ECE59500RL HW2

Robert (Cars) Chandler — chandl71@purdue.edu

Problem 1

1.1

Any matrix with only nonzero eigenvectors is necessarily invertible. So, we seek to prove that $I - \gamma P^\pi$ has only nonzero eigenvectors.

Using the properties of eigenvalues, if λ is an eigenvalue of P^π, then $1 - \gamma\lambda$ is an eigenvalue of $I - \gamma P^\pi$. So, if we can show that $\gamma\lambda \neq 1$ for all λ, then we can prove the matrix is invertible. Since we are in the discounted setting, we know that $0 < \gamma < 1$. So, if we can show that $\lambda \leq 1$, then we will have achieved our goal.

We know that P^π is a row-stochastic matrix, which is to say that each element must be in the interval $[0, 1]$ and the sum of the elements in each row must be 1. Given the definition of the eigenvalue and eigenvector:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

We begin by analyzing the RHS of the equation.

If we consider some $\lambda > 1$ to exist, then by definition:

$$\|\lambda\mathbf{x}\|_\infty = |\lambda|\|\mathbf{x}\|_\infty$$

and since $\lambda > 1$, then

$$\|\lambda\mathbf{x}\|_\infty = |\lambda|\|\mathbf{x}\|_\infty > \|\mathbf{x}\|_\infty$$

Now, looking at the LHS,

$$\|P^\pi\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_\infty$$

This is true because each element in the vector resulting from $P^\pi \mathbf{x}$ will be some linear combination of the components of $\mathbf{x}$ where the scalars are at least 0 and at most 1 and sum to 1. So, the greatest magnitude possible from this operation occurs in the case where a row of P^π has a 1 that aligns with the maximum magnitude of $\mathbf{x}$ and is zero everywhere else.

Now, we have proven that the LHS is less than or equal to $\|\mathbf{x}\|_\infty$ and that the RHS is greater than $\|\mathbf{x}\|_\infty$. But if these two sides of the equation are to be equal, then this is impossible. Therefore, it is impossible that $\lambda > 1$ for a row-stochastic matrix.

So if $\lambda \leq 1$ for all eigenvalues of P^π, then $1 - \gamma\lambda > 0$, because $0 < \gamma < 1$. Therefore, all eigenvalues of $I - \gamma P^\pi$ are nonzero, which means that $I - \gamma P^\pi$ must also be invertible.

1.2

Beginning with the Bellman consistency equation:

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_{a\sim\pi(s)}\left[R(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[v^\pi(s')\right]\right] \\ &= \sum_{a\in\mathcal{A}}\left[\mathbb{P}(a|s)\left(R(s,a) + \gamma\sum_{s'\in\mathcal{S}}\left[\mathbb{P}(s'|a,s)v^\pi(s')\right]\right)\right] \\ &= \sum_{a\in\mathcal{A}}\left[\mathbb{P}(a|s)R(s,a)\right] + \gamma\sum_{s'\in\mathcal{S}}\left[v^\pi(s')\sum_{a\in\mathcal{A}}\left[\mathbb{P}(a|s)\mathbb{P}(s'|a,s)\right]\right] \end{aligned}$$

We can define some new variables under the policy π:

$$R^\pi(s) = \sum_{a\in\mathcal{A}}\left[\mathbb{P}(a|s)R(s,a)\right]$$

$$P^\pi(s,s') = \sum_{a\in\mathcal{A}}\left[\mathbb{P}(a|s)\mathbb{P}(s'|a,s)\right]$$

These are just like before, except they are now a weighted average across all possible actions. We could define a function $\pi(a,s) = \mathbb{P}(a|s)$ and substitute this into the equations above as well.

So we have

$$v^\pi(s) = R^\pi(s) + \gamma\sum_{s'\in\mathcal{S}}\left[P^\pi(s,s')v^\pi(s')\right]$$

where $P^\pi(s,s') = \mathbb{P}(s'|s)$.

We can form matrices to represent these across all states:

$$\vec{v}^{\pi} = [v^{\pi}(s_1) \ ... \ v^{\pi}(s_n)]^T$$

$$\vec{R}^{\pi} = [R^{\pi}(s_1) \ ... \ R^{\pi}(s_n)]^T$$

$$P^{\pi} = \begin{bmatrix} P^{\pi}(s_1, s_1') & ... & P^{\pi}(s_1, s_n') \\ \vdots & \ddots & \vdots \\ P^{\pi}(s_n, s_1') & ... & P^{\pi}(s_n, s_n') \end{bmatrix}$$

Which allows us to write the equation in matrix form as

$$\begin{aligned} \vec{v}^{\pi} &= \vec{R}^{\pi} + \gamma P^{\pi} \vec{v}^{\pi} \\ \vec{v}^{\pi} - \gamma P^{\pi} \vec{v}^{\pi} &= \vec{R}^{\pi} \\ (I - \gamma P^{\pi}) \vec{v}^{\pi} &= \vec{R}^{\pi} \\ \vec{v}^{\pi} &= (I - \gamma P^{\pi})^{-1} \vec{R}^{\pi} \end{aligned}$$

1.3

Since the reward is stochastic now, we need to start with the definition of the value function and derive a modified Bellman consistency equation:

$$\begin{aligned} v^{\pi}(s) &= \mathbb{E}_{\pi,P,R} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s \right] \\ &= \mathbb{E}_{\pi,P,R} \left[R(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s \right] \\ &= \sum_{a \sim \pi} \left[\pi(a|s) \sum_{s',r} \left[p(s', r|s, a) \left(r(s, a) + \gamma v^{\pi}(s') \right) \right] \right], \quad \forall s \in \mathcal{S} \\ &= \sum_{a \sim \pi} \left[\pi(a|s) \mathbb{E}_r \left[r(s, a) + \mathbb{E}_{s'} \left[\gamma v^{\pi}(s') \right] \right] \right], \quad \forall s \in \mathcal{S} \end{aligned}$$

Our $R^{\pi}(s)$ is now defined as:

$$R^{\pi}(s) = \sum_{a \in \mathcal{A}, r \in \mathcal{R}} [\mathbb{P}(a|s) \mathbb{P}(r|a, s) r]$$

Where r is the value of the current reward in the summation. So, we sum over all possible actions as well as all possible rewards and take into account the probability of each action and

reward as defined by the stochastic reward function R and stochastic policy π to give us an expectation across all possible action and reward combinations for a given state.

From here, we can use the same logic from 1.2 to form our matrices, and the final result is still:

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

Problem 2

2.1

We wish to prove that

$$\left| \max_{x \in X} g_1(x) - \max_{x \in X} g_2(x) \right| \leq \max_{x \in X} |g_1(x) - g_2(x)|$$

It is true that

$$g_1(x) \leq |g_1(x) - g_2(x)| + g_2(x)$$

Since if $g_1(x) \geq g_2(x)$, equality holds above, and if $g_1(x) \leq g_2(x)$, strict inequality holds. It therefore also holds that

$$\max_{x \in X} g_1(x) \leq \max_{x \in X} \left(|g_1(x) - g_2(x)| + g_2(x) \right)$$

and by the triangle inequality (or at least the analagous inequality for the max operator)

$$\max_{x \in X} g_1(x) \leq \max_{x \in X} \left(|g_1(x) - g_2(x)| + g_2(x) \right) \leq \max_{x \in X} (|g_1(x) - g_2(x)|) + \max_{x \in X} g_2(x)$$

so then

$$\max_{x \in X} g_1(x) - \max_{x \in X} g_2(x) \leq \max_{x \in X} (|g_1(x) - g_2(x)|)$$

And if we repeat this process, switching the order of g_1 and g_2:

$$g_2(x) \leq |g_1(x) - g_2(x)| + g_1(x)$$

$$\dots$$

$$\max_{x \in X} g_2(x) - \max_{x \in X} g_1(x) \leq \max_{x \in X} (|g_1(x) - g_2(x)|)$$

And with the inequality proven for both orders of subtraction, we can combine the two conclusions by bringng in absolute value to the LHS:

$$\left|\max_{x\in X} g_1(x) - \max_{x\in X} g_2(x)\right| \leq \max_{x\in X} |g_1(x) - g_2(x)| \quad \blacksquare$$

2.2

We wish to prove that

$$\max_{x\in X, y\in Y} f(x, g(y)) \geq \max_{x\in X} f(x, \max_{y\in Y} g(y))$$

for two scalar-valued functions $f : X \times Z \to \mathbb{R}$ and $g : Y \to Z \subseteq \mathbb{R}$.

By definition,

$$\max_{x\in X, y\in Y} f(x, g(y)) \geq f(x', g(y')), \quad \forall x' \in X, y' \in Y$$

Define y^* as:

$$g(y^*) = \max_{y\in Y} g(y)$$

So

$$\max_{x\in X} f(x, \max_{y\in Y} g(y)) = \max_{x\in X} f(x, g(y^*))$$

Now define x^* as:

$$f(x^*, y^*) = \max_{x\in X} f(x, g(y^*))$$

But $x^* \in X$ and $y^* \in Y$ are just elements in their respective spaces, and we have already declared that

$$\max_{x\in X, y\in Y} f(x, g(y)) \geq f(x', g(y')), \quad \forall x' \in X, y' \in Y$$

for all elements in these spaces. So, if we just consider x^* as some $x' \in X$ and the same for y^*, then:

$$\max_{x\in X, y\in Y} f(x, g(y)) \geq f(x^*, g(y^*)) = \max_{x\in X} f(x, \max_{y\in Y} g(y)) \quad \blacksquare$$

Intuitively, the operator on the LHS sweeps a 2D plane in its search for the maximum, while the RHS first sweeps a 1D line Y and then sweeps a single line of the 2D plane in search for the maximum, so naturally there are more maxima that the LHS may find in comparison.

Problem 3

3.1

Given $v_{t=6}$, we can perform the following step in the value iteration

$$v_7(s) = \max_{a\in\mathcal{A}} \left[R(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}[v_6(s')]\right], \quad \forall s \in \mathcal{S}$$

We can implement this in Python:

```python
import numpy as np
from numpy.typing import NDArray

states = list("bc")
actions = list("xy")


def p(s, a):
    match (s, a):
        case ("b", "x"):
            return np.array([1.0, 0.0])
        case ("b", "y"):
            return np.array([0.2, 0.8])
        case ("c", "x"):
            return np.array([0.0, 1.0])
        case ("c", "y"):
            return np.array([0.6, 0.4])
        case _:
            raise Exception("Invalid state-action combination")


def reward(s, a):
    match (s, a):
        case ("b", "x"):
```

```
            return 0
        case ("b", "y"):
            return 0
        case ("c", "x"):
            return 1
        case ("c", "y"):
            return 1
        case _:
            raise Exception("Invalid state-action combination")


def opt_value(v_current: NDArray, n_iterations=1):
    v_t = v_current.copy()
    r_t = np.array([np.nan, np.nan])
    v_expected_t = np.array([np.nan, np.nan])
    optimal_actions = []

    for t in range(n_iterations):
        i_action = np.array(
            [
                np.argmax(
                    [reward(s, a) + np.sum(p(s, a) * v_t) for a in actions]
                )
                for s in states
            ]
        ).squeeze()

        r_t = np.array(
            [reward(s, actions[i_action[i]]) for i, s in enumerate(states)]
        )
        v_expected_t = np.array(
            [
                np.sum(p(s, actions[i_action[i]]) * v_t)
                for i, s in enumerate(states)
            ]
        )
        optimal_actions = [actions[i] for i in i_action]
        v_t = r_t + v_expected_t

    return v_t, r_t, v_expected_t, optimal_actions
```

```
v_opt_valiter_policy, r_opt, v_expected_opt, optimal_actions = opt_value(
    v_current=np.array([10, 5])
)
```

The value function at $t = 7$ will be:

$$v_7(b) = 10\gamma, \quad v_7(c) = 1 + 8\gamma$$

3.2

Since we are given v^{π_8}, we have already performed the policy evaluation step, so we just need to improve the policy using:

$$\pi_9(s) = \arg\max_{a \in \mathcal{A}} [R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[v^{\pi_8}(s')], \quad \forall s \in \mathcal{S}$$

Our Python implementation already keeps track of the action corresponding to the optimal values in the `optimal_actions` output:

```
v_opt_valiter_policy, r_opt, v_expected_opt, optimal_actions = opt_value(
    v_current=np.array([5, 15])
)
```

The improved policy at $t = 9$ will be:

$$\pi_9(b) = y, \quad \pi_9(c) = x$$

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 $s_{x,y} \in \mathcal{S}$, 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 our state-space 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 using the same x-y coordinate system described above:
>
> $$(0,0),(1,0),(2,0),(3,0),(4,0),(0,1),(1,1),...,(3,4),(4,4)$$

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, StrEnum, auto
import numpy as np
import itertools
from IPython.display import Markdown


class BoardSpace(StrEnum):
    NORMAL = "N"
    MOUNTAIN = "M"
    LIGHTNING = "L"
    TREASURE = "T"


class Action(StrEnum):
    UP = "U"
    RIGHT = "R"
```

```
    DOWN = "D"
    LEFT = "L"


width = 5

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

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

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


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] == BoardSpace.MOUNTAIN
    )


def get_trans_prob(x: int, y: int, a: Action):
    if x < 0 or x >= width or y < 0 or y >= width:
        raise RuntimeError("Invalid coordinates")
```

```
    if a not in Action:
        raise RuntimeError("Invalid action type")

    p_vec = np.zeros(width**2)
    i_state_1d = i_1d(x, y)

    if board[x, y] != BoardSpace.NORMAL:
        p_vec[i_state_1d] = 1
        return p_vec

    direction_coordinates = {
        Action.LEFT: [x - 1, y],
        Action.RIGHT: [x + 1, y],
        Action.UP: [x, y + 1],
        Action.DOWN: [x, y - 1],
    }

    for direction, (x_next, y_next) in direction_coordinates.items():
        prob = 0.85 if direction == a else 0.05
        if is_blocked(x_next, y_next):
            p_vec[i_state_1d] += prob
        else:
            p_vec[i_1d(x_next, y_next)] += prob

    if p_vec.sum() != 1:
        raise RuntimeError("Probability vector did not add to 1")

    return p_vec


# (0, 0), (1, 0), (2, 0) ... (3, 4), (4, 4)
each_state = [(x, y) for y in range(5) for x in range(5)]

p = np.zeros([width**2, width**2])

for x, y in each_state:
    i = i_1d(x, y)
    p[i] = get_trans_prob(x, y, policy[x, y])

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:

$$
\begin{aligned}
P^{\pi}(s_{0,0}|s_{0,0}, \pi(s_{0,0}) = \mathrm{U}) &= 0.1 \\
P^{\pi}(s_{1,0}|s_{0,0}, \pi(s_{0,0}) = \mathrm{U}) &= 0.05 \\
P^{\pi}(s_{0,1}|s_{0,0}, \pi(s_{0,0}) = \mathrm{U}) &= 0.85
\end{aligned}
$$

$$
\begin{aligned}
P^{\pi}(s_{0,0}|s_{1,0}, \pi(s_{1,0}) = \mathrm{R}) &= 0.05 \\
P^{\pi}(s_{1,0}|s_{1,0}, \pi(s_{1,0}) = \mathrm{R}) &= 0.05 \\
P^{\pi}(s_{2,0}|s_{1,0}, \pi(s_{1,0}) = \mathrm{R}) &= 0.85 \\
P^{\pi}(s_{1,1}|s_{1,0}, \pi(s_{1,0}) = \mathrm{R}) &= 0.05
\end{aligned}
$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$P^\pi(s_{1,4}|s_{0,4}, \pi(s_{0,4}) = \text{R}) = 0.85$$

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

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

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

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

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 == BoardSpace.LIGHTNING).squeeze())] = -1
r[i_1d(*np.argwhere(board == BoardSpace.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:

```
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)
```

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

$$v^\pi(s_{3,4}) = 16.9416$$
$$v^\pi(s_{4,4}) = 20$$

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:

$$T \geq \frac{\log\left(\frac{\|\vec{v}_0 - \vec{v}^\pi\|_\infty}{\varepsilon}\right)}{\log\frac{1}{\gamma}} \tag{2}$$

However, we cannot use the analytical solution of v^π, so we need to form some kind of bound on it instead and use that. We know that

$$v^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s\right], \quad \forall s \in \mathcal{S}$$

by definition, and we know that our reward function is bounded on the interval $[-1, 1]$. Therefore, we can say that $|R(s,a)| \leq 1$ so that

$$\|v^\pi\|_\infty \leq 1 \cdot \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

And since $v_0 = \mathbf{0}$, we can say that

$$\|\vec{v}_0 - \vec{v}^\pi\|_\infty \leq \|v^\pi\|_\infty \leq \frac{1}{1-\gamma}$$

So, finally:

$$\frac{\log\left(\frac{1}{\varepsilon(1-\gamma)}\right)}{\log\frac{1}{\gamma}}$$

will give us a conservative estimate of T which is more than (or equal to) the number of iterations actually required to obtain our desired error.

We can evaluate this result using Python, and then perform T iterations by implementing the iteration algorithm above.

```
from numpy.linalg import norm

epsilon = 0.01
v0 = np.zeros(width**2)
n_iterations = int(
    np.ceil(np.log(1 / (epsilon * (1 - gamma))) / np.log(1 / gamma))
)

v_t = v0
v_history = [v0]
for t in range(n_iterations):
    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_t):
    y, x = np.unravel_index(i, [width, width])
    value_text_iter.append(
        rf"v_{{T = {n_iterations}}} (s_{{ {x},{y} }}) &= {val:g} \\"
    )

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

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

$$
\begin{aligned}
v_{T=149}(s_{0,0}) &= 4.33141 \\
v_{T=149}(s_{1,0}) &= 2.63358 \\
v_{T=149}(s_{2,0}) &= 2.69873 \\
v_{T=149}(s_{3,0}) &= 2.87014 \\
v_{T=149}(s_{4,0}) &= 6.07083 \\
v_{T=149}(s_{0,1}) &= 4.69995 \\
v_{T=149}(s_{1,1}) &= 2.60848
\end{aligned}
$$

$$
\begin{aligned}
v_{T=149}(s_{2,1}) &= 0 \\
v_{T=149}(s_{3,1}) &= 0 \\
v_{T=149}(s_{4,1}) &= 6.63549 \\
v_{T=149}(s_{0,2}) &= 5.13614 \\
v_{T=149}(s_{1,2}) &= 2.84411 \\
v_{T=149}(s_{2,2}) &= 3.7889 \\
v_{T=149}(s_{3,2}) &= 5.47824 \\
v_{T=149}(s_{4,2}) &= 7.08005 \\
v_{T=149}(s_{0,3}) &= 5.61512 \\
v_{T=149}(s_{1,3}) &= 0 \\
v_{T=149}(s_{2,3}) &= -19.9904 \\
v_{T=149}(s_{3,3}) &= -14.2895 \\
v_{T=149}(s_{4,3}) &= 16.5873 \\
v_{T=149}(s_{0,4}) &= 12.0127 \\
v_{T=149}(s_{1,4}) &= 13.1333 \\
v_{T=149}(s_{2,4}) &= 14.0129 \\
v_{T=149}(s_{3,4}) &= 16.933 \\
v_{T=149}(s_{4,4}) &= 19.9904
\end{aligned}
$$

We can verify that our desired conditon holds:

```
max_error = norm(v_t - v, ord=np.inf)
```

$$
\|v_T - v^\pi\|_\infty = 0.009591 \leq 0.01
$$

4.1.c

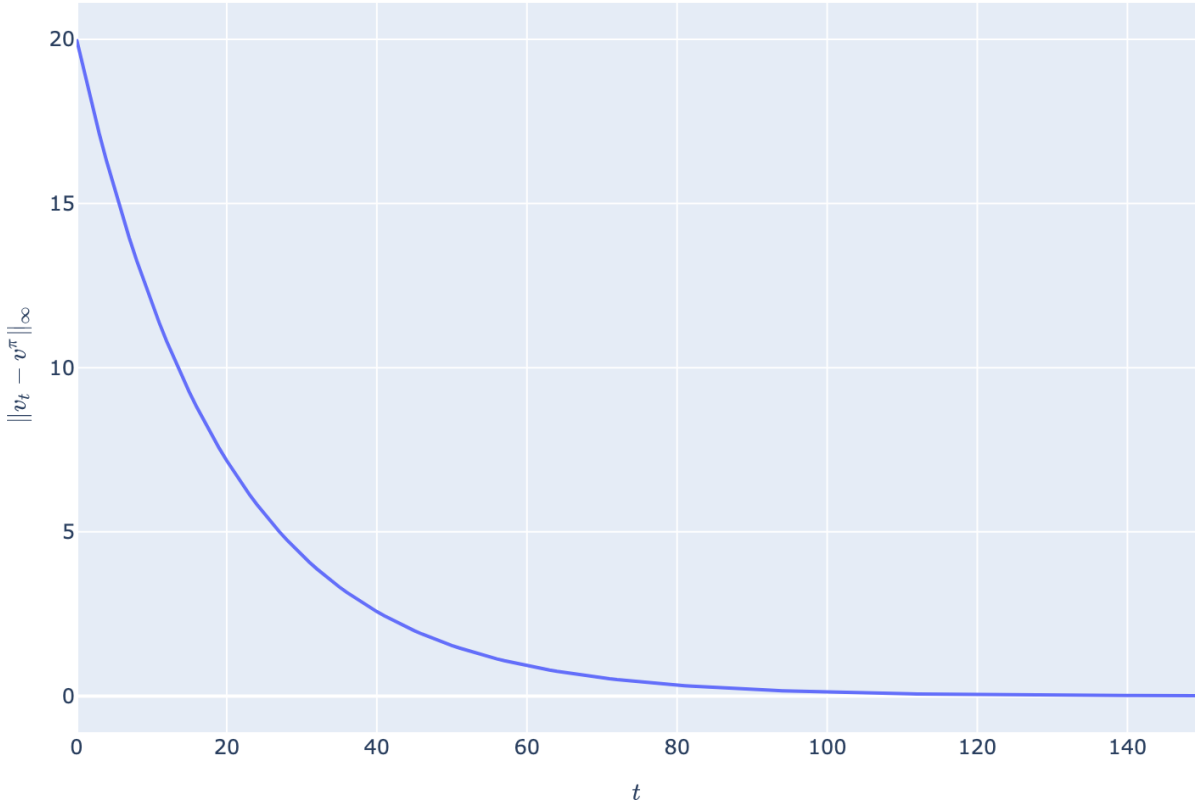
We kept track of the full history of v_t, so we can calculate the error for each timestep and plot it:

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

pio.renderers.default = "png"
```

```
pio.kaleido.scope.default_scale = 2

error_history = norm(v_history - v, ord=np.inf, 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

To perform value iteration, we initialize $\vec{v}_0 = \mathbf{0}$ and then for each iteration which increases t by one, we perform the operation:

$$v_{t+1}(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [v_t(s')] \right], \quad \forall s \in \mathcal{S}$$

We can use the following theorem to determine when we have completed a sufficient number of iterations:

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

This is the same as Equation 2 but with v^* instead of v^π. We can use the same logic as before to bound T since the only difference is that v^* now considers all possible policies, but all the value functions for all policies can still be bound by the reward function as before, such that:

$$\|v^*\|_\infty \leq 1 \cdot \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

Therefore, we can use the same $T = 149$ as before.

```
v0 = np.zeros(width**2)

v_t = v0

epsilon = 0.01

n_iterations = int(
    np.ceil(np.log(1 / (epsilon * (1 - gamma))) / np.log(1 / gamma))
)

for t in range(n_iterations):
    v_t = np.array(
        [
            np.max(
                [
                    r[i_1d(x, y)]
                    + gamma * np.sum(get_trans_prob(x, y, a) * v_t)
                    for a in Action
                ]
            )
            for x, y in each_state
        ]
    )

v_valiter = v_t
```

Now, with $\vec{v}_T$ determined, we need to find the policy corresponding to this value function, which is:

$$\pi_T(s) = \arg\max_{a \in \mathcal{A}} [R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[v_T(s')], \quad \forall s \in \mathcal{S}$$

```
policy_opt = np.full_like(policy, "")

actions_list = list(Action)

for x, y in each_state:
    i_max = np.argmax(
        [
            r[i_1d(x, y)]
            + gamma * np.sum(get_trans_prob(x, y, a) * v_valiter)
            for a in actions_list
        ]
    )

    policy_opt[x, y] = actions_list[i_max].value

# Account for different coordinate systems between numpy and our board
print(np.flipud(policy_opt.T))
```

```
[['R' 'R' 'R' 'R' 'U']
 ['U' 'U' 'U' 'R' 'U']
 ['U' 'R' 'R' 'R' 'U']
 ['U' 'D' 'U' 'U' 'U']
 ['R' 'R' 'R' 'R' 'U']]
```

The learned policy is printed above and its representation corresponds one-to-one with the shape and orientation of the original board given in the problem statement.

> **Note**
>
> The actions prescribed by this policy do not have any meaning for the special spaces where the user cannot move. The actions for these spaces are UP simply because that was the first action in the enumeration of actions.

Lastly, we can calculate the value function for this policy by re-calculating our P matrix and then using the analytical solution:

```
p_opt = np.array(
    [get_trans_prob(x, y, policy_opt[x, y]) for x, y in each_state],
)

v_opt_valiter_policy = (
    np.linalg.inv(np.identity(width**2) - gamma * p_opt) @ r
)

value_text = []
for val, (x, y) in zip(v_opt_valiter_policy, each_state):
    value_text.append(rf"v^{{\pi_T}}(s_{{ {x},{y} }}) &= {val:g} \\")

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

The $v^{\pi_T}(s)$ is shown for each state below:

$$
\begin{aligned}
v^{\pi_T}(s_{0,0}) &= 11.8594 \\
v^{\pi_T}(s_{1,0}) &= 12.6235 \\
v^{\pi_T}(s_{2,0}) &= 13.4931 \\
v^{\pi_T}(s_{3,0}) &= 14.3797 \\
v^{\pi_T}(s_{4,0}) &= 15.3223 \\
v^{\pi_T}(s_{0,1}) &= 11.3527 \\
v^{\pi_T}(s_{1,1}) &= 11.8921 \\
v^{\pi_T}(s_{2,1}) &= 0 \\
v^{\pi_T}(s_{3,1}) &= 0 \\
v^{\pi_T}(s_{4,1}) &= 16.3265 \\
v^{\pi_T}(s_{0,2}) &= 11.9941 \\
v^{\pi_T}(s_{1,2}) &= 12.5169 \\
v^{\pi_T}(s_{2,2}) &= 13.3594 \\
v^{\pi_T}(s_{3,2}) &= 16.1985 \\
v^{\pi_T}(s_{4,2}) &= 17.3965 \\
v^{\pi_T}(s_{0,3}) &= 12.7438 \\
v^{\pi_T}(s_{1,3}) &= 0 \\
v^{\pi_T}(s_{2,3}) &= -20 \\
v^{\pi_T}(s_{3,3}) &= 15.7238
\end{aligned}
$$

$$
\begin{aligned}
v^{\pi_T}(s_{4,3}) &= 18.607 \\
v^{\pi_T}(s_{0,4}) &= 13.577 \\
v^{\pi_T}(s_{1,4}) &= 14.4667 \\
v^{\pi_T}(s_{2,4}) &= 15.4148 \\
v^{\pi_T}(s_{3,4}) &= 18.5082 \\
v^{\pi_T}(s_{4,4}) &= 20
\end{aligned}
$$

4.3

To begin our policy iteration algorithm, we initialize the policy with a uniform random distribution over $\mathcal{A}$:

```
seed = 129831982479182749128498217 6
rng = np.random.default_rng(seed)

int_to_action = np.vectorize(lambda i: actions_list[i])

policy_0 = int_to_action(rng.integers(low=0, high=4, size=(width, width)))

print(policy_0)
```

```
[['R' 'R' 'L' 'R' 'D']
 ['U' 'R' 'L' 'L' 'D']
 ['D' 'L' 'R' 'R' 'U']
 ['U' 'L' 'R' 'L' 'R']
 ['R' 'D' 'R' 'D' 'D']]
```

The policy is shown above using the original board shape and orientation.

Now, for each iteration, we evaluate π_t by computing v^{π_t} and then we improve the policy by updating its action for each state so that it yields the maximum return according to:

$$
\pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} [R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[v^{\pi_t}(s')], \quad \forall s \in \mathcal{S}
$$

To determine how many iterations are necessary, we use the following theorem:

$$
\|v^{\pi_T} - v^*\|_\infty \leq \gamma^T \|v^{\pi_0} - v^*\|_\infty
$$

We can substitute our desired accuracy ε in and solve for T:

$$T \geq \frac{\log\left(\frac{\|v^{\pi_0} - v^*\|_\infty}{\varepsilon}\right)}{\log(\frac{1}{\gamma})}$$

This is the same form we are used to seeing.

Using the same logic as before, we can bound v^* by:

$$\|v^*\|_\infty \leq 1 \cdot \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

Therefore:

$$\|v^{\pi_0} - v^*\|_\infty \leq \|v^{\pi_0}\|_\infty + \frac{1}{1-\gamma}$$

So we can get a conservative upper bound on T by substituting this into the inequality solved for T:

$$T \geq \frac{\log\left(\frac{\|v^{\pi_0}\|_\infty + (1-\gamma)^{-1}}{\varepsilon}\right)}{\log\left(\frac{1}{\gamma}\right)}$$

Depending on how good our randomly generated initial policy is, T may vary.

We solve for T and perform our iterations:

```
def evaluate_policy(policy):
    p_policy = np.array(
        [get_trans_prob(x, y, policy[x, y]) for x, y in each_state],
    )

    v_policy = np.linalg.inv(np.identity(width**2) - gamma * p_policy) @ r

    return v_policy


def improve_policy(policy, v_policy):
    for x, y in each_state:
        i_max = np.argmax(
            [
```

```
                r[i_1d(x, y)]
                + gamma * np.sum(get_trans_prob(x, y, a) * v_policy)
                for a in actions_list
            ]
        )
        policy[x, y] = actions_list[i_max]


policy_t = policy_0

v_policy_0 = evaluate_policy(policy_0)

n_iterations = int(
    np.ceil(
        np.log((norm(v_policy_0, np.inf) + (1 / (1 - gamma))) / epsilon)
        / np.log(1 / gamma)
    )
)

for i in range(n_iterations):
    v_policy_t = evaluate_policy(policy_t)

    improve_policy(policy_t, v_policy_t)

policy_opt = policy_t

print(np.flipud(policy_opt.T))
```

```
[['R' 'R' 'R' 'R' 'U']
 ['U' 'U' 'U' 'R' 'U']
 ['U' 'R' 'R' 'R' 'U']
 ['U' 'D' 'U' 'U' 'U']
 ['R' 'R' 'R' 'R' 'U']]
```

The learned policy is shown above. The number of iterations was $T = 162$. We can evaluate the policy to view the value function:

```
v_policy_opt = evaluate_policy(policy_opt)
value_text = []
for val, (x, y) in zip(v_policy_opt, each_state):
    value_text.append(rf"v^{{\pi_T}}(s_{{ {x},{y} }}) &= {val:g} \\")
```

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

The value $v^{\pi_T}(s)$ is shown for each state below:

$$
\begin{aligned}
v^{\pi_T}(s_{0,0}) &= 11.8594 \\
v^{\pi_T}(s_{1,0}) &= 12.6235 \\
v^{\pi_T}(s_{2,0}) &= 13.4931 \\
v^{\pi_T}(s_{3,0}) &= 14.3797 \\
v^{\pi_T}(s_{4,0}) &= 15.3223 \\
v^{\pi_T}(s_{0,1}) &= 11.3527 \\
v^{\pi_T}(s_{1,1}) &= 11.8921 \\
v^{\pi_T}(s_{2,1}) &= 0 \\
v^{\pi_T}(s_{3,1}) &= 0 \\
v^{\pi_T}(s_{4,1}) &= 16.3265 \\
v^{\pi_T}(s_{0,2}) &= 11.9941 \\
v^{\pi_T}(s_{1,2}) &= 12.5169 \\
v^{\pi_T}(s_{2,2}) &= 13.3594 \\
v^{\pi_T}(s_{3,2}) &= 16.1985 \\
v^{\pi_T}(s_{4,2}) &= 17.3965 \\
v^{\pi_T}(s_{0,3}) &= 12.7438 \\
v^{\pi_T}(s_{1,3}) &= 0 \\
v^{\pi_T}(s_{2,3}) &= -20 \\
v^{\pi_T}(s_{3,3}) &= 15.7238 \\
v^{\pi_T}(s_{4,3}) &= 18.607 \\
v^{\pi_T}(s_{0,4}) &= 13.577 \\
v^{\pi_T}(s_{1,4}) &= 14.4667 \\
v^{\pi_T}(s_{2,4}) &= 15.4148 \\
v^{\pi_T}(s_{3,4}) &= 18.5082 \\
v^{\pi_T}(s_{4,4}) &= 20
\end{aligned}
$$