ECE276B-HW1

Zhenyu Wu | PID: A69030822

April 18, 2025

1 Q1

1.1 Q1.1

Below is my hand-written solution and derivation process.

1.

$$
\begin{array}{cc}
 & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\
\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} &
\begin{bmatrix}
0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0.3 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.7 & 0 & 0 & 0.3 \\
0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.6 & 0.4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6 \\
0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6
\end{bmatrix}
\end{array}
$$

Figure 1: Q1.1 transition matrix

1.2 Q1.2

To find the stationary distribution, we need to find the left eigenvector of the P martix that satisfy $w^T P = 1 \cdot w^T$, which is equivalent to find the right eigenvector of P^T. Below is my hand-written solution and derivation process. For solving the linear equations, I use the python sympy package to do the elimination (I attached my code at the end of pdf).

2. $w^T P = 1 \cdot w^T$

$$P^T w = 1 \cdot w \Rightarrow \begin{matrix} w_6 = \frac{4}{3} w_8 \\ w_7 = \frac{2}{3} w_8 \end{matrix}$$

$$(P^T - I)w = 0$$

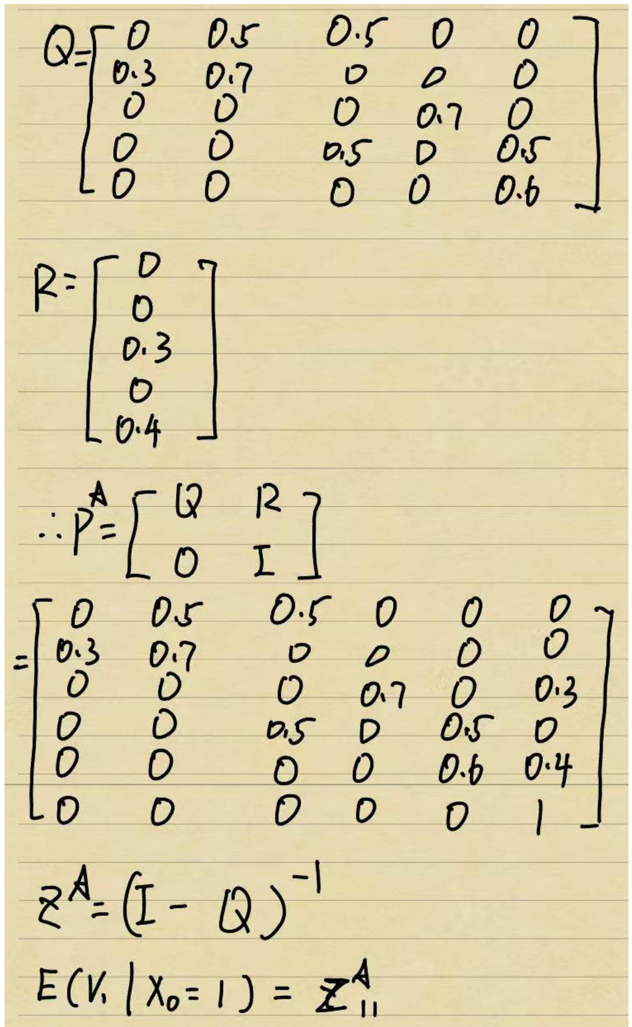
let $w_8 = 1$

$$\Rightarrow w = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4/3 \\ 2/3 \\ 1 \end{bmatrix}$$

Figure 2: Q1.2 stationary distribution

1.3 Q1.3

The given MC doesn't have absorbing state (with $P_{ii} = 1$). But we can find that the state $6, 7, 8$ forms a absorbing and recurrent class that once we enter state 6,7 or 8, we can't leave the class. Therefore, we can treat state $6, 7, 8$ as a single absorbing state to model the original MC as a absorbing MC, and use the fundamental matrix to solve this problem. Below is my hand-written solution and derivation process.



$$Q = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.6 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 \\ 0 \\ 0.3 \\ 0 \\ 0.4 \end{bmatrix}$$

$$\therefore P^A = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0 & 0.3 \\ 0 & 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0.6 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Z^A = (I - Q)^{-1}$$

$$E(V_1 | X_0 = 1) = Z^A_{11}$$

Figure 3: Q1.3

The answer for this problem is 2.0 which is the $Z^A_{1,1} = 2.0$.

1.4 Q1.4

Based on the definition of mean first passage time, we are asked to find $M_{18} = E(\tau_8 | x_0 = 1)$, and we have

$$M_{18} = 1 + \sum_{k \neq 8} P_{1k} M_{k8}$$

Therefore, for each $k \neq 8$, we can write a equation of M_{k8}. In total we would have a system of equations with 7 equations and 7 unknown, and we can solve it to find M_{18}. Below is my hand-written solution and derivation process.

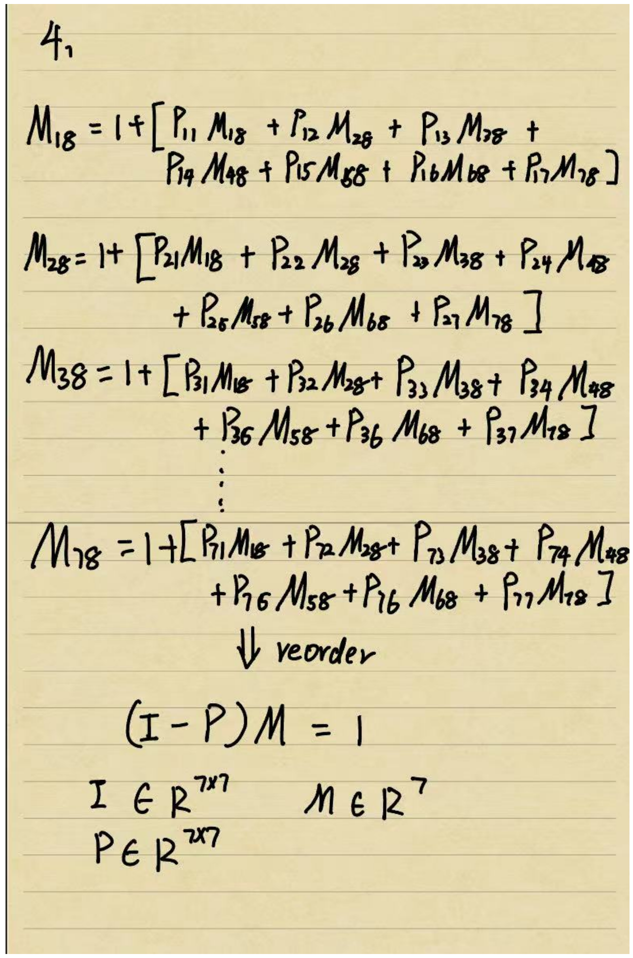


Figure 4: Q1.4

After solving this via sympy package, the $M_{18} = 12.9833333333333$.

2 Q2

2.1 Q2.1

Firstly, I draw the tree of states at $T = 4$ and found the valid trajectories that visit all state at $T = 4$

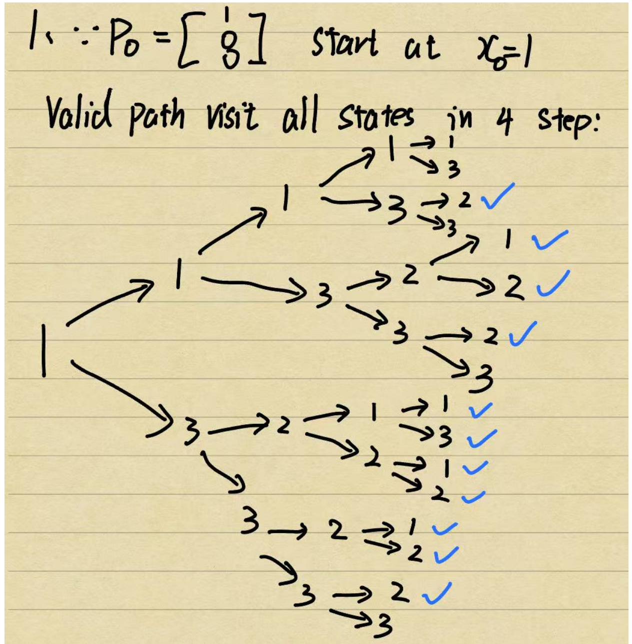


Figure 5: Q2.1

Then, for each valid trajectory $\tau = (x_0, x_1, x_2, x_3, x_4)$, we can compute $P(\tau) = P_{x_0,x_1} P_{x_1,x_2} P_{x_2,x_3} P_{x_3,x_4}$ by the Markov assumption and Bayes Rule. Then probability that all states have been visited by T=4 is just $\sum P(\tau)$. The answer I get is $\sum P(\tau) = 0.7668$.

2.2 Q2.2

To find the $V_0(1)$, by definition of MRP, we have

$$V_0(1) = E(q(x_4) + \sum_{t=0}^{3} l(x_t))$$

$$V_0(1) = \sum_{\tau} P(\tau)(q(x_4) + \sum_{t=0}^{3} l(x_t))$$

Where $P(\tau)$ is the probability of a possible trajectory $\tau = (x_0, x_1, x_2, x_3, x_4)$. To solve this question, I wrote following python code

```
states = [0, 1, 2]
T = 4
initial_state = 0

valid_paths = []
valid_path_probs = []
valid_costs = []

# Generate all paths of T=4
for path in product(states, repeat=T):
    full_path = (initial_state,) + path
    if set(full_path) == {0, 1, 2}:
        prob = 1.0
        cost = 0.0
```

```

        # Probability of path
        for i in range(1, len(full_path)):
            prob *= P[full_path[i - 1], full_path[i]]
        valid_paths.append(full_path)
        valid_path_probs.append(prob)

        # Cost of path
        for i in range(0, len(full_path) - 1):
            cost += visit_cost(full_path[i]+1)
        cost += terminate_cost(full_path[-1] + 1)
        valid_costs.append(cost * prob)
# Value at t=0 of state 1
value = sum(valid_costs)
```

As the result, $V_0(1) = 11.145600000000002$.

3 Q3

3.1 Q3.1

Below is my hand-written solution and derivation process.

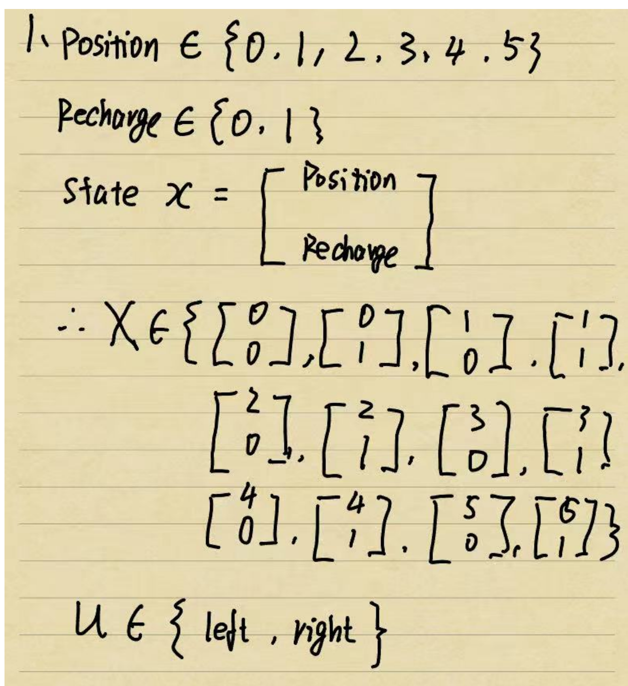
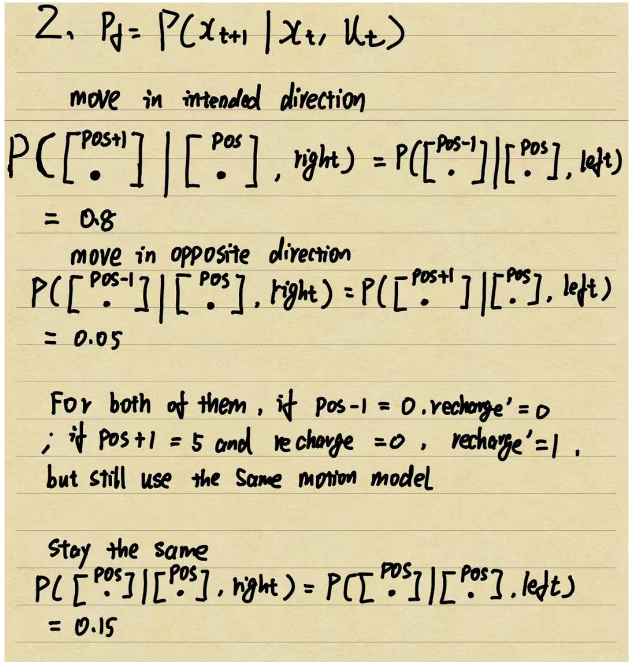


Figure 6: Q3.1

3.2 Q3.2

Below is my hand-written solution and derivation process.



2. $P_f = P(x_{t+1} \mid x_t, u_t)$

move in intended direction

$$P\left(\begin{bmatrix} pos+1 \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, right\right) = P\left(\begin{bmatrix} pos-1 \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, left\right)$$

$= 0.8$

move in opposite direction

$$P\left(\begin{bmatrix} pos-1 \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, right\right) = P\left(\begin{bmatrix} pos+1 \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, left\right)$$

$= 0.05$

For both of them, if $pos-1 = 0$, $recharge' = 0$; if $pos+1 = 5$ and $recharge = 0$, $recharge' = 1$, but still use the same motion model

Stay the same

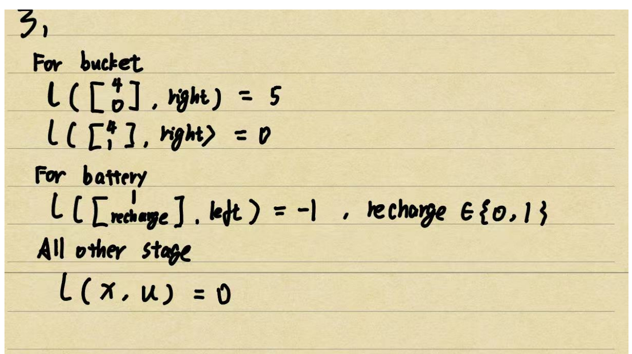
$$P\left(\begin{bmatrix} pos \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, right\right) = P\left(\begin{bmatrix} pos \\ \cdot \end{bmatrix} \middle| \begin{bmatrix} pos \\ \cdot \end{bmatrix}, left\right)$$

$= 0.15$

Figure 7: Q3.2

3.3 Q3.3

Below is my hand-written solution and derivation process.



3.

For bucket

$l\left(\begin{bmatrix} 4 \\ 0 \end{bmatrix}, right\right) = 5$

$l\left(\begin{bmatrix} 4 \\ 1 \end{bmatrix}, right\right) = 0$

For battery

$l\left(\begin{bmatrix} 1 \\ recharge \end{bmatrix}, left\right) = -1$, $recharge \in \{0, 1\}$

All other stage

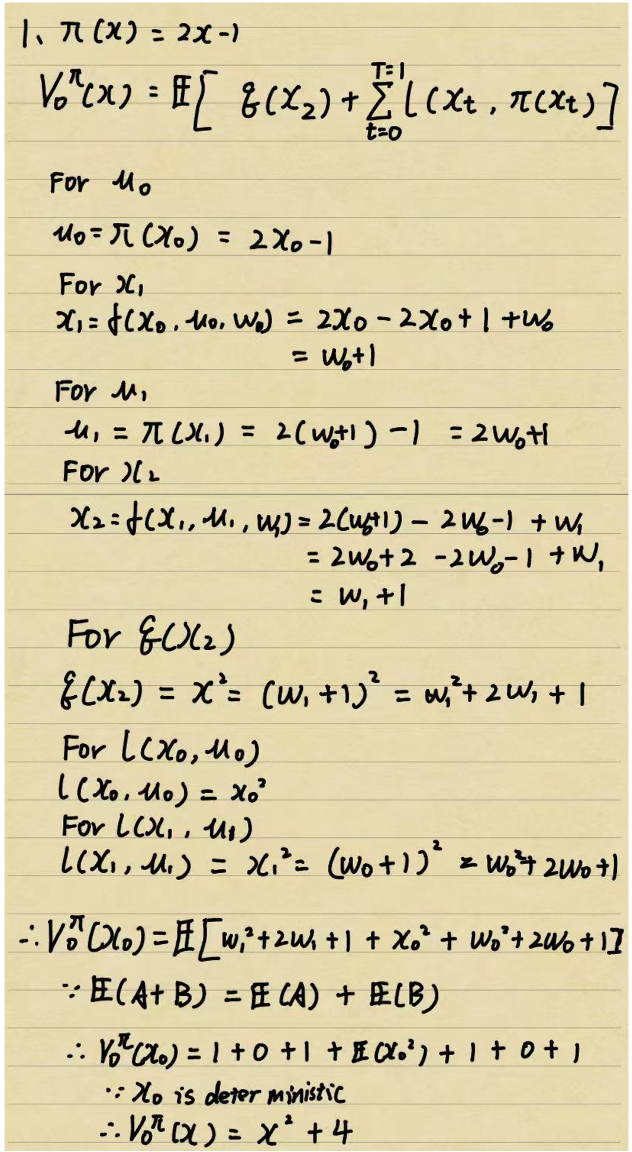
$l(x, u) = 0$

Figure 8: Q3.3

4 Q4

4.1 Q4.1

Below is my hand-written solution and derivation process.



1、$\pi(x) = 2x - 1$

$V_0^\pi(x) = \mathbb{E}\left[g(x_2) + \sum_{t=0}^{T=1} l(x_t, \pi(x_t)) \right]$

For u_0

$u_0 = \pi(x_0) = 2x_0 - 1$

For x_1

$x_1 = f(x_0, u_0, w_0) = 2x_0 - 2x_0 + 1 + w_0$
$= w_0 + 1$

For u_1

$u_1 = \pi(x_1) = 2(w_0+1) - 1 = 2w_0 + 1$

For x_2

$x_2 = f(x_1, u_1, w_1) = 2(w_0+1) - 2w_0 - 1 + w_1$
$= 2w_0 + 2 - 2w_0 - 1 + w_1$
$= w_1 + 1$

For $g(x_2)$

$g(x_2) = x^2 = (w_1+1)^2 = w_1^2 + 2w_1 + 1$

For $l(x_0, u_0)$

$l(x_0, u_0) = x_0^2$

For $l(x_1, u_1)$

$l(x_1, u_1) = x_1^2 = (w_0+1)^2 = w_0^2 + 2w_0 + 1$

$\therefore V_0^\pi(x_0) = \mathbb{E}[w_1^2 + 2w_1 + 1 + x_0^2 + w_0^2 + 2w_0 + 1]$

$\because \mathbb{E}(A+B) = \mathbb{E}(A) + \mathbb{E}(B)$

$\therefore V_0^\pi(x_0) = 1 + 0 + 1 + \mathbb{E}(x_0^2) + 1 + 0 + 1$

$\because x_0$ is deter ministic

$\therefore V_0^\pi(x) = x^2 + 4$

Figure 9: Q4.1

$V_0^\pi(x) = x^2 + 4$

4.2 Q4.2

Below is my hand-written solution and derivation process.

2、For u_0

$u_0 = u(x_0) = 3x_0$

For x_1

$x_1 = f(x_0, u_0, w_0) = 2x_0 - 3x_0 + w_0 = w_0 - x_0$

For $l(x_0, u_0)$

$l(x_0, u_0) = x_0^2$

For u_1

$u_1 = u(x_1) = 3x_1 = 3w_0 - 3x_0$

For $l(x_1, u_1)$

$l(x_1, u_1) = x_1^2 = (w_0 - x_0)^2 = w_0^2 - 2w_0x_0 + x_0^2$

For x_2

$$
\begin{aligned}
x_2 = f(x_1, u_1, w_1) &= 2x_1 - u_1 - w_1 \\
&= 2w_0 - 2x_0 - 3w_0 + 3x_0 - w_1 \\
&= x_0 - w_0 - w_1
\end{aligned}
$$

For $g(x_2)$

$$
\begin{aligned}
g(x_2) = x_2^2 &= (x_0 - w_0 - w_1)^2 \\
&= x_0^2 - 2x_0w_0 - 2x_0w_1 + w_0^2 \\
&\quad + 2w_0w_1 + w_1^2
\end{aligned}
$$

$\because$ Independent, $\mathbb{E}(w_0) = \mathbb{E}(w_1) = 0$

$\therefore \mathbb{E}(x_0w_0) = \mathbb{E}(x_0) \cdot \mathbb{E}(w_0) = 0$

$\mathbb{E}(x_0w_1) = \mathbb{E}(x_0) \cdot \mathbb{E}(w_1) = 0$

$\mathbb{E}(w_0w_1) = \mathbb{E}(w_0)\,\mathbb{E}(w_1) = 0$

Figure 10: Q4.2 part 1

$\mathbb{E}(l(x_0, u_0)) = \mathbb{E}(x_0^2) = x_0^2$

$$
\begin{aligned}
\mathbb{E}(l(x_1, u_1)) &= \mathbb{E}(w_0^2) - 2\mathbb{E}(x_0w_0) + \mathbb{E}(x_0^2) \\
&= 1 - 0 + x_0^2 = x_0^2 + 1
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}(g(x_2)) &= \mathbb{E}(x_0^2) - 2\mathbb{E}(x_0w_0) - \\
&\quad 2\mathbb{E}(x_0w_1) + \mathbb{E}(w_0^2) + 2\mathbb{E}(w_0w_1) \\
&\quad + \mathbb{E}(w_1^2) \\
&= x_0^2 - 0 - 0 + 1 + 0 + 1 \\
&= x_0^2 + 2
\end{aligned}
$$

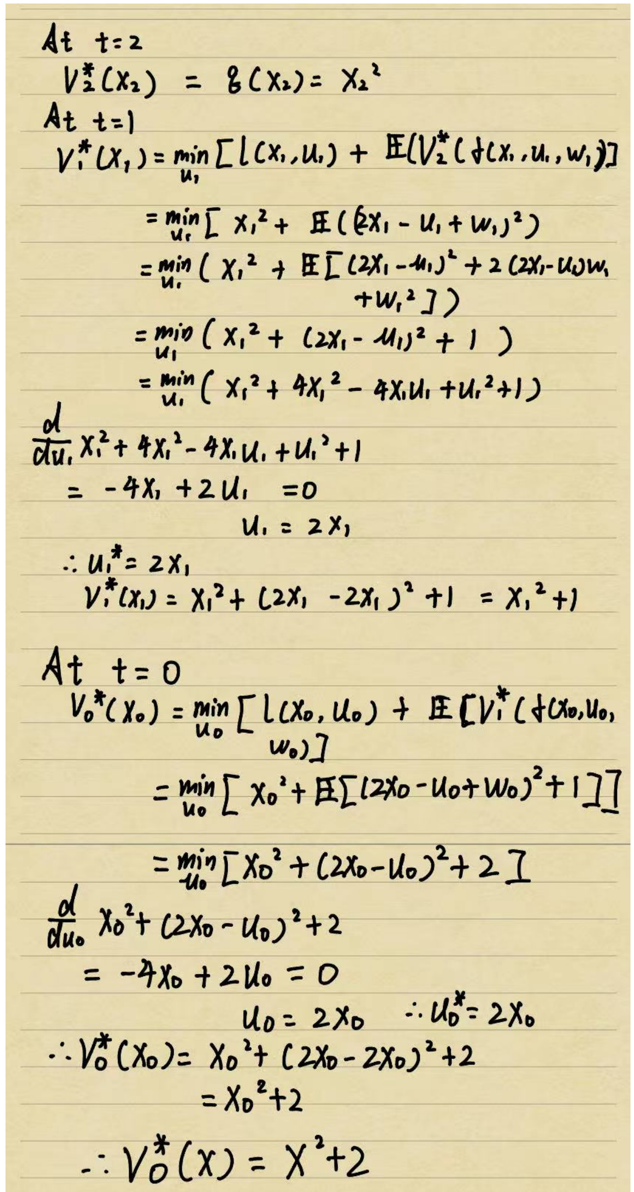
$$
\begin{aligned}
V_0^u(x_0) &= \mathbb{E}[g(x_2) + l(x_1, u_1) + \\
&= l(x_0, u_0)] \\
&= 3x_0^2 + 3 \\
&= 3x^2 + 3
\end{aligned}
$$

Figure 11: Q4.2 part 2

$V_0^u(x) = 3x^2 + 3$

4.3 Q4.3

We can use the dynamic programming to solve this problem. Below is my hand-written solution and derivation process.



At $t=2$

$$V_2^*(x_2) = g(x_2) = x_2^2$$

At $t=1$

$$V_1^*(x_1) = \min_{u_1} [l(x_1,u_1) + \mathbb{E}(V_2^*(f(x_1,u_1,w_1)))]$$

$$= \min_{u_1} [x_1^2 + \mathbb{E}((2x_1 - u_1 + w_1)^2)]$$

$$= \min_{u_1} (x_1^2 + \mathbb{E}[(2x_1 - u_1)^2 + 2(2x_1 - u_1)w_1 + w_1^2])$$

$$= \min_{u_1} (x_1^2 + (2x_1 - u_1)^2 + 1)$$

$$= \min_{u_1} (x_1^2 + 4x_1^2 - 4x_1u_1 + u_1^2 + 1)$$

$$\frac{d}{du_1} x_1^2 + 4x_1^2 - 4x_1u_1 + u_1^2 + 1$$

$$= -4x_1 + 2u_1 = 0$$

$$u_1 = 2x_1$$

$$\therefore u_1^* = 2x_1$$

$$V_1^*(x_1) = x_1^2 + (2x_1 - 2x_1)^2 + 1 = x_1^2 + 1$$

At $t=0$

$$V_0^*(x_0) = \min_{u_0} [l(x_0,u_0) + \mathbb{E}[V_1^*(f(x_0,u_0,w_0))]]$$

$$= \min_{u_0} [x_0^2 + \mathbb{E}[(2x_0 - u_0 + w_0)^2 + 1]]$$

$$= \min_{u_0} [x_0^2 + (2x_0 - u_0)^2 + 2]$$

$$\frac{d}{du_0} x_0^2 + (2x_0 - u_0)^2 + 2$$

$$= -4x_0 + 2u_0 = 0$$

$$u_0 = 2x_0 \quad \therefore u_0^* = 2x_0$$

$$\therefore V_0^*(x_0) = x_0^2 + (2x_0 - 2x_0)^2 + 2 = x_0^2 + 2$$

$$\therefore V_0^*(x) = x^2 + 2$$

Figure 12: Q4.3

$V_0^*(x) = x^2 + 2$

4.4 Q4.4

Below is my hand-written solution and derivation process.

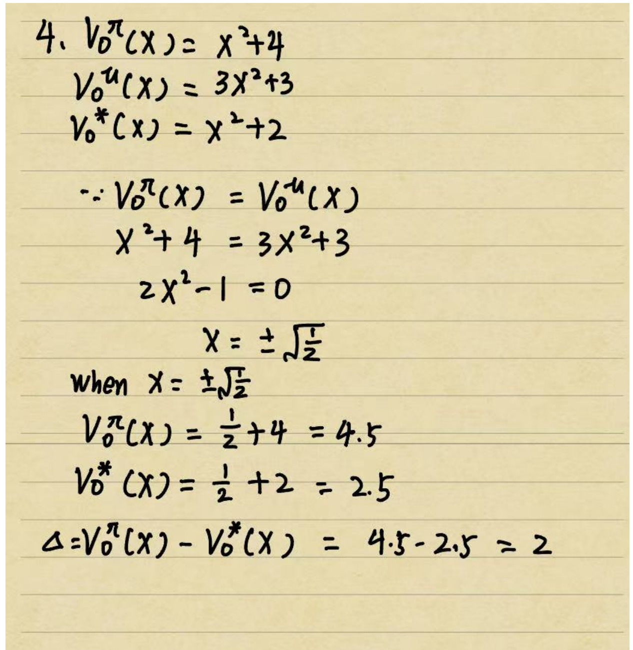


Figure 13: Q4.4

References

In [11]:
```python
import numpy as np
import matplotlib.pyplot as plt
import sympy
from itertools import product
```

Question 1

In [3]:
```python
# Question 1.1
P = np.array([
    [0, 0.5, 0.5, 0, 0, 0, 0, 0],
    [0.3, 0.7, 0.0, 0, 0, 0, 0, 0],
    [0.0, 0.0, 0.0, 0.7, 0, 0, 0, 0.3],
    [0.0, 0.0, 0.0, 0.5, 0.5, 0, 0, 0],
    [0.0, 0.0, 0.0, 0.0, 0.6, 0.4, 0, 0],
    [0.0, 0.0, 0.0, 0.0, 0., 0.5, 0.5, 0],
    [0.0, 0.0, 0.0, 0.0, 0., 0.4, 0., 0.6],
    [0.0, 0.0, 0.0, 0.0, 0., 0.4, 0., 0.6],
])
```

In [4]:
```python
A = P.T - np.eye(P.shape[0])
A = np.hstack((A, np.zeros(P.shape[0]).reshape(-1, 1)))
mat = sympy.Matrix(A)
print(mat.rref())
```

```
(Matrix([
[1, 0, 0, 0, 0, 0, 0,                    0, 0],
[0, 1, 0, 0, 0, 0, 0,                    0, 0],
[0, 0, 1, 0, 0, 0, 0, 2.06514699521048e-16, 0],
[0, 0, 0, 1, 0, 0, 0, 2.89120579329468e-16, 0],
[0, 0, 0, 0, 1, 0, 0, 1.44560289664734e-16, 0],
[0, 0, 0, 0, 0, 1, 0,    -1.33333333333333, 0],
[0, 0, 0, 0, 0, 0, 1,   -0.666666666666667, 0],
[0, 0, 0, 0, 0, 0, 0,                    0, 0]]), (0, 1, 2, 3, 4, 5, 6))
```

In [5]:
```python
x = np.array([0, 0, 0, 0, 0, 4/3, 2/3, 1]).reshape(-1, 1)
print(P.T @ x)
```

```
[[0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [1.33333333]
 [0.66666667]
 [1.        ]]
```

In [6]:
```python
# Question 1.3
Q = np.array([
    [0, 0.5, 0.5, 0, 0],
    [0.3, 0.7, 0.0, 0, 0],
    [0.0, 0.0, 0.0, 0.7, 0],
    [0.0, 0.0, 0.5, 0.0, 0.5],
    [0.0, 0.0, 0.0, 0.0, 0.6]
])
Z = np.eye(Q.shape[0]) - Q
```

```
Z_inv = np.linalg.inv(Z)
print(Z_inv)
```

```
[[2.         3.33333333 1.53846154 1.07692308 1.34615385]
 [2.         6.66666667 1.53846154 1.07692308 1.34615385]
 [0.         0.         1.53846154 1.07692308 1.34615385]
 [0.         0.         0.76923077 1.53846154 1.92307692]
 [0.         0.         0.         0.         2.5       ]]
```

In [10]:

```
# Question 1.4
sub_A = P[:7, :7]
sub_A = np.eye(sub_A.shape[0]) - sub_A
sub_mat = np.hstack([sub_A, np.ones(sub_A.shape[0]).reshape(-1, 1)])
sub_mat = sympy.Matrix(sub_mat)
print(sub_mat.rref())
```

```
(Matrix([
[1, 0, 0, 0, 0, 0, 0, 12.9833333333333],
[0, 1, 0, 0, 0, 0, 0, 16.3166666666667],
[0, 0, 1, 0, 0, 0, 0,             7.65],
[0, 0, 0, 1, 0, 0, 0,              9.5],
[0, 0, 0, 0, 1, 0, 0,              7.5],
[0, 0, 0, 0, 0, 1, 0,              5.0],
[0, 0, 0, 0, 0, 0, 1,              3.0]]), (0, 1, 2, 3, 4, 5, 6))
```

Question 2

In [13]:

```
P = np.array([
    [0.1, 0., 0.9],
    [0.7, 0.3, 0.0],
    [0.0, 0.4, 0.6]
])
def visit_cost(state):
    return 2 * state
def terminate_cost(state):
    return -1 * state
```

In [10]:

```
# Question 2.1
p1 = P[0,0] * P[0,0] * P[0,2] * P[2,1]

p2 = P[0,0] * P[0,2] * P[2,1] * P[1,0]

p3 = P[0,0] * P[0,2] * P[2,1] * P[1,1]

p4 = P[0,0] * P[0,2] * P[2,2] * P[2,1]

p5 = P[0,2] * P[2,1] * P[1,0] * P[0,0]

p6 = P[0,2] * P[2,1] * P[1,0] * P[0,2]

p7 = P[0,2] * P[2,1] * P[1,1] * P[1,1]

p8 = P[0,2] * P[2,1] * P[1,1] * P[1,0]

p9 = P[0,2] * P[2,2] * P[2,1] * P[1,0]

p10 = P[0,2] * P[2,2] * P[2,1] * P[1,1]
```

```python
p11 = P[0,2] * P[2,2] * P[2,2] * P[2,1]
print(p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9 + p10 + p11)
```

0.7668

In [17]:
```python
# Question 2.2
states = [0, 1, 2]
T = 4
initial_state = 0

valid_paths = []
valid_path_probs = []
valid_costs = []

# Generate all paths of T=4
for path in product(states, repeat=T):
    full_path = (initial_state,) + path
    if set(full_path) == {0, 1, 2}:
        prob = 1.0
        cost = 0.0
        for i in range(1, len(full_path)):
            prob *= P[full_path[i - 1], full_path[i]]
        valid_paths.append(full_path)
        valid_path_probs.append(prob)
        for i in range(0, len(full_path) - 1):
            cost += visit_cost(full_path[i]+1)
        cost += terminate_cost(full_path[-1] + 1)
        valid_costs.append(cost * prob)
print(sum(valid_costs))
```

11.145600000000002

In [ ]: