}) &= 0.85 \\
P^\pi(s_{1,1}|s_{1,0}, \pi(s_{1,0}) = \text{R}) &= 0.05 \\
\\
P^\pi(s_{1,0}|s_{2,0}, \pi(s_{2,0}) = \text{R}) &= 0.05 \\
P^\pi(s_{2,0}|s_{2,0}, \pi(s_{2,0}) = \text{R}) &= 0.1 \\
P^\pi(s_{3,0}|s_{2,0}, \pi(s_{2,0}) = \text{R}) &= 0.85 \\
\\
P^\pi(s_{2,0}|s_{3,0}, \pi(s_{3,0}) = \text{U}) &= 0.05 \\
P^\pi(s_{3,0}|s_{3,0}, \pi(s_{3,0}) = \text{U}) &= 0.9 \\
P^\pi(s_{4,0}|s_{3,0}, \pi(s_{3,0}) = \text{U}) &= 0.05 \\
\\
P^\pi(s_{3,0}|s_{4,0}, \pi(s_{4,0}) = \text{U}) &= 0.05 \\
P^\pi(s_{4,0}|s_{4,0}, \pi(s_{4,0}) = \text{U}) &= 0.1 \\
P^\pi(s_{4,1}|s_{4,0}, \pi(s_{4,0}) = \text{U}) &= 0.85 \\
\\
P^\pi(s_{0,0}|s_{0,1}, \pi(s_{0,1}) = \text{U}) &= 0.05 \\
P^\pi(s_{0,1}|s_{0,1}, \pi(s_{0,1}) = \text{U}) &= 0.05
\end{aligned}
$$

$$P^\pi(s_{1,1}|s_{0,1},\pi(s_{0,1})=\mathrm{U})=0.05$$
$$P^\pi(s_{0,2}|s_{0,1},\pi(s_{0,1})=\mathrm{U})=0.85$$

$$P^\pi(s_{1,0}|s_{1,1},\pi(s_{1,1})=\mathrm{D})=0.85$$
$$P^\pi(s_{0,1}|s_{1,1},\pi(s_{1,1})=\mathrm{D})=0.05$$
$$P^\pi(s_{1,1}|s_{1,1},\pi(s_{1,1})=\mathrm{D})=0.05$$
$$P^\pi(s_{1,2}|s_{1,1},\pi(s_{1,1})=\mathrm{D})=0.05$$

$$P^\pi(s_{2,1}|s_{2,1},\pi(s_{2,1})=\mathrm{D})=1$$

$$P^\pi(s_{3,1}|s_{3,1},\pi(s_{3,1})=\mathrm{D})=1$$

$$P^\pi(s_{4,0}|s_{4,1},\pi(s_{4,1})=\mathrm{U})=0.05$$
$$P^\pi(s_{4,1}|s_{4,1},\pi(s_{4,1})=\mathrm{U})=0.1$$
$$P^\pi(s_{4,2}|s_{4,1},\pi(s_{4,1})=\mathrm{U})=0.85$$

$$P^\pi(s_{0,1}|s_{0,2},\pi(s_{0,2})=\mathrm{U})=0.05$$
$$P^\pi(s_{0,2}|s_{0,2},\pi(s_{0,2})=\mathrm{U})=0.05$$
$$P^\pi(s_{1,2}|s_{0,2},\pi(s_{0,2})=\mathrm{U})=0.05$$
$$P^\pi(s_{0,3}|s_{0,2},\pi(s_{0,2})=\mathrm{U})=0.85$$

$$P^\pi(s_{1,1}|s_{1,2},\pi(s_{1,2})=\mathrm{U})=0.05$$
$$P^\pi(s_{0,2}|s_{1,2},\pi(s_{1,2})=\mathrm{U})=0.05$$
$$P^\pi(s_{1,2}|s_{1,2},\pi(s_{1,2})=\mathrm{U})=0.85$$
$$P^\pi(s_{2,2}|s_{1,2},\pi(s_{1,2})=\mathrm{U})=0.05$$

$$P^\pi(s_{1,2}|s_{2,2},\pi(s_{2,2})=\mathrm{R})=0.05$$
$$P^\pi(s_{2,2}|s_{2,2},\pi(s_{2,2})=\mathrm{R})=0.05$$
$$P^\pi(s_{3,2}|s_{2,2},\pi(s_{2,2})=\mathrm{R})=0.85$$
$$P^\pi(s_{2,3}|s_{2,2},\pi(s_{2,2})=\mathrm{R})=0.05$$

$$P^\pi(s_{2,2}|s_{3,2},\pi(s_{3,2})=\mathrm{R})=0.05$$
$$P^\pi(s_{3,2}|s_{3,2},\pi(s_{3,2})=\mathrm{R})=0.05$$

$$P^{\pi}(s_{4,2}|s_{3,2}, \pi(s_{3,2}) = \mathrm{R}) = 0.85$$
$$P^{\pi}(s_{3,3}|s_{3,2}, \pi(s_{3,2}) = \mathrm{R}) = 0.05$$

$$P^{\pi}(s_{4,1}|s_{4,2}, \pi(s_{4,2}) = \mathrm{R}) = 0.05$$
$$P^{\pi}(s_{3,2}|s_{4,2}, \pi(s_{4,2}) = \mathrm{R}) = 0.05$$
$$P^{\pi}(s_{4,2}|s_{4,2}, \pi(s_{4,2}) = \mathrm{R}) = 0.85$$
$$P^{\pi}(s_{4,3}|s_{4,2}, \pi(s_{4,2}) = \mathrm{R}) = 0.05$$

$$P^{\pi}(s_{0,2}|s_{0,3}, \pi(s_{0,3}) = \mathrm{L}) = 0.05$$
$$P^{\pi}(s_{0,3}|s_{0,3}, \pi(s_{0,3}) = \mathrm{L}) = 0.9$$
$$P^{\pi}(s_{0,4}|s_{0,3}, \pi(s_{0,3}) = \mathrm{L}) = 0.05$$

$$P^{\pi}(s_{1,3}|s_{1,3}, \pi(s_{1,3}) = \mathrm{U}) = 1$$

$$P^{\pi}(s_{2,3}|s_{2,3}, \pi(s_{2,3}) = \mathrm{L}) = 1$$

$$P^{\pi}(s_{3,2}|s_{3,3}, \pi(s_{3,3}) = \mathrm{L}) = 0.05$$
$$P^{\pi}(s_{2,3}|s_{3,3}, \pi(s_{3,3}) = \mathrm{L}) = 0.85$$
$$P^{\pi}(s_{4,3}|s_{3,3}, \pi(s_{3,3}) = \mathrm{L}) = 0.05$$
$$P^{\pi}(s_{3,4}|s_{3,3}, \pi(s_{3,3}) = \mathrm{L}) = 0.05$$

$$P^{\pi}(s_{4,2}|s_{4,3}, \pi(s_{4,3}) = \mathrm{U}) = 0.05$$
$$P^{\pi}(s_{3,3}|s_{4,3}, \pi(s_{4,3}) = \mathrm{U}) = 0.05$$
$$P^{\pi}(s_{4,3}|s_{4,3}, \pi(s_{4,3}) = \mathrm{U}) = 0.05$$
$$P^{\pi}(s_{4,4}|s_{4,3}, \pi(s_{4,3}) = \mathrm{U}) = 0.85$$

$$P^{\pi}(s_{0,3}|s_{0,4}, \pi(s_{0,4}) = \mathrm{R}) = 0.05$$
$$P^{\pi}(s_{0,4}|s_{0,4}, \pi(s_{0,4}) = \mathrm{R}) = 0.1$$
$$P^{\pi}(s_{1,4}|s_{0,4}, \pi(s_{0,4}) = \mathrm{R}) = 0.85$$

$$P^{\pi}(s_{0,4}|s_{1,4}, \pi(s_{1,4}) = \mathrm{R}) = 0.05$$
$$P^{\pi}(s_{1,4}|s_{1,4}, \pi(s_{1,4}) = \mathrm{R}) = 0.1$$
$$P^{\pi}(s_{2,4}|s_{1,4}, \pi(s_{1,4}) = \mathrm{R}) = 0.85$$

$$P^\pi(s_{2,3}|s_{2,4}, \pi(s_{2,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{1,4}|s_{2,4}, \pi(s_{2,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{2,4}|s_{2,4}, \pi(s_{2,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{3,4}|s_{2,4}, \pi(s_{2,4}) = \text{R}) = 0.85$$

$$P^\pi(s_{3,3}|s_{3,4}, \pi(s_{3,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{2,4}|s_{3,4}, \pi(s_{3,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{3,4}|s_{3,4}, \pi(s_{3,4}) = \text{R}) = 0.05$$
$$P^\pi(s_{4,4}|s_{3,4}, \pi(s_{3,4}) = \text{R}) = 0.85$$

$$P^\pi(s_{4,4}|s_{4,4}, \pi(s_{4,4}) = \text{U}) = 1$$

All other possible state transitions have probability 0.

Moving onto $\vec{R}^\pi$:

$$\vec{R}^\pi = \begin{bmatrix} R(s_{1,1}, \pi(s_{1,1}) \\ R(s_{1,2}, \pi(s_{1,2}) \\ \cdots \\ R(s_{4,4}, \pi(s_{4,4})) \end{bmatrix}_{|\mathcal{S}|\times 1}$$

So given the reward function described, we just have a simple vector with two nonzero elements:

```
r = np.zeros(width * width, dtype=int)
r[i_1d(*np.argwhere(board == Space.LIGHTNING).squeeze())] = -1
r[i_1d(*np.argwhere(board == Space.TREASURE).squeeze())] = 1
```

$$\vec{R}^\pi = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,1]^T$$

So, now we can substitute in our values into Equation 1 and solve:

```python
gamma = 0.95
v = np.linalg.inv(np.identity(width**2) - gamma * p) @ r
value_text = []
for i, val in enumerate(v):
    y, x = np.unravel_index(i, [width, width])
    value_text.append(rf"v^{{\pi}}(s_{{ {x},{y} }}) &= {val:g} \\")

value_text = "\n".join(value_text)
```

$$
\begin{aligned}
v^{\pi}(s_{0,0}) &= 4.33896 \\
v^{\pi}(s_{1,0}) &= 2.64128 \\
v^{\pi}(s_{2,0}) &= 2.70644 \\
v^{\pi}(s_{3,0}) &= 2.87785 \\
v^{\pi}(s_{4,0}) &= 6.07858 \\
v^{\pi}(s_{0,1}) &= 4.70749 \\
v^{\pi}(s_{1,1}) &= 2.61613 \\
v^{\pi}(s_{2,1}) &= 0 \\
v^{\pi}(s_{3,1}) &= 0 \\
v^{\pi}(s_{4,1}) &= 6.64324 \\
v^{\pi}(s_{0,2}) &= 5.14368 \\
v^{\pi}(s_{1,2}) &= 2.85117 \\
v^{\pi}(s_{2,2}) &= 3.79493 \\
v^{\pi}(s_{3,2}) &= 5.48513 \\
v^{\pi}(s_{4,2}) &= 7.08781 \\
v^{\pi}(s_{0,3}) &= 5.62269 \\
v^{\pi}(s_{1,3}) &= 0 \\
v^{\pi}(s_{2,3}) &= -20 \\
v^{\pi}(s_{3,3}) &= -14.2964 \\
v^{\pi}(s_{4,3}) &= 16.5959 \\
v^{\pi}(s_{0,4}) &= 12.0203 \\
v^{\pi}(s_{1,4}) &= 13.1409 \\
v^{\pi}(s_{2,4}) &= 14.0205 \\
v^{\pi}(s_{3,4}) &= 16.9416
\end{aligned}
$$

$$v^\pi(s_{4,4}) = 20$$

4.1.b

Let $\vec{v}_0 = \mathbf{0}$. Then for each step in the iteration,

$$\vec{v}_{t+1} = \vec{R}^\pi + \gamma P^\pi \vec{v}_t$$

To determine T, the number of iterations we need to make in order to obtain $\|v_T - v^\pi\|_\infty \leq 0.01$, we can use the following theorem:

$$T \geq \frac{\log\left(\frac{\|\vec{v}_0 - \vec{v}^\pi\|_\infty}{\varepsilon}\right)}{\log \frac{1}{\gamma}}$$

Which we can solve using Python, and then perform T iterations by implementing the equation above:

```
epsilon = 0.01
v0 = np.zeros(width**2)
n_timesteps = int(
    np.ceil(np.log(np.max(v0 - v) / epsilon) / np.log(1 / gamma))
)

v_t = v0
v_history = [v0]
for t in range(n_timesteps):
    v_t = r + gamma * p @ v_t
    v_history.append(v_t)

v_history = np.array(v_history)

value_text_iter = []
for i, val in enumerate(v):
    y, x = np.unravel_index(i, [width, width])
    value_text_iter.append(
        rf"v_{{T = {n_timesteps}}} (s_{{ {x},{y} }}) &= {val:g} \\"
    )

value_text_iter = "\n".join(value_text_iter)
```

We perform $T = 149$ iterations, and our final v_T is:

$$
\begin{aligned}
v_{T=149}(s_{0,0}) &= 4.33896 \\
v_{T=149}(s_{1,0}) &= 2.64128 \\
v_{T=149}(s_{2,0}) &= 2.70644 \\
v_{T=149}(s_{3,0}) &= 2.87785 \\
v_{T=149}(s_{4,0}) &= 6.07858 \\
v_{T=149}(s_{0,1}) &= 4.70749 \\
v_{T=149}(s_{1,1}) &= 2.61613 \\
v_{T=149}(s_{2,1}) &= 0 \\
v_{T=149}(s_{3,1}) &= 0 \\
v_{T=149}(s_{4,1}) &= 6.64324 \\
v_{T=149}(s_{0,2}) &= 5.14368 \\
v_{T=149}(s_{1,2}) &= 2.85117 \\
v_{T=149}(s_{2,2}) &= 3.79493 \\
v_{T=149}(s_{3,2}) &= 5.48513 \\
v_{T=149}(s_{4,2}) &= 7.08781 \\
v_{T=149}(s_{0,3}) &= 5.62269 \\
v_{T=149}(s_{1,3}) &= 0 \\
v_{T=149}(s_{2,3}) &= -20 \\
v_{T=149}(s_{3,3}) &= -14.2964 \\
v_{T=149}(s_{4,3}) &= 16.5959 \\
v_{T=149}(s_{0,4}) &= 12.0203 \\
v_{T=149}(s_{1,4}) &= 13.1409 \\
v_{T=149}(s_{2,4}) &= 14.0205 \\
v_{T=149}(s_{3,4}) &= 16.9416 \\
v_{T=149}(s_{4,4}) &= 20
\end{aligned}
$$

We can verify that our desired conditon holds:

```
max_error = np.max(v_t - v)
```

So,

$$\Vert v_T - v^\pi \Vert_\infty = 0.009591 \leq 0.01$$

And if we were to have performed just one less iteration, this value would be above the desired $\varepsilon = 0.01$.

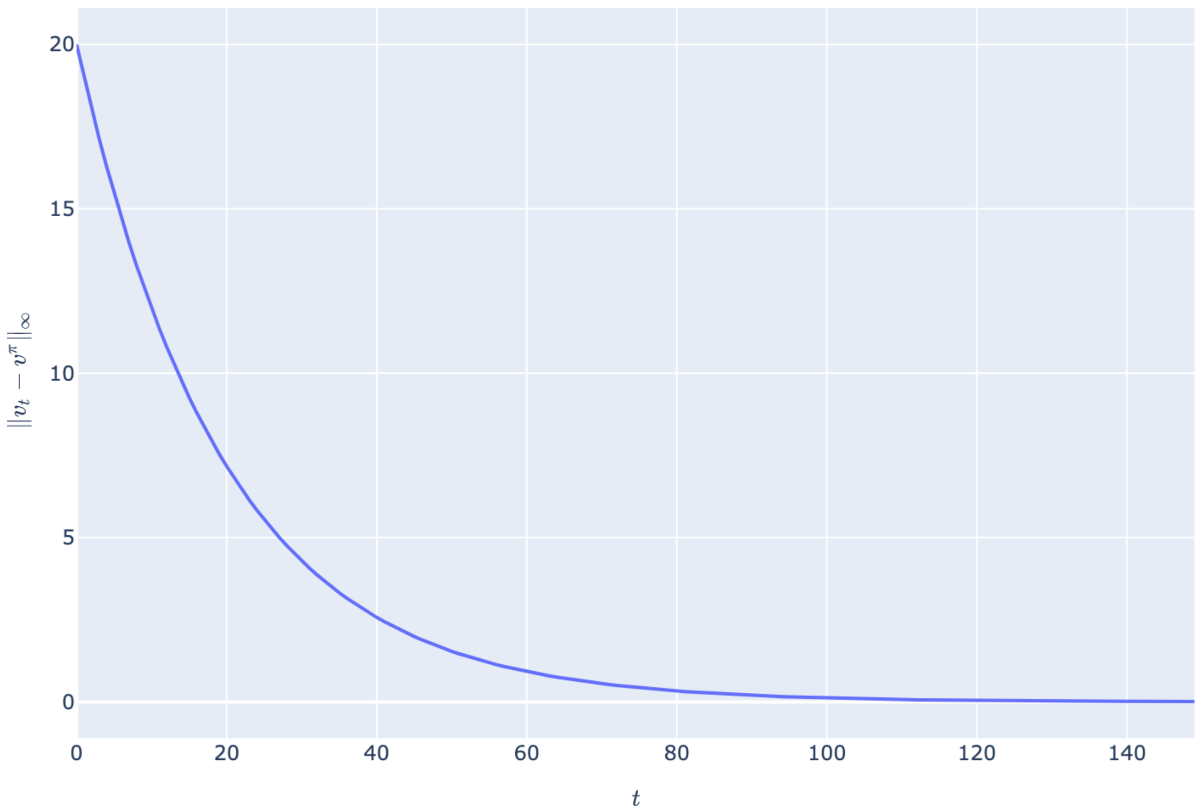
4.1.c

We kept track of the full history of v_t, so we can calculate the error for each timestep and plot it:

```
import plotly.graph_objects as go
import plotly.io as pio

pio.renderers.default = "png"
pio.kaleido.scope.default_scale = 2

error_history = np.max(v_history - v, axis=1)
go.Figure(
    data=[go.Scatter(y=error_history, mode="lines")],
    layout=dict(
        xaxis_title="$t$", yaxis_title=r"$\Vert v_t - v^\pi \Vert_{\infty}$"
    ),
)
```



4.2

We can use use the following theorem to obtain the number of iterations required: for an accuracy level of ε in estimating the optimal value function, we can run the value iteration algorithm for T iterations such that:

$$T \geq \frac{\log\left(\frac{\|\vec{v}_0 - \vec{v}^*\|_\infty}{\varepsilon}\right)}{\log \frac{1}{\gamma}}$$

which ensures that $\|v_T - v^*\|_\infty \leq \varepsilon$

```
v_0 = np.zeros(width**2)
```