確率進化モデル

山岸　敦

東京大学経済学部３年・尾山ゼミ

2014 9/21 (ゼミ合宿)

発表の指針

- 確率進化モデル（KMR モデル・Ellison モデル）の分析、比較をします
- 「理論」の骨格を解説したのち、両者の違いをコンピューターシミュレーションによる「実験」の結果を示して実際に確かめます
- python コード全体の解説はしませんが、みなさまの参考になる？ かもしれない「発想」に関わる部分は抜き出して解説します
- 最後に、課題と今後の展望を示して締めくくります

KMR モデルのおさらい

- 夏学期のゼミで扱いましたが、念のため KMR モデル（逐次改訂・自分と対戦はしないバージョン）を復習します
- 100 人の学生がいるとします。彼らは windows を使うか、mac を使うかどちらかの戦略を取ります
- 集団内の各学生は次のような状況におかれていると仮定します
  - 生徒は、基本的にいままでの戦略を継続してプレーします
  - しかし、各 $t = 1, 2, 3...$ 期に一人だけ、「やる気」が出て戦略を変更しようとします（こういう状況を inertia ＝惰性が存在する状態といいます）
  - 現在の他人の戦略を見て、それに対して最適になるように戦略を変更します。将来どうなるか？ は考えません（myopic ＝近視眼的に戦略を変更）

KMR モデルのおさらい

- 行動変更の際、一定の確率で最適行動を取るのでなく、「実験」します
- ϵ の確率で、windows か mac かランダムに選ぶとします。つまり、$\frac{\epsilon}{2}$ の確率で（近視眼的な）最適行動とは逆の行動を取ります
- t を大きくして、ϵ を小さくした時どんな均衡に至るか？というのが問いです
- 先取りすれば 2×2 対称ゲームの場合、リスクドミナントな均衡に至る、というのが結論です

Table：ゲーム 1（鹿狩りゲーム）

学生 A/ 学生 B	mac	windows
mac	(4, 4)	(0, 3)
windows	(3, 0)	(2, 2)

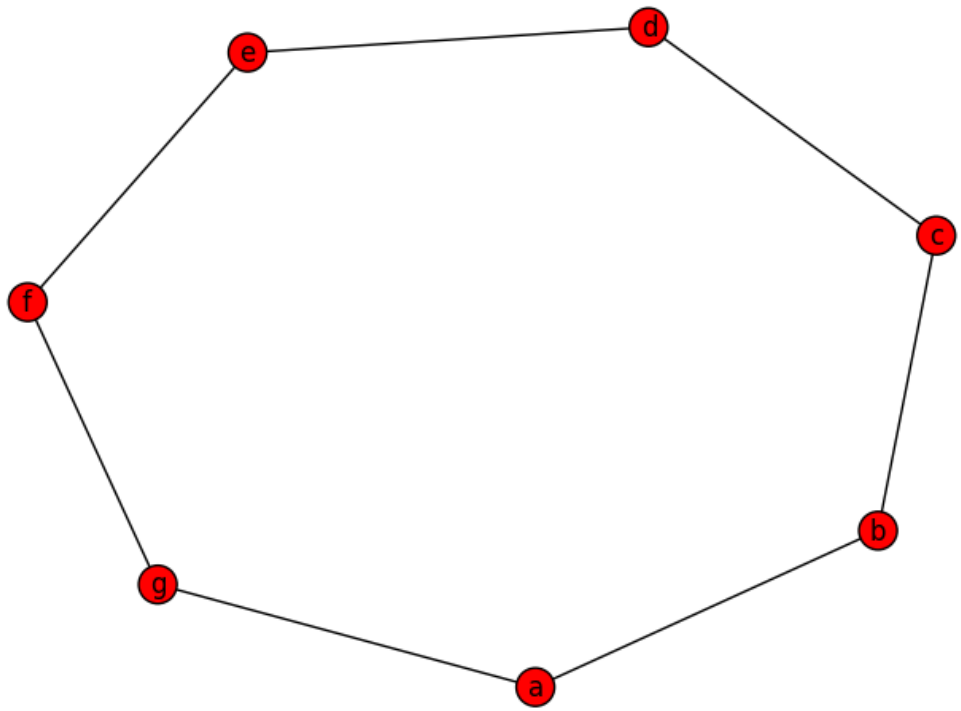
- (mac, mac) 均衡では、各人が行動を変えた時が失う利得は 1 である一方、(windows, windows) 均衡では 2
- よって、(windows, windows) 均衡の方が、安定していると考えられます。これを「(windows, windows) は (mac, mac) をリスク支配している」と言います
- (mac, mac) の方がパレート効率的なことに注意してください

- KMR 論文[1] によれば、t が十分大きく ϵ が限りなく 0 に近い時、確率 1 でリスク支配する均衡に収束することが知られています
- 数学的な証明はここでは省きますが、夏学期にプログラムを書いて実感したことと思います

[1] Kandori et al. (1993): "Learning, Mutation, and Long Run Equilibria in Games ," *Econometrica*, 61, 29-56.

Ellison モデル

- KMR では「自分以外の全員」と対戦していましたが、Ellison モデルでは「ご近所」としか対戦しません
- 典型例として、下図のように円形のネットワークを想定します。各人はお隣さんとしか対戦しません



- Ellison の論文[2] によれば、t が十分大きく ϵ が限りなく 0 に近い時、KMR モデル同様確率 1 でリスク支配する均衡に収束することが知られています
- しかし、KMR よりも「収束がずっと早い」という特徴があります
- そのことを直感的に説明してみます

[2]Ellison,G. (1993): "Learning, Local Interaction,and Coordination ," *Econometrica*, 61, 1047-1071.

Table：ゲーム 1（鹿狩りゲーム）

学生 A/ 学生 B	mac	windows
mac	(4, 4)	(0, 3)
windows	(3, 0)	(2, 2)

- このゲームをもう一度見てみましょう。集団内で mac 戦略を取る人の割合を x とすると、mac の期待利得は $4x$、windows の期待利得は $3x + 2(1 - x) = 2 + x$ です。
- （厳密には、自分がどっちの戦略かにも依りますが）だいたい $x \geq 2/3$ なら mac が最適に、$x \leq 2/3$ なら window が最適になります。
- 仮にこの集団が 10000 人とすれば、同時に 3334 人以上が実験しないと（ϵ^{3334} のオーダー）、(mac,mac) から (windows,windows) になりません
- (windows,windows) にいつか至ることは証明されているのですが、大集団では収束が非常に遅いのです

- 一方、Ellison モデルではこれが解決されています。mac 戦略を M、windows 戦略を W と表し、初期状態が [M,M,M,……M] とします。
- ここで、[M,M,W,M,M…M] と、一人だけ W に変わったとします。
- すると、W の両隣の M を取る人にとっては W が最適になります。一度彼らが W に変われば、そのまた隣も W が最適になり、そのまた隣も…
- このように、Ellison モデルでは小さな「実験」が集団全体に伝播するメカニズムが備わっているのです。
- これを、プログラムを書いて確認してみましょう

```
In [2]: x = ellison22_coordination(N=101,n=50)

In [8]: for i in range(101):
            x.players[i].action=0

In [10]: x.play(X=1000000,epsilon=0.01)

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure : シミュレーション結果:KMR

```
In [5]: y = ellison22_coordination(N=101,n=1)

In [6]: for i in range(101):
            y.players[i].action=0

In [7]: y.play(K=10000,epsilon=0.01)

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Figure： シミュレーション結果:Ellison

Young ゲーム

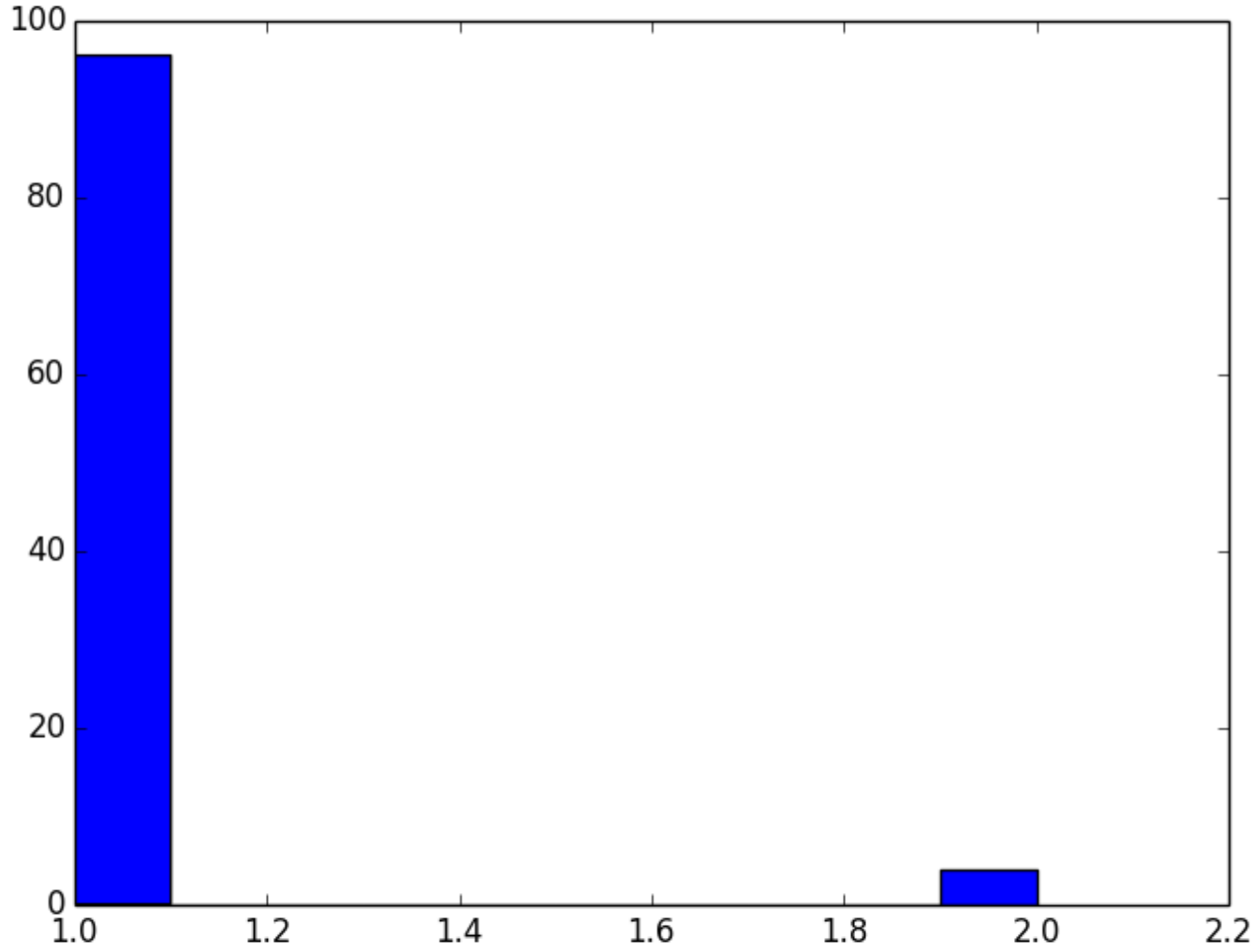
- KMR(1993) では、議論は 2×2 ゲームに絞られていました
- しかし、KMR と近親関係にあるモデルを扱った Young の論文 [3] では、興味深い 3×3 ゲームが取り扱われています (Young ゲーム)
- リスクドミナントな均衡があるのに、それに KMR モデルでは収束しない例です

[3] Young,P. (1993): "The Evolution of Conventions ," *Econometrica* 61, 57-84.

Table : ゲーム 2（Young ゲーム）

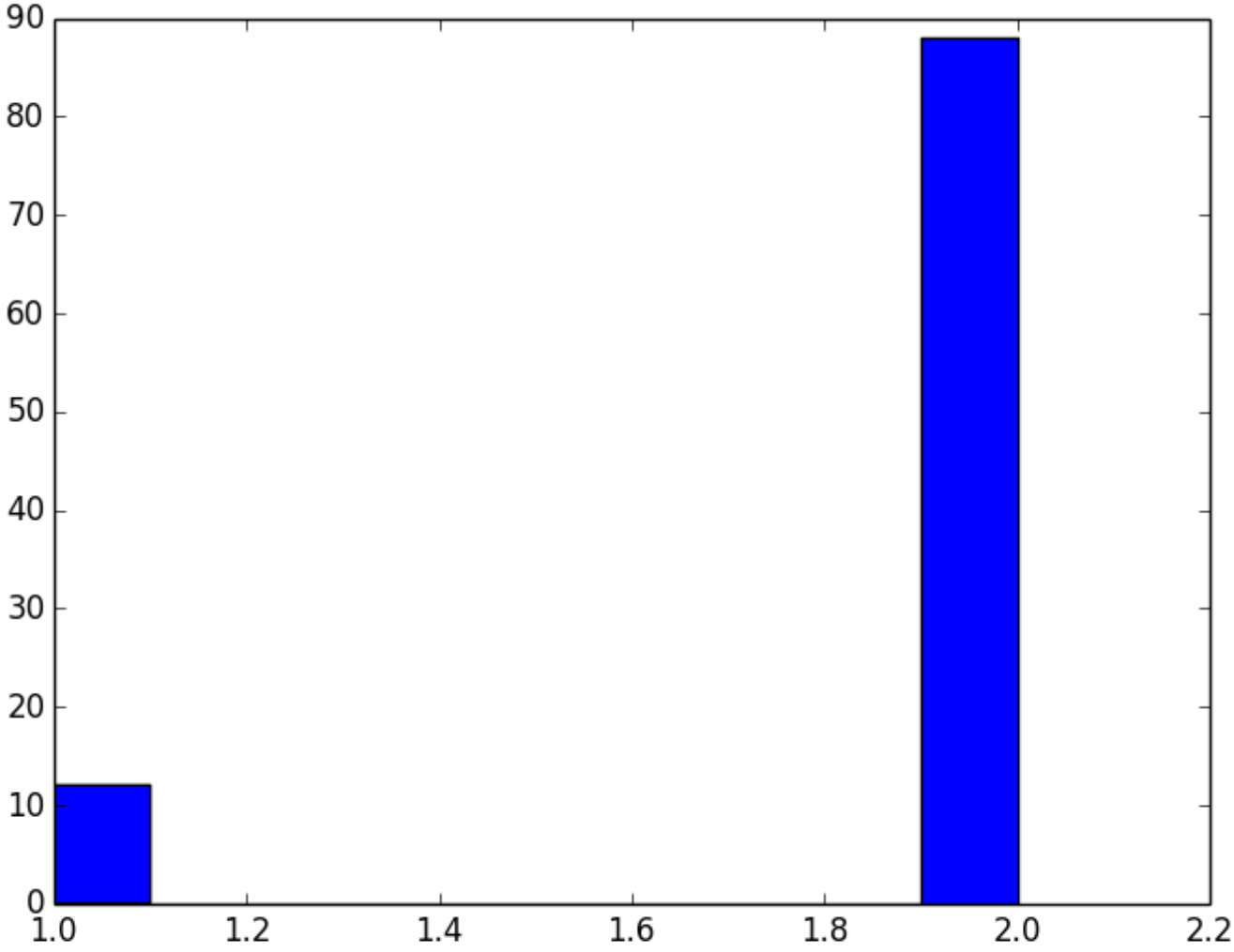
1/ 2	A	B	C
A	(6, 6)	(0, 5)	(0, 0)
B	(5, 0)	(7, 7)	(5, 5)
C	(0, 0)	(5, 5)	(8, 8)

- リスクドミナントは「一対一」の関係でしか定義されないので、2×2 ゲームに落として考えます
- B をないものとして A と C を比べると C は A をリスク支配します。同様にして B も支配するので、このゲームのリスクドミナントな均衡は (C, C) です
- ところが、（2×2 ゲームではかならずそうであったはずの）KMR ではこれに収束しません



Figure： シミュレーション結果:KMR(11 人集団：100000 期後 $\epsilon = 0.01$)

- 一方、Ellison モデルでは (C, C) 均衡に収束します！！
- A 戦略は、両隣が A を取っていない限り最適にならないので真っ先に淘汰されます
- すると B と C だけの集団になりますが、２戦略ではリスク支配する均衡に収束することはすでに見たとおりです
- よって (C, C に収束します)
- （厳密に、どうしてそうなのかはよくわかっていません）



Figure： シミュレーション結果:Ellison（同設定。両隣と対戦）

補足：KMR の結果について

- Kandori and Rob[4] では、より広いクラスのゲームに対し考察が行われました。
- そこでは、ゲームが"MBP"という性質を満たすことがリスクドミナント均衡に必ず収束するための十分条件だと示されました
- このゲームは MBP を満たさないので、リスクドミナントな均衡に必ず収束する保証はなかったのです

[4]Kandori, M and Rob, R(1998): "Bandwagon Effects and Long Run Technology Choice ," *Games and Economic Behavior* 22, 30-60.

補足：KMR の結果について

- MBP の定義を数式で書くと
  $u_{ij} - u_{ji} > u_{ik} - u_{jk}$ for $i \neq j \neq k$
  （u_{ij} は、戦略 j に対して戦略 i のもたらす利得を示します）
- 言葉では、「i の j に対するアドバンテージは、自分と戦うとき最大化される」と言えます
- このゲームは MBP を満たさないので、リスクドミナントな均衡に必ず収束する保証はなかったのです

Table：ゲーム 3（Young ゲーム改）

1/ 2	A	B	C
A	(6, 6)	(0, 5)	(0, 4)
B	(5, 0)	(7, 7)	(5, 5)
C	(4, 0)	(5, 5)	(8, 8)

- ちょっと利得をいじって、こうすると MBP を満たすようになります

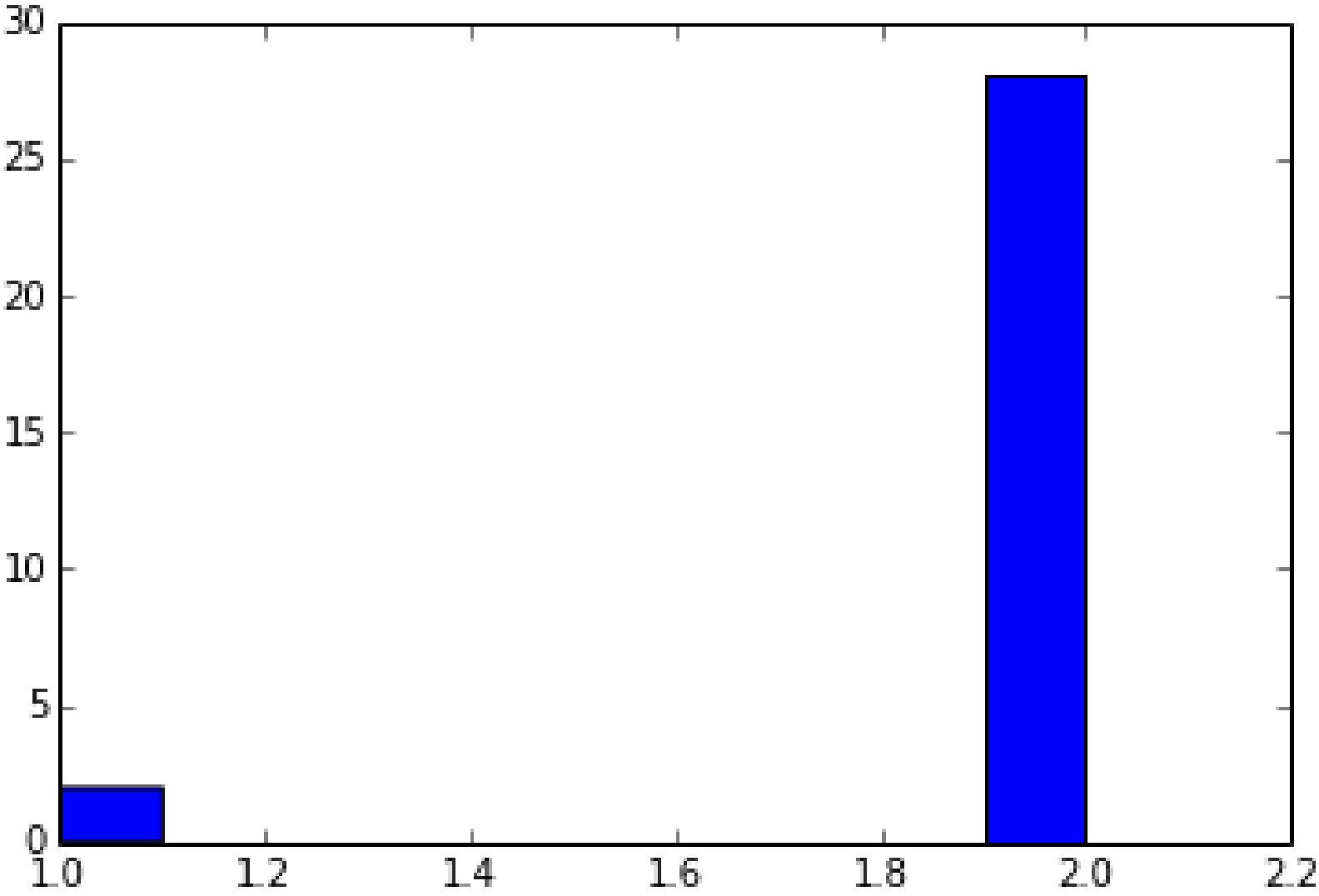


Figure : シミュレーション結果:Young ゲーム改 (7 人集団、5000000 期後、$\epsilon = 0.01$)

プログラム解説

- では、簡単にコードの解説をしようと思います
- 全文やるのは大変なので、要点だけにします

プログラム解説 1

- 各プレイヤーを表すのに、「Player」というクラスを利用してみました
- 夏学期最後に扱った”Schelling’s Segregation Model”のコードの発想を拝借しました

```
class Agent:

    def __init__(self, type):
        self.type = type
        self.draw_location()
...
agents = [Agent(0) for i in range(num_of_type_0)]
```

- 同じ発想で…

```
class Player:
    def __init__(self):
        self.init_action()

class ellison33(Player):
 def __init__(self, N=10, n=1, payoffs]):
      self.players = [Player() for i in range(N)]
```

- ここで、「クラス継承」というものを使ってみました。（見よう見まねであんまりよくわかってません…）

プログラム解説 2

- 繰り返しを避ける設計思想を採用しています。例えば…

```
def update_rational(self):
 # function used when a player is "rational"

def update_irrational(self):
# function used when a player is irrational

def play(self, X=10000, epsilon=0):
     for i in range(X):
         if random.uniform(0, 1) > epsilon:
             self.update_rational()
         else:
             self.update_irrational()
```

- 「合理的モード」と「非合理的モード」の行動を別々の関数にしたので、ゲームをプレーさせる関数たちはシンプルになりました

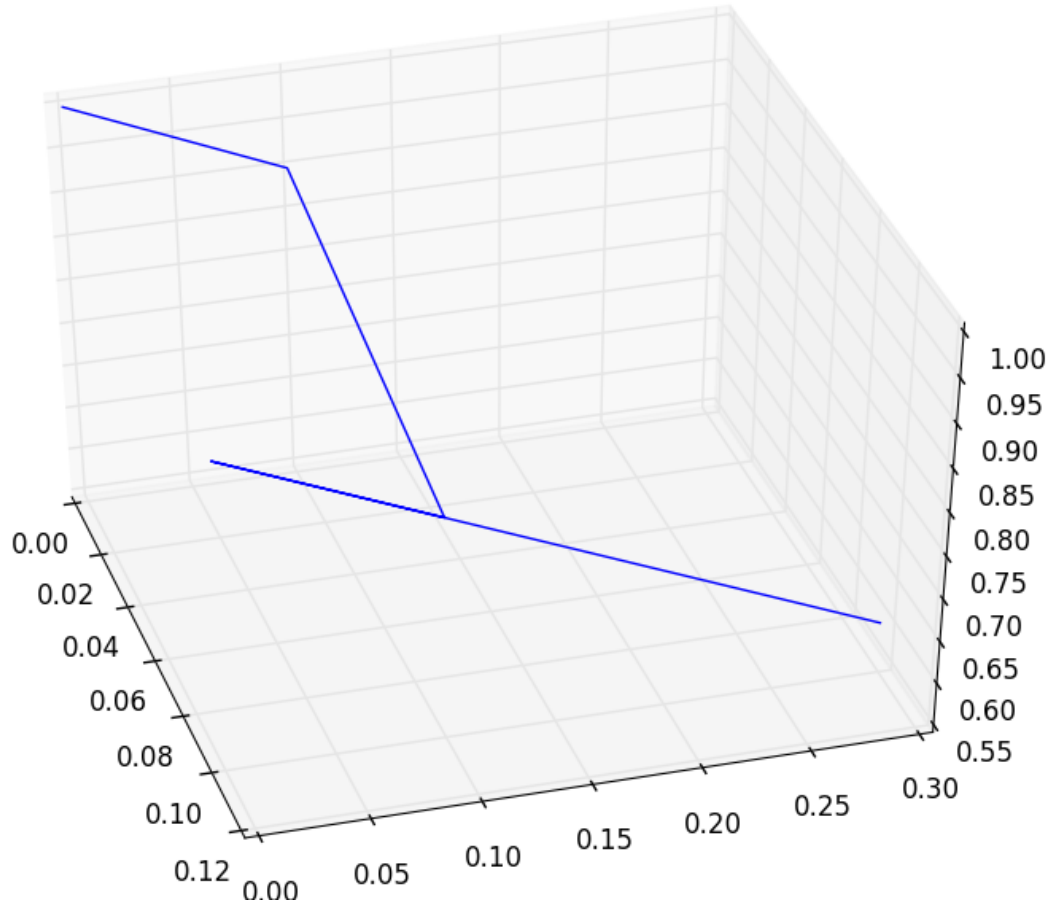
プログラム解説 3

- このプログラムの良くないところも挙げておきます（今後なんとか改善します）

```
if ev0[0] > ev1[0] and ev0[0] > ev2[0]:
  self.players[d].action = 0

elif ev1[0] > ev0[0] and ev1[0] > ev2[0]:
  self.players[d].action = 1

elif ev2[0] > ev0[0] and ev2[0] > ev1[0]:
  self.players[d].action = 2

elif ev1[0] == ev0[0] and ev1[0] > ev2[0]:
 self.players[d].action = random.choice([0, 1])
...
```

- 期待利得を計算して行動を決定する場面ですが、if 文の羅列でださい上に、3 戦略でないバージョンを作るときに描き直さないといけません。

- 視覚化がうまくいっていません
- 「状態の推移」を表したグラフ（のつもり）なのですが、なんのことやら…

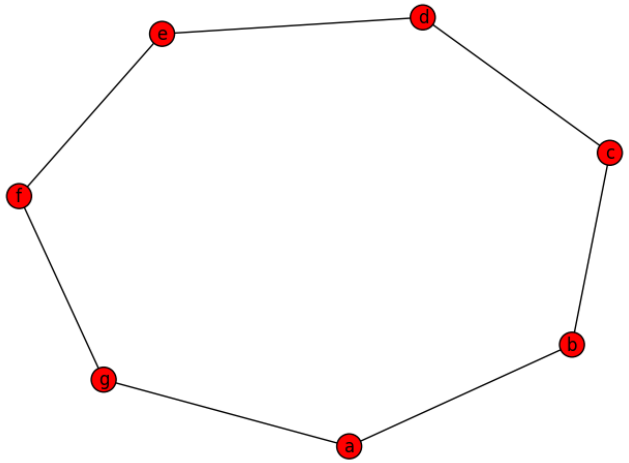


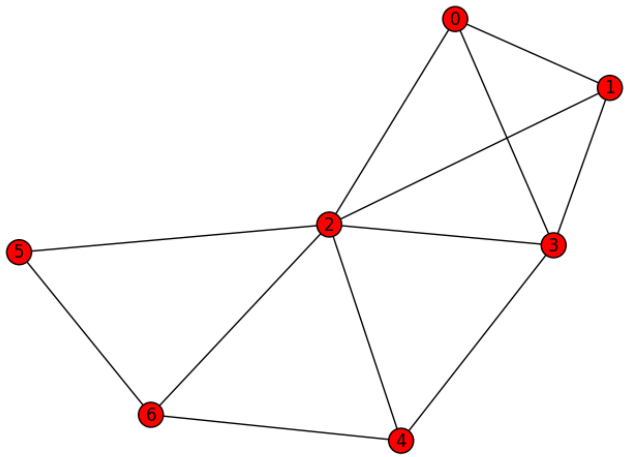
- うまく視覚化できるように、共闘してくれる方歓迎です

今後の方針、課題

- 今のプログラムの説明でも触れましたが、より一般的でかつ分析に使いやすいプログラムを目指さなくてはいけません
- 今後何を分析するか？ に関しては
  - 別の利得表についても KMR と Ellison の比較をする
  - さらに別の確率進化モデルについても考える
  - Ellison モデルで想定する「円形」でない、より複雑なネットワークについても考える
- 3 つ目の点をちょっとだけ掘り下げて、この発表を終わりたいと思います

ネットワークの複雑さを導入





- この話は「ネットワーク班」とかなり被ってきます（つながりが見えてきた？）
- 連携してより複雑（だけど現実に近い）挑戦していきたいところです

まとめ

- KMR は集団全体と、Ellison は「ご近所」だけと対戦
- この違いは、以下の点に明確に影響してくる
  - 収束の速度
  - 収束先（３戦略以上のとき ex.Young ゲーム）
- Ellison モデルを「円形ネットワーク」と捉えた時、拡張の方向性が見える