賽局理論與應用 Game Theory and Its Applications Assignment 1

Student ID: 312551077
Department: 資科工碩
Name: 薛祖恩

- Source Code Explanation

There are four screenshots in total, and the explanation will be provided below each screenshot.

```
1  # fictitious play implementation
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # input payoff matrix
6  payoffMatrix = input("""
7  Please input payoff matrix row by row.
8  Example(prisoner's dilemma): (-1, -1), (-3, 0), (0, -3), (-2, -2)
9  """)
10
11 # parse payoff matrix
12 payoffMatrix = payoffMatrix.replace('(', '').replace(')', '').replace(' ', '').split(',')
13 payoffMatrix = [int(i) for i in payoffMatrix]
14 payoffMatrix = np.array(payoffMatrix).reshape(-1, 2)
15
16 # set up iterations
17 iterations = 3000
18
19 # initial state
20 initialStrategyProfiles = ['(1, 1)', '(1, 2)', '(2, 1)', '(2, 2)']
21
22 # store strategy profiles for displaying
23 strategyProfiles = []
24
25 # set up figure size
26 plt.figure(figsize=(16, 8))
27
```

We import the libraries we need (line2-3).

The user can input the payoff matrix of the game (line6-8), the format is introduced with the output. Take prisoner's dilemma as an example, the input should be "(-1, -1), (-3, 0), (0, -3), (-2, -2)" representing u(r1, c1) = (-1, -1), u(r1, c2) = (-3, 0), u(r2, c1) = (0, -3), and u(r2, c2) = (-2, -2). The user input will then be stored into the variable "payoffMatrix".

The variable "payoffMatrix" will be preprocessed and reshaped into np arrays (line12-14). For example, the payoffMatrix of the prisoner's dilemma we have mentioned above after the preprocess will be:

[[-1 -1]
 [-3 0]
 [0 -3]
 [-2 -2]]

The iteration will be set up first (line 17).

Since the assignment this time only care about four strategy profiles (line 20), i.e., 2 strategies for each player. We set up a variable “initialStrategyProfiles” and store all the possible combination of the strategy profiles, this will be used as the initial round for the user to start their game.

“strategyProfiles” stores the result start from each initial strategy profiles after the iteration (line 23), that is, there will be four lists in “strategyProfiles”, the size of each list will be the amount of the variable “iterations” (line 17).

Set up for the size of the figure (line 26).

```
28  # play game
29  for count, initialStrategyProfile in enumerate(initialStrategyProfiles):
30      # player A belief means the player b's history, vice versa
31      playerABelief = [0, 0]
32      playerBBelief = [0, 0]
33      playerAPayoff = [0, 0]
34      playerBPayoff = [0, 0]
35      StrategyProfileResult = []
36      for i in range(iterations):
37          aChoice = 0
38          bChoice = 0
39          if i == 0:
40              # parse initial state
41              initialStrategyProfile = initialStrategyProfile.replace('(', '').replace(')', '').replace(' ', '').split(',')
42              if initialStrategyProfile[0] == '1':
43                  playerBBelief[0] += 1
44                  aChoice = 1
45              else:
46                  playerBBelief[1] += 1
47                  aChoice = 2
48              if initialStrategyProfile[1] == '1':
49                  playerABelief[0] += 1
50                  bChoice = 1
51              else:
52                  playerABelief[1] += 1
53                  bChoice = 2
```

The outer for loop iterate through the variable “initialStrategyProfiles” (line 29), which means that this program will consider all the possible initial round in the game. Then, the variables “playerABelief”, “playerBBelief”, “playerAPayoff”, “playerBPayoff”, and “StrategyProfileResult” are all initialized (line 31-35). “playerABelief” is the same meaning as the player B’s history choices, and vice versa. The meaning of “playerAPayoff” and “playerBPayoff” are the same as described in the question description.

The inner loop (line 36) means that the program will run this much time to see whether the result of the game will converge or not.

“aChoice” and “bChoice” are the strategy that this player chooses each round (line 37, 38).

The first if statement (line 36-53) deals with the first iteration, which is the initial round (line 39). First, the program will preprocess the “initialStrategyProfile” to a list which only contains two elements (line 41), the first is player A’s decision and the second is player B’s decision. Then, if player A’s decision is 1 (line 42), then the first element of player B’s belief (the same meaning as player A history) will be added by 1 (line 43), also update the variable “aChoice” by 1 (line 44); if player A’s decision is 2 (line 45),

then the second element of player B's belief (the same meaning as player A history) will be added by 1 (line 46), also update the variable "aChoice" by 2 (line 47). Same thing for player B, player A's belief, and "bChoice" (line 48-53).

```
54          else:
55              # update belief
56              if playerAPayoff[0] > playerAPayoff[1]:
57                  playerBBelief[0] += 1
58                  aChoice = 1
59              elif playerAPayoff[0] < playerAPayoff[1]:
60                  playerBBelief[1] += 1
61                  aChoice = 2
62              else:
63                  aRandomChoice = np.random.randint(1, 2)
64                  playerBBelief[aRandomChoice - 1] += 1
65                  aChoice = aRandomChoice
66              if playerBPayoff[0] > playerBPayoff[1]:
67                  playerABelief[0] += 1
68                  bChoice = 1
69              elif playerBPayoff[0] < playerBPayoff[1]:
70                  playerABelief[1] += 1
71                  bChoice = 2
72              else:
73                  bRandomChoice = np.random.randint(1, 2)
74                  playerABelief[bRandomChoice - 1] += 1
75                  bChoice = bRandomChoice
```

If it isn't the first iteration (line 54), then the choice of player A and player B will depend on the payoff matrix of themselves (line 56-75), if the first element(strategy) in the payoff matrix of player A is greater than the second element(strategy) in the payoff matrix of player A (line 56), that means player A is going to choose strategy 1, thus "aChoice" is confirmed as 1 (line 58). Thus, player A is going to choose the first strategy representing the first element of player B's belief should be added by 1 (line 57). For the situation where the first element(strategy) in the payoff matrix of player A is less than the second element(strategy) in the payoff matrix of player A (line 59), that means player A is going to choose strategy 2, thus "aChoice" is confirmed as 2 (line 61). Thus, player A is going to choose the second strategy representing the second element of player B's belief should be added by 1 (line 60). If both element of the payoff matrix are equal, then we randomly choose one strategy as the strategy of this round (line 62-65). We can apply the same logic to player B (line 66-75).

```
76          # player A payoff
77          playerAPayoff[0] = playerABelief[0] * payoffMatrix[0][0] + playerABelief[1] * payoffMatrix[1][0]
78          playerAPayoff[1] = playerABelief[0] * payoffMatrix[2][0] + playerABelief[1] * payoffMatrix[3][0]
79          # player B payoff
80          playerBPayoff[0] = playerBBelief[0] * payoffMatrix[0][1] + playerBBelief[1] * payoffMatrix[2][1]
81          playerBPayoff[1] = playerBBelief[0] * payoffMatrix[1][1] + playerBBelief[1] * payoffMatrix[3][1]
82          # Combine strategy profiles into a single integer
83          # A1B1 = 1, A1B2 = 2, A2B1 = 3, A2B2 = 4
84          strategy_profile = aChoice * 2 + bChoice - 2
85          StrategyProfileResult.append(strategy_profile)
86      strategyProfiles.append(StrategyProfileResult)
87
88      # Create subplots
89      plt.subplot(2, 2, count + 1)
90      y_ticks = [1, 2, 3, 4]
91      playerStrategyProfile = ["(r1, c1)", "(r1, c2)", "(r2, c1)", "(r2, c2)"]
92      plt.plot(range(1, 1 + iterations), strategyProfiles[count], marker='o', markersize=0.75)
93      xlabel = plt.xlabel(f'P(r*), P(c*): ({playerBBelief[0] / iterations:.4f}, \
94  {playerBBelief[1] / iterations:.4f}), \
95  ({playerABelief[0] / iterations:.4f}, \
96  {playerABelief[1] / iterations:.4f})'
97      )
98      xlabel.set_color('red')
99      plt.ylabel('Strategy Profile (Combined)')
100     plt.gca().set_yticks(y_ticks)
101     plt.gca().set_yticklabels(playerStrategyProfile)
102     # plt.title(f'Initial Strategy Profile: {initialStrategyProfile}')
103     plt.title(f'Starts from strategy profile: {playerStrategyProfile[count]}')
104
105 plt.tight_layout()
106 plt.show()
```

The next step we are going to do is to calculate the payoff of player A and player B after the current round. The way to calculate the first element(strategy) of player A's payoff matrix is the first element of player A's belief multiplied by the player A's utility in the payoff matrix where player A chose r1 and player 2 chose c1 added by the second element of player A's belief multiplied by the player A's utility in the payoff matrix where player A chose r1 and player B chose c2 (line 77). To make it clear, the formula will be like:

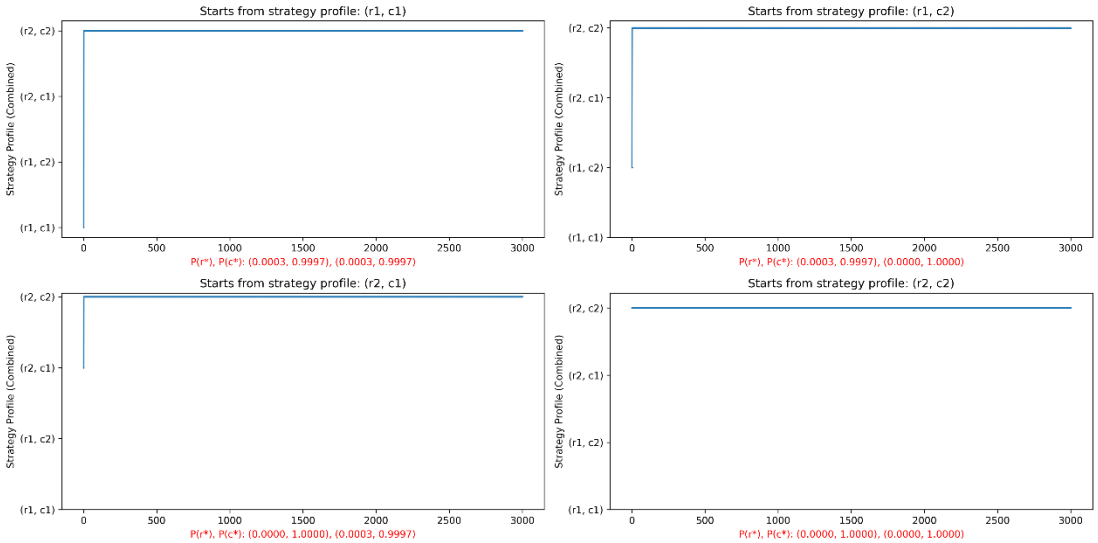
$$Payoff_a = \begin{bmatrix} n(c1\ history) \times u_a(r1, c1) + n(c2\ history) \times u_a(r1, c2) \\ n(c1\ history) \times u_a(r2, c1) + n(c2\ history) \times u_a(r2, c2) \end{bmatrix}^T$$

We can apply the same logic to player A's payoff and player B's payoff (line 76-81). The final step is to create subplots to show the information of each starting strategy profile (line 88-106), the y axis is the strategy profile (line 90, 91, 100, 101), that is, (r1, c1), (r1, c2), (r2, c1), (r2, c2); the x axis is the iteration. Through the figure we can easily see whether the strategy profile converges or not. There is also some useful information in the figure: y label (line 99), title of each subplot (line 103), and text (line 93-98) displayed at the bottom of each subplot. The text include the probability of strategies the player might choose, that is, P(r1, r2) and P(c1, c2), which represents the probability of each strategy each player would make.

- Q1

The input should be: (-1, -1), (1, 0), (0, 1), (3, 3)

The results are shown as follows:

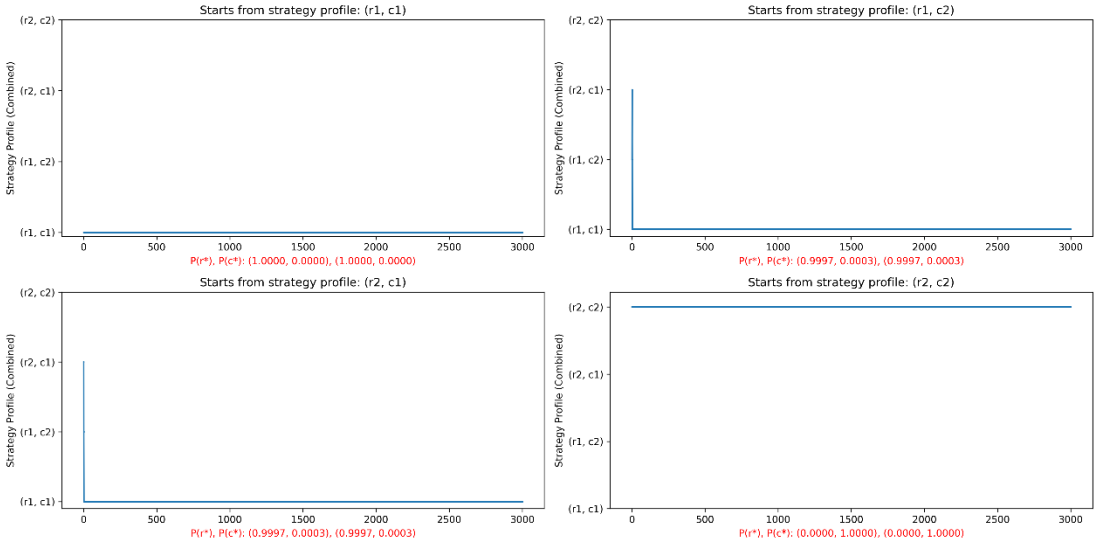


Yes. The results show that no matter what the initial strategy profile is, it will eventually converge to the pure-strategy NE, (r2, c2).

- Q2

The input should be: (2, 2), (1, 0), (0, 1), (3, 3)

The results are shown as follows:

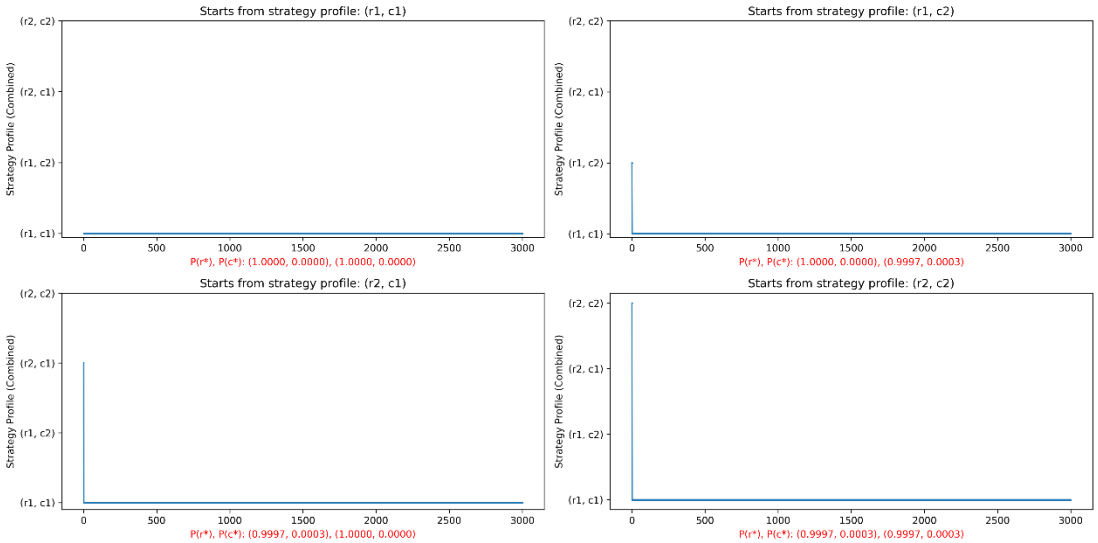


Yes. The result shows that fictitious play can converge to both two pure-strategy NE, when the initial strategy profile is (r2, c2), that is, (3, 3) it converges to (r2, c2); the rest of the initial strategy profile converges to (r1, c1), which is (2, 2).

- Q3

The input should be: (1, 1), (0, 0), (0, 0), (0, 0)

The results are shown as follows:

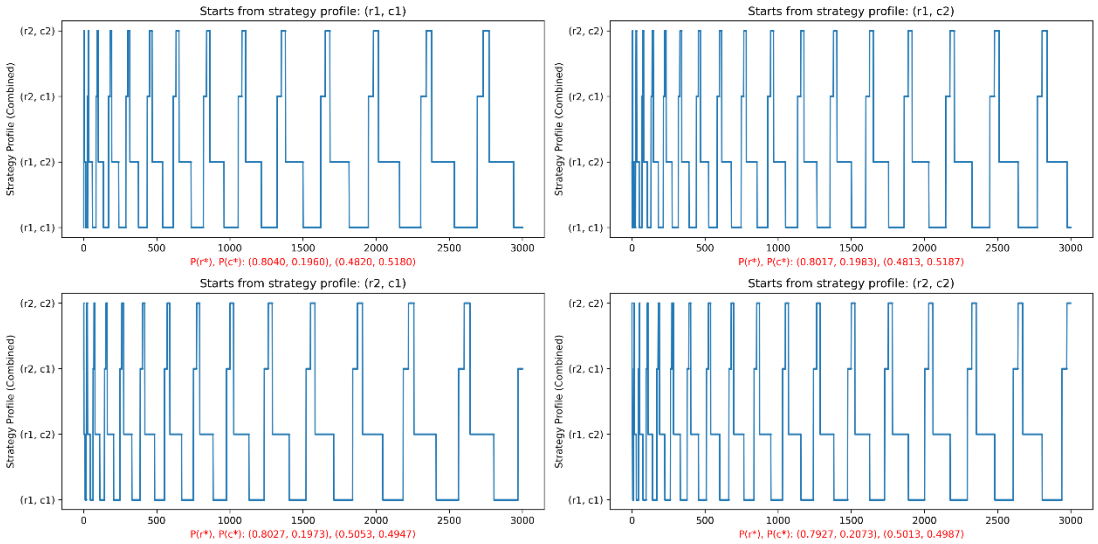


Just one of them. The result shows that although there are two NEs (r1, c1) and (r2, c2), all of the situations only converge to (r1, c1), the reason is because for initial strategy profile (r1, c2), (r2, c1), and (r2, c2), the payoff is always 0, so the player will randomly choose his/her strategy, once both of the players choose (r1, c1), then the player's payoff will become [1, 0], and this current result leads all of the results after this will always be (r1, c1), thus converges.

- Q4

The input should be: (0, 1), (2, 0), (2, 0), (0, 4)

The results are shown as follows:

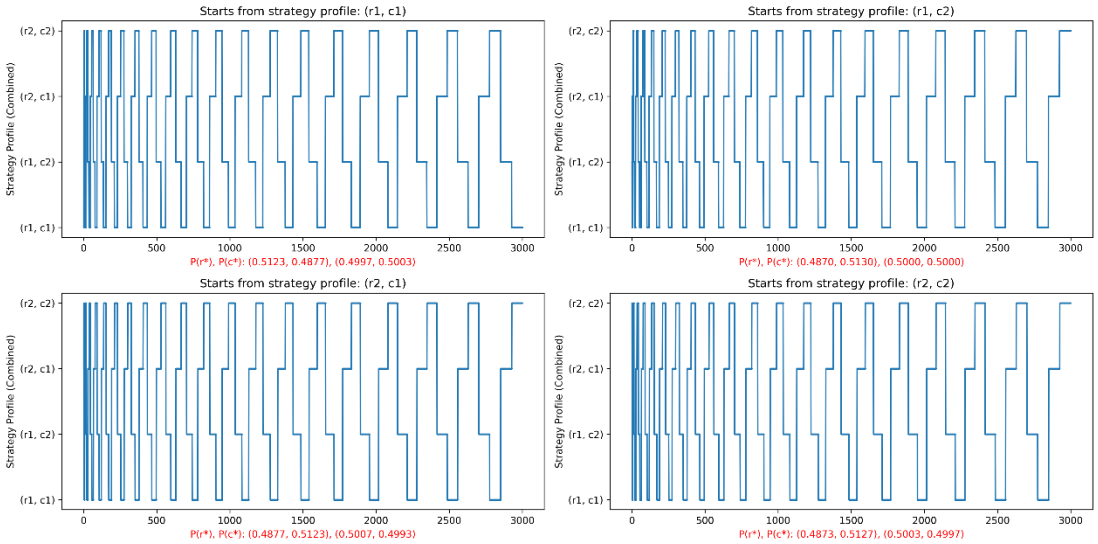


Yes. The result shows that when the initial strategy profile is (r1, c1), then P(r1) = 0.8040, P(r2) = 0.1960, P(c1) = 0.4820, P(c2) = 0.5180, the results of the others are approximately the same, thus we can say that fictitious play will converge to the mixed-strategy P(r1) = 0.8, P(r2) = 0.2, P(c1) = 0.5, and P(c2) = 0.5.

- Q5

The input should be: (0, 1), (1, 0), (1, 0), (0, 1)

The results are shown as follows:

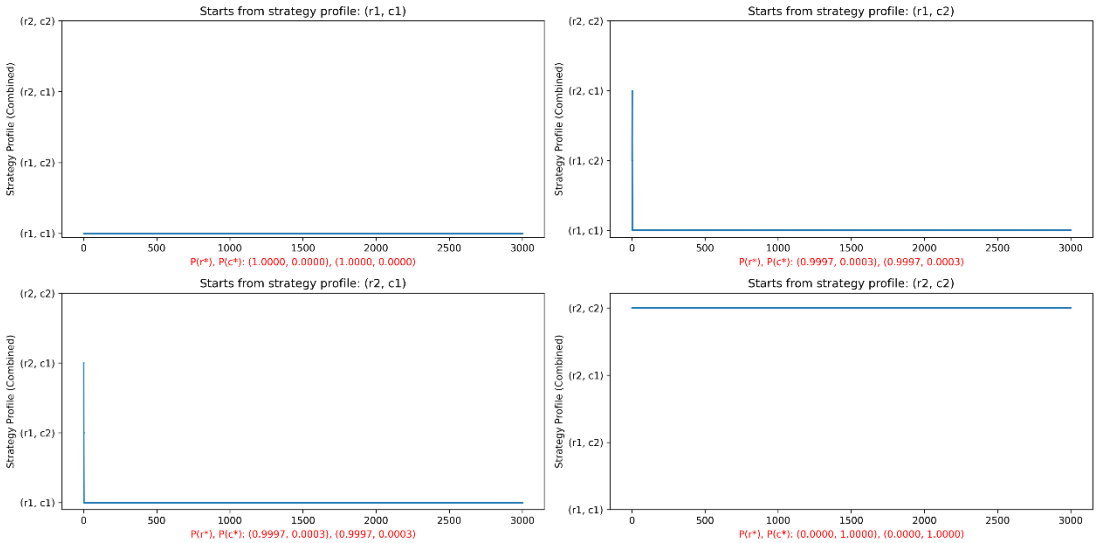


Yes. The result shows that no matter what the starting strategy profile is, you can always find out that there exists an infinite best reply path: (r2, c2), (r1, c2), (r1, c1), (r2, c1).

- Q6

The input should be: (10, 10), (0, 0), (0, 0), (10, 10)

The results are shown as follows:

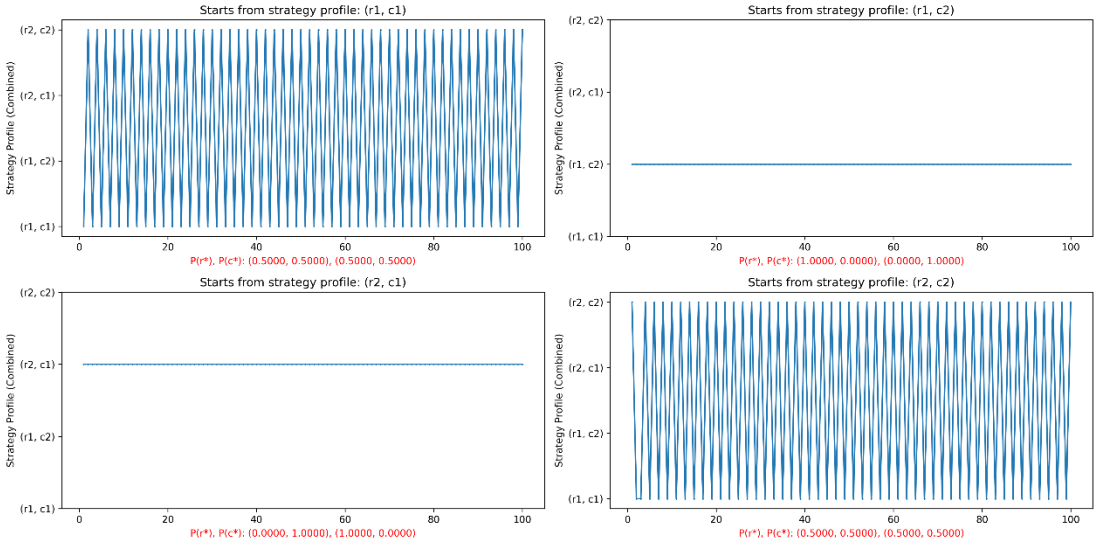


The result shows that it can only converge to the pure-strategy NE, not the mixed-strategy NE, since once the player's payoff is not zero (similar logic to Q3), it converges.

- Q7

The input should be: (0, 0), (1, 1), (1, 1), (0, 0)

The results are shown as follows:

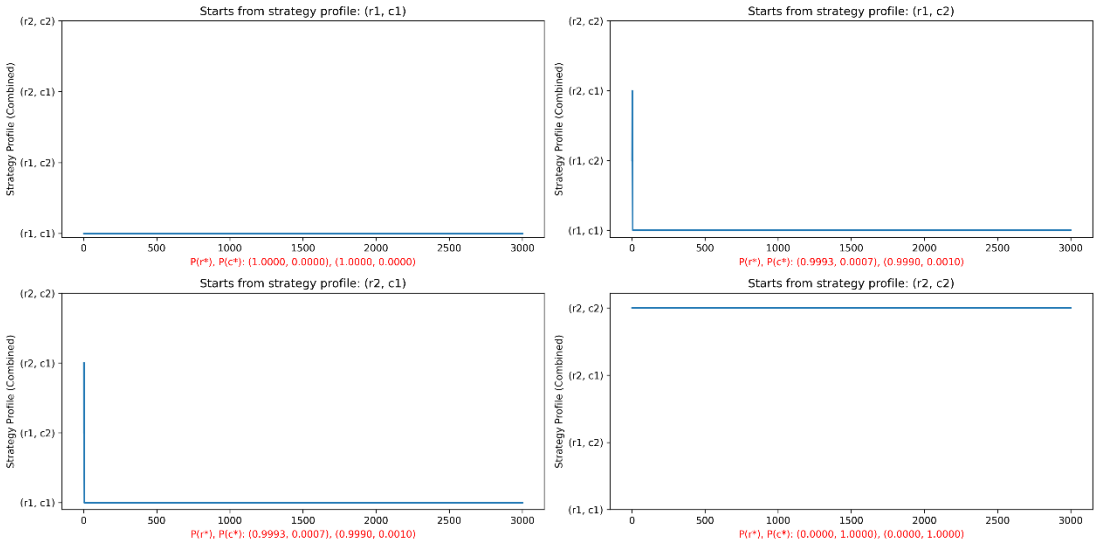


The results show that all can converge to NE, including (r1, c2), (r2, c1), and mixed-strategy P(r1, r2, c1, c2) = (0.5, 0.5, 0.5, 0.5). The reason is because for initial strategy profile (r1, c2) and (r2, c1), the player's payoff will let it converge. On the other hand, about the mixed-strategy NE, if the initial strategy profile is (r1, c1), player A will choose r2 since the payoff matrix is [0, 1] and so does player B, we can apply the same

logic to the initial state (r2, c2), the path will be: (r1, c1), (r2, c2), (r1, c1), (r2, c2), ….
Thus P(r1, r2, c1, c2) = (0.5, 0.5, 0.5, 0.5).

- Q8

The input should be: (3, 2), (0, 0), (0, 0), (2, 3)
The results are shown as follows:

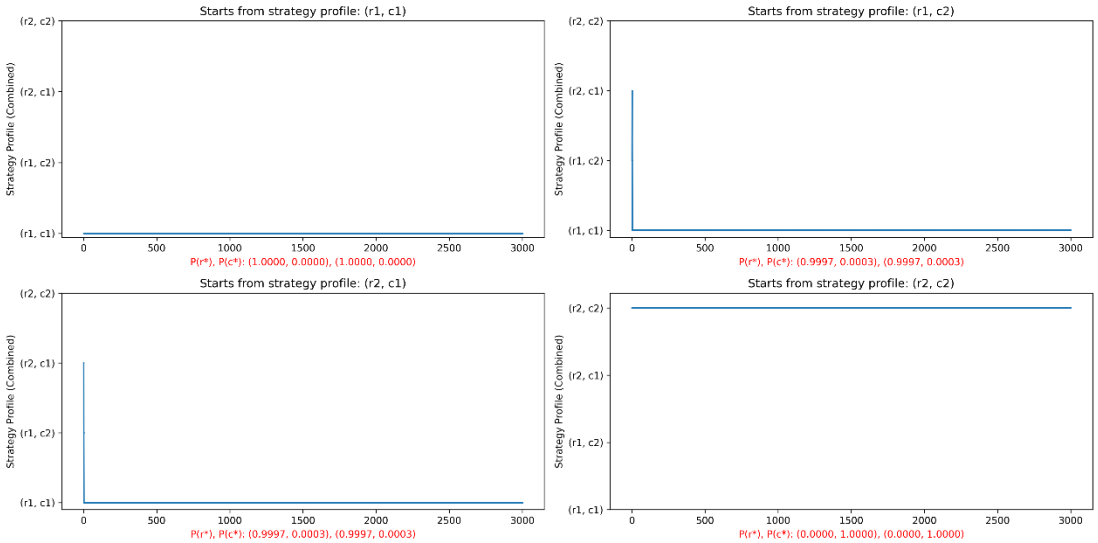


The result shows that it can only converge to the pure-strategy NE. For (r1, c1) and (r2, c2), the player's payoff will let it converge; for the rest of the two, it will choose random until the strategy profile (r1, c1) or (r2, c2) is chosen, then it converges.

- Q9

The input should be: (3, 3), (0, 2), (2, 0), (1, 1)

The results are shown as follows:



The result shows that it only converges to pure-strategy NEs, that is, (r1, c1) and (r2, c2), since once the player's payoff changes, it converges. For the mixed-strategy NE, it will never converge, since all the player payoffs leads to converge to the pure-strategy NE.

- Q10

Unfortunately, no. Take rock-paper-scissors as the example, the Nash Equilibrium is when only two players play the same strategy. However, after implementing the 3x3 version fictitious play, we can easily tell that the fictitious play method does not converge to a Nash Equilibrium. Thus, it is not reliable to find Nash Equilibrium by fictitious play for every game matrix.