数理生物学演習

第10回　進化ゲーム理論：Pythonによる進化ダイナミクスの解析

廣瀬　草太郎（HIROSE Sotaro）
soymthty[at]gmail.com
数理生物学研究室

本日の内容

進化ゲーム理論の基礎的な内容を紹介

1. ゲームモデル
   - 囚人のジレンマ（Prisoner’s Dilemma）

2. 進化ゲームモデル
   - タカ・ハトゲーム（Hawk-Dove game）
   - タカ・ハトゲームの拡張（Hawk-Dove-Retaliator game）

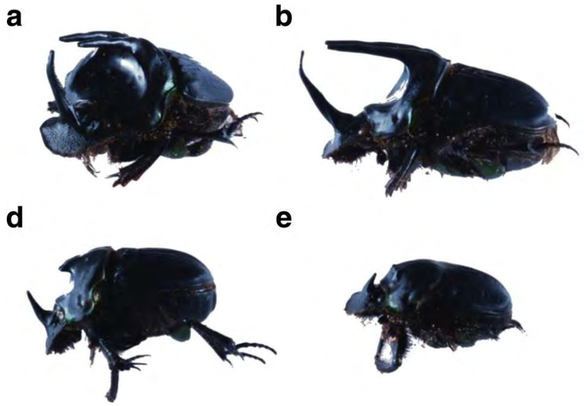
3. 研究紹介

生物に見られる様々な個体間相互作用

表現型（e.g. 形態、行動様式）は、各個体が相互作用する過程で重要な役割を果たす。

闘争

Perez et al., *Anim. Behav.*, 2015



Cummings et al., *J. Insect Behav.*, 2018

配偶者選択

Freyer et al., *Interface Focus*, 2018

性比

Huy Pham et al., *J. Asia-Pac. Entomol.*, 2024

ある個体の適応度が、他個体の表現型の頻度に影響される（頻度依存性を示す）場合、どのような表現型が適応的になるのかを明らかにしたい。

ゲーム理論（Game Theory）

目的や目標、選好性が対立しうる主体（プレイヤー、エージェント）が相互作用して意思決定が行われる状況を、説明・予測することのできる数学的手法。

Neumann & Morgenstern, 1944. “Theory of Games and Economic Behavior” で、経済学的・社会学的問題を分析するための手法として確立された。

▼

古典的なゲーム理論における仮定

1. 各プレイヤーは、明確に定義された利得関数をそれぞれ持ち、ゲームの結果として得られる自身の利得（Pay-off）の最大化を目標とする。

2. プレイヤーは、自分・相手の戦略の選択肢・利得の値を認識しており（完全合理性；Perfect rationality）、全てのプレイヤーが合理性を持つと知っている状態を、全員が知っていて…という構造が無限に続く（共有知識；Common knowledge）。

Szabó & Fáth, *Phys. Rep.*, 2007

ナッシュ均衡（Nash Equilibrium）

**どのプレイヤーも、採用する戦略を変更しても利得が増えず、
この均衡から逸脱することが出来ないような戦略の組み合わせ。**

定義

全ての $s_i \in S_i \ \ \forall i$ に対して、プレイヤー i の利得 $\pi_i(s_i, s_{-i})$ が、

$$\pi_i\left(s_i^*, s_{-i}^*\right) \geq \pi_i\left(s_i, s_{-i}^*\right)$$

を満たすとき、戦略の組 $s^* = \left(s_i^*, s_{-i}^*\right)$ はナッシュ均衡である。

S_i: プレイヤー $i (i = 1, \ldots, N)$ が取りうる全ての戦略

s_{-i}: プレイヤー i 以外のプレイヤーが取る戦略の組

Rao, 2005

囚人のジレンマ（Prisoner’s Dilemma）

2人のプレイヤーの協力・裏切りの相互作用をモデル化した2人ゲーム

利得に関する仮定

- **自分が協力＆相手が協力**
  相互協力に対する報酬（Reward for mutual cooperation; R）
- **自分が協力＆相手が裏切り**
  愚か者の利得（Sucker’s pay-off; S）
- **自分が裏切り＆相手が協力**
  裏切りの誘惑（Temptation to defect; T）
- **自分が裏切り＆相手が裏切り**
  相互裏切りに対する罰
  （Punishment for mutual defection; P）
- 利得の値に関して、$S < P < R < T$ を満たす。

囚人のジレンマの利得表
（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1 ＼ プレイヤー2	協力（Cooperate）	裏切り（Defect）
協力（Cooperate）	(R, R)	(S, T)
裏切り（Defect）	(T, S)	(P, P)

Doebeli & Hauert, *Ecol. Lett.*, 2005; Szabó & Fáth, *Phys. Rep.*, 2007

囚人のジレンマにおける純粋戦略ナッシュ均衡の導出

各プレイヤーが、協力戦略か裏切り戦略を採用する場合のナッシュ均衡を導出する。

利得の計算例

- **プレイヤー1が協力戦略、プレイヤー2が裏切り戦略を採用する場合：** $s^* = (s_1^* = C, s_2^* = D)$

  プレイヤー i の利得を $\pi_i(s_i, s_{-i})$ とすると、

$$\begin{cases} \pi_1(C, D) = S \\ \pi_2(D, C) = T \end{cases} \qquad S_i = \{C, D\} \ \ (i = 1, 2)$$

▼

$$\begin{cases} \pi_1(C, D) \geq \pi_1(D, D) \\ \pi_2(D, C) \geq \pi_2(C, C) \end{cases}$$

を満たさないため、
$s^* = (s_1^* = C, s_2^* = D)$ はナッシュ均衡ではない。

囚人のジレンマの利得表
（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1 ＼ プレイヤー2	協力（Cooperate）	裏切り（Defect）
協力（Cooperate）	(R, R)	(S, T)
裏切り（Defect）	(T, S)	(P, P)

Doebeli & Hauert, *Ecol. Lett.*, 2005; Szabó & Fáth, *Phys. Rep.*, 2007

囚人のジレンマにおける純粋戦略ナッシュ均衡の導出

利得の計算の続き

- **プレイヤー1が裏切り戦略、プレイヤー2が協力戦略を採用する場合：** $s^* = (s_1^* = D, s_2^* = C)$

$$\begin{cases} \pi_1(D, C) = T \\ \pi_2(C, D) = S \end{cases} \blacktriangleright \begin{cases} \pi_1(D, C) \geq \pi_1(C, C) \\ \pi_2(C, D) \geq \pi_2(D, D) \end{cases}$$ を満たさないため、ナッシュ均衡ではない。

- **プレイヤー1が協力戦略、プレイヤー2が協力戦略を採用する場合：** $s^* = (s_1^* = C, s_2^* = C)$

$$\begin{cases} \pi_1(C, C) = R \\ \pi_2(C, C) = R \end{cases} \blacktriangleright \begin{cases} \pi_1(C, C) \geq \pi_1(D, C) \\ \pi_2(C, C) \geq \pi_2(D, C) \end{cases}$$ を満たさないため、ナッシュ均衡ではない。

- **プレイヤー1が裏切り戦略、プレイヤー2が裏切り戦略を採用する場合：** $s^* = (s_1^* = D, s_2^* = D)$

$$\begin{cases} \pi_1(D, D) = P \\ \pi_2(D, D) = P \end{cases} \blacktriangleright \begin{cases} \pi_1(D, D) \geq \pi_1(C, D) \\ \pi_2(D, D) \geq \pi_2(C, D) \end{cases}$$ を満たすため、ナッシュ均衡である。

Doebeli & Hauert, *Ecol. Lett.*, 2005; Szabó & Fáth, *Phys. Rep.*, 2007

NashPy

2人ゲームを解析するためのPythonライブラリ

▼

混合戦略ナッシュ均衡などを手軽に解析できる。

囚人のジレンマの場合、
プレイヤー i が確率 p_i^* で協力戦略、確率 $1-p_i^*$ で裏切り戦略を採用する戦略の組 $p^* = (p_i^*, p_{-i}^*)$ が、

$$\pi_i\left(p_i^*, p_{-i}^*\right) \geq \pi_i\left(p_i, p_{-i}^*\right)$$

を満たす。

p_{-i}：プレイヤー i 以外のプレイヤーが取る戦略の組

Nashpy

Navigation

Tutorial: building and finding the equilibrium for a game
How to
Nashpy Game Theory Text book
Discussion and code reference
Contributor documentation
Indices and tables

Quick search

Go

Welcome to Nashpy's documentation!

A python library for 2 player games.

- Tutorial: building and finding the equilibrium for a game
  - Introduction to game theory
  - Installing Nashpy
  - Creating a game
  - Calculating the utility of a pair of strategies
  - Computing Nash equilibria
  - Learning in games
- How to
  - Install Nashpy
  - Create a Normal Form Game
  - Calculate utilities
  - Check if a strategy is a best response
  - Handle Degenerate games
  - Use the minimax theorem
  - Solve with support enumeration
  - Solve with vertex enumeration
  - Solve with Lemke Howson
  - Use fictitious play
  - Use stochastic fictitious play
  - Use replicator dynamics
  - Use replicator dynamics with mutation
  - Use asymmetric replicator dynamics
  - Use Moran processes
  - Obtain fixation probabilities
  - Obtain a repeated game
  - Use Moran processes on replacement graph

https://nashpy.readthedocs.io/en/stable/

囚人のジレンマの混合戦略ナッシュ均衡をNashPyで解析する


```
# 01-01. NashPyのインストール
!pip install nashpy
```

```
# 01-02. 必要なライブラリのインポート
import nashpy as nash
import numpy as np
```

```
# 01-03. 囚人のジレンマにおける混合戦略ナッシュ均衡の導出

# 利得
s = 1  # 愚か者の利得
p = 3  # 相互裏切りに対する罰
r = 5  # 相互協力に対する報酬
t = 7  # 裏切りの誘惑
```

```
# 利得行列（プレイヤー1）
A = np.array([[r, s],
              [t, p]])

# 利得行列（プレイヤー2）
B = np.array([[r, t],
              [s, p]])

# 「囚人のジレンマ」の構築
PDG = nash.Game(A, B)

# 混合戦略ナッシュ均衡の導出
equilibria = PDG.support_enumeration()

for eq in equilibria:
    print("Nash equilibrium (Player 1, Player 2):")
    print("  Player 1 strategy:", eq[0])
    print("  Player 2 strategy:", eq[1])
```

古典的なゲーム理論から進化ゲーム理論へ

古典的なゲーム理論

- 「合理性を持ち、意思決定を行う主体であるプレイヤーが、
  他個体の行動を考慮して自分の行動の結果を先読みし、
  自身の利得を最大にするためにどのような行動を取るのか」を考察する。

▼

進化ゲーム理論

- プレイヤーの完全合理性を仮定せずに、
  「個体がある戦略を採用し行動した結果が上手くいくほど、
  適応度（子孫の数）が大きくなり、戦略が子孫に遺伝する」という状況を考え、
  どのような戦略が集団内に広がっていくのか（**進化ダイナミクス**）を考察する。

ゲーム理論を生物学に応用可能な形式に拡張することで、進化ゲーム理論が発展した。

Szabó & Fáth, *Phys. Rep.*, 2007

進化ゲーム理論（Evolutionary Game Theory）

Smith & Price, Nature, 1973. “The Logic of Animal Conflict”

- 発達した武器を持つ生物に見られる儀式的闘争や、
  武器が発達せず闘争が激化しない生物における持久戦など、
  非危機的な闘争の進化を数理モデルで説明することを試みた。

Perez et al., *Anim. Behav.*, 2015

- 自然選択の下でどのような行動戦略が安定的に維持されるかを定式化するため、
  進化的に安定な戦略（Evolutionarily Stable Strategy; ESS）という概念を提唱。

Leimar & McNamara, *Phil. Trans. R. Soc. B*, 2023; Smith, *Am. Nat.*, 1976

ESSとは

生物集団全体が採用した場合に、異なる戦略を持つ変異型戦略が侵入出来ない戦略。

▼

戦略 x^* を採用する野生型集団が、異なる戦略 x' を採用する変異型に対し、
適応度 W に関して

(i) $W(x', x^*) \leq W(x^*, x^*)$

(ii) $W(x', x^*) = W(x^*, x^*)$ を満たす x' $(x' \neq x^*)$ について、

$W(x', x') < W(x^*, x')$

のいずれかを満たすとき、戦略 x^* は進化的に安定な戦略（ESS）である。

Leimar & McNamara, *Phil. Trans. R. Soc. B*, 2023; Smith, *Am. Nat.*, 1976

タカ・ハトゲーム（Hawk-Dove game）

資源を巡って争う2個体間の闘争に関する2人ゲーム

- **タカ戦略（Hawk, Escalate）**
  戦いをエスカレートさせ、武器を使って相手を倒そうとする。
- **ハト戦略（Dove, Display）**
  儀式的な闘争を行い、戦いがエスカレートすると逃げ出す。

Smith, *Am. Nat.*, 1976; Smith & Parker, *Anim. Behav.*, 1976

タカ・ハトゲーム（Hawk-Dove game）の利得に関する仮定

- **タカ戦略 vs ハト戦略**
  必ずタカ戦略が勝利し、資源 v の利得を獲得する。

- **タカ戦略 vs タカ戦略**
  戦いがエスカレートし、敗者は損失 c を支払う
  （勝率は1/2）。

- **ハト戦略 vs ハト戦略**
  戦いはエスカレートせず、
  勝者は資源 v の利得を得る（勝率は1/2）。

- ただし、$v, c > 0$ を満たす。

タカ・ハトゲームの利得表
（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1 ＼ プレイヤー2	タカ（H）	ハト（D）
タカ（H）	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$	$(v, 0)$
ハト（D）	$(0, v)$	$\left(\frac{v}{2}, \frac{v}{2}\right)$

Smith, *Am. Nat.*, 1976; Smith & Parker, *Anim. Behav.*, 1976

タカ・ハトゲームにおける純粋戦略ナッシュ均衡の導出

各プレイヤーが、タカ戦略かハト戦略を採用する場合のナッシュ均衡を導出する。

利得の計算例

- **プレイヤー1がタカ戦略、プレイヤー2がハト戦略を採用する場合：** $s^* = (s_1^* = H, s_2^* = D)$

  $v < c$ を満たす、怪我のリスクが高い闘争を仮定し、
  プレイヤー i の利得を $\pi_i(s_i, s_{-i})$ とすると、

$$\begin{cases} \pi_1(H, D) = v \\ \pi_2(D, H) = 0 \end{cases} \qquad S_i = \{H, D\} \ \ (i = 1, 2)$$

▼

$$\begin{cases} \pi_1(H, D) \geq \pi_1(D, D) \\ \pi_2(D, H) \geq \pi_2(H, H) \end{cases}$$

を満たすため、
$s^* = (s_1^* = H, s_2^* = D)$ はナッシュ均衡である。

タカ・ハトゲームの利得表
（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1 ＼ プレイヤー2	タカ（H）	ハト（D）
タカ（H）	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$	$(v, 0)$
ハト（D）	$(0, v)$	$\left(\frac{v}{2}, \frac{v}{2}\right)$

Smith, *Am. Nat.*, 1976; Smith & Parker, *Anim. Behav.*, 1976

タカ・ハトゲームにおける純粋戦略ナッシュ均衡の導出

利得の計算の続き

- **プレイヤー1がハト戦略、プレイヤー2がタカ戦略を採用する場合：** $s^* = (s_1^* = D, s_2^* = H)$

$$\begin{cases} \pi_1(D,H) = 0 \\ \pi_2(H,D) = v \end{cases} \blacktriangleright \begin{cases} \pi_1(D,H) \geq \pi_1(H,H) \\ \pi_2(H,D) \geq \pi_2(D,D) \end{cases}$$ を満たすため、ナッシュ均衡である。

- **プレイヤー1がタカ戦略、プレイヤー2がタカ戦略を採用する場合：** $s^* = (s_1^* = H, s_2^* = H)$

$$\begin{cases} \pi_1(H,H) = \frac{v-c}{2} \\ \pi_2(H,H) = \frac{v-c}{2} \end{cases} \blacktriangleright \begin{cases} \pi_1(H,H) \geq \pi_1(D,H) \\ \pi_2(H,H) \geq \pi_2(D,H) \end{cases}$$ を満たさないため、ナッシュ均衡ではない。

- **プレイヤー1がハト戦略、プレイヤー2がハト戦略を採用する場合：** $s^* = (s_1^* = D, s_2^* = D)$

$$\begin{cases} \pi_1(D,D) = \frac{v}{2} \\ \pi_2(D,D) = \frac{v}{2} \end{cases} \blacktriangleright \begin{cases} \pi_1(D,D) \geq \pi_1(H,D) \\ \pi_2(D,D) \geq \pi_2(H,D) \end{cases}$$ を満たさないため、ナッシュ均衡ではない。

Smith, *Am. Nat.*, 1976; Smith & Parker, *Anim. Behav.*, 1976

タカ・ハトゲームの混合戦略ナッシュ均衡をNashPyで解析する


```
# 02-01. 必要なライブラリのインポート
import nashpy as nash  # ランタイムの接続が解除された場合は、再度NashPyをインストールする（01-01）
import numpy as np
import matplotlib.pyplot as plt
```

```
# 02-02. タカ・ハトゲームにおける混合戦略ナッシュ均衡の導出

# 利得
v = 2  # 資源
c = 4  # 闘争のコスト

# 利得行列（プレイヤー1）
A = np.array([[(v - c)/2, v],
              [0,         v/2]])

# 利得行列（プレイヤー2）
B = np.array([[(v - c)/2, 0],
              [v,         v/2]])

# 「タカ・ハトゲーム」の構築
HDG = nash.Game(A, B)

# 混合戦略ナッシュ均衡の導出
equilibria = HDG.support_enumeration()

for eq in equilibria:
    print("Nash equilibrium (Player 1, Player 2):")
    print("  Player 1 strategy:", eq[0])
    print("  Player 2 strategy:", eq[1])
```

Replicator dynamicsによる進化ダイナミクスの解析

進化ゲーム理論

- 「個体がある戦略を採用し行動した結果が上手くいくほど、
  適応度（子孫の数）が大きくなり、戦略が子孫に遺伝する」という状況を考え、
  適応度の大きい戦略が、時間をかけて集団内に広がっていく（**進化ダイナミクス**）。

▼

Replicator dynamics

- n 個の戦略が存在する生物集団において、戦略 i の頻度 x_i の時間変化を
  微分方程式で表現することで、戦略の進化ダイナミクスを解析する。

$$\frac{dx_i}{dt} = x_i(f_i - \phi) \quad (i = 1, \ldots, n)$$

戦略 j に対する戦略 i の適応度：a_{ij}

戦略 i の適応度の期待値：$f_i = \sum_{j=1}^{n} x_j a_{ij}$

集団の平均適応度：$\phi = \sum_{i=1}^{n} x_i f_i$

タカ・ハトゲームにおけるReplicator dynamics

集団内の各個体がタカ戦略かハト戦略のどちらかを採用する純粋戦略を考える。

タカ戦略、ハト戦略を採用する個体の適応度

$$
\begin{aligned}
f_H &= x_H \cdot a_{HH} + x_D \cdot a_{HD} \\
&= x_H \cdot \frac{v-c}{2} + (1-x_H) \cdot v \\
&= v - x_H \cdot \frac{v+c}{2}
\end{aligned}
\qquad
\begin{aligned}
f_D &= x_H \cdot a_{DH} + x_D \cdot a_{DD} \\
&= x_H \cdot 0 + (1-x_H) \cdot \frac{v}{2} \\
&= (1-x_H) \cdot \frac{v}{2}
\end{aligned}
$$

x_H, x_D：集団内のタカ戦略、ハト戦略の頻度 $(x_H + x_D = 1)$

タカ・ハトゲームの利得表

（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1 ＼ プレイヤー2	タカ（H）	ハト（D）
タカ（H）	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$	$(v, 0)$
ハト（D）	$(0, v)$	$\left(\frac{v}{2}, \frac{v}{2}\right)$

Replicator dynamics

$$
\begin{aligned}
\frac{dx_H}{dt} &= x_H(f_H - \phi) \\
&= x_H[f_H - \{x_H f_H + (1-x_H) f_D\}] \\
&= x_H(1-x_H)(f_H - f_D)
\end{aligned}
$$

タカ・ハトゲームの進化ダイナミクスを解析する

1. 平衡点 x_H^* を求める

$g(x_H) = \dfrac{dx_H}{dt} = x_H(1-x_H)(f_H - f_D)$ に対し、$g(x_H^*) = 0$ を満たす。

2. 局所安定性解析

- 1変数のreplicator dynamicsでは、導関数 $g'(x_H^*)$ の符号で平衡点の安定性を判別する。

平衡点 $x_H = x_H^*$ の周りでテイラー展開すると、

$$g(x_H) = g(x_H^*) + g'(x_H^*)(x_H - x_H^*) + \frac{g''(x_H^*)}{2!}(x_H - x_H^*)^2 + \cdots$$

$x_H - x_H^*$ が十分に小さいと仮定し、2次以上の項を無視すると、

$$g(x_H) = \frac{dx_H}{dt} \approx g'(x_H^*)(x_H - x_H^*)$$

$$\therefore x_H = x_H^* + (x_H(0) - x_H^*)\exp\left(g'(x_H^*)t\right)$$

x_H が平衡点に近づいていくか、
遠ざかっていくかを決める。

$g'(x_H^*) < 0$ を満たす
平衡点 x_H^* は局所安定

$g'(x_H^*) > 0$ を満たす
平衡点 x_H^* は局所不安定

タカ・ハトゲームの進化ダイナミクス

1. 平衡点 x_H^* を求める

$$g(x_H) = x_H(1-x_H)\frac{v-cx_H}{2} = 0$$

$$\therefore x_H^* = 0, 1, \frac{v}{c}$$

2. 局所安定性解析

$$g'(x_H) = \frac{1}{2}\{(1-2x_H)(v-cx_H) - cx_H(1-x_H)\}$$

▼

x_H^* の値をそれぞれ導関数 $g'(x_H)$ に代入し、導関数の符号を確認する。

- $g'(x_H^*) < 0$ を満たす平衡点 x_H^* は局所安定
- $g'(x_H^*) > 0$ を満たす平衡点 x_H^* は局所不安定

タカ・ハトゲームにおけるダイナミクスの可視化

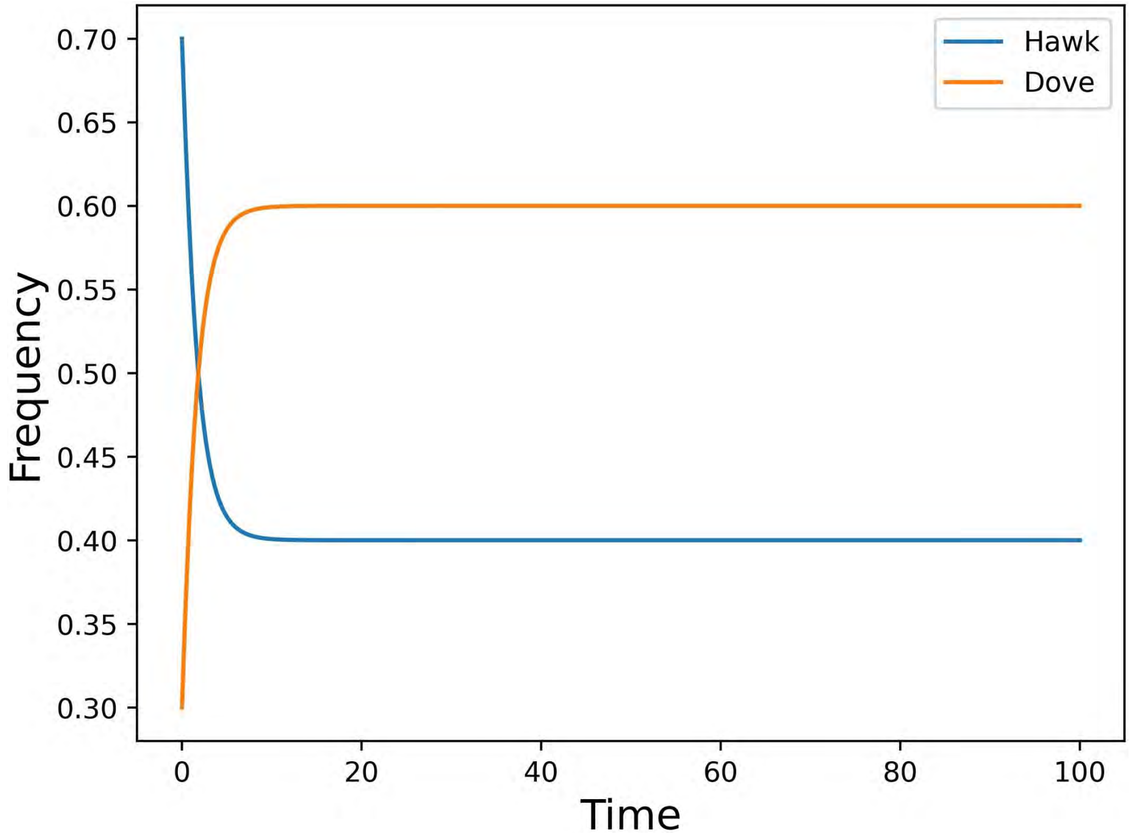

```
# 02-03. オイラー法 - Replicator dynamicsに従うタカ・ハトゲームの
進化ダイナミクスの解析 -

# 利得
v = 2  # 資源
c = 5  # 闘争のコスト

# 利得行列
A = np.array([[(v - c)/2, v],
              [0,         v/2]])

# パラメータ
t = 0
dt = 0.01
t_end = 100
i_end = int(t_end / dt)

# タカ戦略・ハト戦略の初期頻度
x = np.array([0.1, # タカ戦略（H）
              0.9, # ハト戦略（D）
              ])

# データを格納するリスト
t_list = [t]
xH_list = [x[0]]  # タカ戦略（H）
xD_list = [x[1]]  # ハト戦略（D）
```

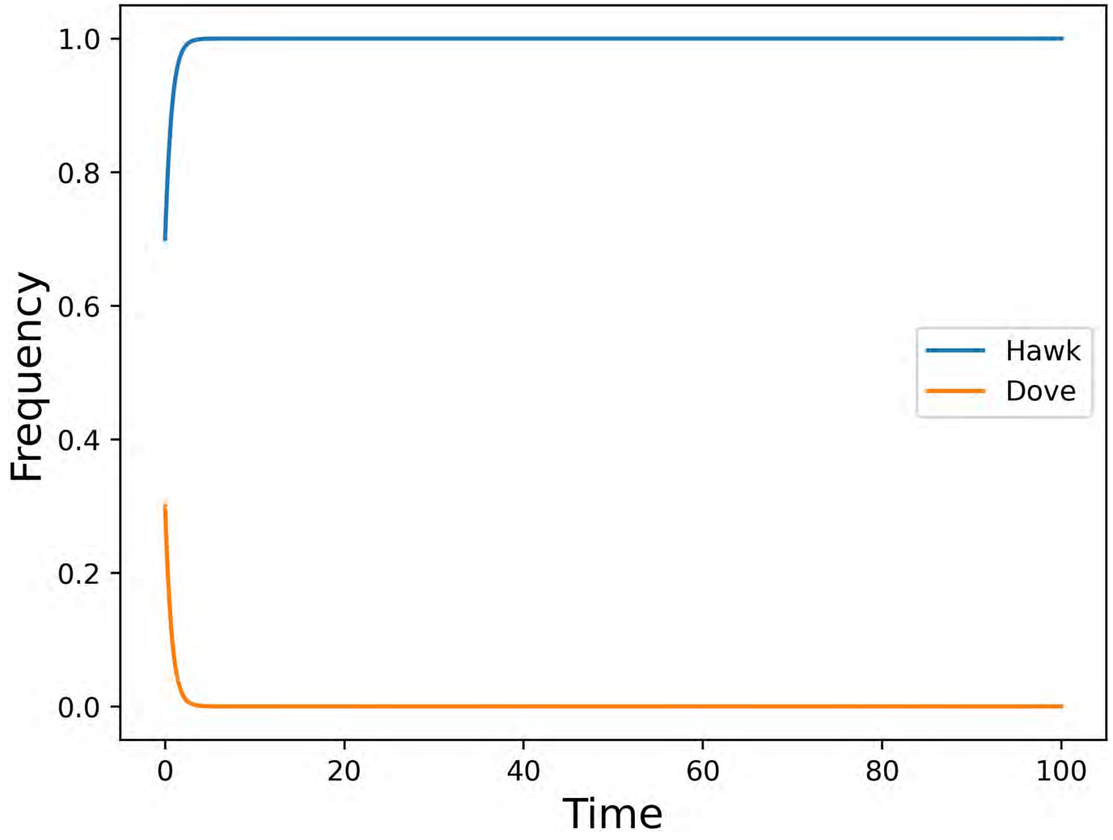
```
# 数値計算
for i in range(i_end):
  t =  t + dt
  f = A @ x             # 各戦略の適応度の期待値
  phi = x @ f           # 集団の平均適応度
  dx = x * (f - phi)  # Replicator dynamics: dx/dt

  x = x + dt * dx      # オイラー法による近似計算

  # 数値計算の誤差への対策
  x = np.clip(x, 0, 1)  # 頻度の閉区間[0,1]の維持
  x /= np.sum(x)  # 正規化による頻度の維持（xH + xD = 1）

  # リストへの格納
  t_list.append(t)
  xH_list.append(x[0])
  xD_list.append(x[1])
```

```
# 02-04. 可視化

fig, ax = plt.subplots(dpi=100)
ax.plot(t_list, xH_list, label="Hawk")
ax.plot(t_list, xD_list, label="Dove")

ax.set_xlabel("Time", fontsize=15)
ax.set_ylabel("Frequency", fontsize=15)
ax.legend()
plt.show()
```

タカ・ハトゲームにおけるダイナミクスの可視化

パラメータ v や c の値を変えて、ダイナミクスの変化を確認してみる。





報復のあるタカ・ハトゲーム（Hawk-Dove-Retaliator game）

タカ・ハトゲームを拡張し、第3の戦略「報復戦略（Retaliator）」を組み込んだゲーム

- **タカ戦略（Hawk, Escalate）**
  戦いをエスカレートさせ、武器を使って相手を倒そうとする。
- **ハト戦略（Dove, Display）**
  儀式的な闘争を行い、戦いがエスカレートすると逃げ出す。

＋

- **報復戦略（Retaliator）**
  相手がタカ戦略であれば闘争をエスカレートさせ、
  ハト戦略・報復戦略であれば儀式的な闘争を行う。

Bomze, *Biol. Cybern.*, 1983

Hawk-Dove-Retaliator gameの利得に関する仮定

- **タカ戦略 vs ハト戦略**
  必ずタカ戦略が勝利し、資源vの利得を獲得する。

- **タカ戦略 vs タカ戦略（報復戦略 vs タカ戦略）**
  戦いがエスカレートし、敗者は損失cを支払う
  （勝率は1/2）。

- **ハト戦略 vs ハト戦略（報復戦略 vs 報復戦略）**
  戦いはエスカレートせず、
  勝者は資源vの利得を得る（勝率は1/2）。

- **報復戦略 vs ハト戦略**
  戦いはエスカレートせず、
  勝者は資源vの利得を得る（勝率は1/2）。

  ただし、報復戦略はハト戦略に対して
  僅かな優位性εを持つ。

Hawk-Dove-Retaliator gameの利得表
（[プレイヤー1の利得], [プレイヤー2の利得]）

プレイヤー1＼プレイヤー2	タカ（H）	ハト（D）	報復（R）
タカ（H）	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$	$(v, 0)$	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$
ハト（D）	$(0, v)$	$\left(\frac{v}{2}, \frac{v}{2}\right)$	$\left(\frac{v}{2}-\varepsilon, \frac{v}{2}+\varepsilon\right)$
報復（R）	$\left(\frac{v-c}{2}, \frac{v-c}{2}\right)$	$\left(\frac{v}{2}+\varepsilon, \frac{v}{2}-\varepsilon\right)$	$\left(\frac{v}{2}, \frac{v}{2}\right)$

Bomze, *Biol. Cybern.*, 1983

Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化


```python
# 03-01. 必要なライブラリのインポート
import numpy as np
import matplotlib.pyplot as plt
```

```python
# 03-02. オイラー法 - Replicator dynamicsに従うHawk-Dove-Retaliator gameの進化ダイナミクスの解析 -

# 利得
v = 2    # 資源
c = 4    # 闘争のコスト
e = 0.1  # ハト戦略に対する報復戦略の優位性

# 利得行列
A = np.array([[(v - c)/2, v,       (v - c)/2],
              [0,         v/2,     v/2 - e],
              [(v - c)/2, v/2 + e, v/2]])

# パラメータ
t = 0
dt = 0.01
t_end = 100
i_end = int(t_end / dt)

# タカ戦略・ハト戦略・報復戦略の初期頻度
x = np.array([0.6, # タカ戦略（H）
              0.2, # ハト戦略（D）
              0.2  # 報復戦略（R）
              ])
```

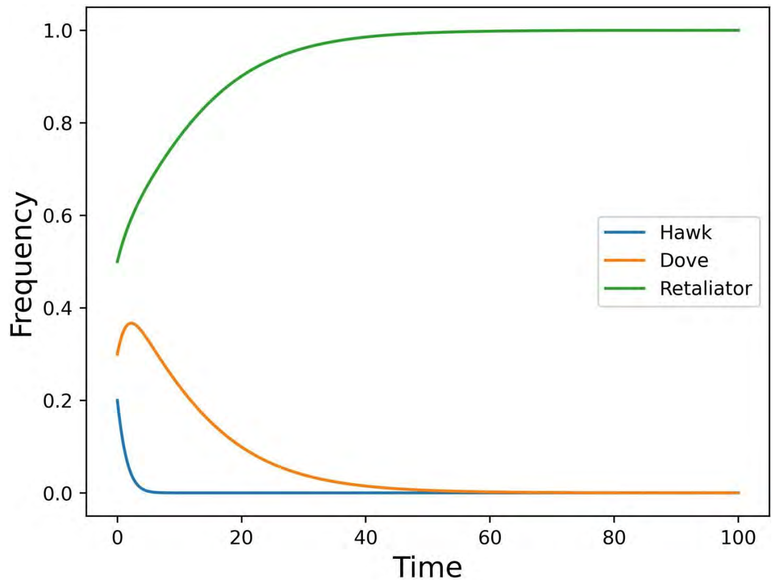
```python
# データを格納するリスト
t_list = [t]
xH_list = [x[0]]  # タカ戦略（H）
xD_list = [x[1]]  # ハト戦略（D）
xR_list = [x[2]]  # 報復戦略（R）

# 数値計算
for i in range(i_end):
  t =  t + dt
  f = A @ x             # 各戦略の適応度の期待値
  phi = x @ f           # 集団の平均適応度
  dx = x * (f - phi)  # Replicator dynamics: dx/dt

  x = x + dt * dx      # オイラー法による近似計算

  # 数値計算の誤差への対策
  x = np.clip(x, 0, 1)  # 頻度の閉区間[0,1]の維持
  x /= np.sum(x)          # 正規化による頻度の維持（xH + xD = 1）

  # リストへの格納
  t_list.append(t)
  xH_list.append(x[0])
  xD_list.append(x[1])
  xR_list.append(x[2])
```

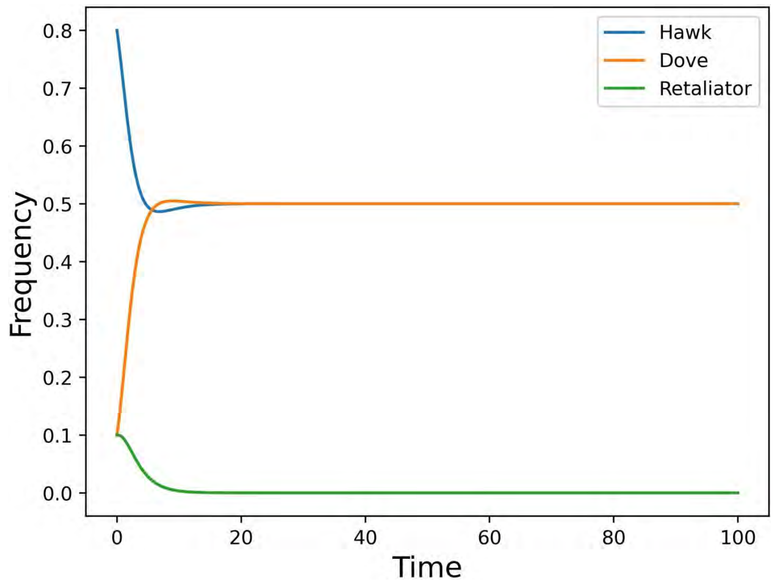
Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化

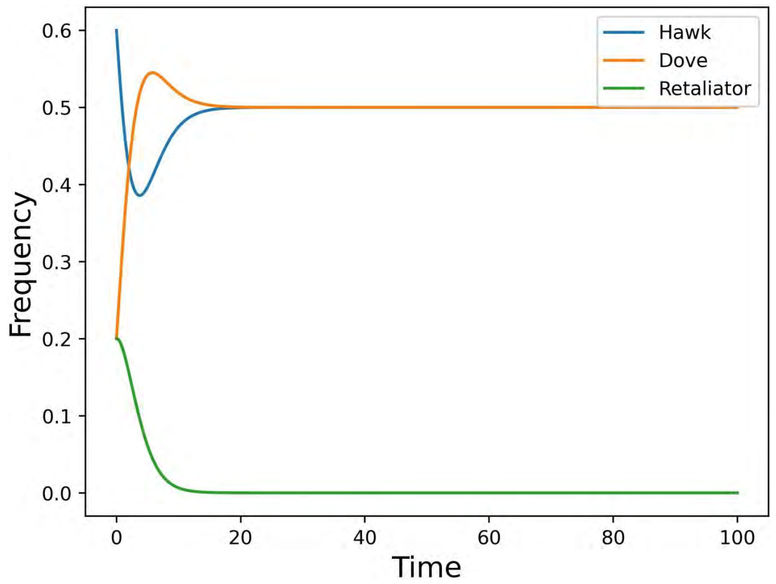

```
# 03-03. 可視化

fig, ax = plt.subplots(dpi=100)

ax.plot(t_list, xH_list, label="Hawk")
ax.plot(t_list, xD_list, label="Dove")
ax.plot(t_list, xR_list, label="Retaliator")

ax.set_xlabel("Time", fontsize=15)
ax.set_ylabel("Frequency", fontsize=15)
ax.legend()
plt.show()
```

Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化

パラメータ v や c 、ε の値を変えて、ダイナミクスの変化を確認してみる。





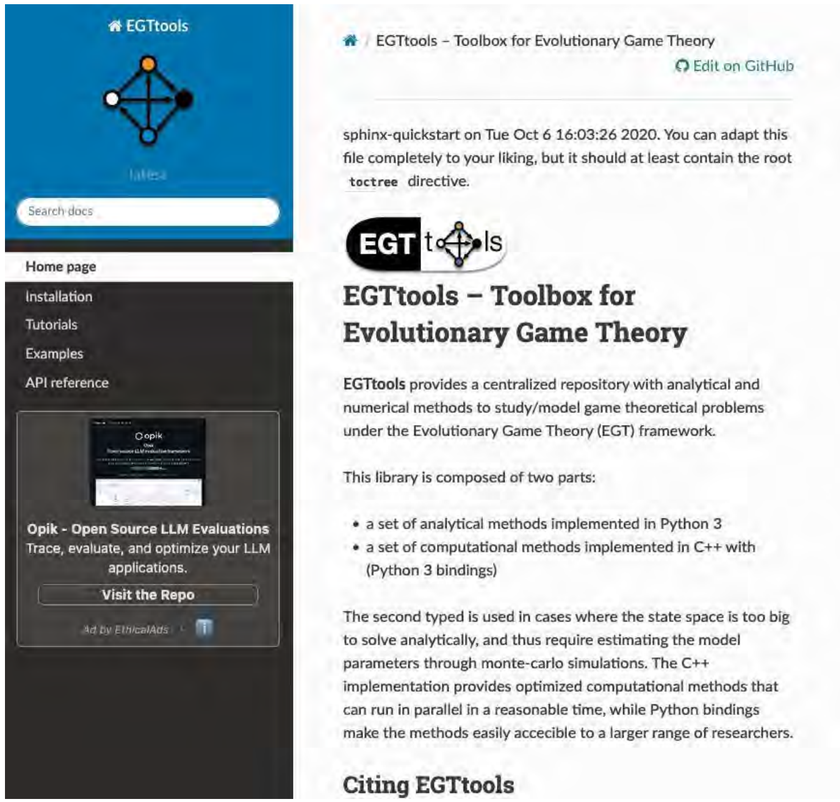


EGTtools

- 進化ゲーム理論に基づいて、ゲーム理論的な問題をモデル化するための解析的・数値的手法を統合した、C++/Pythonライブラリ

▼

タカ戦略、ハト戦略、報復戦略の頻度をそれぞれ軸に取った三角図（2D-simplex plot）として相図を描き、ダイナミクスを数値的に解析する。



Fernández Domingos et al., *iScience*, 2023

https://egttools.readthedocs.io/en/latest/index.html

Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化


```
# 04-01. EGTtoolsのインストール
!pip install egttools
```

```
# 04-02. 必要なライブラリのインポート
import numpy as np
import matplotlib.pyplot as plt
import egttools as egt
from egttools.plotting.simplified import
plot_replicator_dynamics_in_simplex
```

```
#04-03. パラメータの用意

# 利得
v = 2     # 資源
c = 4     # 闘争のコスト
e = 0.1   # ハト戦略に対する報復戦略の優位性

# 利得行列
A =  np.array([[(v - c)/2, v,        (v - c)/2],
               [0,         v/2,      v/2 - e],
               [(v - c)/2, v/2 + e, v/2]])

type_labels = ['Hawk', 'Dove', 'Retaliator']
```

Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化


```
#04-04. Hawk-Dove-Retaliator gameの相図（三角図）のプロット

fig, ax = plt.subplots(dpi=100)

simplex, gradient_function, roots, roots_xy, stability =
plot_replicator_dynamics_in_simplex(A, ax = ax)
plot = (simplex
        .draw_triangle(linewidth=1.4)
        .add_vertex_labels(type_labels, epsilon_bottom=0.15, fontsize=13)
        .draw_stationary_points(roots_xy, stability)
        .draw_gradients(arrowsize=1.5,
                        linewidth=1.3, zorder = 1, density=1.5)
        .add_colorbar())

ax.axis('off')
ax.set_aspect('equal')
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, simplex.top_corner + 0.05))
```

```
# 三角図に目盛となる点線をプロットする
# 三角図の頂点
V1 = np.array([0.0, 0.0])
V2 = np.array([1.0, 0.0])
V3 = np.array([0.5, np.sqrt(3)/2])

def bary_to_xy(b):  # バリセンター座標をxyに変換
    return b[0]*V1 + b[1]*V2 + b[2]*V3

steps = 10  # 目盛の分割数

# 点線のプロット
for i in range(1, steps):
    t = i / steps

    # Hawk
    p1 = bary_to_xy((t, 0, 1 - t))
    p2 = bary_to_xy((t, 1 - t, 0))
    ax.plot([p1[0], p2[0]], [p1[1], p2[1]],
            color='gray', lw=0.7, linestyle='dotted', zorder=0)

    # Dove
    p1 = bary_to_xy((0, t, 1 - t))
    p2 = bary_to_xy((1 - t, t, 0))
    ax.plot([p1[0], p2[0]], [p1[1], p2[1]],
            color='gray', lw=0.7, linestyle='dotted', zorder=0)

    # Retaliator
    p1 = bary_to_xy((0, 1 - t, t))
    p2 = bary_to_xy((1 - t, 0, t))
    ax.plot([p1[0], p2[0]], [p1[1], p2[1]],
            color='gray', lw=0.7, linestyle='dotted', zorder=0)
```

Hawk-Dove-Retaliator gameにおけるダイナミクスの可視化

パラメータ v や c 、ε の値を変えて、ダイナミクスの変化を確認してみる。

黒色の点：安定な平衡点
白色の点：不安定な平衡点
灰色の点：鞍点