Fictitious Play by Python

Naoya Sho

Oyama Seminar, University of Tokyo

28 Jun 2014

Today's contents

Fictitious Play

- A dynamic learning rule where each players rationally behaves based on the belief for the opponents' strategy.
- At t round, each player presumes that the opponents follows *the empirical frequency of strategies* from round 1 to $t-1$
- The learning process can be replicated by programming. In this excersise, two types of games below are covered.
  - Matching Pennies
  - 2×2 coordination game

Fictitious Play in a case of 2 players

- Diffrence equation of the belief

  $x_0(t)$ can be recursively written as

$$x_0(t+1) = x_0(t) + \frac{1}{t+2}(a_1(t) - x_0(t))$$

  where $x_0(t)$ is the player 0's belief about the player 1's behavior at time t and $a_1(t)$ is the player 1's action at time t.

Basic Algorithm

- A payoff of all the players is given in the form of a matrix.
- Buid a best responce function which
  - takes the payoff matrix and belief about the opponent's action as inputs
  - returns the action that maximize the expected payoff as an output
- For $t = 0, 1, 2...$, iteratively compute each player's action and belief and make a list of belief over the whole procedure.

Example games

- One example of Matching Pennies game is defined as below:

	Action 0	Action 1
Action 0	$1, -1$	$-1, 1$
Action 1	$-1, 1$	$1, -1$

- One example of 2×2 coordination game is defined as below:

	Action 0	Action 1
Action 0	$4, 4$	$0, 3$
Action 1	$3, 0$	$2, 2$

- Note that 2×2 coordination game is a symmetric game, where each player has the same payoff structure.

Programming codes 1

- Extracted codes for Matching Pennies game with brief comments

```
pay = ((1,-1),(-1,1),
       (-1,1),(1,-1))

def sep(p, pay):
    return ((pay[0][a], pay[1][a]),
            (pay[2][a], pay[3][a]))
```

- First, input the payoff structure in the form of matrix called `pay`
- Then set up the function `sep(a, pay)`, which
  - takes the index of the player and payoff matrix as inputs
  - returns the each player's individual payoff matrix dropping the opponent's payoff. (For later use.)

Programming codes 2

- Construct the best responce function.

```
xmat = np.empty((len(players), len(players)))

def br(p,x):
    xmat[p] = (1-x, x)
    expay = np.dot(np.array(sep(p, pay)), xmat[p])

    if expay[0] == expay[1]:
        return random.randint(0,1)

    else:
        return expay.argmax()
```

- p indicates pth player and x denotes his belief about the opponent's action. xmat is a empty 2×2 matrix where pth row has player p's expectation for opponent's each action.
- expay is a vector obtained by calculating the product of player p's individual payoff matrix and belief vector.
- Finally, returns the biggest element of expay if there is only one (Not very general).

Programming codes 3

- Compute the games iteratively.

```
def playgame(trials):

    x0 = random.uniform(0,1)
    x1 = random.uniform(0,1)

    for i in range(1000):
        a0 = br(0, x0)
        a1 = br(1, x1)
        x0.append(x[i]+(a[1]-x[i])/(i+2))
        x1.append(x[i]+(a[0]-x[i])/(i+2))
```

- Here, I used a function so that I can easily try different number of trials.
- Choose the action utilizing `br(p, x)` and put it into the difference equation.
- This part can be neatly summarized by using `for` loop for each player. (but I gave up)

Matching Pennies: Transition of belief

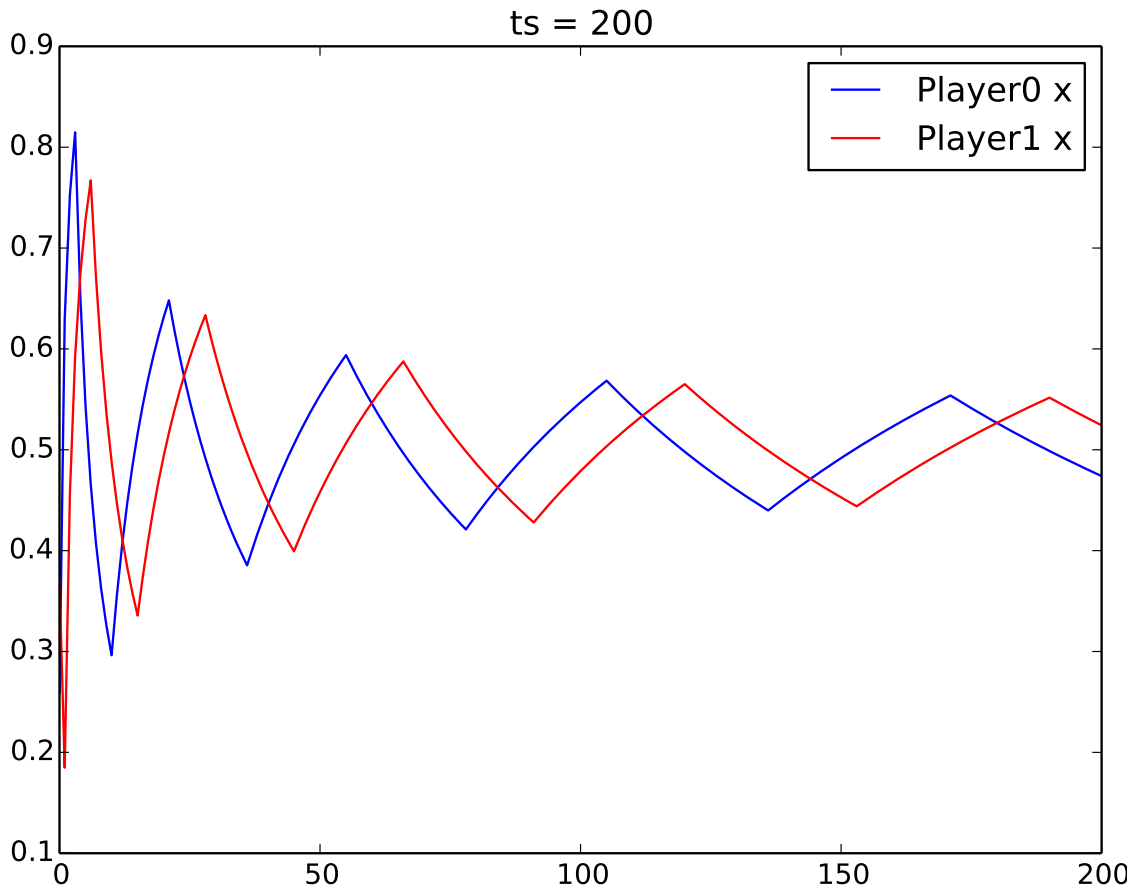

Figure : Transition of belife in Matching Pennis game for 200 times

Matching Pennies: Histogram of the terminal belief

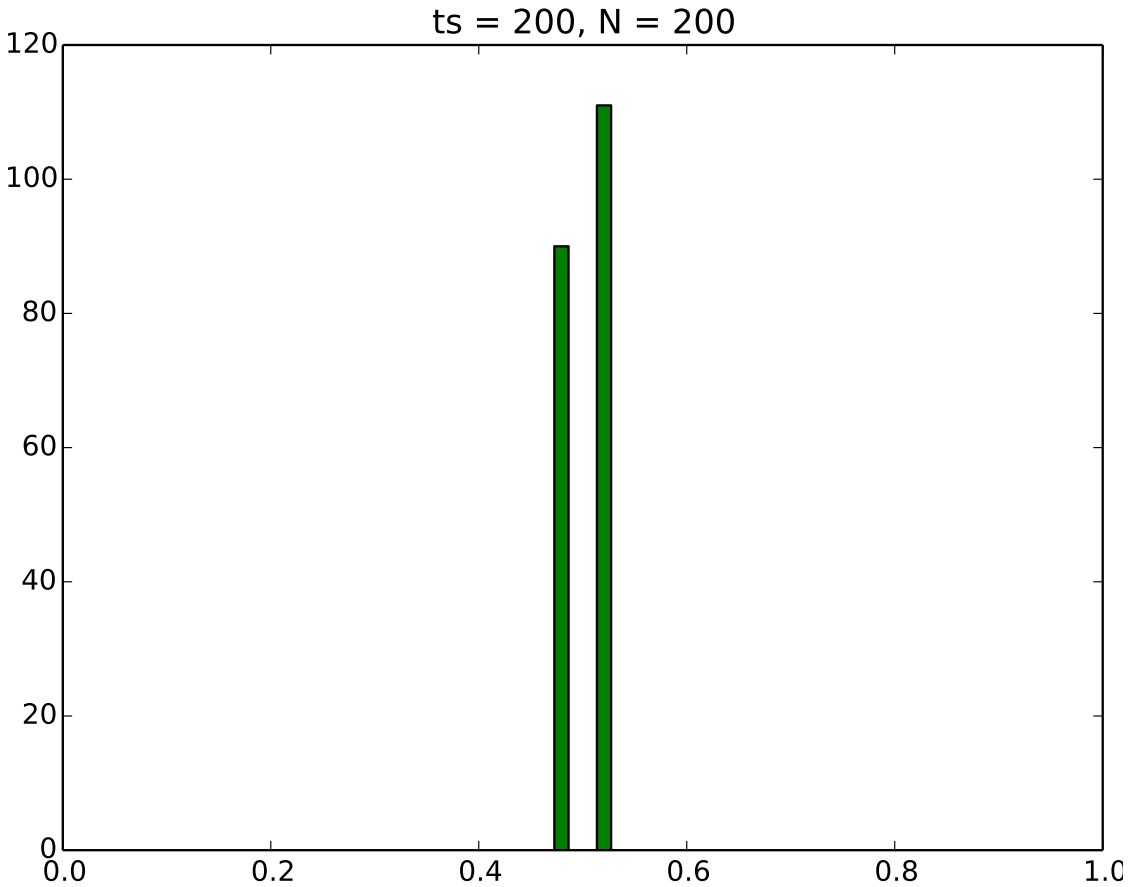

Figure : 200 iterations of Matching Pennis game for 200 times

2×2 coordination game: Transition of belief pattern 1

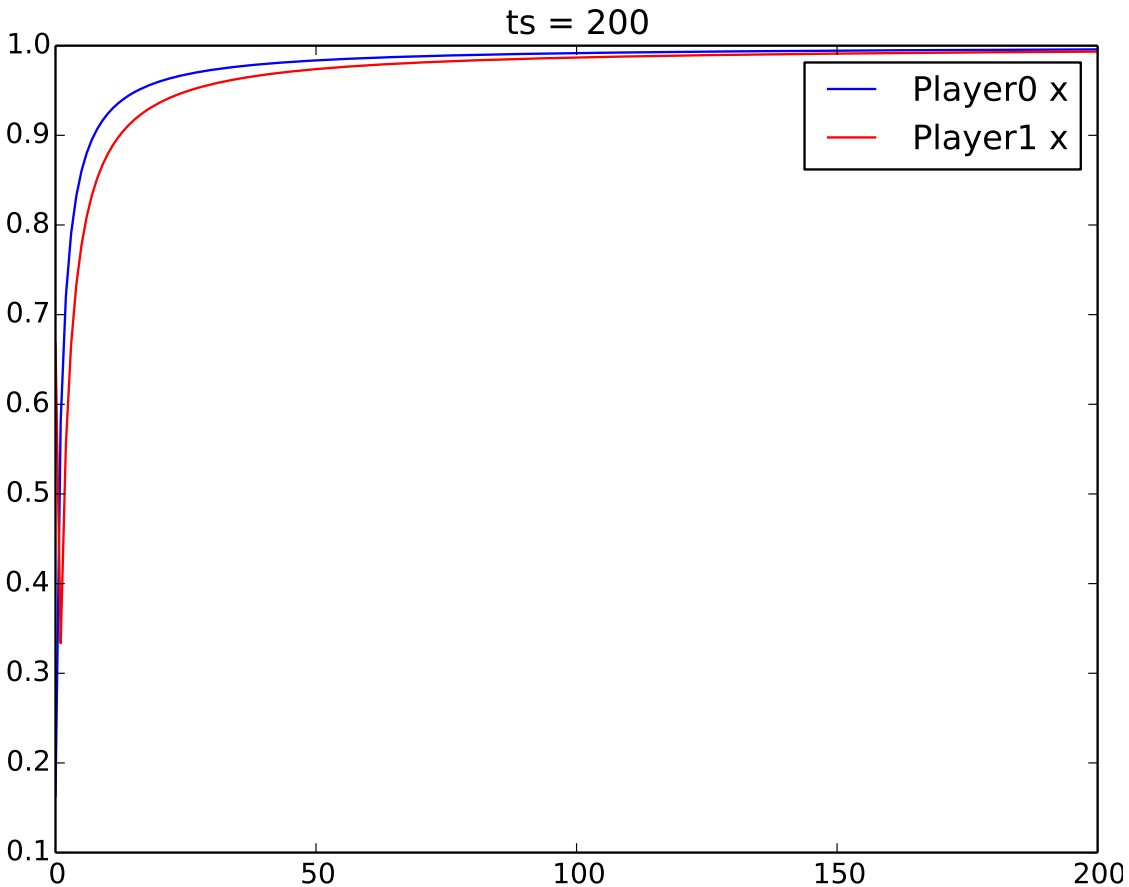

Figure : 2×2 coordination game for 200 times

2×2 coordination game: Transition of belief pattern 2

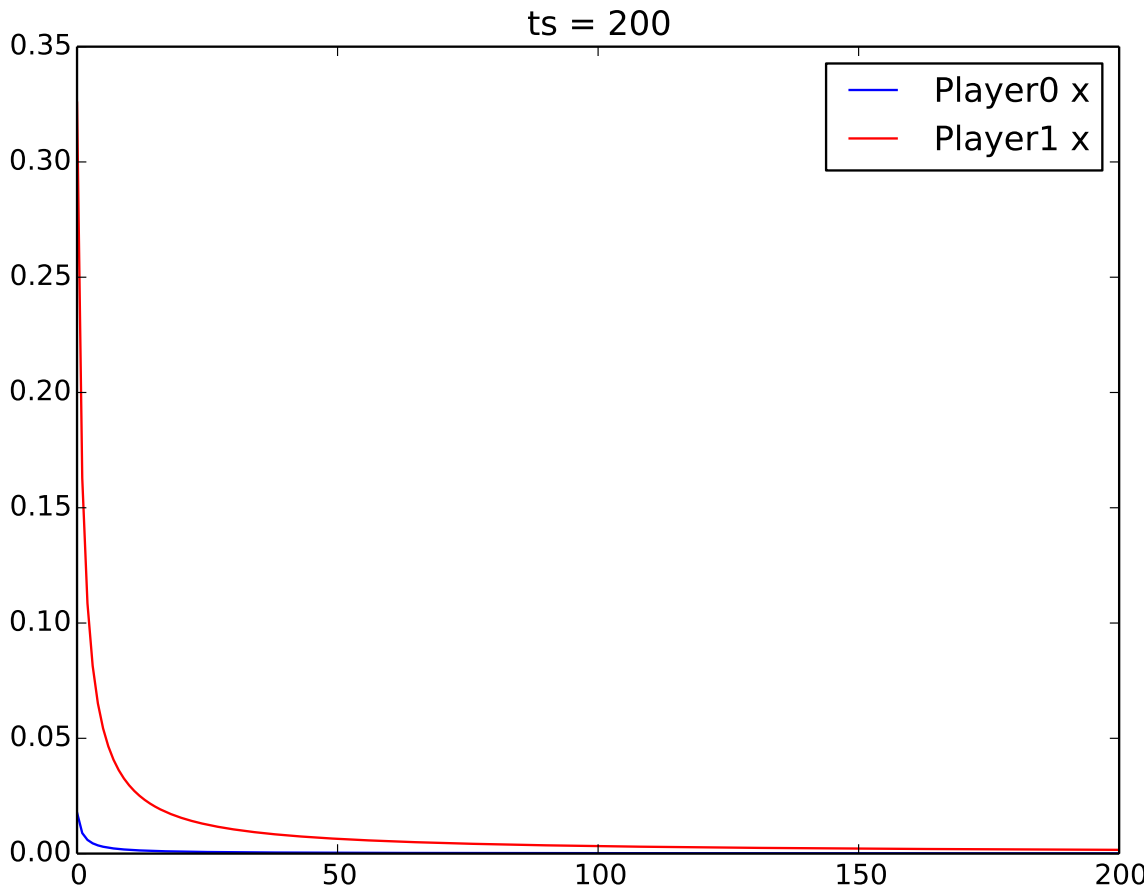

Figure : 2×2 coordination game for 200 times

2×2 coordination game: Histogram of the terminal belief

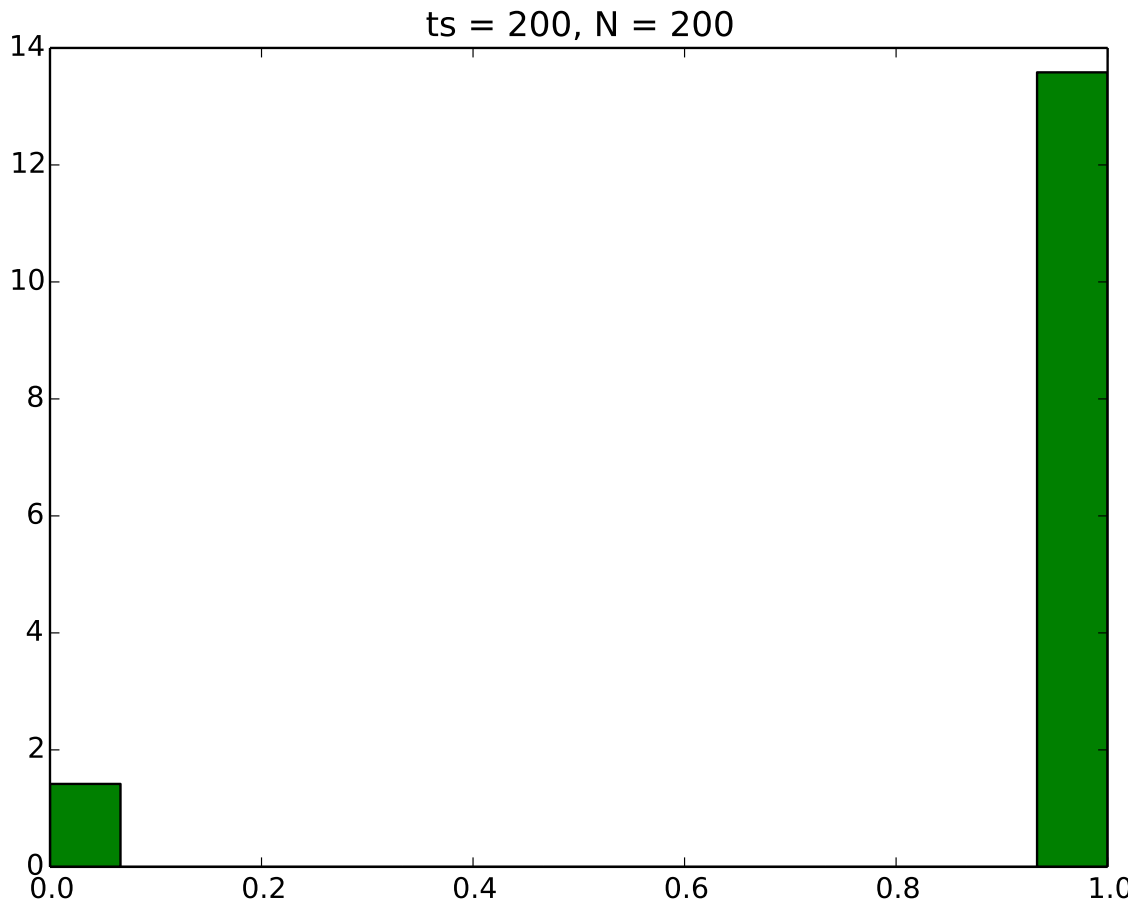

Figure : 200 iterations of 2×2 coordination game for 200 times

For further improvements

- OOP can be introduced. I intentionally often used functions so that the transition is smooth. (But not tried yet.)
- Introducing `for` loop for players is a bit clumsy. In the loop for p, I sometimes have to use p as a index for matrices, so end up with messy codes with tons of indexed matrices and vectors.