Python Program for Fictitious Play

Matching Pennies Game

Tanaka Masaki

The University of Tokyo

Introduction

- Fictitious Play のシミュレーションプログラム
- 使用言語は Python
- プレイするゲームは “Matching Pennis Game”

What's Fictitious Play?

- 戦略形ゲームが、$t = 1, 2, \ldots$ の各期にプレイされる
- 他のプレーヤーの戦略に対する予測
  - → t 期の他のプレーヤーは、1 期から $t-1$ 期に選択した分布に等しい確率で各純粋戦略を選択する
- この予測の下で、各プレイヤーがそれに対する最適反応戦略を選択
- このような動学モデルを "Fictitious Play" と呼ぶ。

What's Fictitious Play?

- t 期において、プレーヤー 0 は…
  - プレーヤー 1 が確率 $1 - x_0(t)$ で行動 0 をとる
  - プレーヤー 1 が確率 $x_0(t)$ で行動 1 をとる

  と予測しているものとする
- プレーヤー 1 に対しても同様
- この下で、期待利得が最大になるように行動を決定する (期待利得が等しい場合は等確率で選ぶ)
- なお、この $x_0(t)$ を「プレーヤー 0 の、t 時点におけるプレイヤー 1 の行動に関する信念 (belief)」と呼ぶこととする

What's Fictitious Play?

- 初期信念 $x_0(0)$ は [0,1] 上の一様分布に従ってランダムに与えられる。
- 各 $t \geq 1$ 時点において、プレーヤー 1 が過去とった行動を $a_1(0), \ldots, a_1(t-1)$ とすると、

$$x_0(t) = \frac{x_0(0) + a_1(0) + a_1(1) + \cdots + a_1(t-1)}{t+1} \qquad (1)$$

と書ける

What's Fictitious Play?

- ここで (1) 式は、

$$\begin{aligned} x_0(t+1) &= \frac{x_0(0)+a_1(0)+a_1(1)+\cdots+a_1(t)}{t+2} \\ &= \frac{(t+1)x_0(t)+a_1(t)}{t+2} \\ &= x_0(t)+\frac{1}{t+2}(a_1(t)-x_0(t)) \end{aligned}$$

と再帰的に書ける

What's Fictitious Play?

- つまり、

$$\begin{aligned} x_0(t+1) &= x_0(t) + \frac{1}{t+2}(a_1(t) - x_0(t)) \\ x_1(t+1) &= x_1(t) + \frac{1}{t+2}(a_0(t) - x_1(t)) \end{aligned}$$

  という連立 1 階漸化式を考えれば良いことになる。
- ただし、
  - $x_0(0), x_1(0) \sim Uniform[0,1]$
  - $a_0(t)$ は $x_0(t)$ に対する最適反応
  - $a_1(t)$ は $x_1(t)$ に対する最適反応
  - 最適反応が複数ある場合は等確率でランダムに選ぶ

Matching Pennis Game

- 次のような利得表を持つ "Matching Pennis Game" を例として考える

	行動 0	行動 1
行動 0	1,-1	-1,1
行動 1	-1,1	1,-1

Program Code

```
import matplotlib.pyplot as plt
import random

cycles = 2000
# number of cycles in one simulation
times = 100
# number of simulations
profits = [[1,-1], [-1, 1], [-1,1], [1, -1]]
# matrix of profits
x0 = random.uniform(0, 1)
# the beginning belief of player0
x1 = random.uniform(0, 1)
# the beginning belief of player1
num = 0
#saving png/pdf images
    with the name "fictitious_hist[num].png/pdf"
#if num = None, images won' t be saved
```

Program Code

```
last_x0_values = []
for t in range(times):
  x0_values = []
  x1_values = []
  for c in range(cycles):
    x0_values.append(x0)
    x1_values.append(x1)

    # decision of player0's act at c
    if profits[0][0]*(1-x0)+profits[1][0]*x0
         > profits[2][0]*(1-x0)+profits[3][0]*x0:
       a0 = 0
    elif profits[0][0]*(1-x0)+profits[1][0]*x0
        < profits[2][0]*(1-x0)+profits[3][0]*x0:
       a0 = 1
    else :
       a0 = random.choice([0,1])
```

Program Code

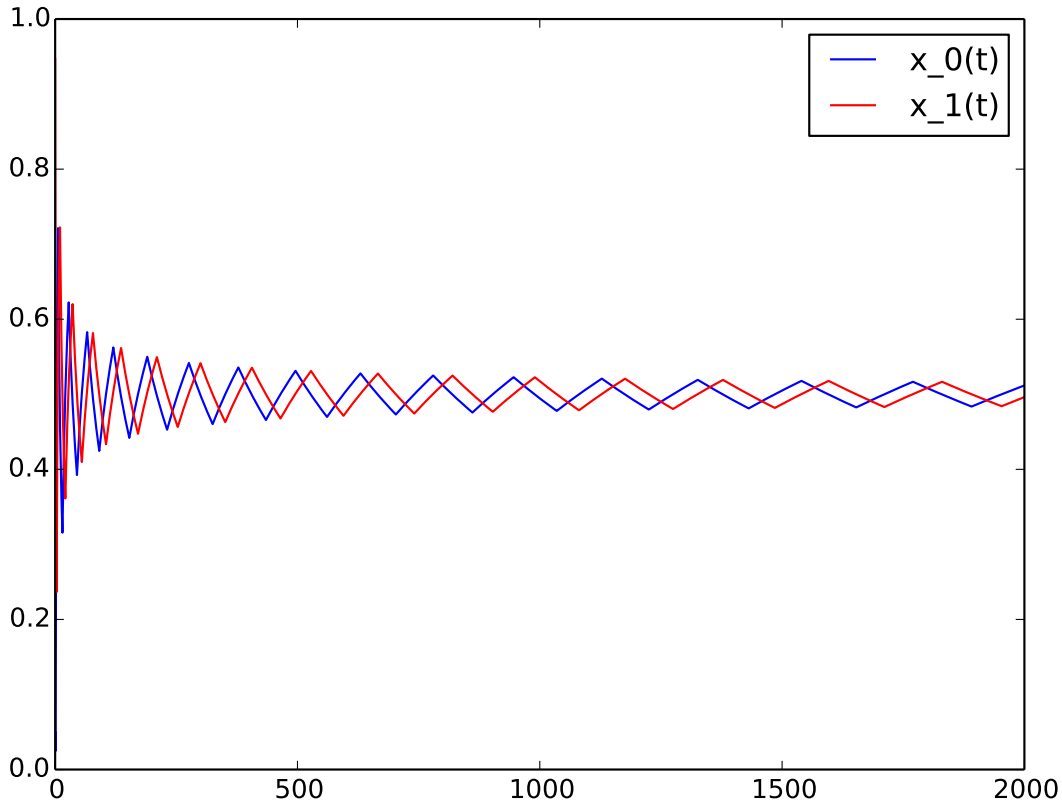
```
    # decision of player1's act at c
    if profits[0][1]*(1-x1)+profits[2][1]*x1
          > profits[1][1]*(1-x1)+profits[3][1]*x1:
      a1 = 0
    elif profits[0][1]*(1-x1)+profits[2][1]*x1
          < profits[1][1]*(1-x1)+profits[3][1]*x1:
      a1 = 1
    else :
      a1 = random.choice([0,1])

    if c< cycles :
       #updating beliefs
       x0 =  (a1 + x0*(t+1))/(t+2)
       x1 =  (a0 + x1*(t+1))/(t+2)
last_x0_values.append(x0)
```

Program Code

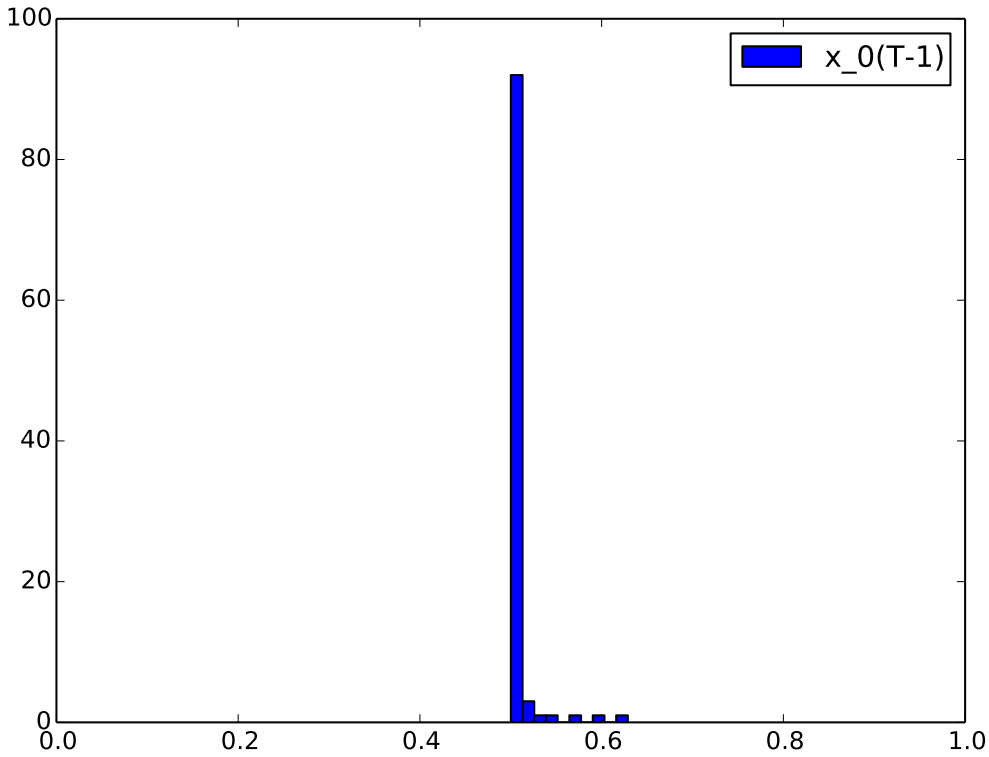
```
plt.subplot()
plt.hist(last_x0_values, facecolor='b'
                        , label='x_0(T-1)')
plt.xlim(0, 1)
plt.ylim(0, 100)
plt.legend()
if num != None:
    plt.savefig('fictitious_hist'
                        + str(num) + '.png')
    plt.savefig('fictitious_hist'
                        + str(num) + '.pdf')
plt.show()
```

Figures



- 両プレーヤーの信念がどのように推移しているか？
- 2000 サイクルの “Matching Pennis Game”

Figures



- プレーヤー 0 が最終的にそのような信念に辿り着いたのか？
- 2000 サイクルの “Matching Pennis Game” を 100 回行って集計

Explanation of the Code

- 次のサイクルを繰り返すことで、"Fictitious Play" を実現
  - 保持している信念をリストに登録
  - その信念をもとに期待利得を計算し、その期の行動を決定する
  - その期における相手の行動を踏まえて、信念を更新
- 保持している信念の登録を期首に持ってきている理由

  → 初期信念の抜け落ち・最終期に更新された信念の誤登録を防ぐため

Conclution : Further Improvements

- 利得表を行列として読み込ませるようにすれば、全体的にスッキリする。
  - 期待利得も行列の掛け算で計算可能
- 期待利得の計算を行って、その期の行動を決める過程は、関数化すべき