

# HOMEWORK 7

EE 578B - Winter 2021

Due Date: Wednesday, Mar 3rd, 2021 @ 11:59 PM

```
In [2]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
import cvxpy as cp
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

```
In [3]: import warnings
warnings.simplefilter('ignore')
```

Consider the Markov Decision Process with the following graph and action structure. (SEE PDF)

## 1. Transition Kernel Constraints



(PTS:0-2)

Write down the incidence matrices for the graph.

$$E_i \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{E}|}, \quad E_o \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{E}|}, \quad P \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}, \quad A \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}, \quad W \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{A}|}$$

The incidence matrices can be written as follows

$$E_i = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$E_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

■ (PTS:0-2)

For the incidence matrices given above show the following identities

$$\begin{aligned} \mathbf{1}^T E_i &= \mathbf{1}^T E_o = \mathbf{1}^T \\ \mathbf{1}^T A &= \mathbf{1}^T P = \mathbf{1}^T \\ \mathbf{1}^T W &= \mathbf{1}^T \\ E_i W &= P, \quad E_o W = A \end{aligned}$$

where the dimension of each  $\mathbf{1}$  is determined by context.

For the first identity,

$$\begin{aligned} \mathbf{1}^T E_i &= [1 \quad 1 \quad 1 \quad 1] * \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1] \\ \mathbf{1}^T E_o &= [1 \quad 1 \quad 1 \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1] \end{aligned}$$

Thus, we see that  $\mathbf{1}^T E_i = \mathbf{1}^T E_o = \mathbf{1}^T$ .

For the second identity,

$$\begin{aligned} \mathbf{1}^T A &= [1 \quad 1 \quad 1 \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1] \\ \mathbf{1}^T P &= [1 \quad 1 \quad 1 \quad 1] * \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1] \end{aligned}$$

Thus, we can see that  $\mathbf{1}^T A = \mathbf{1}^T P = \mathbf{1}^T$ .

For the third identity,

$$1^T W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Thus, we can see that  $1^T W = 1^T$ .

For our last identity,

$$E_i W = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} = P$$

$$E_o W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = A$$

Thus, we can see that  $E_i W = P$  and  $E_o W = A$ .



(PTS:0-2)

Consider two policies with the following actions chosen from each state

**Policy 1:** State 1: Action 1, State 2: Action 2,  
State 3: Action 4, State 4: Action 6

**Policy 2:** State 1: Action 1, State 2: Action 2,  
State 3: 50\% Action 3, State 4: 50\% Action 5  
50\% Action 4, 50\% Action 6

Write each policy in matrix form  $\Pi \in \mathbb{R}^{6 \times 4}$ . Compute the corresponding Markov matrix  $M = P\Pi$ . Also show that  $A\Pi = I$  for each policy.

The policy matrices  $\Pi$  can be written as follows

$$\Pi_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Pi_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$$

The corresponding Markov matrices are computed as follows,

$$M_1 = P\Pi_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 0 & 1 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 \\ 1 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}$$

$$M_2 = P\Pi_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 0 & 1 & 0 & 0.75 \\ 0 & 0 & 0.25 & 0 \\ 1 & 0 & 0 & 0.25 \\ 0 & 0 & 0.75 & 0 \end{bmatrix}$$

Proving that  $A\Pi = I$  for each policy,

$$A\Pi_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I$$

$$A\Pi_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I$$

(PTS:0-4)

The stationary (state) distribution associated with each Markov chain is the solution to the equation  $\rho = M\rho$ . Compute this stationary distribution by finding the eigenvector with eigenvalue 1. (You can use the function `eig` in Matlab or `numpy.linalg.eig` in Python.). Make sure to scale the eigenvector so that it is an appropriate probability distribution that sums to 1 and has all positive values. Compute the corresponding action distribution  $y$  as  $y = \Pi\rho$ .

In [4]:

```
M_1 = np.matrix([[0, 1, 0, 0.5], [0, 0, 0.5, 0], [1, 0, 0, 0.5], [0, 0, 0.5, 0]])
M_2 = np.matrix([[0, 1, 0, 0.75], [0, 0, 0.25, 0], [1, 0, 0, 0.25], [0, 0, 0.75, 0]])

w, v = np.linalg.eig(M_1)
print(np.round_(w, decimals=1), '\n', np.round_(v, decimals=1))

w, v = np.linalg.eig(M_2)
print(np.round_(w, decimals=1), '\n', np.round_(v, decimals=1))

[-0.5+0.7j -0.5-0.7j  1. +0.j   0. +0.j ]
[[-0.2+0.6j -0.2-0.6j  0.5+0.j  -0.4+0.j ]
 [-0.2-0.3j -0.2+0.3j  0.3+0.j  -0.4+0.j ]
 [ 0.6+0.j   0.6-0.j   0.7+0.j   0. +0.j ]
 [-0.2-0.3j -0.2+0.3j  0.3+0.j   0.8+0.j ]]
[-0.5+0.8j -0.5-0.8j  1. +0.j   0. +0.j ]
[[ 0.6+0.j   0.6-0.j   0.5+0.j  -0.2+0.j ]
 [-0.1+0.1j -0.1-0.1j  0.2+0.j  -0.6+0.j ]
 [-0.2-0.6j -0.2+0.6j  0.7+0.j   0. +0.j ]
 [-0.3+0.4j -0.3-0.4j  0.5+0.j   0.8+0.j ]]
```

The stationary distribution as identified using `numpy.linalg.eig` (as coded above) results in,

$$\rho_1 = [0.32 \quad 0.32 \quad 0.36 \quad 0]^T$$

$$\rho_2 = [0.18 \quad 0.18 \quad 0.64 \quad 0]^T$$

The corresponding action distribution  $y$  is calculated as,

$$y_1 = \Pi_1 \rho_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.32 \\ 0.32 \\ 0.36 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix}$$

$$y_2 = \Pi_2 \rho_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.18 \\ 0.18 \\ 0.64 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.18 \\ 0.32 \\ 0.32 \\ 0 \\ 0 \end{bmatrix}$$

(PTS:0-2)

Show that each  $y$  from the previous part satisfies  $Py = Ay$  and  $\mathbf{1}^T y = 1$ . Compute the corresponding edge mass vector for each  $x = Wy$ . Show that  $x$  is in the nullspace of  $E = E_i - E_o$ .

For each  $y$ , we can show that  $Py = Ay$  and  $\mathbf{1}^T y = 1$ ,

$$Py_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.18 \\ 0.32 \\ 0.18 \end{bmatrix}$$

$$Ay_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.32 \\ 0.36 \\ 0 \end{bmatrix}$$

$$Py_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.16 \\ 0.18 \\ 0.48 \end{bmatrix}$$

$$Ay_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0.18 \\ 0.18 \\ 0.32 \\ 0.32 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.18 \\ 0.64 \\ 0 \end{bmatrix}$$

$$\mathbf{1}^T y_1 = [1 \ 1 \ 1 \ 1 \ 1 \ 1] * \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix} = 1$$

$$\mathbf{1}^T y_2 = [1 \ 1 \ 1 \ 1 \ 1 \ 1] * \begin{bmatrix} 0.18 \\ 0.18 \\ 0.32 \\ 0.32 \\ 0 \\ 0 \end{bmatrix} = 1$$

The corresponding edge mass vector is calculated as,

$$x_1 = Wy_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.36 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.32 \\ 0 \\ 0.18 \\ 0 \\ 0 \end{bmatrix}$$

$$x_2 = Wy_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.18 \\ 0.18 \\ 0.48 \\ 0.16 \\ 0 \\ 0 \end{bmatrix}$$

## Infinite Horizon, Average Reward

Consider the following optimization problem for finding the optimal steady-state action distribution  $y \in \mathbb{R}^{|\mathcal{A}|}$

$$\begin{aligned} \max_y \quad & r^T y \\ \text{s.t.} \quad & Py = Ay, \mathbf{1}^T y = 1, y \geq 0 \end{aligned} \tag{1}$$

for reward vector  $r \in \mathbb{R}^{|\mathcal{A}|}$ .

### ► (PTS:0-2)

Write the dual optimization problem with dual variables  $\lambda \in \mathbb{R}$  associated with the constraint  $\mathbf{1}^T y = 1$ ,  $v \in \mathbb{R}^{|\mathcal{S}|}$  associated with constraint  $Py = Ay$ ,  $\mu \in \mathbb{R}_+^{|\mathcal{A}|}$  associated with the constraint  $y \geq 0$ .

We can write the dual optimization problem as,

$$\begin{aligned} \max_{v, \lambda, \mu} \quad & \lambda \\ \text{s.t.} \quad & \lambda \mathbf{1}^T + v^T A = r^T + v^T P + \mu^T, \mu \geq 0 \end{aligned}$$

### ► (PTS:0-2)

The KKT conditions at optimum (for either the primal or dual problem) are given by

$$\begin{aligned} r^T - \lambda \mathbf{1}^T + v^T (P - A) + \mu^T &= 0, \quad \mu \geq 0 \\ Py - Ay &= 0, \quad \mathbf{1}^T y = 1, \quad y \geq 0 \\ \mu^T y &= 0 \end{aligned}$$

Use these conditions to show that  $\lambda$  is an upper bound on the primal objective  $r^T y$  for any feasible  $y$ . What does  $\mu^T y$  represent for a specific  $y$ ? What does the condition  $\mu^T y = 0$  imply about the

optimal  $y$ ?

We can use our first condition and solve for  $r^T$  such that

$$r^T = \lambda 1^T - v^T(P - A) - \mu^T$$

Substituting this relationship into our relationship  $r^T y$  we find,

$$r^T y = (\lambda 1^T - v^T(P - A) - \mu^T) y$$

$$r^T y = \lambda 1^T y - v^T(Py - Ay) - \mu^T y$$

From our other KKT conditions, we can see that  $Py - Ay = 0$  and  $\mu^T y = 0$ , thus,

$$r^T y = \lambda y$$

As we can see from this relationship,  $\lambda$  must be upper bound of our primal objective.

$u^T y$  represents that when optimized, either our inefficiency for an action is zero or the mass flow for an action is zero. This implies that at the optimal  $y$  that the inefficiency is zero.

#### ■ (PTS:0-4)

Use cvx or cvxpy to solve the above optimization problem for the transition kernel given initially and each reward vector

$$r^T = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$
$$r^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

What is the optimal joint distribution  $y$  in each case? What is the expected average reward  $r^T y$  in each case?

The solutions to the above optimization problem are given as outputs from the code below. In addition, each expected average reward  $r^T y$  is printed out as well.

In [5]:

```
# Establish our values for input parameters
P = np.matrix([[0, 1, 0, 0, 1, 0.5], [0, 0, 0, 0.5, 0, 0], [1, 0, 0, 0, 0, 0.5], [0, 0,
A = np.matrix([[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 1, 0, 0], [0, 0, 0, 0,
r_T = np.matrix([1, 2, 3, 4, 5, 6]) # Define r_T as reward vector
one_T = np.matrix([1, 1, 1, 1, 1, 1])

# Define and solve the CVXPY problem.
y = cp.Variable(shape=(6, 1))
prob = cp.Problem(cp.Maximize(r_T @ y), [P @ y == A @ y, one_T @ y == 1, y >= 0])
prob.solve()

# Print result.
print('For r =', r_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution y is:")
print(np.round_(y.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob

y1 = y.value
```



```
print('Expected Average Reward:', r_T @ y.value)
```

For  $r = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$   
The optimal value is 3.8

A solution  $y$  is:

```
[[0.2]
 [0. ]
 [0.4]
 [0. ]
 [0. ]
 [0.4]]
```

A dual solution is:

```
[[ 1.4]
 [ 2.1]
 [-1.4]
 [-2.2]]
```

and

```
[[3.8]]
```

Expected Average Reward:  $\begin{bmatrix} 3.8 \end{bmatrix}$

In [6]:

```
r_T = np.matrix([1, 1, 1, 1, 1, 1]) # Define r_T as reward vector

# Define and solve the CVXPY problem.
y = cp.Variable(shape=(6, 1))
prob = cp.Problem(cp.Maximize(r_T @ y), [P @ y == A @ y, one_T @ y == 1, y >= 0])
prob.solve()

# Print result.
print('For r =', r_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution y is:")
print(np.round_(y.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob

y2 = y.value

print('Expected Average Reward:', r_T @ y.value)
```

For  $r = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$   
The optimal value is 1.0

A solution  $y$  is:

```
[[0.3]
 [0.1]
 [0.2]
 [0.2]
 [0.1]
 [0.2]]
```

A dual solution is:

```
[[ -0.]
 [ -0.]
 [ -0.]
 [ -0.]]
```

and

```
[[1.]]
```

Expected Average Reward:  $\begin{bmatrix} 1. \end{bmatrix}$

## ▀ (PTS:0-2)

What is the steady-state state distribution associated with each solution  $\rho = Ay$ ?

What is the optimal policy associated with  $y$ ? Use the formula

$$(\pi_s)_a = \frac{y_a}{\rho_s} = \frac{y_a}{\sum_{a \in \mathcal{A}_s} y_a}$$

You could also put the policy in matrix form using the formula

$$\Pi = \text{diag}(y)A^T \text{diag}(\rho)^{-1}$$

The steady-state distributions and policies are outputted to console using the code below.

In [7]:

```
A = np.matrix([[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]])

rho_1 = A @ y1
rho_2 = A @ y2

print('Rho 1:\n', np.round_(rho_1, decimals=1))
print('Rho 2:\n', np.round_(rho_2, decimals=1))

Pi_1 = np.diag(np.squeeze(y1)) @ A.T @ np.linalg.inv(np.diag(np.squeeze(np.asarray(rho_1))))
print('Pi 1:\n', np.round_(Pi_1, decimals=1))

Pi_2 = np.diag(np.squeeze(y2)) @ A.T @ np.linalg.inv(np.diag(np.squeeze(np.asarray(rho_2))))
print('Pi 2:\n', np.round_(Pi_2, decimals=2))
```

```
Rho 1:
[[0.2]
 [0. ]
 [0.4]
 [0.4]]
Rho 2:
[[0.3]
 [0.1]
 [0.4]
 [0.3]]
Pi 1:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]]
Pi 2:
[[1.  0.  0.  0. ]
 [0.  1.  0.  0. ]
 [0.  0.  0.49 0. ]
 [0.  0.  0.51 0. ]
 [0.  0.  0.  0.42]
 [0.  0.  0.  0.58]]
```

## ▀ (PTS:0-2) Now suppose you apply the policy

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0.2 \\ 0 & 0 & 0 & 0.8 \end{bmatrix}$$

What reward do you achieve in each case? (Hint: compute  $\rho$  such that  $\rho = P\Pi\rho$  and then  $y$  using  $y = \Pi\rho$ .) How much does this reward differ from the optimal average reward? How does this difference relate to the quantity  $\mu^T y$  where  $\mu$  is the optimal dual variable?

The reward and average reward are calculated below using the code. We can see that our rewards are smaller when compared to our previously calculated values.

```
In [8]: Pi_new = np.matrix([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0.2, 0], [0, 0, 0.8, 0], [0, 0,
w, v = np.linalg.eig(P @ Pi_new)
#print(np.round_(w, decimals=1), '\n', np.round_(v, decimals=1))

rho_new = np.real(v[2, :])
y_new = Pi_new @ rho_new.T
print('Reward:\n', np.round_(y_new, decimals=1))

r_T = np.matrix([1, 2, 3, 4, 5, 6]) # Define r_T as reward vector
print('New Average Reward (Case 1):\n', r_T @ y_new)
r_T = np.matrix([1, 1, 1, 1, 1, 1]) # Define r_T as reward vector
print('New Average Reward (Case 2):\n', r_T @ y_new)
```

```
Reward:
[[-0.6]
 [-0.6]
 [ 0.1]
 [ 0.6]
 [-0. ]
 [-0. ]]
New Average Reward (Case 1):
[[0.79264503]]
New Average Reward (Case 2):
[[-0.53027565]]
```

### 3. Finite Horizon, Total Reward

Consider the following optimization problem for finding the optimal finite horizon policy.

$$\begin{aligned} \max_{y(t), t \in \mathcal{T}} \quad & \sum_{t=0}^{T-1} r(t)^T y(t) + g^T A y(T) \\ \text{s.t.} \quad & A y(0) = \rho(0), \quad y(0) \geq 0 \\ & A y(t+1) = P y(t), \quad y(t+1) \geq 0, \quad t \in \mathcal{T} \end{aligned} \tag{2}$$

where  $\mathcal{T} = \{0, \dots, T-1\}$ ,  $\rho(0) \in \mathbb{R}^{|\mathcal{S}|}$  is a given initial state distribution, and  $g \in \mathbb{R}^{|\mathcal{S}|}$  is a final cost on each of the states.



(PTS:0-4)

Write the dual optimization problem with dual variables  $v(0) \in \mathbb{R}^{|\mathcal{S}|}$  associated with the constraint  $Ay(0) = \rho(0)$ ,  $v(t+1) \in \mathbb{R}^{|\mathcal{S}|}$  associated with constraint  $Py(t) = Ay(t+1)$ , and  $\mu(t) \in \mathbb{R}_+^{|\mathcal{A}|}$  associated with the constraint  $y(t) \geq 0$ .

We can write the dual optimization problem such that,

$$\begin{aligned} \min_{v, \mu} \quad & v(0)^T \rho(0) \\ \text{s.t.} \quad & v(T)^T A = g^T A + \mu(T)^T, \mu(T) \geq 0 \\ & v(t)^T A = r(t)^T + v(t+1)^T P + \mu(t)^T, \mu(t) \geq 0 \end{aligned}$$

where  $t = \{0, \dots, T-1\}$ .

#### ■ (PTS:0-4)

The KKT optimality conditions for the primal and dual optimization problems are given by

$$\begin{aligned} g^T A - v(T)^T A + \mu(T)^T &= 0, \quad \mu(T) \geq 0 \\ r(t)^T + v(t+1)^T P - v(t)^T A + \mu(t)^T &= 0, \quad \mu(t) \geq 0, \quad t \in \mathcal{T} \\ Ay(0) &= \rho(0), \quad y(0) \geq 0 \\ Ay(t+1) &= Py(t), \quad y(t+1) \geq 0, \quad t \in \mathcal{T} \\ \mu(t)^T y(t) &= 0, \quad t \in \mathcal{T}, t = T \end{aligned}$$

Use these conditions to show that  $v(0)^T \rho(0)$  is an upper bound on the primal objective  $\sum_t r(t)^T y(t) + g^T Ay(T)$  for any feasible  $y(t)$  that satisfies the mass flow equations. What does  $\sum_t \mu(t)^T y(t)$  represent for a specific mass flow  $y(t), t \in \mathcal{T}$ .

Starting with our first two conditions, we can solve for  $r(t)^T$  and  $g^T A$  respectively such that,

$$\begin{aligned} g^T A &= v(T)^T A - \mu(T)^T \\ r(t)^T &= -v(t+1)^T P + v(t)^T A - \mu(t)^T \end{aligned}$$

We can now substitute these two relationships into our primal objective such that,

$$\begin{aligned} & \sum_{t=0}^{T-1} (-v(t+1)^T P + v(t)^T A - \mu(t)^T) y(t) + (v(T)^T A - \mu(T)^T) y(T) \\ & \sum_{t=0}^{T-1} -v(t+1)^T Py(t) + v(t)^T Ay(t) - \mu(t)^T y(t) + v(T)^T Ay(T) - \mu(T)^T y(T) \end{aligned}$$

Substituting some of our other conditions we see that,

$$\sum_{t=0}^{T-1} -v(t+1)^T Ay(t+1) + v(t)^T Ay(t) + v(T)^T Ay(T) - \mu(T)^T y(T)$$

As we can see, when we apply the summation most of our terms will cancel out expect for our first time stamp such,

$$v(0)^T Ay(0) = v(0)^T \rho(0)$$

The summation of  $\mu$  and  $y$  represents that at any given time step either the inefficiency is zero or the mass flow at a given  $y$  is zero.



#### (PTS:0-4)

Use cvx or cvxpy to solve the above optimization problem for the MDP given initially with the following rewards

$$r(t)^T = [2 \quad 1 \quad 2 \quad 1 \quad 2 \quad 1] \text{ for } t \in \mathcal{T}, \quad g^T = [1 \quad 1 \quad 1 \quad 1]$$

for ten time steps  $T = 10$  and initial distribution  $\rho(0) = [0.25 \quad 0.25 \quad 0.25 \quad 0.25]^T$

What is the optimal action distribution  $y(t)$  at each time step? What is the expected total reward  $\sum_t r(t)^T y(t)$ ?

In [33]:

```
g_T = np.matrix([1, 1, 1, 1])
r_T = np.matrix([2, 1, 2, 1, 2, 1])
rho0 = np.matrix([0.25, 0.25, 0.25, 0.25])

# Define and solve the CVXPY problem.
T = 10
y = cp.Variable(shape=(6, T))

obj = r_T @ y + g_T @ A @ y[:, T-1]
constraints = []
print(A.shape, y[:, 0].shape, rho0.shape)
constraints.append(A@y[:, 0] == rho0)
constraints.append(y[:, 0] >= 0)
for t in range(T):
    constraints.append(A@y[:, t+1] == P@y[:, t])
    constraints.append(y[:, t+1] >= 0)

prob = cp.Problem(cp.Maximize(obj), constraints)
prob.solve()

# Print result.
print('For r =', r_T)
print("The optimal value is", np.round_(prob.value, decimals = 1))
print("\nA solution y is:")
print(np.round_(y.value, decimals=1))
print("\nA dual solution is: ")
print(np.round_(prob.constraints[0].dual_value, decimals=1), '\nand \n', np.round_(prob

print('Expected Average Reward:', r_T @ y.value)
```

(4, 6) (6,) (4, 1)

-----  
ValueError Traceback (most recent call last)

<ipython-input-33-065536386ca8> in <module>

```
11 constraints = []
12 print(A.shape, y[:, 0].shape, rho0.shape)
----> 13 constraints.append(A@y[:, 0] == rho0)
14 constraints.append(y[:, 0] >= 0)
15 for t in range(T):
```

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in cast\_op(self, other)

```
45 """
46 other = self.cast_to_const(other)
```

```

---> 47         return binary_op(self, other)
      48     return cast_op
      49

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in __eq__(self, other)
      641     """Equality : Creates a constraint ``self == other``.
      642     """
--> 643     return Equality(self, other)
      644
      645     @_cast_other

~\Anaconda3\lib\site-packages\cvxpy\constraints\zero.py in __init__(self, lhs, rhs, constr_id)
      100     """
      101     def __init__(self, lhs, rhs, constr_id=None):
--> 102         self._expr = lhs - rhs
      103         super(Equality, self).__init__([lhs, rhs], constr_id)
      104

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in cast_op(self, other)
      45     """
      46     other = self.cast_to_const(other)
---> 47     return binary_op(self, other)
      48     return cast_op
      49

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in __sub__(self, other)
      514     """Expression : The difference of two expressions.
      515     """
--> 516     return self + -other
      517
      518     @_cast_other

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in cast_op(self, other)
      45     """
      46     other = self.cast_to_const(other)
---> 47     return binary_op(self, other)
      48     return cast_op
      49

~\Anaconda3\lib\site-packages\cvxpy\expressions\expression.py in __add__(self, other)
      500     return self
      501     self, other = self.broadcast(self, other)
--> 502     return cvxtypes.add_expr()([self, other])
      503
      504     @_cast_other

~\Anaconda3\lib\site-packages\cvxpy\atoms\affine\add_expr.py in __init__(self, arg_groups)
      31     # For efficiency group args as sums.
      32     self._arg_groups = arg_groups
---> 33     super(AddExpression, self).__init__(*arg_groups)
      34     self.args = []
      35     for group in arg_groups:

~\Anaconda3\lib\site-packages\cvxpy\atoms\atom.py in __init__(self, *args)
      44     self.args = [Atom.cast_to_const(arg) for arg in args]
      45     self.validate_arguments()
---> 46     self._shape = self.shape_from_args()
      47     if len(self._shape) > 2:
      48         raise ValueError("Atoms must be at most 2D.")

~\Anaconda3\lib\site-packages\cvxpy\atoms\affine\add_expr.py in shape_from_args(self)
      39     """Returns the (row, col) shape of the expression.
      40     """

```

```

---> 41         return u.shape.sum_shapes([arg.shape for arg in self.args])
      42
      43     def expand_args(self, expr):

~\Anaconda3\lib\site-packages\cvxpy\utilities\shape.py in sum_shapes(shapes)
      47         raise ValueError(
      48             "Cannot broadcast dimensions " +
---> 49             len(shapes)*" %s" % tuple(shapes))
      50
      51     longer = shape if len(shape) >= len(t) else t

```

**ValueError:** Cannot broadcast dimensions (4,) (4, 1)

#### (PTS:0-4)

What is the policy  $\Pi(t)$  chosen at each time step? Use the formula

$$(\pi_s)_a(t) = \frac{y_a(t)}{\rho_s(t)} = \frac{y_a(t)}{\sum_{a \in \mathcal{A}_s} y_a(t)}$$

where  $\rho(t) = Ay(t)$ .

In [ ]:



(PTS:0-4) Now suppose you apply the policy

$$\Pi(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0.2 \\ 0 & 0 & 0 & 0.8 \end{bmatrix}$$

at each time step. Start by computing  $y(0) = \Pi(0)\rho(0)$ .  $\rho(t)$  is then given by  $Py(0) = \rho(1)$ . Use  $\rho(1)$  to compute  $y(1) = \Pi(1)\rho(1)$ , etc. What total reward do you achieve? What is the quantity  $\sum_t \mu(t)^T y(t)$ ? How does this relate the total reward to the optimal total reward?

In [ ]: