學號:110303585

# Artificial Intelligence

# Homework 4

# Reinforcement learning

姓名: 劉育辰

學號:110303585

系級:機械 4C

A.

(1) State value iteration Vk*(s)

Code:

```
def v_value(agent, max_iter):
    shape = agent.env.map.shape
    V = np.zeros(shape)
    V[agent.end] = 1
    V[agent.bomb] = -1
    policy = np.full(shape, "", dtype=object)

    for iteration in range(max_iter):
        print("========={}===========".format(iteration))
        new_V = np.copy(V)

        for i in range(shape[0]):
            for j in range(shape[1]):
                state = (i, j)
                print(state)
                max_value = -np.inf
                best_action = None

                if agent.env.map[state] == agent.env.wall:
                    continue

                if agent.env.map[state] == agent.env.final:
                    new_V[state] = 1
                    continue

                if agent.env.map[state] == agent.env.bomb:
                    new_V[state] = -1
```

這裡 V[]是個 table 會記錄每個格子的 predict reward，這段會先對 V[]做初始化，再來遍歷每個格子做算 v-value 的計算。

```python
                new_V[state] = -1
                continue

            """
            calculate v* processing
            """
            for action in agent.actions:
                value = 0
                next_state = agent.get_next_state(state, action)
                print(next_state)
                reward = agent.get_reward(state, next_state, V)

                value = 0 + agent.gamma * (reward[0]*0.8 + reward[1]*0.1 + reward[2]*0.1)
                value = round(float(value), 2)
                print("{} = 0 + {} * ({}*0.8 + {}*0.1) + {}*0.1".format(value, agent.gamma, reward[0], reward[1], reward[2]))

                if value > max_value:
                    max_value = value
                    best_action = action

            new_V[state] = max_value
            policy[state] = best_action
            print("\n")
    if np.array_equal(V, new_V):
        break
    V = new_V
```

這裡是跟 ppt 上的計算方法一樣，先得到可能會到達的方向，再根據機率計算 V-value，拿後只存取最大 reward 的動作，最後回傳 V[]和 policy table。

(2) Q-value iteration Qk*(s,a)

Code:

```python
class Q_TABLE:
    def __init__(self):
        self.table = []

    def get_items(self):
        items = {
            "pos":(0,0),
            "up": 0.0,
            "down": 0.0,
            "left": 0.0,
            "right": 0.0
        }
        return items
```

因為 Q-TABLE 會上下左右的 reward 都存取，所以特別設計一個結構，紀錄每個格子的 4 個動作的 reward 和位置。

```python
        """
        Get Q_table, but use V*to procsssing
        """
        for action in agent.actions:
            value = 0
            next_state = agent.get_next_state(state, action)
            reward = agent.get_reward(state, next_state, V)

            value = 0 + agent.gamma * (reward[0]*0.8 + reward[1]*0.1 + reward[2]*0.1)
            value = round(float(value), 2)
            q[action] = value

            if value > max_value:
                max_value = value
                best_action = action

        new_V[state] = max_value
        policy[state] = best_action
        row.append(q)
    q_table.insert(0, row)

if np.array_equal(V, new_V):
    break
```

作法上和剛剛的 v-value 的過程差不多，但多了 q[action] = value 來記錄每個 action 的 value，這樣才畫得出和 ppt 上一樣的圖。

## (3) Policy iteration

Code:

```python
def policy_iteration(agent):
    shape = agent.env.map.shape
    V = np.zeros(shape)
    V[agent.end] = 1
    V[agent.bomb] = -1
    policy = np.full(shape, "", dtype=object)

    for i in range(shape[0]):
        for j in range(shape[1]):
            if agent.env.map[i, j] not in [agent.env.wall, agent.env.final, agent.env.bomb
                policy[i, j] = np.random.choice(agent.actions)
    time = 0
```

這段會隨機選定 policy，並對 V[]做初始化。

```python
# ====== Policy Evaluation ======
while True:

    delta = 0
    new_V = np.copy(V)
    for i in range(shape[0]):
        for j in range(shape[1]):
            state = (i, j)

            if agent.env.map[state] in [agent.env.wall, agent.env.final, agent.env.bomb]:
                continue

            action = policy[state]
            next_states = agent.get_next_state(state, action)
            rewards = agent.get_reward(state, next_states, V)

            value = sum([
                0.8 * (0 + agent.gamma * rewards[0]),
                0.1 * (0 + agent.gamma * rewards[1]),
                0.1 * (0 + agent.gamma * rewards[2])
            ])
            value = round(float(value), 4)
            delta = max(delta, abs(value - V[state]))
            new_V[state] = value

    V = new_V
    time += 1
```

這段會根據剛剛隨機給定的 policy 做 v-value 的計算。

View Go Run Terminal Help

V.py U | policy.py U

olicy_iteration

```python
olicy_iteration(agent):
    # ====== Policy Improvement ======
    policy_stable = True

    for i in range(shape[0]):
        for j in range(shape[1]):
            state = (i, j)

            if agent.env.map[state] in [agent.env.wall, agent.env.final, agent.env.bomb]:
                continue

            old_action = policy[state]
            best_value = -np.inf
            best_action = None

            for action in agent.actions:
                next_states = agent.get_next_state(state, action)
                rewards = agent.get_reward(state, next_states, V)
                value = sum([
                    0.8 * (0 + agent.gamma * rewards[0]),
                    0.1 * (0 + agent.gamma * rewards[1]),
                    0.1 * (0 + agent.gamma * rewards[2])
                ])
                value = round(float(value), 4)
                if value > best_value:
                    best_value = value
                    best_action = action

            policy[state] = best_action
```

這段才會開始做策略上的改進，根據每個格子的 v-value 來做策略的變換直到收斂。

B.

My Calculate:

$k = 1$

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | | 0 | -1 |
| 0 | 0 | 0 | 0 |

$k = 2$

~~(3,4)~~ ~~(3,4)~~ (2,3)

| | | | |
|---|---|---|---|
| 0 | 0 | 0.72 | 1 |
| 0 | | 0 | -1 |
| 0 | 0 | 0 | 0 |

(0,0)

$(\uparrow)(0,0) = 0 + 0.9(0 \cdot 0.8 + 0 \cdot 0.1 + 0 \cdot 0.1) = 0$

$(\uparrow)(0,1) = 0 + 0.9(0) = 0$

$(\uparrow)(0,2) = 0 + 0.9(0) = 0$

$(\uparrow)(0,3) = 0 + 0.9(0) = 0$

$(\uparrow)(1,0) = 0 + 0.9(0) = 0$

$(1,1) = 0 + 0.9(0) = 0$ (const)

$(\leftarrow)(1,2) = 0 + 0.9(0) = 0$

$(1,3) = -1$ (const)

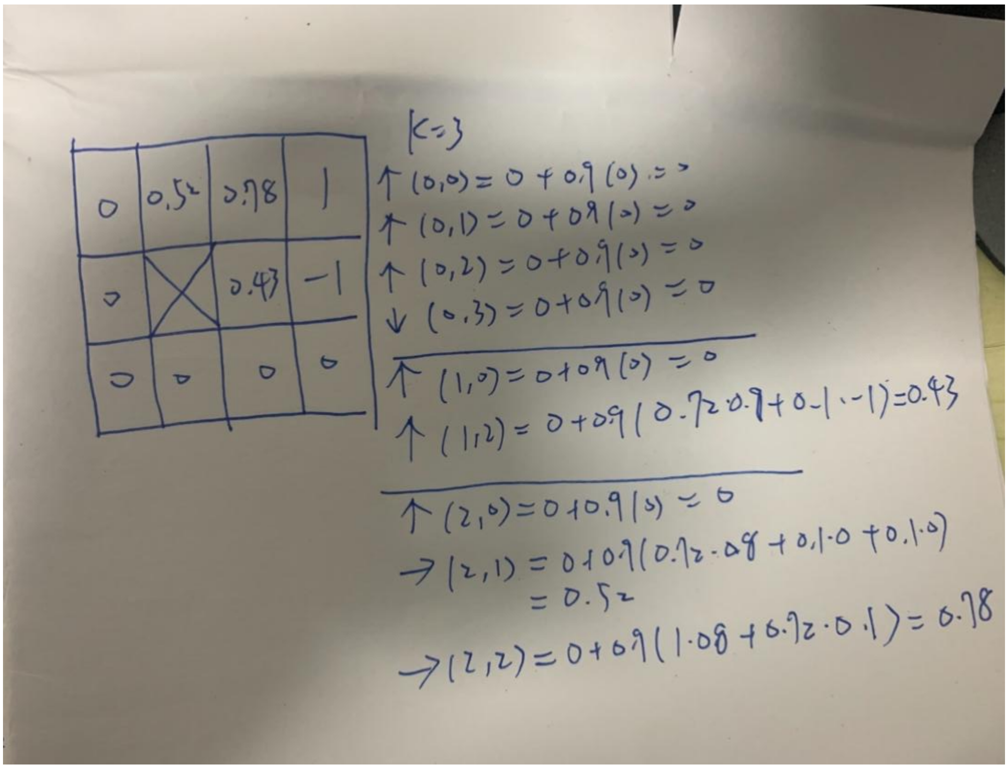
$(\uparrow)(2,0) = 0 + 0.9(0) = 0$

$(\uparrow)(2,1) = 0 + 0.9(0) = 0$

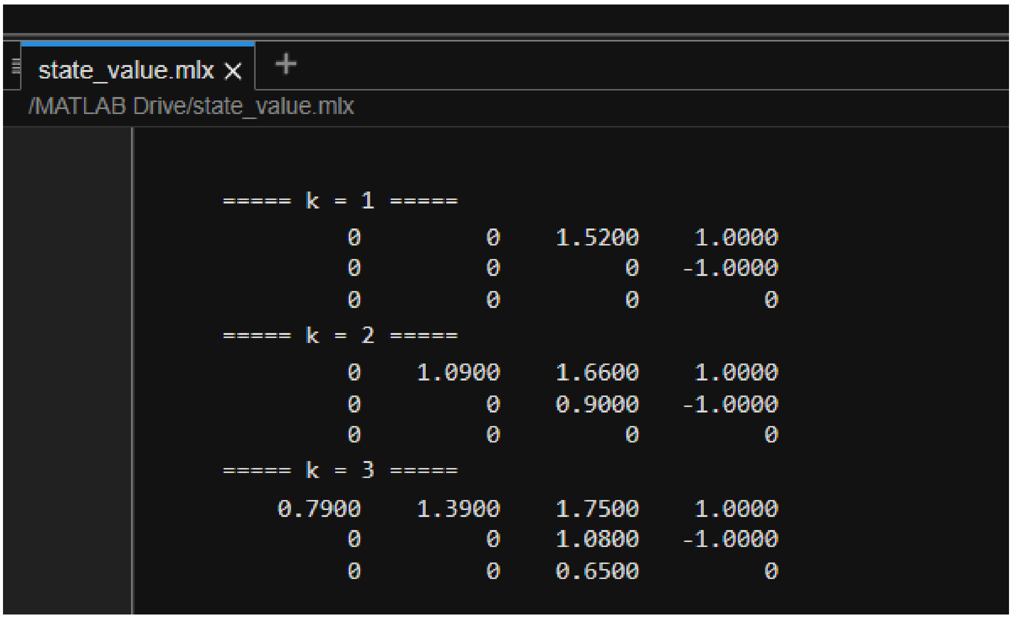
$(\rightarrow)(2,2) = 0 + 0.9(1 \cdot 0.8 + 0 \cdot 0.1 + 0 \cdot 0.01) = 0.72$

$(2,3) = 1$ (const)

結果和 V-value 和 Q-value 相近，但是 MATLAB 的結果確有




MATLAB:




```
===== k = 1 =====
     0         0    1.5200    1.0000
     0         0         0   -1.0000
     0         0         0         0
===== k = 2 =====
     0    1.0900    1.6600    1.0000
     0         0    0.9000   -1.0000
     0         0         0         0
===== k = 3 =====
0.7900    1.3900    1.7500    1.0000
     0         0    1.0800   -1.0000
     0         0    0.6500         0
```

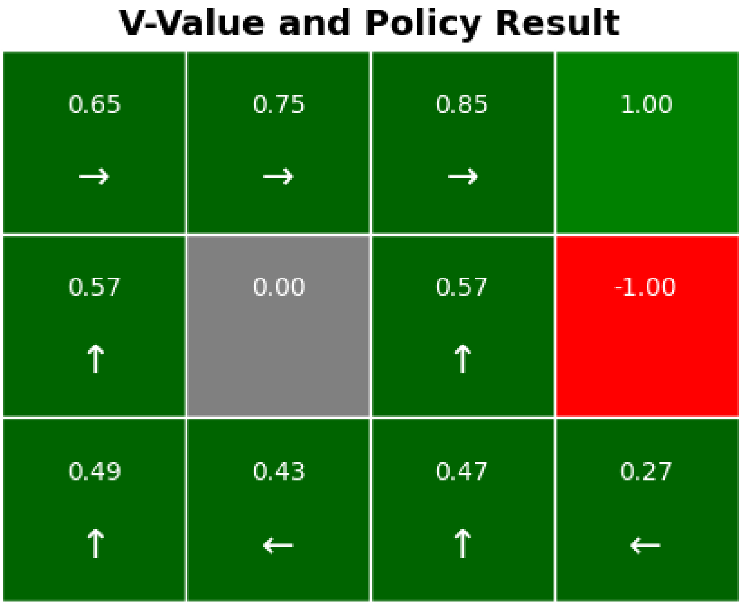
從我的計算結果和 MATLAB 的結果比較可以知道，我的計算

結果和 V-value 和 Q-value 相近，但是 MATLAB 的結果確有

大於 1 的數值出現，可能是 MATHLAB 的計算方式更考慮多步累積 reward 得到的結果。

C.

Result:





Discussion:

從上面三個結果的圖可看出，V-value 和 Q-value 的 policy 一樣，差在 Q-value 的 table 會多了其他三個 action 的值，但前面這兩個跟 Policy iteration 的圖有些差別，在(0,2)這格，Policy iteration 認為往右比較好，再來各格子的值也和前兩個的不一樣，我用程式跑得到的是經過 5 次遞迴就收斂了，但是前兩個方法要經過 12 次左右才收斂。

Conclusion:

結論是我用 code 跑 Q-VALUE 和 V-VALUE 結果更接近 ppt 上的結果，用手算的數值也能證明我的 code 跑的是對的，可是透過 Policy iteration 卻得到不一樣的 reward 數值，但 policy 卻是是一樣的，如果要找出最佳的 action，也是直線往上走，再往右走的終點。討論最後 policy 的決策化，最佳的入線有達到要求，所以三個方法都可行。