Game Theory 2018 Fall Final Project
进化博弈中的复制者动态实验探究

张义飞 — 201821080630
yidadaa@qq.com

UESTC — November 27, 2018

1 进化博弈论的起源

进化博弈论最初由R.A. Fisher在尝试解释哺乳动物群体的性别比例总是趋于相等时提出的。Fisher当时面临的课题是：为什么在某些动物群体中大多数雄性不参与交配的情况下，群体的性别比依然保持相等？在这些物种中，不交配的雄性似乎是群体的累赘，对物种的进化没有积极作用。Fisher意识到，如果我们用后代（孙子辈）来衡量个体适应性，那么个体的适应性将取决于人口中雄性和雌性的分布。Fisher指出，这种情况下，进化动态将会使固定在相等数量的雄性和雌性上。也就是说，个体适应性依赖与群体中男女的相对频率，这为进化提供了战略要素。

Fisher的论证在理论上可以理解为博弈，但是他并没有阐明。1961年，R.C. Lewontin在《进化与博弈》中首次明确地将博弈论与进化生物学结合起来。1972年，Maynard Smith在《博弈论与进化对抗》中定义了**进化稳定策略**（Evolutionarily Stable Strategy，简称ESS）。后来，Maynard Smith and Price的《动物冲突的逻辑》使得ESS的概念广为人知。从此，越来越多的经济学家和社会学家展开了对进化博弈论的研究。

2 进化博弈论的主要方法

进化博弈论中有两种方法。第一种方法由Maynard Smith and Price在使用ESS进行理论分析时提出；第二种方法则构造了一个与人口中的策略频率相关的显性模型，并且研究了模型中的进化动力学属性。

第一种方法提供了研究进化稳定性的静态方法，之所以称之为“静态”，是因为即使给出了进化稳定性的定义，其并不会随着群体的行为改变而改变。与之相反，第二种方法并不直接定义进化稳定性，一旦确定了群体演化模型，那么所有用来研究动态系统的标准稳定性概念都可以直接应用其中。

2.1 进化稳定性的定义

我们使用鹰-鸽博弈（Hawk-Dove Game）作为第一中方法的例子。在这种博弈框架下，两种个体同时竞争总量固定为V的资源。每个个体都使用如下描述的策略：

	鹰	鸽子
鹰	$1/2(V-C)$	V
鸽子	0	$V/2$

Table 1: 鹰-鸽博弈的支付矩阵

- **鹰** 使用攻击性策略，除非受伤或者胜利，否则不会停止攻击行为；
- **鸽** 如果对方攻击，则立即撤退。

我们作出如下假设：

1 如果两个个体均采用攻击策略，则冲突始终会结束，并且两者受伤概率一致；

2 冲突会引起个体适应度减少恒定值C；

3 当鹰遇见鸽子，鹰会获得全部资源；

4 两只鸽子相遇，平分资源。

则支付矩阵如表1所示。

如果一个策略是进化稳定的，则当群体中的所有个体都完全遵循其策略时，没有突变体（使用新策略的个体）能成功入侵。下面给出形式化表述：使用$\Delta F(s_1, s_2)$表示遵循策略s_1的个体与遵循策略s_2的个体竞争时的个体适应性变化量；使用$F(s)$表示遵循策略s的群体适应性之和；假设每种群体拥有初始适应性F_0。如果使用σ表示进化稳定策略，而μ表示突变体，则有：

$$\begin{aligned} F(\sigma) &= F_0 + (1-p)\Delta F(\sigma,\sigma) + p\Delta F(\sigma,\mu) \\ F(\sigma) &= F_0 + (1-p)\Delta F(\sigma,\sigma) + p\Delta F(\sigma,\mu) \end{aligned} \tag{1}$$

其中p表示群体中突变体的比例。

由于σ是进化稳定的，则遵循σ的个体适应性一定大于遵循的μ的个体，则有$F(\sigma) > F(\mu)$。由于p很小，则还需满足以下条件中的一个：

$$\begin{aligned} &\Delta F(\sigma,\sigma) > \Delta F(\mu,\sigma) \\ &\Delta F(\sigma,\sigma) = \Delta F(\mu,\sigma) \text{且} \Delta F(\sigma,\mu) > \Delta F(\mu,\mu) \end{aligned} \tag{2}$$

即在与原生策略群体对抗时，原生策略群体一定能获得比突变体更高的收益；如果两者收益相等，则

	合作	背叛
合作	(R, R')	(S, T')
背叛	(T, S')	(P, P')

Table 2: 囚徒困境的支付矩阵

在与突变体对抗时，原生群体也会比突变体获得更高的收益。

通过给出进化稳定策略的形式化表述，我们可以确信，在鹰-鸽游戏中，鸽子的策略不是进化稳定的，因为鸽子群体可以被鹰策略入侵。如果竞争资源V大于受伤代价C，则鹰策略是进化稳定的。

2.2 为群体指定动态性

我们使用著名的囚徒困境来作为第二种方法的例子。在囚徒困境中，每个个体都可以选择“合作”和“背叛”两种策略，表2给出了囚徒困境的一般形式的支付矩阵。其中$T > R > P > S$且$T' > R' > P' > S'$。假定此支付矩阵在群体中保持一致。

如果群体中个体持续与他人进行博弈，那么群体中使用各策略的个体比例将会如何变化？我们可以从自然群体中的引入以下假设：首先，我们假设群体足够大，这样可以使用合作和背叛的个体比例来描述群体状态，我们使用p_c和p_d来分别描述合作者和背叛者的比例；然后，我们使用W_C和W_D来表示合作者和背叛者群体的平均适应性，并使用$\bar{W}$表示整个群体的平均适应性。则W_C和W_D以及$\bar{W}$表示如下：

$$\begin{aligned} W_C &= F_0 + p_c \Delta F(C, C) + p_d \Delta F(C, D) \\ W_D &= F_0 + p_c \Delta F(D, C) + p_d \Delta F(D, D) \\ \bar{W} &= p_c W_C + p_d W_D \end{aligned} \tag{3}$$

然后，我们假设下一代群体中合作者和背叛者的分布依赖于当前代中合作者和背叛者的分布：

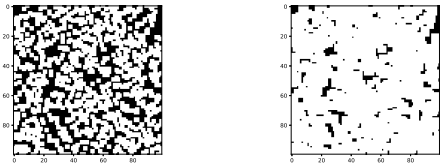
$$\begin{aligned} p_c' &= \frac{p_c W_C}{\bar{W}} \\ p_d' &= \frac{p_d W_D}{\bar{W}} \end{aligned} \tag{4}$$

重写以上公式为：

$$\begin{aligned} (p_c' - p_c) &= \frac{p_c W_C - \bar{W}}{\bar{W}} \\ (p_d' - p_d) &= \frac{p_d W_D - \bar{W}}{\bar{W}} \end{aligned} \tag{5}$$

如果我们假设每一代之间，各个策略占比变化很小，则得到微分方程：

$$\begin{aligned} \frac{dp_c}{dt} &= \frac{p_c W_C - \bar{W}}{\bar{W}} \\ \frac{dp_d}{dt} &= \frac{p_d W_D - \bar{W}}{\bar{W}} \end{aligned} \tag{6}$$



(a) $round = 0$ (b) $round = 1$

(c) $round = 2$ (d) $round = 4$

Figure 1: $T = 2.80, P = 0.10, R = 1.10, S = 0.00$

由W_C和W_D的构成：

$$\begin{aligned} W_C &= F_0 + p_c R + p_d S \\ W_D &= F_0 + p_c T + p_d P \end{aligned} \tag{7}$$

且由于$T > R, P > S$，则有$W_D > \bar{W} > W_C$，则可以得到：

$$\begin{aligned} \frac{W_D - \bar{W}}{\bar{W}} &> 0 \\ \frac{W_C - \bar{W}}{\bar{W}} &< 0 \end{aligned} \tag{8}$$

即随着时间推移，群体中背叛者的数量会逐渐增加，而合作者的数量则会逐渐减少。

3 复制者动态的实验模拟

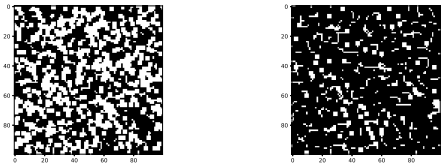
前文中，我们已经介绍了进化博弈论中的一些重要概念，本节我们在此概念的基础上进行实验模拟。

在群体演化过程中，个体策略的更新使用复制者动态进行更新，即每个个体会在每一轮博弈结束后，观察其邻近个体的收益，并选择收益最高的邻居的策略作为其下一轮博弈的策略。

3.1 囚徒困境的博弈模拟

根据2.2中描述的囚徒困境博弈，我们使用棋盘格局来模拟每个个体之间的博弈，在100×100的棋盘中，每个个体与其周围的8个邻居（边界上的个体的邻居会少于8）进行博弈，然后根据支付矩阵获得收益，每轮博弈后，每个个体使用复制者动态更新自身策略。

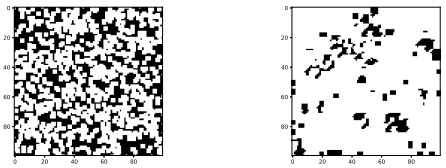
在初始状态下，棋盘中随机分布着90%的合作者和10%的背叛者，棋盘中黑色表示合作者，白色表示背叛者。图3展示了参数为$T = 2.80, P = 0.10, R = 1.10, S = 0.00$时，群体中各个策略群体



(a) $round = 0$ (b) $round = 2$

(c) $round = 4$ (d) $round = 6$

Figure 2: $T = 1.20, P = 0.10, R = 1.10, S = 0.00$



(a) $round = 0$ (b) $round = 10$

(c) $round = 15$ (d) $round = 5$

Figure 3: $T = 1.61, P = 0.10, R = 1.01, S = 0.00$

的变化，可以看到随着时间推移，所有的个体都采用了背叛策略，也就是说，在这个博弈场景下，背叛策略是进化稳定的。

然而，当支付矩阵设置为$T = 1.20, P = 0.10, R = 1.10, S = 0.00$，进化动态将会在两种状态之间稳定震荡，如图2所示。

图3表示，当支付矩阵设置为$T = 1.61, P = 0.10, R = 1.01, S = 0.00$时，可以发现，合作者们成功抵御住了背叛者策略的入侵。

3.2 分蛋糕博弈模拟

分蛋糕问题描述如下：假设两个个体同时分总量为C的蛋糕，每个个体可以选择自己的分蛋糕策略s，且$0 \leq s \leq C$，若双方需求之和小于等于总量C，则双方都可以获得与需求相符的蛋糕；但是若需求之和超过了总量C，那么双方获得的蛋糕数量为0。

考虑如下的进化模型：假设我们拥有了一个相当规模的群体，并且个体之间随机配对并重复进行

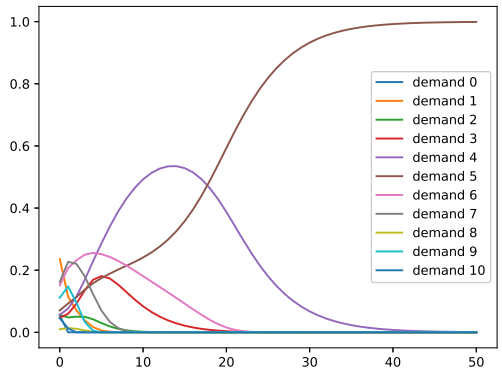


Figure 4: 公平分割进化

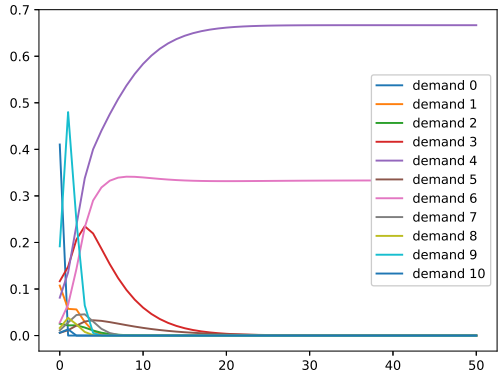


Figure 5: 非对称分割进化

分蛋糕博弈，博弈完成后根据复制者动态更新其各自的策略。简便起见，我们假设蛋糕被分为10个等份，每个个体的策略有11种可能性：需求0份、1份...10份。在复制者动态中，群体的状态使用向量$[p_0, p_1, ..., o_10]$表示，其中p_i表示使用策略“需求i份”的群体比例。

不同的初始条件可能产生两种不同的格局。我们使用2.2种提到的数学模型对群体状态进行迭代。图4的初始向量为$[0.0544685, 0.236312, 0.0560727, 0.0469244, 0.0562243, 0.0703294, 0.151136, 0.162231, 0.0098273, 0.111366, 0.0451093]$，最终群体中均分策略（即需求5份策略）占据主流。

图5则展示了不均衡分割情况下，群体进化动态的走向，可以看到，需求4份和需求6份的群体各自占据了一定比例，并达到了稳定状态。

4 总结

本文对进化博弈论的几个基本概念进行了阐释，并利用进化博弈论的主要方法针对囚徒困境和分蛋糕博弈问题进行了模拟，进化博弈论的确是社会和经济学分析中一个十分重要的概念。

5 附录

```
"""
Paper link: https://plato.stanford.edu/entries/game-evolutionary/#SenFai

This code implements section 2.2 of the paper.
"""
import numpy as np
import json
from matplotlib import pyplot as plt

N = 100

def simulate(payoffs, rounds=10, skip=1, escape=[]):
  """
  Simulate the generation with payoff matrix T/R/P/S.

  In Nowak and May's model, each individual on the lattice plays the prisoner's
  dilemma with their eight nearest neighbors. At the end of each game round, an
  individual compares her score with that of her neighbors.

  - If one of her neighbors earned a higher score, that player will adopt the
    strategy used by her most successful neighbor (presumably using some kind of
    randomization process to break ties).
  - If no neighbor earned a higher score, that player will continue using the
    same strategy for the next round of play.
  All individuals switch strategies at the same time, and all have the same
  payoff structure.

  Link: https://plato.stanford.edu/entries/game-evolutionary/notes.html

  Arguments:
    payoffs: payoff matrix with struture [T, R, P, S]
  """
  # The simulation is performed on a N * N square lattice and starts with
  # the same initial configuation with 90% cooperators and 10% defectors.
  # 0 denotes cooperator, 1 denotes defector
  agent_matrix = np.random.choice(2, size=(N, N), p=(0.9, 0.1))

  [T, R, P, S] = payoffs
  payoff_matrix = [[R, S], [T, P]]

  for i in range(rounds):
    score_matrix = np.zeros((N, N))
    # In each round, each player plays with its eight neighbors.
    for r in range(N):
      for c in range(N):
        strategy_r_c = agent_matrix[r, c]
        for n_r in [r - 1, r, r + 1]:
          for n_c in [c - 1, c, c + 1]:
            if (n_r == r and n_c == c) or (n_r < 0 or n_r >= N) or\
              (n_c < 0 or n_c >= N):
              continue
```

```
            else:
              neighbor_strategy = agent_matrix[n_r, n_c]
              score_matrix[r, c] += \
                payoff_matrix[strategy_r_c][neighbor_strategy]
              score_matrix[n_r, n_c] += \
                payoff_matrix[neighbor_strategy][strategy_r_c]
    # At the end of round, player_r_c changes its strategy.
    for r in range(N):
      for c in range(N):
        strategy_r_c = agent_matrix[r, c]
        max_neighbor_score = score_matrix[r, c]
        max_neighbor_strategy = strategy_r_c
        for n_r in [r - 1, r, r + 1]:
          for n_c in [c - 1, c, c + 1]:
            if (n_r == r and n_c == c) or (n_r < 0 or n_r >= N) or \
              (n_c < 0 or n_c >= N):
              continue
            else:
              neighbor_strategy = agent_matrix[n_r, n_c]
              if score_matrix[n_r, n_c] > max_neighbor_score:
                max_neighbor_score = score_matrix[n_r, n_c]
                max_neighbor_strategy = neighbor_strategy
        agent_matrix[r, c] = max_neighbor_strategy
    if i % skip == 0 and i not in escape:
      filename = '_'.join(['T=%0.2f' % T, 'P=%0.2f' % P, 'R=%0.2f' % R,\
        'S=%0.2f' % S, 'round=%d' % i])
      plt.imshow(agent_matrix, cmap=plt.cm.gray, vmin=0, vmax=1)
      plt.savefig('./img/%s.eps' % filename)

if __name__ == "__main__":
    simulate([2.8, 1.1, 0.1, 0], 5, 1, [3])
    simulate([1.2, 1.1, 0.1, 0], 8, 2)
    simulate([1.61, 1.01, 0.1, 0], 20, 5)
```

```
"""
Paper link: https://plato.stanford.edu/entries/game-evolutionary/#SenFai

This code implements section 4.1 of the paper.
"""
import numpy as np
import json
from matplotlib import pyplot as plt
from scipy.ndimage.filters import gaussian_filter1d

C = 10 # amount of cake
N = 500

def simulate_2(frequency_of_demands, rounds=50, label='dist'):
  """
  A strategy for a player, in this game, consists of an amount of cake that he
  would like.The set of possible strategies for a player is thus any amount
  between 0 and C.
  - If the sum of strategies for each player is less than or equal to C,
    each player receives the amount he asked for.
  - However, if the sum of strategies exceeds C, no player receives anything.

  Arguments:
    frequency_of_demands: A vector with length 11.
  """
  frequency_of_demands = np.array(frequency_of_demands)\
    / np.sum(frequency_of_demands)
  freq_sum = [frequency_of_demands.copy()]
  for i in range(rounds):
    w_c = np.zeros(C + 1)
    for demand_i in range(C + 1):
      for demand_k in range(C + 1):
        w_c[demand_i] += demand_i * frequency_of_demands[demand_k]\
          if demand_i + demand_k <= C else 0
    w_c *= np.array(frequency_of_demands)
    w_c = w_c / np.sum(w_c)
    frequency_of_demands = w_c
    freq_sum.append(frequency_of_demands.copy())
  legends = []
  freq_sum = np.array(freq_sum)
  plt.clf()
  for i in range(C + 1):
    legends.append('demand␣%d' % i)
    plt.plot(freq_sum[:, i])
  plt.legend(legends)
  # plt.show()
  plt.savefig('./img/%s.eps' % label)

if __name__ == "__main__":
  dist_1 = [0.0544685, 0.236312, 0.0560727, 0.0469244, 0.0562243, 0.0703294,
    0.151136, 0.162231, 0.0098273, 0.111366, 0.0451093]
  dist_2 = [0.410376, 0.107375, 0.0253916, 0.116684, 0.0813494, 0.00573677,
    0.0277155, 0.0112791, 0.0163166, 0.191699, 0.00607705]
  simulate_2(dist_1, 50, label='dist_1')
  simulate_2(dist_2, 50, label='dist_2')
```