

Fictitious Play

山岸 敦

2014/6/28

構成

- ▶ Fictitious Play とは？
- ▶ シミュレーション結果の解説
 - ▶ Matching Pennies Game
 - ▶ Coordination Game
- ▶ Python コードの解説

Fictitious Play の解説

- ▶ Fictitious play では、相手の前回の行動により、「相手がどの手をどんな確率で出してくるか」についての予想（信念）が変化する状況が想定されます。
- ▶ さらにどの時点でも、プレイヤーは「その時点での自信の信念に照らして最適」な行動をします。このとき、各人の信念の動きはどうなるのでしょうか？
- ▶ 信念の推移を定式化すると、（導出は省略しますが）

$x_0(t)$ は

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

と再帰的に書くことができます。

Fictitious Play の解説

- ▶ Fictitious play では、相手の前回の行動により、「相手がどの手をどんな確率で出してくるか」についての予想（信念）が変化する状況が想定されます。
- ▶ さらにどの時点でも、プレイヤーは「その時点での自信の信念に照らして最適」な行動をします。このとき、各人の信念の動きはどうなるのでしょうか？
- ▶ 信念の推移を定式化すると、（導出は省略しますが）

$x_0(t)$ は

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

と再帰的に書くことができます。

Fictitious Play の解説

- ▶ Fictitious play では、相手の前回の行動により、「相手がどの手をどんな確率で出してくるか」についての予想（信念）が変化する状況が想定されます。
- ▶ さらにどの時点でも、プレイヤーは「その時点での自信の信念に照らして最適」な行動をします。このとき、各人の信念の動きはどうなるのでしょうか？
- ▶ 信念の推移を定式化すると、（導出は省略しますが）

$x_0(t)$ は

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

と再帰的に書くことができます。

Matching Pennies Game の解説

- ▶ Matching Pennies Game の利得行列は

$$\begin{pmatrix} (1, -1) & (-1, 1) \\ (-1, 1) & (1, -1) \end{pmatrix}$$

です。ナッシュ均衡は両戦略に確率 (0.5, 0.5) ずつ付与する混合戦略のみであることがわかります。

- ▶ お互いの信念が (0.5, 0.5) に収斂するならば、ナッシュ均衡が実現する、と考えてよいでしょう
- ▶ 本当にそうなるか、シミュレーションした結果を示します。

Matching Pennies Game の解説

- ▶ Matching Pennies Game の利得行列は

$$\begin{pmatrix} (1, -1) & (-1, 1) \\ (-1, 1) & (1, -1) \end{pmatrix}$$

です。ナッシュ均衡は両戦略に確率 (0.5, 0.5) ずつ付与する混合戦略のみであることがわかります。

- ▶ お互いの信念が (0.5, 0.5) に収斂するならば、ナッシュ均衡が実現する、と考えてよいでしょう
- ▶ 本当にそうなるか、シミュレーションした結果を示します。

Matching Pennies Game の解説

- ▶ Matching Pennies Game の利得行列は

$$\begin{pmatrix} (1, -1) & (-1, 1) \\ (-1, 1) & (1, -1) \end{pmatrix}$$

です。ナッシュ均衡は両戦略に確率 (0.5, 0.5) ずつ付与する混合戦略のみであることがわかります。

- ▶ お互いの信念が (0.5, 0.5) に収斂するならば、ナッシュ均衡が実現する、と考えてよいでしょう
- ▶ 本当にそうなるか、シミュレーションした結果を示します。

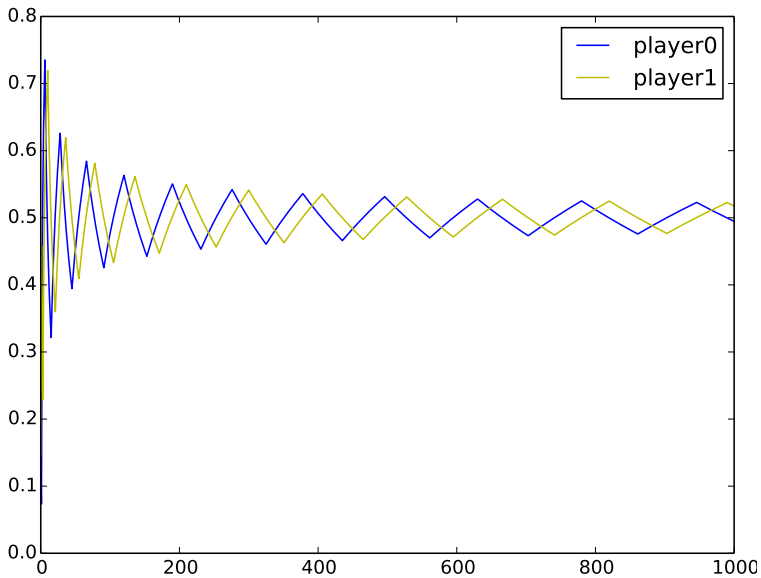


Figure : Matching Pennies Game

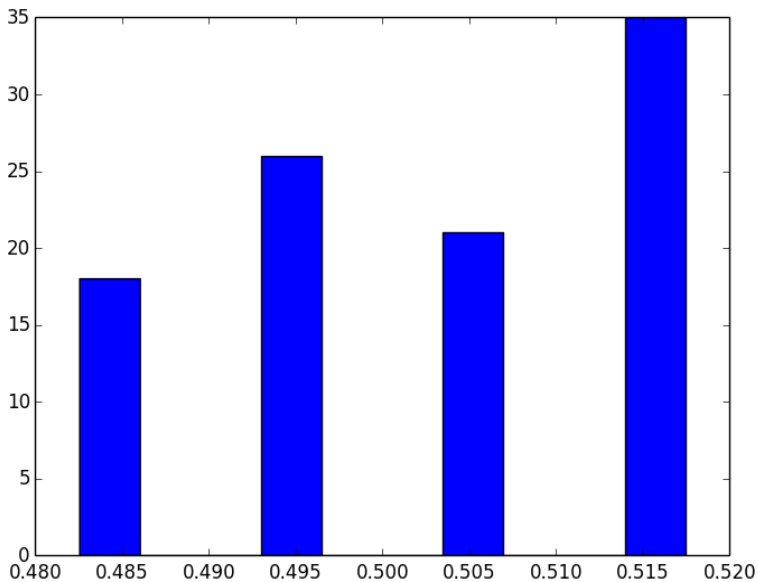


Figure : Matching Pennies Game

Coordination Game の解説

- ▶ Coordination Game の利得行列は

$$\begin{pmatrix} (4, 4) & (0, 3) \\ (3, 0) & (2, 2) \end{pmatrix}$$

です。ナッシュ均衡は純粋戦略の組 $(0,0)$ 、 $(1,1)$ および、2人とも確率 $(2/3, 1/3)$ ずつ付与する混合戦略の3つです。

- ▶ 先程とちがって、ナッシュ均衡が複数あります。このケースではどのようなプレイがなされるのでしょうか
- ▶ シミュレーションした結果を示します。

Coordination Game の解説

- ▶ Coordination Game の利得行列は

$$\begin{pmatrix} (4, 4) & (0, 3) \\ (3, 0) & (2, 2) \end{pmatrix}$$

です。ナッシュ均衡は純粋戦略の組 $(0,0)$ 、 $(1,1)$ および、2人とも確率 $(2/3, 1/3)$ ずつ付与する混合戦略の3つです。

- ▶ 先程とちがって、ナッシュ均衡が複数あります。このケースではどのようなプレイがなされるのでしょうか
- ▶ シミュレーションした結果を示します。

Coordination Game の解説

- ▶ Coordination Game の利得行列は

$$\begin{pmatrix} (4, 4) & (0, 3) \\ (3, 0) & (2, 2) \end{pmatrix}$$

です。ナッシュ均衡は純粋戦略の組 $(0,0)$ 、 $(1,1)$ および、2人とも確率 $(2/3, 1/3)$ ずつ付与する混合戦略の3つです。

- ▶ 先程とちがって、ナッシュ均衡が複数あります。このケースではどのようなプレイがなされるのでしょうか
- ▶ シミュレーションした結果を示します。

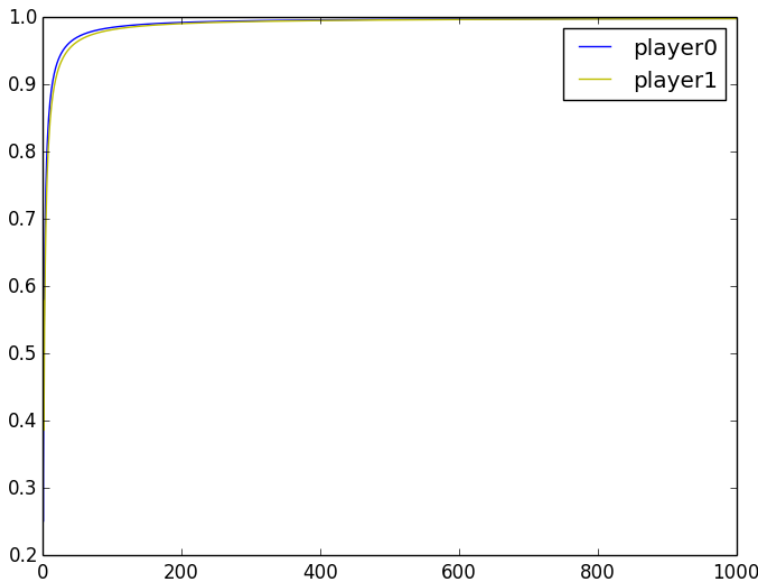
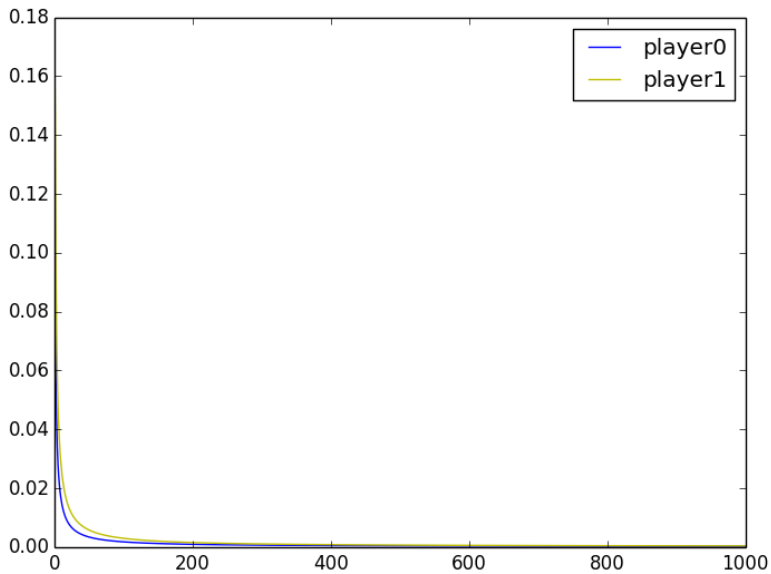


Figure : Coordination Game



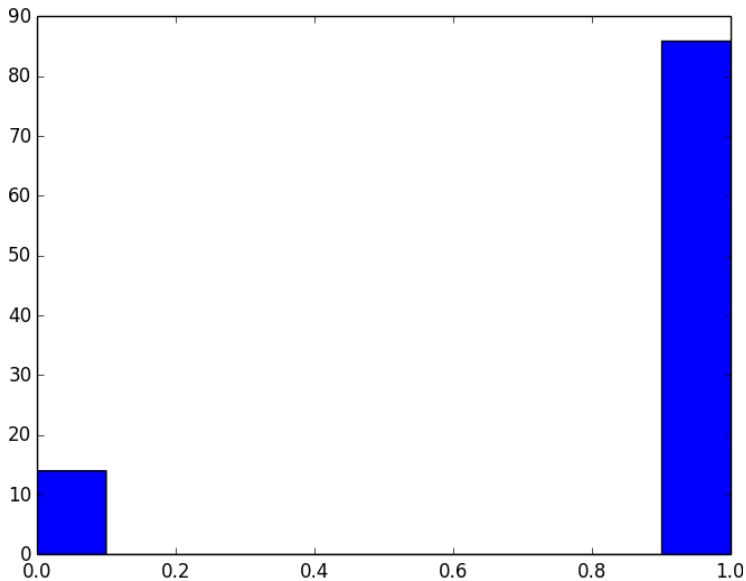


Figure : Coordination Game

補足: Coordination Game

- ▶ 各プレイヤーは「相手が $2/3$ より大きい確率で 0 をプレイするなら 0 を、それが $2/3$ より小さければ 1 を」必ず選択します。よって、混合戦略均衡で、たまたまお互いに 0 ないし 1 を取ればそちらの均衡に移るとわかります。ヒストグラムを眺めると、混合戦略に収斂した回数はゼロです。
- ▶ 「両者にとって、相手が均衡（純）戦略を取る確率が p 以上の時、自分もその均衡（純）戦略を取るのが最適である」ならば、その戦略の組み合わせは p -ドミナントと呼ばれます。
- ▶ $(0,0)$ は $2/3$ ドミナント、 $(1,1)$ は $1/3$ ドミナントです。一般に「相手が均衡（純）戦略を取る確率がより低くても最適」という意味で p が小さいほうが安全で起こりやすい均衡と予想されます。
- ▶ ヒストグラムを眺めると、 $(0,0)$ 均衡と比して $(1,1)$ 均衡がプレイされる頻度が圧倒的に大きいですが、これによりある（それなりにもっともらしい）信念形成過程を仮定した下で p ドミナントの性質が確認できた、と言えると思います。

補足: Coordination Game

- ▶ 各プレイヤーは「相手が $2/3$ より大きい確率で 0 をプレイするなら 0 を、それが $2/3$ より小さければ 1 を」必ず選択します。よって、混合戦略均衡で、たまたまお互いに 0 ないし 1 を取ればそちらの均衡に移るとわかります。ヒストグラムを眺めると、混合戦略に収斂した回数はゼロです。
- ▶ 「両者にとって、相手が均衡（純）戦略を取る確率が p 以上の時、自分もその均衡（純）戦略を取るのが最適である」ならば、その戦略の組み合わせは p -ドミナントと呼ばれます。
- ▶ $(0,0)$ は $2/3$ ドミナント、 $(1,1)$ は $1/3$ ドミナントです。一般に「相手が均衡（純）戦略を取る確率がより低くても最適」という意味で p が小さいほうが安全で起こりやすい均衡と予想されます。
- ▶ ヒストグラムを眺めると、 $(0,0)$ 均衡と比して $(1,1)$ 均衡がプレイされる頻度が圧倒的に大きいですが、これによりある（それなりにもっともらしい）信念形成過程を仮定した下で p ドミナントの性質が確認できた、と言えると思います。

補足: Coordination Game

- ▶ 各プレイヤーは「相手が $2/3$ より大きい確率で 0 をプレイするなら 0 を、それが $2/3$ より小さければ 1 を」必ず選択します。よって、混合戦略均衡で、たまたまお互いに 0 ないし 1 を取ればそちらの均衡に移るとわかります。ヒストグラムを眺めると、混合戦略に収斂した回数はゼロです。
- ▶ 「両者にとって、相手が均衡（純）戦略を取る確率が p 以上の時、自分もその均衡（純）戦略を取るのが最適である」ならば、その戦略の組み合わせは p -ドミナントと呼ばれます。
- ▶ $(0,0)$ は $2/3$ ドミナント、 $(1,1)$ は $1/3$ ドミナントです。一般に「相手が均衡（純）戦略を取る確率がより低くても最適」という意味で p が小さいほうが安全で起こりやすい均衡と予想されます。
- ▶ ヒストグラムを眺めると、 $(0,0)$ 均衡と比して $(1,1)$ 均衡がプレイされる頻度が圧倒的に大きいですが、これによりある（それなりにもっともらしい）信念形成過程を仮定した下で p ドミナントの性質が確認できた、と言えると思います。

補足: Coordination Game

- ▶ 各プレイヤーは「相手が $2/3$ より大きい確率で 0 をプレイするなら 0 を、それが $2/3$ より小さければ 1 を」必ず選択します。よって、混合戦略均衡で、たまたまお互いに 0 ないし 1 を取ればそちらの均衡に移るとわかります。ヒストグラムを眺めると、混合戦略に収斂した回数はゼロです。
- ▶ 「両者にとって、相手が均衡（純）戦略を取る確率が p 以上の時、自分もその均衡（純）戦略を取るのが最適である」ならば、その戦略の組み合わせは p -ドミナントと呼ばれます。
- ▶ $(0,0)$ は $2/3$ ドミナント、 $(1,1)$ は $1/3$ ドミナントです。一般に「相手が均衡（純）戦略を取る確率がより低くても最適」という意味で p が小さいほうが安全で起こりやすい均衡と予想されます。
- ▶ ヒストグラムを眺めると、 $(0,0)$ 均衡と比して $(1,1)$ 均衡がプレイされる頻度が圧倒的に大きいですが、これによりある（それなりにもっともらしい）信念形成過程を仮定した下で p ドミナントの性質が確認できた、と言えると思います。

Matching Pennies Game: コードの解説

- ▶ まずは必要な物を import し、利得を nparray で設定します。これを用いると後々期待利得の計算などがラクになります。

```
from __future__ import division
import matplotlib.pyplot as plt
import random
import numpy as np
```

```
#defining variables and functions that are useful
```

```
payoff_0 = np.array([[1,-1],[-1,1]])
payoff_1 = np.array([[1,-1],[-1,1]])
```

Matching Pennies Game: コードの解説

- ▶ 次に、必要な関数を定義していきます。ついでに、初期信念もここで設定しています。

```
def set_intbelief():  
    int_belief = random.uniform(0,1)  
    return np.array([1-int_belief,int_belief])  
    # belief about the opponent's actions  
  
belief0 = set_intbelief()  
belief1 = set_intbelief()  
  
def expected_value(payoff,beliefs):  
    return np.dot(payoff,beliefs)  
# returns expected values of each action as a vector
```

Matching Pennies Game: コードの解説

- ▶ 引き続き、必要な関数を定義していきます。あと、後にグラフを書くのに使うリスト Trajectory を設定しています。

```
def take_action(x)
: # this takes a vector as an argument

if x[0] > x[1]:
return 0
elif x[0] < x[1]:
return 1
else:
return random.randint(0,1)

# lists used later to draw the graph
trajectory0 = [belief0[1]]
trajectory1 = [belief1[1]]
```

Matching Pennies Game: コードの解説

- ▶ for 文で、ゲームをプレイ。信念の軌跡は Trajectory に保存

```
for i in range(1000):
```

```
    ev0 = expected_value(payoff_0,belief0)
```

```
    ev1 = expected_value(payoff_1,belief1)
```

```
    action0 = take_action(ev0)
```

```
    action1 = take_action(ev1)
```

```
    # updating beliefs
```

```
    m = belief0[1] + (action1 - belief0[1])/(i + 2)
```

```
    n = belief1[1] + (action0 - belief1[1])/(i + 2)
```

```
    belief0 = np.array([1-m,m])
```

```
    belief1 = np.array([1-n,n])
```

```
    trajectory0.append(belief0[1])
```

```
    trajectory1.append(belief1[1])
```


Matching Pennies Game: コードの解説

- ▶ 描画します。#を取ると画像が保存できます。

```
plt.plot(trajjectory0, 'b-', label='player0')
plt.plot(trajjectory1, 'y-', label='player1')
plt.legend()
#plt.savefig
#("fictitious_graph1.0.png"
, bbox_inches="tight", pad_inches=0)

plt.show()
```

- ▶ ヒストグラムもほぼ同様のプログラムです。このプログラムを、さらに for 文でメタ的に包み込む形になります。

まとめ

- ▶ 二種類のゲームをプレイさせて、どんな戦略の組が均衡になるか調べることができます。今回僕が試したゲーム以外では、永遠に均衡にたどり着かないケースもありそうです。
- ▶ ヒストグラムを描くときに for ループを二重にするという原始的手法を用いましたが、処理回数が指数的に増加しているのでもっさりしています。どう改善できるのでしょうか…
- ▶ 2人2戦略ゲームについては、それなりに一般性の高いコードが書けたと思います。しかし、ここから n 人、 n 戦略へと拡張するならば複雑性が増し、class を定義して整理していく必要が生まれそうです。
- ▶ 今回は、class については逆に複雑になる + 恩恵が薄い気がしたので避けましたが、もう少し複雑なバージョンを class で書いて将来的には使いこなせるように練習したい。

まとめ

- ▶ 二種類のゲームをプレイさせて、どんな戦略の組が均衡になるか調べることができます。今回僕が試したゲーム以外では、永遠に均衡にたどり着かないケースもありそうです。
- ▶ ヒストグラムを描くときに for ループを二重にするという原始的手法を用いましたが、処理回数が指数的に増加しているのでもっさりしています。どう改善できるのでしょうか…
- ▶ 2人2戦略ゲームについては、それなりに一般性の高いコードが書けたと思います。しかし、ここから n 人、 n 戦略へと拡張するならば複雑性が増し、class を定義して整理していく必要が生まれそうです。
- ▶ 今回は、class については逆に複雑になる + 恩恵が薄い気がしたので避けましたが、もう少し複雑なバージョンを class で書いて将来的には使いこなせるように練習したい。

まとめ

- ▶ 二種類のゲームをプレイさせて、どんな戦略の組が均衡になるか調べることができます。今回僕が試したゲーム以外では、永遠に均衡にたどり着かないケースもありそうです。
- ▶ ヒストグラムを描くときに for ループを二重にするという原始的手法を用いましたが、処理回数が指数的に増加しているのでもっさりしています。どう改善できるのでしょうか…
- ▶ 2人2戦略ゲームについては、それなりに一般性の高いコードが書けたと思います。しかし、ここから n 人、 n 戦略へと拡張するならば複雑性が増し、class を定義して整理していく必要が生まれそうです。
- ▶ 今回は、class については逆に複雑になる + 恩恵が薄い気がしたので避けましたが、もう少し複雑なバージョンを class で書いて将来的には使いこなせるように練習したい。

まとめ

- ▶ 二種類のゲームをプレイさせて、どんな戦略の組が均衡になるか調べることができます。今回僕が試したゲーム以外では、永遠に均衡にたどり着かないケースもありそうです。
- ▶ ヒストグラムを描くときに for ループを二重にするという原始的手法を用いましたが、処理回数が指数的に増加しているのでもっさりしています。どう改善できるのでしょうか…
- ▶ 2人2戦略ゲームについては、それなりに一般性の高いコードが書けたと思います。しかし、ここから n 人、 n 戦略へと拡張するならば複雑性が増し、class を定義して整理していく必要が生まれそうです。
- ▶ 今回は、class については逆に複雑になる + 恩恵が薄い気がしたので避けましたが、もう少し複雑なバージョンを class で書いて将来的には使いこなせるように練習したい。

まとめ

- ▶ 二種類のゲームをプレイさせて、どんな戦略の組が均衡になるか調べることができます。今回僕が試したゲーム以外では、永遠に均衡にたどり着かないケースもありそうです。
- ▶ ヒストグラムを描くときに for ループを二重にするという原始的手法を用いましたが、処理回数が指数的に増加しているのでもっさりしています。どう改善できるのでしょうか…
- ▶ 2人2戦略ゲームについては、それなりに一般性の高いコードが書けたと思います。しかし、ここから n 人、 n 戦略へと拡張するならば複雑性が増し、class を定義して整理していく必要が生まれそうです。
- ▶ 今回は、class については逆に複雑になる + 恩恵が薄い気がしたので避けましたが、もう少し複雑なバージョンを class で書いて将来的には使いこなせるように練習したい。