卒業論文

繰り返し囚人のジレンマゲームに対する2種アプローチについての考察

令和６年度

岩手大学理工学部

物理・材料理工学科　数理・物理コース

氏名　漆原　知喜

（指導教員　宮島　信也）

目次

第 1 章

序論

1.1　研究の背景

本節の内容は, 書籍 [1] を参考に執筆した.

1.1.1　囚人のジレンマゲーム

応用数学の一分野に,「ゲーム理論」という分野がある. ゲーム理論とは「利害関係のある意思決定主体間の相互作用を数理モデルと論理を用いて研究する学問」である. この理論の先駆者であるフォン＝ノイマンとモルゲンシュテルンの発想は, 原子や分子といった自然界の構成要素で全体を説明するような物理学的手法を, 社会の分析に応用しようということだった. ここで重要なのは, ゲーム理論で分析対象となる人間は「意志を持って行動する」という点で原子や分子と本質的に異なることである. したがって, ゲーム理論の研究は決して容易ではない. しかし, この大きな特徴がゲーム理論の研究を興味深いものにする所以であり, 特に経済学における各種分析に大きな貢献を果たしている.

ゲーム理論の中でも最も有名な題材の一つに「囚人のジレンマゲーム」がある. 状況は以下の通りである.

囚人のジレンマゲーム

警察署に 2 人の共犯罪容疑者 A, B が連れられてきた. しかし, 決定的な証拠がないため, 2 人は別々の部屋で尋問されている. 2 人にはそれぞれ「自白する」,「自白しない」の 2 つの選択肢が与えられる. A, B ともに自白した場合はそれぞれ懲役 5 年, ともに自白しなかった場合はそれぞれ懲役 2 年が課せられる. 一方のみが自白した場合, 自白した方は無罪, 他方は懲役 30 年が課せられる.

このゲームにおける利得を「刑期の (-1) 倍」と定義すれば, 各戦略に対する利得は下の表 1.1 の

ように整理することができる. 表の第 1 列は容疑者 A の取る行動を, 第 1 行は容疑者 B の取る行動を示している. また, 表の要素 (X, Y) は容疑者 A, B がそれぞれ X, Y の利得を得ることを表す. このように, 各プレイヤーの戦略の組み合わせに対してそれぞれの利得を示す表し方を**戦略形（正規形）**と呼ぶ.

表 1.1: 囚人のジレンマ 戦略形

容疑者 A / 容疑者 B	自白しない	自白する
自白しない	$(-2, -2)$	$(-30, 0)$
自白する	$(0, -30)$	$(-5, -5)$

ゲーム理論では一般に, 全てのプレイヤー（ここでは容疑者 A, B）が利得（の期待値）を最大にするように行動すると仮定する. この仮定のもとで上のゲームを考えると, 容疑者 A, B のどちらにとっても「自白する」することが有効な戦略となる. なぜなら, 相手が自白しなければ, 自分が自白することで無罪になることができ, 相手が自白していた場合では自白することで刑期を 30 年から 5 年に縮めることができるからである. しかし, 2 人の利得を合計して考えた場合, 最も合理的な戦略は 2 人が共に「自白しない」を選択することである. このように, **各プレイヤーが自分の利得を最大にするように行動しても, 全体としての利得は最大化されない**. これが「**ジレンマ**」と呼ばれる理由となっている.

利得の和が 0 であるゲームのことを「**ゼロサムゲーム**」といい, 利得の和が一定でないゲームのことを「**非ゼロサムゲーム**」という. 囚人のジレンマゲームは非ゼロサムゲームである. 前者においては一方のプレイヤーの得が他方のプレイヤーの損となり, 純粋に競争的になるだけである. しかし, 後者においては単なる競争の状況ではなく, 協力して行動を取ることで全体の利得をさらに増やすことができる場合がある. 囚人のジレンマゲームでは, ともに自白しないことで刑期を短くできることがそれに対応する.

1.1.2 繰り返しゲーム

囚人のジレンマゲームの例では, ゲームは 1 回限りプレイされることが前提であった. しかし, 現実社会には市場競争や軍備拡張競争, 環境破壊などといった囚人のジレンマゲームのような問題が数多く存在し, それらは長期間にわたって継続的である. これを考慮すると, ゲーム理論の問題を現実社会の問題にモデルとして当てはめる際に, ゲームが多数回繰り返しプレイされる状況を考えるというのは自然な発想である.

ゲーム理論では, 同一のゲームが無限に繰り返しプレイされる状況を一般に**繰り返しゲーム**という. 繰り返しゲームでプレイヤーが過去のプレイの結果に依存して行動を選択できる場合, 協力, 裏切り, 仕返しなどの多様な行動パターンが可能となることが知られている. 実は, このことが囚人のジレンマの解消法の 1 つになるということがゲーム理論の重要な論点としてある. 囚人のジレンマの解消法については, 次章で詳しく述べる.

1.2　繰り返しゲームをシミュレーションすることの有用性

囚人のジレンマゲームを繰り返しゲームとした数理モデルを考える. すなわち,「2 人のプレイヤーが自身の戦略に基づいて行動を選択し, その組み合わせに応じて戦略形で定められた利得を得る」ということを繰り返すようなモデルを考える. このような数理モデルは, 例えば 2 企業間の値下げ競争や 2 個体間での資源の取り合いといった題材に対して適用することができる. これをシミュレーションすることのメリットは, ある戦略を持った 2 個体の利得や意思決定の時間発展やその他の統計量を可視化し, 分析することができる点である. また, モデル内において戦略や利得のパラメータに関する部分を変更するだけで, 様々な条件を簡単に試すことができる.

ここで, 繰り返し囚人のジレンマゲームにおいてプレイヤーの人数を 2 人から $N(\gg 2)$ 人へと拡張してみよう. 対戦方法としては, 例えば「自身以外の $(N-1)$ 人からランダムに選んだ 1 人と囚人のジレンマゲームをする」という方法や,「各プレイヤーを 1 つのセルとみなした二次元格子空間を考えて, 自身の近接セル（に対応したプレイヤー）全員とそれぞれ囚人のジレンマゲームをする」といった方法を考えることができる. ここで, 後者の対戦方法を採用した数理モデルを設計し, これをシミュレーションすることの有用性について説明する. まず, このモデルの最大の特徴は 1 次元モデルではなく 2 次元モデルであることである. これにより, 空間的に大量に配置された利己的個体の, 近接相互作用による利得や意思決定の時間発展を可視化することが可能となる. 1.1.2 節で述べた通り, 現実社会には様々な囚人のジレンマゲームが存在し, 人間のみならず企業や国家までもを囚人に例えることができる. そして, このような個々の物体の相互作用について調べるには, 1 次元モデルよりも自由度が高く, 状態の時間発展をより現実的な視点から観察可能な 2 次元モデルでの分析が妥当である.

個々の物体の振る舞いを記述するモデルが存在したときに, それらを多数用意して互いに相互作用させたものを**多体系モデル**と呼ぶ（書籍 [2] にて定義）. 2 次元モデルに関する上述したような利点から, 多数プレイヤーによる繰り返し囚人のジレンマゲームを 2 次元**多体系モデル**によってシミュレーションし考察することは, 現実社会における多くの協力現象について理解を深めたり, また予測を立てることに有用である.

1.3　囚人のジレンマゲームに対する 2 種のアプローチ

前節で導入した, プレイヤー数が N である繰り返し囚人のジレンマゲームをシミュレーションすることを考える. 毎時間ステップに各プレイヤーは, それぞれの戦略に従って自身の行動を選択するのであるが, 書籍 [2] ではこの行動選択を行う代表的な方法として 2 種のアプローチが紹介されている. これらを以下で説明する.

1 つ目は**進化論的アプローチ**である. このアプローチは, 自分が対戦した相手が得た得点を参照して, 一番得点が高いプレイヤーが取った行動を真似するというものである. 環境に適応した行動が生き残り, そうでない行動は淘汰されることを反映している.

2 つ目は**強化学習アプローチ**である. それぞれの行動の価値を, 毎回のゲームに応じてアップデートしていくという方針である. このアプローチでは, 強化学習の技法の一つである Q 学習やアスピレーション学習などが使用される.

1.4 囚人のジレンマゲームを背景とした先行研究

囚人のジレンマゲームをモデルとした先行研究には, Q 学習を用いた研究が多い. 森山ら [3] は, プレイヤーが強化学習を行うエージェントであると仮定したとき, 学習率が所与の下で「協調」の行動価値関数が「裏切り」のものよりも大きくなる協調関係の回数についての定理を導出した. それを踏まえ田口ら [4] は, ゲームプレイヤーを N 人と拡張した繰り返し囚人のジレンマゲームを扱い, 行動選択による利得の差によって協調行動を選択するエージェントの数が急激に変化することを実験的に明らかにした.

二次元格子空間上で囚人のジレンマゲームをモデルとした先行研究もいくつか存在している. 大原ら [5] は, 各プレイヤーが過去の対戦結果をもとに戦略を変えるという設定の下, 対戦相手が選ばれる近傍の範囲が小さいほど協調行動が進化することを数値実験により示した. また藤原ら [6] は, 協調関係にあるプレイヤー集団内にフリーライダー（集団に協力せずに利益だけを搾取する個体）が一定確率で発生した場合の影響について研究した.

1.5 本研究の目的と新規性

本研究の目的は, 繰り返し囚人のジレンマゲームをモデルとした新たな数理モデルを 2 つ提案し, それらのモデルに基づいてシミュレーションを行い, 結果を考察することである. この数理モデルの概要を説明する. プレイヤー数を $N(\gg 2)$ 人とする. そして, 各プレイヤーを 1 つのセルとみなした二次元格子空間上での繰り返し囚人のジレンマゲームにおける協力現象の時間発展を追うことを考える. 書籍 [2] では各プレイヤーの行動選択を行う代表的な手法として, **進化論的アプローチ**と**強化学習アプローチ**の 2 種類が紹介されていることを 1.3 節で述べた. 本研究ではこれらを参考に, 2 種のアプローチについて筆者が提案した数理モデルを用いて, 二次元格子空間上での繰り返し囚人のジレンマゲームをシミュレーションする.

本研究の新規性を述べる. まず, 書籍 [2] では代表的な 2 種のアプローチの存在が紹介されているだけであり, 具体的な数理モデルやシミュレーション結果については言及されていない. したがって, 本研究で著者が提案する数理モデルはオリジナリティを持っている. 本研究の先行研究との違いについても述べる. まず先行研究 [3, 4, 5, 6] では, 囚人のジレンマゲームをモデルとした数理モデルに対して**進化論的アプローチ**もしくは**強化学習アプローチ**からしか研究を行っていないが, 本研究ではこれら両方のアプローチから研究を行っている. また二次元格子空間上での繰り返し囚人のジレンマゲームに対して, 強化学習の手法を用いたシミュレーションを試みている研究は皆無である. 以上の理由から, 本研究は新規性を持つといえる.

本論文は次のような構成をとる.

第 2 章では本シミュレーションを行う上で前提となる知識を基礎理論として紹介する.

第 3 章では次章以降でのシミュレーションに際して登場する各種ゲームの定義を行う.

第 4 章及び第 5 章では本論文の核となる 2 種のアプローチに基づいたシミュレーションを実行し, 結果を考察する.

第 6 章では結論を述べ, 本論文を総括する.

第 2 章

基礎理論

2.1　ゲーム理論に関する補足

本節の内容は, 書籍 [1] を参考に執筆した.

2.1.1　ゲーム理論による定式化

ゲーム理論におけるゲームは, 本質的に「プレイヤー」,「戦略」,「利得」,「情報構造と手番」の 4 つの要素から構成されている. 以下, これら 4 要素について簡単に説明する.

プレイヤーとは,「誰がプレイするのか」ということである. プレイヤーが 2 人の場合は 2 人ゲームといい, 3 人以上の場合は n 人ゲームと呼ぶ.

次に**戦略**とは, 相手がこうしてきたら自分はこうするというような「状況に応じた自分の行動計画」という意味である.

人々は満足度を高めるように, また企業は利潤を高めるように行動すると仮定する. 満足度や利潤のように目的となる指標をまとめて**利得**と呼ぶ. 第 1 章で述べた通り, 全プレイヤーはこの利得を最大にするように行動すると想定する. また, ゲーム理論においてこの利得は以下の 2 つの条件を満たすとする.

- **期待値が計算可能であること**
- **利得の間の相対関係だけが意味を成すこと**
  （全ての利得パラメータの値を C 倍（$C \in \mathbb{R}_{>0}$）してもゲームの結果が不変であること）

情報構造と手番は, ゲームが進行するにつれ, 誰が何を知っているのかと次は誰が行動するのかを記述する部分である. 行動が同時に起こる場合を**同時手番**, 順番に行動する場合を**逐次手番**という.

2.1.2　支配戦略による予測

戦略 A と戦略 B を比べて, 相手が何をしてきても戦略 A の方が戦略 B よりも高い利得をもたらす場合, 戦略 A は戦略 B を**優越する（支配する）**という. ある戦略が全ての戦略を優越していると

き, この戦略は**支配戦略**と呼ばれ, この支配戦略を取ることで利得が最大化される. 互いに支配戦略が存在する場合, それぞれのプレイヤーは支配戦略を取ることが予測される. 第 1 章における囚人のジレンマゲームでは「自白する」が支配戦略となっており, 容疑者 A, B 共に「自白する」ことが予測できる.

2.1.3 一般の囚人のジレンマ

一般に囚人のジレンマとは, 以下の 3 つの性質を満たすものをいう.

囚人のジレンマの 3 つの性質

1. 自分にとって相手が何をしてきても, 非協力的に行動した方が利得が高い（非協力的に行動することが支配戦略）
2. 自分が非協力的に行動すると相手の利得は下がる
3. 互いが私的利益を追求することで, 協力した時と比べてそれぞれの利得が下がってしまう

そして, 戦略形は下の表 2.1 のようになる.

表 2.1: 一般の囚人のジレンマ 戦略形

容疑者 A / 容疑者 B	自白しない	自白する
自白しない	(a, a)	(c, b)
自白する	(b, c)	(d, d)

ただし, 利得 $a,\ b,\ c,\ d$ は以下を満たす.

$$b > a > d > c \ \wedge \ 2a > b + c$$

一般の囚人のジレンマにおける戦略について説明を加える. このゲームでは, 容疑者 A, B のどちらにとっても「自白する」することが支配戦略である. したがって, **「自白する」ことが「非協力的な行動」,「自白しない」ことが「協力的な行動」に対応する**.

次に, 利得 $a,\ b,\ c,\ d$ が満たす関係式について説明する. まず, 上述した「囚人のジレンマの 3 つの性質」の性質 1, 性質 2, 性質 3 はそれぞれ $a < b \ \wedge \ c < d,\ a > c \ \wedge \ b > d,\ a > d$ に対応する. これらから $b > a > d > c$ が導かれる. 加えて,「一方が協力し他方が協力しなかった結果, 双方が協力し合った場合以上の合計利得をもたらした」という事態を防ぐために, $2a > b + c$ も要求される（囚人のジレンマゲームでは, 互いが協力し合った場合に合計利得が最大になることを想定している）. 以上から, 利得 $a,\ b,\ c,\ d$ が満たす関係式 $b > a > d > c \ \wedge \ 2a > b + c$ が導かれる.

2.1.4 「評判」による囚人のジレンマへの対応

1.1.1 節で見た通り, 囚人のジレンマゲームでは個人の最善と社会の最善が一致していない. そこで, このゲームは多分野にわたる学者の関心を集め, 様々な対策が考えられた.

解決法の一つとして, 取引が繰り返し行われる場合には,「**評判**」を使うことが有効な方法として存在する. 協力しないプレイヤーはよい評判を失うとし, よい評判のプレイヤーにのみ他のプレイヤーが取引に協力的に応じるとする. これは,**「協力しない人には将来, 相手からの協力が保証されない」という罰則が設けられている**と解釈することができる. この罰則の存在により, プレイヤーは評判の下落を恐れ, 自分勝手な行動を控えることになる. 長期的な関係の中でこのような罰則が可能になれば, 利己的なプレイヤーがゲームの中で協力関係を築くことができる.

2.2 強化学習の全体像

本節の内容は, 書籍 [7] を参考に執筆した.

2.2.1 強化学習とは

人工知能関連のキーワードとしての強化学習の位置付けについて説明する. まず**機械学習**とは, 何かしらの目的を達成するための知識や行動を, データを読み込ませることで機械に獲得させるための技術である. すなわち, 人工知能を実現する技術の一つである.「機械」はモデルとも呼ばれ, 実体はパラメータを持った数式になる. このパラメータを, 与えたデータに合うよう調整する作業が「学習」になる.

機械学習の代表的方法は, 学習プロセスによって以下のように大きく 3 つに大別される.

- **教師あり学習**
  データと正解（ラベル）をセットで与えて, データを与えたらラベルが出力されるよう, モデルのパラメータを調整する.
- **教師なし学習**
  データのみを与えて, データの特徴（構造や表現）を抽出できるようにパラメータを調整する.
- **強化学習**
  行動により報酬が得られる環境（タスク）を与えて, 各状態で報酬につながる行動が出力されるように, モデルのパラメータを調整する.

このように, 強化学習は機械学習における一つの学習手法となっている.

2.2.2 マルコフ決定過程と方策

強化学習では, 環境から与えられる報酬を最大化する**方策**（行動の選択方法）の学習という形で問題が定式化される. これが非常に幅広い応用を持ち, ゲーム AI やロボットの制御, レコメンドへの利用や囲碁・将棋といったボードゲームにまでも用いられている. これらは全て,「今の状態に応じて行動を決定し, その結果として成功や失敗などのフィードバックがあり, 次の状態に移る」という共通の構造を持っている. このフィードバックは, **報酬**という形で定式化される. ここで, 行動の決定を行う主体を**エージェント**といい, 状態の遷移や報酬などを与える主体を**環境**という.

強化学習に現れる状態, 行動, 報酬という構造は, 数学的には**マルコフ決定過程（MDP）**を用いて定式化される.

マルコフ決定過程は, 次の 5 つの組のことをいう.

マルコフ決定過程の 5 要素

1. 状態の集合 $\boldsymbol{S}$
   あり得る全ての状態を集めた集合を $\boldsymbol{S}$ と書く. 一つひとつの状態は $s \in \boldsymbol{S}$ と小文字の s で記す. 状態の確率変数は大文字の S で記す.
2. 行動の集合 $\boldsymbol{A}_s$
   各状態 s について, その状態で選択可能な行動全体の集合を $\boldsymbol{A}_s$ と書く. 一つひとつの行動は $a \in \boldsymbol{A}_s$ と小文字の a で記す. 全ての状態 s で $\boldsymbol{A}_s$ が共通の場合や, 状態の区別が重要でない場合は, 単に $\boldsymbol{A}$ と書くこともある. 行動の確率変数は大文字の A で記す.
3. 報酬の確率分布 $p_r(r \mid s, a)$
   マルコフ決定過程では, 行動を選択した後, 報酬と呼ばれる数値 $r \in \mathbb{R}$ が手に入る. 状態 s で行動 a を選択した場合の報酬 r の確率分布を, $p_r(r \mid s, a)$ と書く. 報酬の確率変数は大文字の R で記す.
4. 状態の確率分布 $p_s(s' \mid s, a)$
   マルコフ決定過程では, 行動を選択した後, 次の状態に遷移する. 状態 s で行動 a を選択した場合の次の状態 s' の確率分布を, $p_s(s' \mid s, a)$ と書く.
5. 初期状態の確率分布 $p_s(s_0)$
   マルコフ決定過程は初期状態 s_0 から始まる. この初期状態 s_0 の確率分布を, $p_s(s_0)$ と書く.

マルコフ決定過程が与えられると, 次のような状態遷移が考えられる.

まず, $p_s(s_0)$ に従って初期状態 s_0 が決定される. そこで, $\boldsymbol{A}_{s_0}$ から行動 $a_0 \in \boldsymbol{A}_{s_0}$ を選択すると, 確率分布 $p_r(r_0 \mid s_0, a_0)$ に従って報酬 r_0 が手に入り, 確率分布 $p_s(s_1 \mid s_0, a_0)$ に従って次の状態 s_1

が決まる. これを繰り返していくことで, $s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_t, a_t, r_t, \cdots$ という状態, 行動, 報酬の列が得られる.

次に, エージェントの定式化である方策について説明する. エージェントは, 各状態 $s \in \boldsymbol{S}$ において, 行動 $a \in \boldsymbol{A}_s$ を選択する. この仕組みを数学的に定式化したものが**方策（policy）**である. 方策 π は, 各状態 $s \in \boldsymbol{S}$ に対して, 行動 $a \in \boldsymbol{A}_s$ を選択する確率分布 $\pi_a(a \mid s)$ として定式化される.

方策 π が確率分布であることには,「マルコフ決定過程が確率を含む概念であるが故に, 方策も確率的な概念を用いるのが自然であるから」,「良い方策がまだ未知である場合は, 各状態 s において様々な行動を試してみる必要があるから」といった理由が存在する.

2.2.3 価値関数と最適方策

各状態での行動により得られる報酬の累積和を**収益**と呼ぶ. **強化学習の目的は, 環境と相互作用して得られる収益を最大化する方策を見つけることである**.

最大化したい収益 g は,

$$g = \sum_{t \geq 0} r_t$$

と表される. しかし, r_t や g が確率的に変動するため, その最大化は単純ではない. そこで, 収益の期待値を最大化することを考える. まず, 収益と報酬の確率変数をそれぞれ G, R_t で表すと,

$$G = \sum_{t \geq 0} R_t$$

である. すると, 期待値の線形性より, 収益の期待値は

$$E[G] = E\left[\sum_{t \geq 0} R_t\right] = \sum_{t \geq 0} E[R_t]$$

と表すことができる.

この期待値は無限和なので, 発散する可能性がある. これを回避するために, $0 \leq \gamma \leq 1$ なる γ（**割引率**と呼ばれる）を用いて G を

$$G = \sum_{t \geq 0} \gamma^t R_t$$

と再定義する. この G を**割引収益**という. なお割引率は, **長期の状況で収益を考える場合, 現時点から見た将来の報酬の価値は若干低い**ということを反映する役割を持っている.

結果的に, 我々は

$$E[G] = E\left[\sum_{t \geq 0} \gamma^t R_t\right] = \sum_{t \geq 0} \gamma^t E[R_t]$$

を最大化すればよいことがわかった. この値を**期待割引収益**という.

特に, 状態が s であり, そこで行動 a を取った場合, それ以降の収益の条件付き期待値を $Q(s,a)$ と書き, これを**行動価値関数**という. すなわち, 行動価値関数 $Q(s_t,a_t)$ は

$$Q(s_t,a_t) = E\left[\sum_{t'\geq t}\gamma^{t'-t}R_{t'}\right]$$

と書ける.

これを用いてさらに計算を進めると, 行動価値関数 $Q(s_t,a_t)$ は

$$\begin{aligned} Q(s_t,a_t) &= E\left[\sum_{t'\geq t}\gamma^{t'-t}R_{t'}\right] \\ &= E[R_t] + \gamma E[R_{t+1}] + \gamma^2 E[R_{t+2}] + \cdots \\ &= E[R_t] + \gamma(E[R_{t+1}] + \gamma E[R_{t+2}] + \cdots) \\ &= E[R_t] + \gamma Q(s_{t+1},a_{t+1}) \end{aligned} \tag{2.1}$$

のように再帰的な式で表現可能であることが示される.

ある方策 π^* が存在して, その方策 π^* を利用すると, 行動価値関数の値が最大化されるとする. この π^* を, **最適方策**という. 最適方策 π^* に従って行動した場合の行動価値関数を $Q^{\pi^*}(s,a)$ と書き, これを**最適行動価値関数**という.

強化学習の究極の目標は, この最適方策 π^* を見つけることだといえる. 現実的に最適方策の発見が不可能な場合は, なるべく最適方策に近い, 価値関数の値が大きくなる方策を見つけることが目標となる. これを, **最適方策の近似**という.

以降, 本論文では行動価値関数の**推定値**を $\hat{Q}(s,a)$ と書き, (行動価値関数の**推定値**) と (行動価値関数の**真の値**) の差の絶対値 $\left|\hat{Q}(s,a) - Q(s,a)\right|$ の値が小さいことを**精度良く行動価値関数 Q を推定している**と表現することにする.

2.2.4 強化学習の学習プロセス

強化学習を実施する際の典型的な流れを以下に示す.

強化学習の流れ

1. 方策 π や行動価値関数の**推定値** $\hat{Q}(s,a)$ を初期化する
2. 方策 π に従って行動を選択し, データを溜める
3. 得られたデータを元に, 方策 π や行動価値関数の**推定値** $\hat{Q}(s,a)$ を更新する
4. 以下, 2 と 3 を学習が十分進むまで繰り返す
5. 学習が十分に進んだら, 状態 s に対して行動価値関数の**推定値** $\hat{Q}(s,a)$ が最大になる行動 a（複数ある場合は, これらからランダムな行動）を選択するような方策を最適方策 π^* とする（最適方策の近似）

この流れのように, データ収集, 価値関数の推定値の更新, 方策の更新を繰り返す手法を **Generalized Policy Iteration（GPI）**という. なお, Iteration とは「繰り返し」や「反復」,「ループ」を意味する英単語である.

最適方策を発見する方法としては, 良い方策 π を直接見つける方法と, 行動価値関数 Q を経由する方法の 2 つが存在する. 以下では, 本研究で採用する後者について述べる.

精度良く行動価値関数 Q の値を推定する方法は, 大きく 2 種類存在する. 1 つ目は現在利用中の方策 π から得られたデータを元に, この方策 π についての行動価値関数 Q^π の値の推定を中心に据える方法で, これは **On-Policy** な手法と呼ばれる. 2 つ目は利用中の方策とは関係なく最適方策 π^* の行動価値関数 Q^{π^*} の値を推定する方法で, これは **Off-Policy** な手法と呼ばれる.

どちらの手法でも, 行動価値関数の値の推定がある程度進んだら, 現在の方策を ϵ-greedy $\hat{Q}$ で更新する. **ϵ-greedy $\hat{Q}$ とは, 確率 ϵ（$0 \leq \epsilon \leq 1$）でランダムな行動を選択し, 確率 $1-\epsilon$ で現在の行動価値関数の推定値 $\hat{Q}$ が最大になる行動（複数ある場合は, これらからランダムな行動）を選択するような方策を表す**. これは, **ϵ 貪欲（ϵ-greedy）**な手法と呼ばれる. このように ϵ-greedy $\hat{Q}$ では, 現在の行動価値関数の**推定値**が最大になる行動を **1 より小さい**確率で選択する. その理由は, **学習の途中段階では, その時点での行動価値関数の推定値の精度は決して高くないため, $\hat{Q}(s,a)$ が最大となる行動以外の行動も選択される可能性を残しておく（取得するデータに多様性をもたせる）ことが望ましい**からである.

2.2.5 Q 学習

最後に, 本研究で用いる Q 学習について簡単に説明する. Q 学習は, 強化学習の様々な分析モデルの一つであり, これは Off-Policy な手法である. Q 学習の考え方を以下に示す.

Q 学習ではまず, 方策 π と最適行動価値関数の推定値 $\hat{Q^{\pi^*}}$ を適当に初期化する. 理想的な目標は,「状態 s_t で行動 a_t を選択した場合の最適行動価値関数の**真の値** $Q^{\pi^*}(s_t, a_t)$ を求めること」である. しかし, 現在手元にある値は, 各 s_t, a_t に対する最適行動価値関数の**推定値** $\hat{Q^{\pi^*}}(s_t, a_t)$ である. よって, これらの推定値を用いて真の値を探るために, 以下のように考える.

式 (2.1) より,

（状態 s_tで行動 a_tを選択した場合の行動価値関数の**真の値**）
＝（状態 s_tで行動 a_tを選択した時の報酬の期待値）
　＋$\gamma \times$（状態 s_{t+1}で行動 a_{t+1}を選択した場合の行動価値関数の**真の値**）

である. したがって,（状態 s_t で行動 a_t を選択した場合の**最適**行動価値関数の**真の値**）は, 次を満たす.

（状態 s_tで行動 a_tを選択した場合の**最適**行動価値関数の**真の値**）
＝（状態 s_tで行動 a_tを選択した時の報酬の期待値）
　＋$\gamma \times$（状態 s_{t+1}において, 最も大きな行動価値関数の値を与えるような行動
　　を選択した場合の行動価値関数の**真の値**）

これを数式で書くと, 以下のようになる.

$$Q^{\pi^*}(s_t, a_t) = E[R_t] + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} Q^{\pi^*}(s_{t+1}, a')$$

本研究における Q 学習では,「状態 s_t で行動 a_t を選択した場合の報酬 r_t」として, 5.5 節にて定義される「報酬の確率分布 $p_r(r_t \mid s_t, a_t)$」に従って決定された**定数値**を採用する. よって, 以下では $E[R_t]$ を定数 r_t で置き換える. すると,

$$Q^{\pi^*}(s_t, a_t) = r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} Q^{\pi^*}(s_{t+1}, a') \tag{2.2}$$

が得られる. この式 (2.2) と, 手元にある最適行動価値関数の**推定値**を用いて最適行動価値関数の**真の値** $Q^{\pi^*}(s_t, a_t)$ を近似的に求める. そのための考え方を以下に示す.

以下の近似を行う.

$$\begin{aligned} Q^{\pi^*}(s_t, a_t) &= r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} Q^{\pi^*}(s_{t+1}, a') \\ &\approx r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} \hat{Q^{\pi^*}}(s_{t+1}, a') \end{aligned}$$

よって, 現在の $\hat{Q^{\pi^*}}(s_t, a_t)$ を $r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} \hat{Q^{\pi^*}}(s_{t+1}, a')$ に近づけるような修正を加える. この修正を（現実的に無限回繰り返すことは不可能なので）有限回繰り返すことによって, 最適行動価値関数の**真の値** $Q^{\pi^*}(s_t, a_t)$ を近似的に求めることができる.

以上の考え方を元に, Q 学習では方策 π と最適行動価値関数の推定値 $\hat{Q^{\pi^*}}$ を適当に初期化した後, 集まった s_t, a_t, r_t, s_{t+1} の組のデータを利用して

$$\hat{Q^{\pi^*}}(s_t, a_t) \longleftarrow \hat{Q^{\pi^*}}(s_t, a_t) + \alpha \delta_t \tag{2.3}$$

のように最適行動価値関数の推定値 $\hat{Q^{\pi^*}}$ を更新していく学習を行う. ここで α は**学習率**と呼ばれる正の実数であり, 更新による変更度合いを表す. また, δ_t は **TD 誤差**と呼ばれ,

$$\delta_t = r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} \hat{Q^{\pi^*}}(s_{t+1}, a') - \hat{Q^{\pi^*}}(s_t, a_t) \tag{2.4}$$

で定義される.

$\hat{Q^{\pi^*}}$ の更新が充分に進んだら, 定期的に現在の方策を ϵ-greedy $\hat{Q^{\pi^*}}$ で更新する. そして, その新しい方策を用いて再びデータ収集を行い, 得られたデータでさらに $\hat{Q^{\pi^*}}$ の更新を進める. このように, Q 学習は GPI の一例となっている.

最後に, 学習率 α に関する補足説明をする. 学習率は機械学習における学習スピードと比例する. 学習率の値が高いと学習スピードは上がる一方で, 最適解を飛び越えて更新してしまう可能性があるため, 精度良く最適行動価値関数を推定できない恐れがある. 反対に学習率の値が低いと, 精度良く最適行動価値関数を推定できるが, 学習スピードが低下してしまう. したがって, **学習率の値は固定せずに, 様々な値を試しながらバランスの良い値を設定することが重要となる**.

第 3 章

シミュレーションに関する各種ゲームの定義

本章では, 次章以降の数理モデルに登場する各種ゲームの定義を行う.

まず, **囚人のジレンマゲームを行う**という言葉を以下に定義する.

囚人のジレンマゲームを行う

個体 A と個体 B がそれぞれ自身の戦略に基づいて行動を**同時に**選択し, その行動を起こす（同時手番）. そして, 個体 A と個体 B の行動の組み合わせに応じて, それぞれは以下の戦略形に従って利得を得る.

表 3.1: 囚人のジレンマ 戦略形

個体 A / 個体 B	協力	裏切り
協力	(a, a)	(c, b)
裏切り	(b, c)	(d, d)

ただし, 利得 $a,\ b,\ c,\ d$ は以下を満たす（2.1.3 節参照）.

$$b > a > d > c > 0 \ \wedge \ 2a > b + c \tag{3.1}$$

戦略形について説明を加える. 2.1.3 節にて「『自白する』ことが『非協力的な行動』,『自白しない』ことが『協力的な行動』に対応する」と述べた. これを踏まえ, 行動の名称を「自白しない」から「協力」へ,「自白する」から「裏切り」へと変更した.

また, 利得を表すパラメータについて, a, b, c, d を**正に限定**しているのは, 2.1.1 で述べたゲーム理論における利得の条件「利得の間の相対関係だけが意味を成すこと」を用いて議論を簡単にするた

めである.

次に, **二次元格子空間上での囚人のジレンマゲーム**を以下に定義する.

二次元格子空間上での囚人のジレンマゲーム

二次元平面上に正方形の格子セルが一様に広がっている. 各格子セルを, ある戦略を持った個体とみなし, 全個体が**同時に**以下に従う.

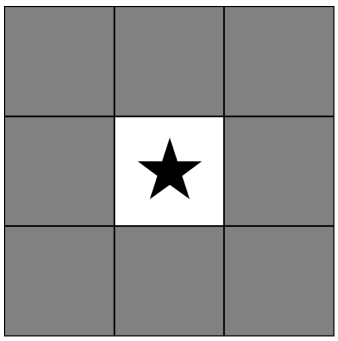


図 3.1: ムーア近傍

各個体は, 右図 3.1 において黒色で塗り潰された近接 8 個体（ムーア近傍という）それぞれと囚人のジレンマゲームを行う. 近接 8 個体それぞれと囚人のジレンマゲームを行う間, 自身の行動選択は同一のものであり, 対戦個体によって行動は変えない. 各個体の獲得利得は, **その個体が近接 8 個体それぞれと囚人のジレンマゲームを行った時に得られた 8 回分の利得の合計**とする.

最後に, **二次元格子空間上での繰り返し囚人のジレンマゲーム**を以下に定義する.

二次元格子空間上での繰り返し囚人のジレンマゲーム

ある時刻に**二次元格子空間上での囚人のジレンマゲーム**を行う. そして, 次の時刻に各個体は改めて自身の戦略に基づいて行動を**同時に**選択し, 再び**二次元格子空間上での囚人のジレンマゲーム**を行う. これを繰り返す.

第 4 章

進化論的アプローチ

本章では, 1 つ目のアプローチである**進化論的アプローチ**に基づいたシミュレーション結果をまとめ, 考察を行う.

4.1 数理モデルの構築

本アプローチでは, 次の数理モデルに従いシミュレーションを行う.

進化論的アプローチの数理モデル

n を正整数とし, モデルの対象空間を $n \times n$ の二次元セルとする. これは, $n \times n$ 体の意思決定個体を表す. また, この対象空間には 1 行目と n 行目, 1 列目と n 列目が接続しているという**周期的境界条件**が課されているとする.「時刻 t（$t \in \mathbb{Z}_{\geq 0}$）」をパラメータとして導入し, 以下の流れを繰り返す. ただし, **初期時刻（時刻 $t = 0$）での各個体の行動はランダムに決定される**とする.

時刻 t と時刻 $(t+1)$ の間に, 各個体は**二次元格子空間上での囚人のジレンマゲーム**を行う. ここで, 各個体が選択する行動は時刻 t で更新された行動とする. 各時刻 $t \geq 1$ に各個体は, **直前に行われた二次元格子空間上での囚人のジレンマゲームに関する**次の確率で行動を a（行動 a は「協力」または「裏切り」のどちらか）に更新する.

$$\frac{\text{自身及び対戦個体計 9 個体のうち, 行動 } a \text{ を選択した個体の獲得利得の合計}}{\text{自身及び対戦個体計 9 個体の獲得利得の合計}}$$

進化論的アプローチは, 各行動選択による獲得利得が, 自身及び対戦個体計 9 個体の獲得利得の合計に対してどれほどの貢献をもたらしたかを考え, その貢献度を確率として自身の次の行動が定まるというものである. 各個体は対戦個体とゲームをする度に, 相手が現在どのような行動を選択しており, その結果どれだけの利得を獲得したのかという情報を手に入れる. そして, より大きな利

得が期待される行動に高確率で更新する. 環境に適応しない行動は淘汰されるというイメージである. **囚人のジレンマゲームでは「裏切り」が支配戦略である（2.1.2 節参照）ため, 時間発展ともに「裏切り」を選択する個体が増加することが予想される**.

4.2 シミュレーションの実行

4.2.1 実行環境

Python IDLE (Integrated DeveLopment Environment) 上でシミュレーションを行った. これは, Python プログラミング言語の統合開発環境で, Python に付属している. **シミュレーションの具体的な内容は, Python におけるデータ可視化ライブラリである Matplotlib を用いて「各時刻 t における, 各個体の行動更新の結果」をアニメーションとして連続的に表示させることである**. Matplotlib の最も重要な機能の 1 つは, 多様なオペレーティングシステム (OS) とグラフィックシステム上で非常にうまく機能することであり, どのような OS を使っていても, どのようなフォーマットで出力しようとしても問題なく動作させることが可能である.

4.2.2 パラメータの設定

進化論的アプローチの数理モデルをシミュレーションする際, モデルに現れるパラメータに設定した具体的な値を以下に示す. ここでは以下のように利得の値に 2 通りの値を設定してそれぞれの場合についてシミュレーションを行った.

1. $(n, a, b, c, d) = (100, 1060, 1100, 1000, 1030)$
2. $(n, a, b, c, d) = (100, 1080, 1140, 1000, 1040)$

4.2.3 ソースコード

シミュレーションを実行するにあたり実装したソースコードを, ソースコード 4.1 に示す. Python の文法について, 筆者は書籍 [8] を用いて学習した.

ソースコード 4.1 について説明する. このソースコードは, 1 つ目のパラメータ設定値に対応したものである. 意思決定個体は計 $100 \times 100 = 10000$ 人であり, これらの個体を 100×100 の 2 次元配列で表現している.

FuncAnimation の引数には, アニメーションにする対象の fig や更新関数, 初期化関数などが入っている. 最終的に変数 time=500 回だけ各個体の行動が高速に更新され, 更新の度にその結果が 2 次元セルとして連続的に表示される. 上のプログラムを Python IDLE 上で開き, Run → Run Module と実行すれば, 協力現象の時間発展を可視化したアニメーションが GIF ファイルとして保存される.

ソースコード 4.1: approach1 prisoner's dilemma simulation

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.animation import PillowWriter

H = 100
W = 100
time = 500 # 更新回数の上限
# 協力は 1
# 裏切りは 0
action = np.random.randint(0, 2, size=(H, W))

# 更新の処理
def update():
    global action
    dH = [-1, 0, 1]
    dW = [-1, 0, 1]
    new_action = np.full((H, W), -1)
    benefit = np.full((H, W), 0)
    for h in range(H):
        for w in range(W):
            for dh in dH:
                for dw in dW:
                    if dh == dw == 0:
                        continue
                    if action[h, w] == 1 and action[(h + dh) % H, (w + dw) %
                         W] == 1:
                        benefit[h, w] += 1060
                    elif action[h, w] == 1 and action[(h + dh) % H, (w + dw)
                        % W] == 0:
                        benefit[h, w] += 1000
                    elif action[h, w] == 0 and action[(h + dh) % H, (w + dw)
                        % W] == 1:
                        benefit[h, w] += 1100
                    else:
                        benefit[h, w] += 1030
    for h in range(H):
        for w in range(W):
            action_list = [1, 0]
            cooperation = 0
```

```
            betrayal = 0
            for dh in dH:
                for dw in dW:
                    if action[(h + dh) % H, (w + dw) % W] == 1:
                        cooperation += benefit[(h + dh) % H, (w + dw) % W]
                    else:
                        betrayal += benefit[(h + dh) % H, (w + dw) % W]
            prob_list = [
                cooperation / (cooperation + betrayal),
                betrayal / (cooperation + betrayal),
            ]
            new_action[h, w] = np.random.choice(a=action_list, p=prob_list)
    action = new_action
    return


# アニメーションで可視化
fig, ax = plt.subplots()
im = ax.imshow(action, cmap="autumn", vmax=1, vmin=0)
text = ax.set_title("time={}".format(0))
ax.set_aspect("equal")

# 初期化関数
def init():
    im.set_data(action)
    text.set_text("time={}".format(0))
    return (im,)


# 更新関数
def update_anim(dt):
    global action
    update()
    im.set_data(action)
    text.set_text("time={}".format(dt))
    return (
        im,
        text,
    )


ani = FuncAnimation(
```

```
    fig,
    update_anim, # 更新関数
    init_func=init,
    frames=np.arange(1,time+1),
    interval=100, # 更新間隔 (ms)
    repeat=True, # 繰り返し表示させる
    blit=True,
)
# アニメーションの保存
ani.save("approach1_prisoner's_dilemma_simulation_1.gif", writer="pillow", dpi
    =200)
```

4.3 シミュレーション結果の可視化及び分析

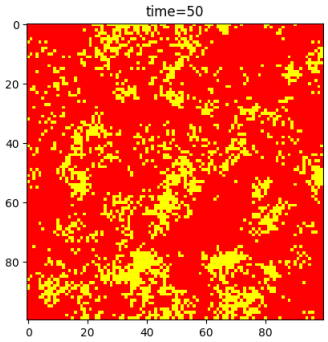
以下では, 1 つ目のパラメータ設定, 2 つ目のパラメータ設定をそれぞれ設定 1, 設定 2 と呼ぶ. 設定 1 と設定 2 それぞれにおけるシミュレーション結果を, 次ページ以降に示す. なお, Matplotlib による可視化にあたり, 書籍 [9] を参考にした.

図 4.1 と図 4.2 は, それぞれの設定における各個体の行動更新の時間発展を可視化したものである. **各時刻 t で行動が「協力」に更新された個体は黄色で,「裏切り」に更新された個体は赤色で示されている**. ここで**協力個体率**とは, その時刻 t において行動を「協力」に更新した個体数が全個体数に占める割合を百分率で表したものである. いずれの設定の場合に関しても, 時間発展とともに支配戦略である「裏切り」を選択する個体が増加していくことがわかる. よって,**「時間発展とともに『裏切り』を選択する個体が増加する」という 4.1 節での予想は的中している**と考えられる.

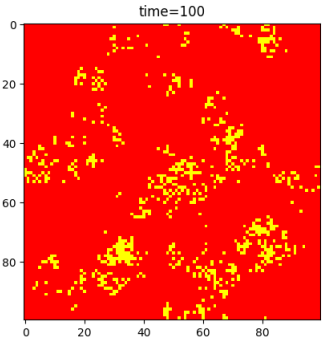
図 4.3 と図 4.4 は, 各行動へ自身の行動を更新した個体数の時間変化をグラフで示したものである. 行動を「協力」に更新した個体数の時間推移は「cooperation count」で, 行動を「裏切り」に更新した個体数の時間推移は「betrayal count」で表されている. betrayal count について見ると, **設定 1 の場合よりも設定 2 の場合の方が行動を「裏切り」に更新する個体が急激に増えていく**ことがわかる.

図 4.5 と図 4.6 は, 近接相互作用の関係にある 2 個体の全組み合わせ数に占める, 相互協力が実現している組み合わせ数の割合を**相互協力率**と定義し, その時間変化をグラフで示したものである. 設定 1 の場合よりも設定 2 の場合の方がほぼ単調に相互協力率は下がっていき, 終盤になると相互協力の関係は消滅してしまっていることがわかる.

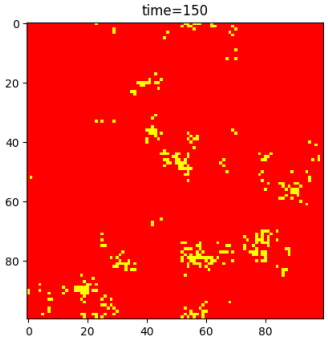
このように, 設計した**進化論的アプローチの数理モデル**は 4.1 節での予想をうまく再現するものであったが, **利得を表すパラメータの値によって, 行動を「裏切り」に更新する個体の増加の勢いに差が見られる**ことがわかった.



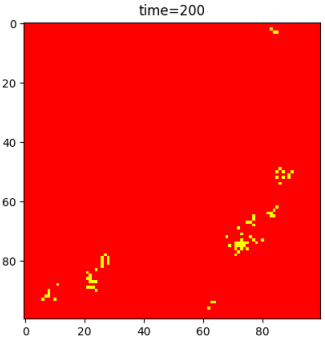
(a) $t = 50$ における行動更新の分布
協力個体率 21.44%



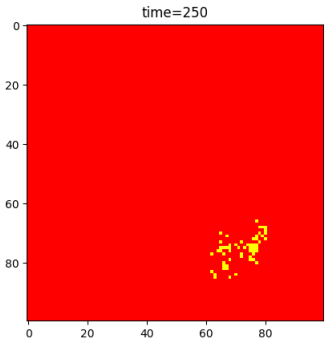
(b) $t = 100$ における行動更新の分布
協力個体率 7.66%



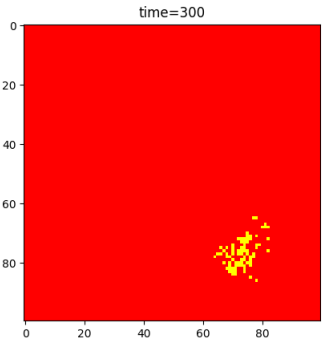
(c) $t = 150$ における行動更新の分布
協力個体率 3.46%



(d) $t = 200$ における行動更新の分布
協力個体率 0.76%

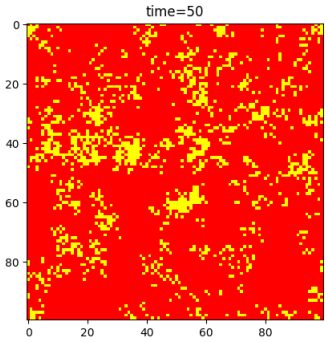


(e) $t = 250$ における行動更新の分布
協力個体率 0.58%

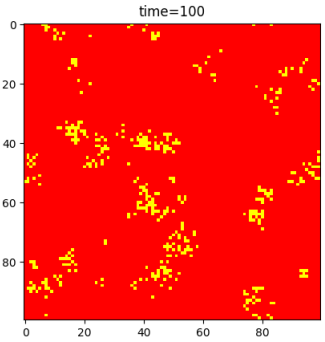


(f) $t = 300$ における行動更新の分布
協力個体率 0.8%

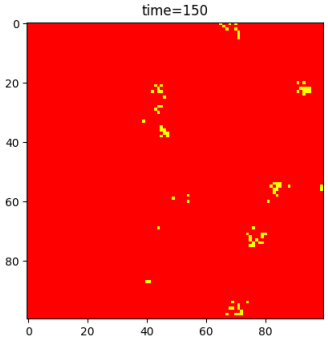
図 4.1: 設定 1 におけるシミュレーション結果



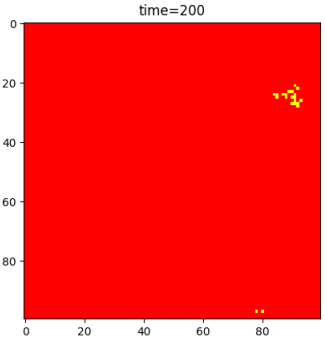
(a) $t = 50$ における行動更新の分布
協力個体率 14.23%



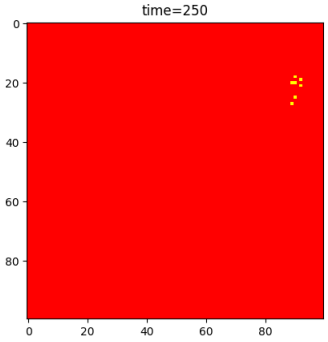
(b) $t = 100$ における行動更新の分布
協力個体率 4.31%



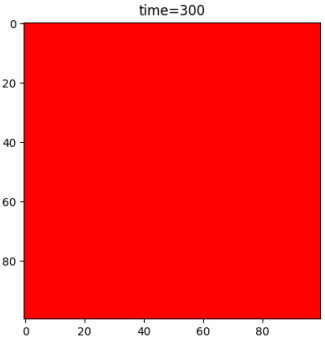
(c) $t = 150$ における行動更新の分布
協力個体率 0.83%



(d) $t = 200$ における行動更新の分布
協力個体率 0.21%



(e) $t = 250$ における行動更新の分布
協力個体率 0.07%



(f) $t = 300$ における行動更新の分布
協力個体率 0.0%

図 4.2: 設定 2 におけるシミュレーション結果

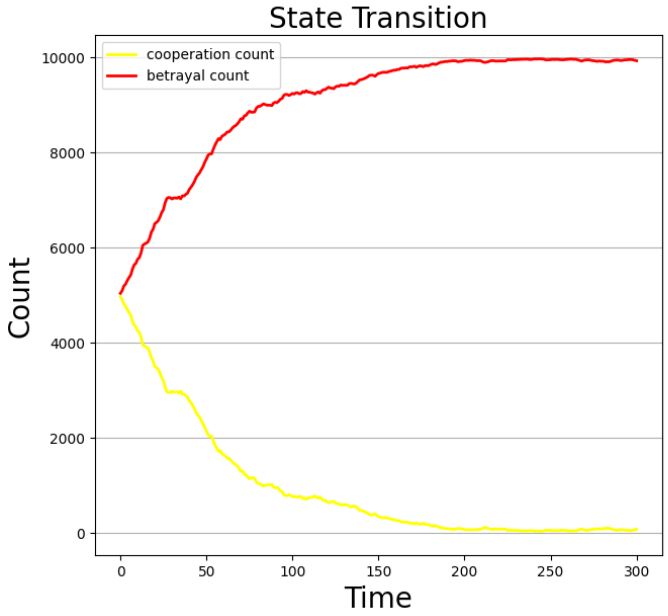


図 4.3: 設定 1 における, 各行動へ自身の行動を更新した個体数の時間変化

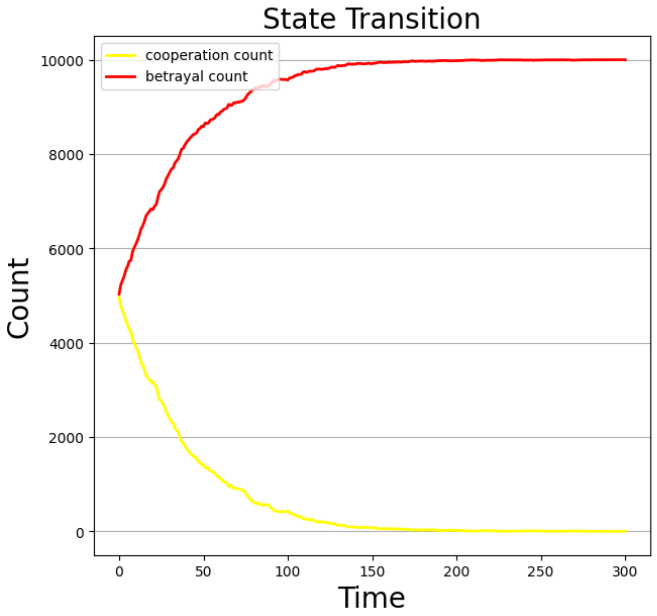


図 4.4: 設定 2 における, 各行動へ自身の行動を更新した個体数の時間変化

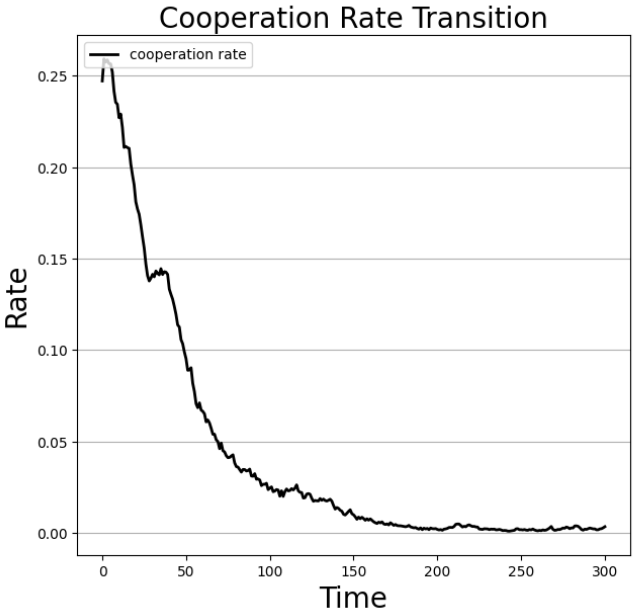


図 4.5: 設定 1 における, 相互協力率の時間変化

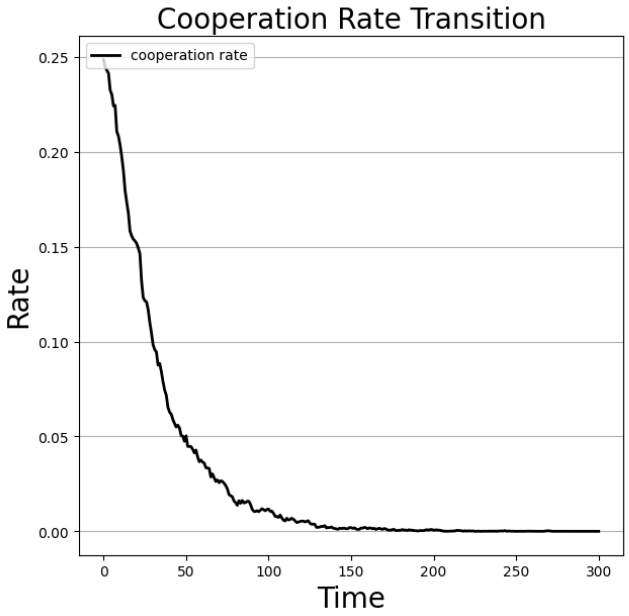


図 4.6: 設定 2 における, 相互協力率の時間変化

4.4 考察

4.4.1 状態の収束に関する考察

シミュレーションの結果から, **十分な時間が経過すると行動「協力」を選択する個体数が 0 に収束する（協力個体率が 0% に収束する）**ことが予想される. 簡単のために, ここでは

- **全ての個体について, 時刻 $t = n$（$\in \mathbb{Z}_{\geq 0}$）で行動を「協力」に更新する確率は共通である**
- **獲得利得の期待値を真の獲得利得であるとみなす**

の 2 点を仮定してこれを証明する.

漸化式の導出

ある 1 個体が時刻 $t = n$（$\in \mathbb{Z}_{\geq 0}$）で行動を「協力」に更新する確率を p_n とおく（行動を「裏切り」に更新する確率は $(1 - p_n)$ となる）. 1 つ目の仮定より, 全個体に同等の p_n が適用される. また, 初期状態の定め方より $p_0 = 1/2$ である. $t = n$ に更新された行動は $(n + 1)$ 回目のゲーム時に参照されることに注意する.

以下では, 全て $(n + 1)$ 回目の二次元格子空間上での囚人のジレンマゲームについて議論する. まず, ある 1 個体が獲得する利得の期待値 E_{n+1} は

$$E_{n+1} = 8 \times \{a \cdot p_n \cdot p_n + b \cdot (1 - p_n) \cdot p_n + c \cdot p_n \cdot (1 - p_n) + d \cdot (1 - p_n) \cdot (1 - p_n)\}$$

と表せる. したがって, 自身及び対戦個体計 9 個体の獲得利得の合計の期待値は $9E_{n+1}$ となる.

次に, ある 1 個体について, それが行動「協力」を選択して獲得する利得の期待値 $E_{\mathrm{co},n+1}$ は

$$E_{\mathrm{co},n+1} = 8 \times \{a \cdot p_n \cdot p_n + c \cdot p_n \cdot (1 - p_n)\}$$

と表せる. 同様に, ある 1 個体について, それが行動「裏切り」を選択して獲得する利得の期待値 $E_{\mathrm{be},n+1}$ は

$$E_{\mathrm{be},n+1} = 8 \times \{b \cdot (1 - p_n) \cdot p_n + d \cdot (1 - p_n) \cdot (1 - p_n)\}$$

と表せる.

よって, $R_{\mathrm{co},n+1}$ を

$$R_{\mathrm{co},n+1} = \frac{\text{自身及び対戦個体計 9 個体のうち, 行動「協力」を選択した個体の獲得利得の合計}}{\text{自身及び対戦個体計 9 個体の獲得利得の合計}}$$

とおけば, 2 つ目の仮定を考慮すると,

$$R_{\mathrm{co},n+1} = \frac{9E_{\mathrm{co},n+1}}{9E_{n+1}} = \frac{E_{\mathrm{co},n+1}}{E_{n+1}}$$

$$= \frac{8 \times \{a \cdot p_n \cdot p_n + c \cdot p_n \cdot (1-p_n)\}}{8 \times \{a \cdot p_n \cdot p_n + b \cdot (1-p_n) \cdot p_n + c \cdot p_n \cdot (1-p_n) + d \cdot (1-p_n) \cdot (1-p_n)\}}$$

$$= \frac{a \cdot p_n \cdot p_n + c \cdot p_n \cdot (1-p_n)}{a \cdot p_n \cdot p_n + b \cdot (1-p_n) \cdot p_n + c \cdot p_n \cdot (1-p_n) + d \cdot (1-p_n) \cdot (1-p_n)}$$

と計算できる. 同様に, $R_{\mathrm{be},n+1}$ を

$$R_{\mathrm{be},n+1} = \frac{\text{自身及び対戦個体計 9 個体のうち, 行動「裏切り」を選択した個体の獲得利得の合計}}{\text{自身及び対戦個体計 9 個体の獲得利得の合計}}$$

とおけば, 2 つ目の仮定を考慮すると,

$$R_{\mathrm{be},n+1} = \frac{9E_{\mathrm{be},n+1}}{9E_{n+1}} = \frac{E_{\mathrm{be},n+1}}{E_{n+1}}$$

$$= \frac{8 \times \{b \cdot (1-p_n) \cdot p_n + d \cdot (1-p_n) \cdot (1-p_n)\}}{8 \times \{a \cdot p_n \cdot p_n + b \cdot (1-p_n) \cdot p_n + c \cdot p_n \cdot (1-p_n) + d \cdot (1-p_n) \cdot (1-p_n)\}}$$

$$= \frac{b \cdot (1-p_n) \cdot p_n + d \cdot (1-p_n) \cdot (1-p_n)}{a \cdot p_n \cdot p_n + b \cdot (1-p_n) \cdot p_n + c \cdot p_n \cdot (1-p_n) + d \cdot (1-p_n) \cdot (1-p_n)}$$

と計算できる.

構築した**進化論的アプローチの数理モデル**によれば, この $R_{\mathrm{co},n+1}$ が p_{n+1} に等しく, $R_{\mathrm{be},n+1}$ が $(1-p_{n+1})$ に等しい. 実際, 上の計算結果を当てはめると

$$R_{\mathrm{co},n+1} + R_{\mathrm{be},n+1} = 1$$

が成り立ち, これは

$$R_{\mathrm{co},n+1} + R_{\mathrm{be},n+1} = p_{n+1} + (1-p_{n+1}) = 1$$

と辻褄が合う. したがって, この議論は妥当であるといえる.

以上の議論から, **冒頭の 2 点の仮定を認めると**, ある 1 個体が時刻 $t = n$（$\in \mathbb{Z}_{\geq 0}$）で行動を「協力」に更新する確率 p_n は, 漸化式

$$p_0 = \frac{1}{2}, \quad p_{n+1}\,(= R_{\mathrm{co},n+1}) = \frac{a(p_n)^2 + cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \quad (n \in \mathbb{Z}_{\geq 0}) \tag{4.1}$$

で定まる数列 $\{p_n\}$ に従うことが示された.

数列 $\{p_n\}$ が収束することの証明

ここでは, 漸化式 (4.1) で定まる数列 $\{p_n\}$ が収束することを

1. **数列 $\{p_n\}$ が有界であること**
2. **数列 $\{p_n\}$ が単調であること**

の 2 点を示すことによって証明する.

定理 1. $b > a > d > c > 0 \ \wedge \ 2a > b + c$ を満たす実数 a, b, c, d と漸化式

$$p_0 = \frac{1}{2}, \quad p_{n+1} = \frac{a(p_n)^2 + cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \quad (n \in \mathbb{Z}_{\geq 0})$$

で定まる数列 $\{p_n\}$ に対し,

$$0 < p_n \leq \frac{1}{2}$$

が成り立つ.

証明. 数学的帰納法により示す.

(I) $n = 0$ のとき

$$p_0 = \frac{1}{2}$$

であるから,

$$0 < p_0 \leq \frac{1}{2}$$

である.

よって, $n = 0$ のとき成り立つ.

(II) $n = k \in \mathbb{Z}_{\geq 0}$ のとき, $0 < p_k \leq \frac{1}{2}$ が成り立つと仮定する. このとき, $0 < p_{k+1} \leq \frac{1}{2}$ を示せばよい.

まず, $0 < p_{k+1}$ を示す. 仮定 $0 < p_k \leq \frac{1}{2}$ より $\frac{1}{2} \leq 1 - p_k < 1$ が従う. これと a, b, c, d が正であることより, 明らかに

$$p_{k+1} = \frac{a(p_k)^2 + cp_k(1-p_k)}{a(p_k)^2 + (b+c)p_k(1-p_k) + d(1-p_k)^2} > 0$$

である.

次に, $p_{k+1} \leq \frac{1}{2}$ を示す. 仮定 $0 < p_k \leq \frac{1}{2}$ より $0 < p_k \leq \frac{1}{2} \leq 1 - p_k < 1$ を得る. これと $b > a > d > c > 0$ を用いると, 漸化式より

$$\begin{aligned} p_{k+1} &= \frac{a(p_k)^2 + cp_k(1-p_k)}{a(p_k)^2 + (b+c)p_k(1-p_k) + d(1-p_k)^2} \\ &= \frac{a(p_k)^2 + cp_k(1-p_k)}{a(p_k)^2 + bp_k(1-p_k) + cp_k(1-p_k) + d(1-p_k)^2} \\ &< \frac{a(p_k)^2 + cp_k(1-p_k)}{a(p_k)^2 + a(p_k)^2 + cp_k(1-p_k) + cp_k(1-p_k)} \\ &= \frac{a(p_k)^2 + cp_k(1-p_k)}{2\{a(p_k)^2 + cp_k(1-p_k)\}} \\ &= \frac{1}{2} \, . \end{aligned}$$

よって, $p_{k+1} \leq \frac{1}{2}$ が成り立つ.

これで, $0 < p_{k+1}$ と $p_{k+1} \leq \frac{1}{2}$ の両方が示されたので, $0 < p_k \leq \frac{1}{2}$ の仮定のもとで $0 < p_{k+1} \leq \frac{1}{2}$ が示された.

以上 (I), (II) より, **定理 1** が示された. □

定理 2. $b > a > d > c > 0 \ \wedge \ 2a > b + c$ を満たす実数 a, b, c, d と漸化式

$$p_0 = \frac{1}{2}, \ \ p_{n+1} = \frac{a(p_n)^2 + cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \quad (n \in \mathbb{Z}_{\geq 0})$$

で定まる数列 $\{p_n\}$ は, 狭義単調減少数列である.

証明. 任意の $n \in \mathbb{Z}_{\geq 0}$ について

$$p_n - p_{n+1} > 0$$

を示せばよい.

$p_n - p_{n+1}$ を計算すると,

$$\begin{aligned}
&p_n - p_{n+1} \\
&= \frac{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \cdot p_n - \frac{a(p_n)^2 + cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \\
&= \frac{a(p_n)^3 + (b+c)(p_n)^2(1-p_n) + dp_n(1-p_n)^2 - a(p_n)^2 - cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \\
&= \frac{-a(p_n)^2(1-p_n) + b(p_n)^2(1-p_n) - cp_n(1-p_n)^2 + dp_n(1-p_n)^2}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \\
&= \frac{(b-a)(p_n)^2(1-p_n) + (d-c)p_n(1-p_n)^2}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2}
\end{aligned}$$

となる.

ここで, **定理 1** より任意の $n \in \mathbb{Z}_{\geq 0}$ について $0 < p_n \leq \frac{1}{2} \leq 1 - p_n < 1$ が成り立つことと, $b > a > d > c > 0$ より

$$p_n - p_{n+1} = \frac{(b-a)(p_n)^2(1-p_n) + (d-c)p_n(1-p_n)^2}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} > 0 \ .$$

以上より, **定理 2** が示された. □

以上**定理 1** 及び**定理 2** より, **数列 $\{p_n\}$ は下に有界な単調減少数列である**ので, 数列 $\{p_n\}$ は収束することが示された.

数列 $\{p_n\}$ が 0 に収束することの証明

これまでで, 数列 $\{p_n\}$ が収束することを証明した. ここでは, その数列 $\{p_n\}$ の極限値が 0 であること, すなわち

$$\lim_{n\to\infty} p_n = 0$$

を示す.

定理 3. $b > a > d > c > 0 \ \wedge \ 2a > b + c$ を満たす実数 a, b, c, d と漸化式

$$p_0 = \frac{1}{2}, \ \ p_{n+1} = \frac{a(p_n)^2 + cp_n(1-p_n)}{a(p_n)^2 + (b+c)p_n(1-p_n) + d(1-p_n)^2} \quad (n \in \mathbb{Z}_{\geq 0})$$

で定まる数列 $\{p_n\}$ に対し,

$$\lim_{n\to\infty} p_n = 0$$

が成り立つ.

証明. 数列 $\{p_n\}$ は収束することが保証されているので, その極限値を α とおくと, α は方程式

$$\alpha = \frac{a\alpha^2 + c\alpha(1-\alpha)}{a\alpha^2 + (b+c)\alpha(1-\alpha) + d(1-\alpha)^2}$$

の解である. この方程式を同値変形すると,

$$\alpha = \frac{a\alpha^2 + c\alpha(1-\alpha)}{a\alpha^2 + (b+c)\alpha(1-\alpha) + d(1-\alpha)^2}$$

$$\iff a\alpha^3 + (b+c)\alpha^2(1-\alpha) + d\alpha(1-\alpha)^2 - a\alpha^2 - c\alpha(1-\alpha) = 0$$

$$\iff \alpha\{\alpha^2(a-b-c+d) + \alpha(-a+b+2c-2d) - c + d\} = 0 \tag{4.2}$$

となる.

(i) $a - b - c + d = 0$ の場合

$a - b - c + d = 0 \iff -a + b = -c + d$ を方程式 (4.2) に代入し, 整理すると

$$\alpha(\alpha - 1)(c - d) = 0$$

が得られる. ここで, $c - d \neq 0$ より $\alpha = 0, 1$.

定理 1 及び**定理 2** より, 数列 $\{p_n\}$ は下に有界な単調減少数列であり, その極限値は下限 $\inf\{p_n\}$ である. また**定理 1** より $0 < p_n \leq \frac{1}{2}$ であるから, $\alpha = 1$ の解は極限値の候補からは除外され, 数列

$\{p_n\}$ の極限値は 0 となる.

(ii) $a-b-c+d \neq 0$ の場合

2 次方程式の解の公式を用いれば, 方程式の解は

$$\alpha = 0,\ \frac{-(-a+b+2c-2d) \pm \sqrt{(-a+b+2c-2d)^2 - 4(a-b-c+d)(-c+d)}}{2(a-b-c+d)}$$

$$= 0,\ \frac{a-b-2c+2d \pm \sqrt{(b-a)^2}}{2(a-b-c+d)}$$

$$= 0,\ \frac{a-b-2c+2d \pm (b-a)}{2(a-b-c+d)}$$

$$= 0, 1,\ \frac{-c+d}{a-b-c+d}\ .$$

ここで $b > a > d > c > 0$ より, $(-c+d)/(a-b-c+d)$ は $a-b-c+d > 0$ の場合は 1 より大きな正の数となり, $a-b-c+d < 0$ の場合は負の数となる.

定理 1 及び**定理 2** より, 数列 $\{p_n\}$ は下に有界な単調減少数列であり, その極限値は下限 $\inf\{p_n\}$ である. また**定理 1** より $0 < p_n \leq \frac{1}{2}$ であるから, $\alpha = 1$ 及び $\alpha = (-c+d)/(a-b-c+d)$ の解は極限値の候補からは除外され, 数列 $\{p_n\}$ の極限値は 0 となる.

以上 (i), (ii) より, **定理 3** が示された. □

定理 3 は, ある 1 個体が時刻 $t = n$（$\in \mathbb{Z}_{\geq 0}$）で行動を「協力」に更新する確率 p_n は $n \to \infty$ で 0 に収束する, すなわち**十分な時間が経過すると協力個体率が 0% に収束する**ことを主張している.

以上の考察により,

- **全ての個体について, 時刻 $t = n$（$\in \mathbb{Z}_{\geq 0}$）で行動を「協力」に更新する確率は共通である**
- **獲得利得の期待値を真の獲得利得であるとみなす**

の 2 点を仮定すれば, **十分な時間が経過すると行動「協力」を選択する個体数は 0 に収束する**ことが示された.

4.4.2 パラメータ設定に関する考察

4.3 節で見た通り, 設定 1 $(a, b, c, d) = (1060, 1100, 1000, 1030)$ の場合よりも設定 2 $(a, b, c, d) = (1080, 1140, 1000, 1040)$ の場合の方が, 行動を「裏切り」に更新する個体が急激に増加した. これは, 設定 2 の場合の方が p_n がより速く 0 に収束することを意味していると考えられる. これら 2 つのパラメータ設定の特徴は, **利得のパラメータの最小値 c は共通であるが, 最大値 b に関しては設定 2 の方が大きい**ということである. それに伴い, 条件 $2a > b + c$ を満たすように a, d の値も異なっている.

以下では, 4.4.1 節で証明した「**十分な時間が経過すると行動「協力」を選択する個体数は 0 に収束する**」という命題が**真であると**認めた上で, **全個体が行動を「裏切り」に更新するようになるまでの経過時間が短い**ことを**状態遷移の勢いが大きい**と表現することにする.

結果から述べると, 今回の考察段階では利得のパラメータから状態遷移の勢いを厳密に評価する定理を導出することはできなかった. しかし, 非常に感覚的ではあるが, 法則なるものは発見できたので, それを以下に記載しておく.

感覚的な法則

1. 全ての利得パラメータの値を C 倍（$C \in \mathbb{R}_{>0}$）しても, 状態遷移の勢いは不変である.
2. 利得のパラメータについて, c の値を共通にした場合は b の値が大きいほど, また b の値を共通にした場合は c の値が小さいほど状態遷移の勢いは大きくなる.

1 について補足する. これは, 漸化式 (4.1) において a, b, c, d 全体を定数倍しても約分により右辺の値が不変であることからわかる. またこの事実は, 2.1.1 節で述べたゲーム理論における利得の条件「**利得の間の相対関係だけが意味を成すこと**」の裏付けとなっている.

2 は, **利得のパラメータ全体のスケールを均一に変化させているわけではない**という点で 1 と異なっている. パラメータの一端を共通させたとき, パラメータ全体の広がり具合が大きいほど漸化式右辺の分母が分子に対して相対的に大きくなるので, 状態遷移の勢いは大きくなると考えられる.

まとめると, 状態遷移の勢いを推測するための（感覚的な）指標は「**利得のパラメータ自体の大きさを考慮した上での, パラメータ全体の広がり具合**」となる. 例えば $(a, b, c, d) = (3, 4, 1, 2)$ とすると, 各利得値は非常に小さいにも関わらず, 状態遷移の勢いは本シミュレーションにおける 2 つの設定の場合よりも格段に大きくなる. これは, 本シミュレーションの設定ではパラメータ全体の広がり具合がパラメータ自体のスケールの約 $1/10$ であるのに対し, $(a, b, c, d) = (3, 4, 1, 2)$ の場合はパラメータ全体の広がり具合がパラメータ自体のスケールと同程度に大きいからである.「**利得のパラメータ自体の大きさを考慮した上での, パラメータ全体の広がり具合**」が大きいほど, 裏切りによる利得のアドバンテージは莫大なものとなる. その結果, 意思決定個体は「協力」よりも「裏切り」が得であるということをより明確に判断し, 状態遷移の勢いが大きくなるのである.

第 5 章

強化学習アプローチ

本章では, 2 つ目のアプローチである**強化学習アプローチ**に基づいたシミュレーション結果をまとめ, 考察を行う.

5.1 数理モデルの構築

本アプローチでは, 次の数理モデルに従いシミュレーションを行う.

強化学習アプローチの数理モデル

n を正整数とし, モデルの対象空間を $n \times n$ の二次元セルとする. これは, $n \times n$ 体の意思決定個体を表す. また, この対象空間には 1 行目と n 行目, 1 列目と n 列目が接続しているという**周期的境界条件**が課されているとする.「時刻 t （$t \in \mathbb{Z}_{\geq 0}$）」をパラメータとして導入し, 時間発展とともに以下の流れを繰り返す.

各時刻 t において, 各個体は**二次元格子空間上での囚人のジレンマゲーム**を行う. ここで各個体の取る行動は, **強化学習の結果により得られた最適方策 π^* に従って, 各個体が時刻 t に選択した行動である**.

一般に強化学習は 1 体のエージェントに対して学習を行うが, 本研究では大量の意思決定個体が存在する**二次元格子空間上での繰り返し囚人のジレンマゲーム**を扱う. そこで本アプローチでは, **1 個体に対する強化学習の結果により得られた最適方策 π^* に全個体が従う**とした数理モデルを構築している.

5.2　なぜ強化学習か

強化学習の目的である「収益の最大化を実現する最適方策の発見」は,「相互協力による社会全体の利得の増大」という 囚人のジレンマゲームの解決法と非常に相性が良いと思われる. なぜなら,「社会全体の利得」を「収益」とみなして, 囚人のジレンマゲームに強化学習の観点からアプローチすることで, 相互協力関係が発生するモデルの実現を期待できるからである.

以下では, シミュレーションの結果として**協力個体率（定義は 4.3 節参照）が 0% に収束しないこと**を**ジレンマの解消**と表現する. そして, **ジレンマが解消する最適方策 π^* が得られるような強化学習モデルを構築することを考える**.

ここで,「各個体が行動『協力』を選択する確率を行動『裏切り』を選択する確率よりも大きくなるように設定してしまえば, わざわざ強化学習などしなくともジレンマが解消するのでは」と思われるかもしれない. 実際, その予想は正しい. しかし, たとえそれが最適方策であったとしても, 本研究ではそれを**人為的にではなく強化学習の結果により導く**ことを重要視している. このアプローチの目的は, 無理にでもジレンマを解消することではなく, あくまで強化学習を通して得た最適方策 π^* を用いてシミュレーションを実行し, 強化学習モデルの妥当性を評価することである.

5.3　強化学習に求められること

前章の**進化論的アプローチ**でのシミュレーションでは, 時間発展とともに協力個体率が徐々に低下していく様子が見られた. 実際, いくつかの仮定のもとでは協力個体率が 0% に収束することが示された（4.4.1 節参照）.

前節の**強化学習アプローチの数理モデル**において, 毎時刻で各個体が**ランダムに行動を選択する**と仮定しよう. すると, 当然のことながら協力個体率は常に 50% 前後を保つ. これは, 各個体がデタラメな戦略を取っても約 50% の協力個体率が維持されるということである. 協力個体率 50% というのは, 数値上ではジレンマの解消に成功しているといえるが,「ランダムに行動を選択する」という戦略は結論としてはあまりにも無価値であり無益である. なぜなら, 現実社会の様々な意思決定個体は通常何かしらの戦略を持っており,「ランダムに行動を更新する」という戦略は戦略がないも同然だからである. よって, **強化学習には「学習の結果により『ランダムに行動を選択する』ではない最適方策 π^* が得られ, それにより協力個体率が 50% を上回る状態が維持された」という成果が求められる**.

5.4　強化学習モデルの構築

本アプローチでは, 次の**強化学習モデル**を用いて強化学習を行う.

強化学習モデル

右図 5.1 のように, 二次元格子空間上における 1 個体とそのムーア近傍の対戦個体（を表す 3×3 のセル）に着目する. 中心に強化学習をさせる個体（エージェント）が配置されている. 時刻の概念を 5.1 節の**強化学習アプローチ**の数理モデルと共通させ, この 3×3 のセルで**二次元格子空間上での繰り返し囚人のジレンマゲーム**を行う. ただし, **対戦個体同士でのゲームは行わず, エージェントと対戦個体でのゲームのみを行う**とする. 本アプローチでは個体 A と個体 B の囚人のジレンマゲームにおいて, 個体 A が「裏切り」を選択し, 個体 B が「協力」を選択した場合, **個体 A は個体 B を大きく裏切った**と表現することにする. 各**二次元格子空間上での囚人のジレンマゲーム**において, エージェントの行動選択は現在の方策に従ったものであるとする. また各対戦個体の行動選択については, **その個体がエージェントに大きく裏切られたことがあれば必ず「裏切り」を選択し, 大きく裏切られたことがなければ「協力」または「裏切り」を等確率で選択する**ものとする.

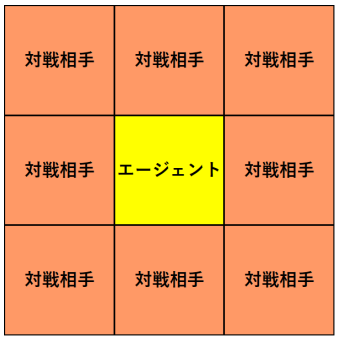


図 5.1: 1 個体のムーア近傍の様子

この強化学習モデルにおける時刻 t での状態 s_t, 行動 a_t, 報酬 r_t に関する流れを以下のように定める. なお s_t, a_t, r_t の具体的な値については, 次節を参照されたい.

1. エージェントの状態 s_t が与えられる.
2. 現在の方策 π に従い, この時刻に行われる**二次元格子空間上での囚人のジレンマゲーム**でのエージェントの行動が選択される. これが, 行動 a_t に対応する.
3. **二次元格子空間上での囚人のジレンマゲーム**が行われる. このゲームでの**エージェントと各対戦個体の獲得利得の合計値**を, エージェントの報酬 r_t とする.

5.1 節で述べたように, このモデルでは 1 個体をエージェントとして強化学習を行う. このモデルのように 3×3 のセルの範囲で強化学習を扱うというアイデアは,「**二次元格子空間上での囚人のジレンマゲーム**において 1 個体が影響を及ぼすのはムーア近傍の範囲である」という事実による.

また, 各対戦個体の行動選択のルールは 2.1.4 節にて導入した**評判**を反映したものになっている. エージェントがある対戦個体を大きく裏切れば, それ以降その対戦個体はエージェントに協力しなくなるので, これはエージェントの「裏切り」という行動に対しての罰則であると考えることができる. **この罰則の存在により, エージェントは行動「協力」を選択することがより大きな収益に繋がることを学習し, ジレンマが解消する最適方策 π^* が得られることが期待できる**.

5.5　強化学習モデルにおけるマルコフ決定過程と方策の定義

本節では, 行動を表す a と表 3.1 の利得のパラメータ a を区別するために, 利得のパラメータをそれぞれ $a_{\mathrm{b}}, b_{\mathrm{b}}, c_{\mathrm{b}}, d_{\mathrm{b}}$ と表記することにする（添字 b は, 利得を表す英語 benefit の頭文字である）.

まず, **強化学習モデル**における**マルコフ決定過程の 5 要素**を以下に定義する（2.2.2 節参照）.

1. 状態の集合 $\boldsymbol{S}$

   $s_0 = 0$ とする. $t \geq 1$ については, 時刻 $(t-1)$ での**二次元格子空間上での囚人のジレンマゲーム**までにエージェントが大きく裏切った対戦個体数を状態 s_t と定義する. すなわち,

   $$\boldsymbol{S} = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$$

   である.
2. 行動の集合 $\boldsymbol{A}_s$

   全ての状態 s_t で $\boldsymbol{A}_{s_t}$ は共通である. $a_t = 0, 1$ をそれぞれ (行動「裏切り」を選択), (行動「協力」を選択) に対応させ,

   $$\boldsymbol{A}_{s_t} = \{0, 1\}$$

   とする（強化学習での行動と戦略形における行動の区別に注意する）.
3. 報酬の確率分布 $p_r(r_t \mid s_t, a_t)$

   対戦個体 8 個体中 s_t 個体は必ず行動「裏切り」を選択し, $(8 - s_t)$ 個体はランダムに行動を選択する. ランダムに行動を選択する $(8 - s_t)$ 個体中 i 個体 $(0 \leq i \leq 8 - s_t)$ が行動「協力」を選択する確率は, 以下で与えられる.

   $$ {}_{8-s_t}C_i \left(\frac{1}{2}\right)^i \left(\frac{1}{2}\right)^{8-s_t-i} = {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{8-s_t} \tag{5.1}$$

   また, 対戦個体 8 個体中 s_t 個体が「裏切り」を選択し, 残りの $(8 - s_t)$ 個体中 i 個体が「協力」を選択した場合の報酬 r_t は, 以下で与えられる.

   $$r_t = \begin{cases} (c_{\mathrm{b}} + b_{\mathrm{b}})s_t + 2a_{\mathrm{b}}i + (c_{\mathrm{b}} + b_{\mathrm{b}})(8 - s_t - i) & (a_t = 1) \\ 2d_{\mathrm{b}}s_t + (b_{\mathrm{b}} + c_{\mathrm{b}})i + 2d_{\mathrm{b}}(8 - s_t - i) & (a_t = 0) \end{cases}$$

   $$= \begin{cases} 2a_{\mathrm{b}}i + (c_{\mathrm{b}} + b_{\mathrm{b}})(8 - i) & (a_t = 1) \\ (b_{\mathrm{b}} + c_{\mathrm{b}})i + 2d_{\mathrm{b}}(8 - i) & (a_t = 0) \end{cases}$$

   以上より, 状態 s_t で行動 a_t を選択した場合の報酬 r_t の確率分布 $p_r(r_t \mid s_t, a_t)$ は以下で与えられる.

   (i) $a_t = 1$ の場合

   $$\begin{cases} P(R_t = 2a_{\mathrm{b}}i + (c_{\mathrm{b}} + b_{\mathrm{b}})(8 - i)) = {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{8-s_t} & (0 \leq i \leq 8 - s_t) \\ P(R_t \neq 2a_{\mathrm{b}}i + (c_{\mathrm{b}} + b_{\mathrm{b}})(8 - i)) = 0 & (0 \leq i \leq 8 - s_t) \end{cases}$$

(ii) $a_t = 0$ の場合

$$\begin{cases} P(R_t = (b_{\mathrm{b}} + c_{\mathrm{b}})i + 2d_{\mathrm{b}}(8-i)) = {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{8-s_t} & (0 \leq i \leq 8 - s_t) \\ P(R_t \neq (b_{\mathrm{b}} + c_{\mathrm{b}})i + 2d_{\mathrm{b}}(8-i)) = 0 & (0 \leq i \leq 8 - s_t) \end{cases}$$

4. 状態の確率分布 $p_s(s_{t+1} \mid s_t, a_t)$

   $a_t = 1$ の場合, 対戦個体を大きく裏切ることはないので, 状態 s_t は不変である. 一方 $a_t = 0$ の場合, 対戦個体の中でランダムな行動選択をする $(8 - s_t)$ 個体中 i 個体 $(0 \leq i \leq 8 - s_t)$ を

   $$ {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{8-s_t-i} = {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{8-s_t} $$

   の確率で大きく裏切ることになる（これは, (5.1) と同様の確率分布である）. よって, 状態 s_t で行動 a_t を選択した場合の次の状態 s_{t+1} の確率分布 $p_s(s_{t+1} \mid s_t, a_t)$ は以下で与えられる.

   (i) $a_t = 1$ の場合

   $$\begin{cases} P(S_{t+1} = s_t) = 1 \\ P(S_{t+1} \neq s_t) = 0 \end{cases}$$

   (ii) $a_t = 0$ の場合

   $$\begin{cases} P(S_{t+1} = s_t + i) = {}_{8-s_t}C_i \left(\frac{1}{2}\right)^{8-s_t} & (0 \leq i \leq 8 - s_t) \\ P(S_{t+1} \neq s_t + i) = 0 & (0 \leq i \leq 8 - s_t) \end{cases}$$

5. 初期状態の確率分布 $p_s(s_0)$

   初期状態 s_0 の確率分布を以下で定義する.

   $$P(S_0 = 0) = 1$$

以上の定義により, **強化学習モデル**における行動価値関数の推定値 $\hat{Q}(s_t, a_t)$ の更新とは

$$\hat{Q}(s, a) = q_{s,a} \quad (s \in \{0,1,2,3,4,5,6,7,8\},\ a \in \{0,1\})$$

としたときに q_{s_t,a_t} の値を更新することを意味する.

次に, **強化学習モデル**における方策を以下に定義する（2.2.2 節参照）.

1. 強化学習の途中段階での方策 π（の更新）について

   これには, 2.2.4 節で導入した ϵ-greedy $\hat{Q}$ を採用する. この**強化学習モデル**では $|\boldsymbol{A}_{s_t}| = 2$ であるので, 確率 ϵ や確率 $1-\epsilon$ で $\boldsymbol{A}_{s_t}$ からランダムな行動を選択することは, 各行動 $a_t \in \boldsymbol{A}_{s_t}$ をそれぞれ確率 $\frac{\epsilon}{|\boldsymbol{A}_{s_t}|} = \frac{\epsilon}{2}$ や確率 $\frac{1-\epsilon}{|\boldsymbol{A}_{s_t}|} = \frac{1-\epsilon}{2}$ で選択することを意味する. よって, 学習途中の方策 π では, 各状態 $s_t \in \boldsymbol{S}$ に対して, 行動 $a_t \in \boldsymbol{A}_{s_t}$ を選択する確率分布 $\pi_a(a_t \mid s_t)$ は以

下で与えられる.

$$\begin{cases} P(A_t=0)=(1-\epsilon)+\frac{\epsilon}{2}=1-\frac{\epsilon}{2},\ P(A_t=1)=\frac{\epsilon}{2} & (\hat{Q}(s_t,0)>\hat{Q}(s_t,1)) \\ P(A_t=0)=\frac{\epsilon}{2},\ P(A_t=1)=(1-\epsilon)+\frac{\epsilon}{2}=1-\frac{\epsilon}{2} & (\hat{Q}(s_t,0)<\hat{Q}(s_t,1)) \\ P(A_t=0)=\frac{1-\epsilon}{2}+\frac{\epsilon}{2}=\frac{1}{2},\ P(A_t=1)=\frac{1-\epsilon}{2}+\frac{\epsilon}{2}=\frac{1}{2} & (\hat{Q}(s_t,0)=\hat{Q}(s_t,1)) \end{cases}$$

2. 強化学習終了後, **強化学習アプローチの数理モデル**にて各個体が従う最適方策 π^* について本強化学習では, **Algorithm 1**（5.6.2 節参照）**の実行回数**を**学習回数**と定義し, また $\boldsymbol{i}$ **回の学習により得られた行動価値関数の推定値を元に導かれた最適方策**を $\boldsymbol{\pi^{*i}}$ と定義する. 2.2.4 節の「強化学習の流れ」では,「状態 s に対して行動価値関数の**推定値** $\hat{Q}(s,a)$ が最大になる行動 a（複数ある場合は, これらからランダムな行動）を選択するような方策を最適方策 π^* とする」と述べた. しかし, **強化学習 1 回目で精度の良い行動価値関数の推定値** $\boldsymbol{\hat{Q}(s,a)}$ **が得られるとは考えにくい**. そこで, 本論文では最適方策 π^{*i} を以下のように定める.

本論文における最適方策 $\boldsymbol{\pi^{*i}}$ の定義

まず, **強化学習モデル**を用いて**学習回数が $\boldsymbol{i}$ 回の強化学習を実行する Algorithm**（5.6.2 節参照）を 100 回実行する. これにより, 行動価値関数の（最終的な）推定値 $\hat{Q}^i(s,a)$ のデータが 100 件得られる. これら 100 件のデータをそれぞれ $\hat{Q}^i_1(s,a)$, $\hat{Q}^i_2(s,a)$, $\cdots$, $\hat{Q}^i_{100}(s,a)$ とする. このとき, 本論文における最適方策 π^{*i} では, 各状態 $s_t \in \boldsymbol{S}$ に対して, 行動 $a_t \in \boldsymbol{A}_{s_t}$ を選択する確率分布 ${\pi^{*i}}_a(a_t \mid s_t)$ は以下で与えられる.

$$P(A_t=0)=1-\frac{\xi}{100},\ P(A_t=1)=\frac{\xi}{100}$$

ここで, ξ は $\hat{Q}^i_j(s_t,0)<\hat{Q}^i_j(s_t,1)$ を満たす整数 j $(1\leq j\leq 100)$ の個数である.

この最適方策 π^{*i} は,「**2.2.4 節での最適方策の定義に従った場合に, 全データ 100 件の中でその行動が選択されるようなデータ件数の割合をその行動選択の確率とする**」という考えに基づいて定義している.

5.6 強化学習の実行

強化学習モデルにおける報酬 r_t の累積和を収益として, Q 学習を実行する.

5.6.1 方策 π 及び最適行動価値関数の推定値 $\hat{Q^{\pi^*}}$ の初期化

Q 学習を実行するにあたり, まずは方策 π を以下のような確率分布 $\pi_a(a_t \mid s_t)$ で初期化する. これを, **初期方策 π_0** と呼ぶことにする.

$$\begin{cases} P(A_t=0)=1-\frac{\epsilon}{2},\ P(A_t=1)=\frac{\epsilon}{2} & (\hat{Q^{\pi^*}}(s_t,0)>\hat{Q^{\pi^*}}(s_t,1)) \\ P(A_t=0)=\frac{\epsilon}{2},\ P(A_t=1)=1-\frac{\epsilon}{2} & (\hat{Q^{\pi^*}}(s_t,0)<\hat{Q^{\pi^*}}(s_t,1)) \\ P(A_t=0)=P(A_t=1)=\frac{1}{2} & (\hat{Q^{\pi^*}}(s_t,0)=\hat{Q^{\pi^*}}(s_t,1)) \end{cases}$$

次に, 最適行動価値関数の推定値 $\hat{Q^{\pi^*}}$ を任意の $s \in \boldsymbol{S}, a \in \boldsymbol{A}_s$ に対して $\hat{Q^{\pi^*}}(s,a)=0$ で初期化する.

ここで, この初期化を考慮すると初期方策 π_0 は

$$P(A_t=0)=P(A_t=1)=\frac{1}{2}$$

となる.

5.6.2 Q 学習のアルゴリズム

強化学習モデルに基づいて実行する Q 学習のアルゴリズムについて説明する. 本 Q 学習では, **二次元格子空間上での囚人のジレンマゲーム**を 10 回繰り返し行うことを, (1 回の) **二次元格子空間上での繰り返し囚人のジレンマゲーム**とする. この**二次元格子空間上での繰り返し囚人のジレンマゲーム**をさらに 10 回繰り返し行い, 終えたら現在の最適行動価値関数の推定値 $\hat{Q^{\pi^*}}(s,a)$ を更新し, 現在の方策 π を ϵ-greedy $\hat{Q^{\pi^*}}$ で更新する. この一連の流れを **Algorithm 1** とし, 1 回の学習と定義する.

この Q 学習のアルゴリズムを以下に示す.

—Q 学習のアルゴリズム—

各パラメータの値 $a, b, c, d, \alpha, \gamma, \epsilon$ が与えられたとき, **学習回数が i 回の強化学習を実行する Algorithm** を以下に示す.

学習回数が i 回の強化学習を実行する Algorithm

Step 1: 任意の $s \in \boldsymbol{S}, a \in \boldsymbol{A}_s$ に対して, $\hat{Q^{\pi^*}}(s,a)$ を $\hat{Q^{\pi^*}}(s,a)=0$ で初期化する. また, 方策 π を初期方策 π_0 で初期化する.

Step 2: **Algorithm 1** を i 回繰り返し行う.

Step 3: 現在の $\hat{Q^{\pi^*}}(s,a)$ を, 最適行動価値関数 $Q^{\pi^*}(s,a)$ の最終的な推定値 $\hat{Q^{\pi^*}}^{i}(s,a)$ として採用する.

各種 **Algorithm** を以下に示す. **Algorithm 1** は, **$\hat{Q^{\pi^*}}(s,a)$ や π が所与の下, 1 回の学習を行うアルゴリズム**である. **Algorithm 2** は, 加えて $\hat{Q^{\pi^*}}_{\mathrm{new}}(s,a)$ が所与の下, **二次元格子空間上での繰り返し囚人のジレンマゲームを実行するアルゴリズム**である. **Algorithm 3** は, 加えて時刻 t とエージェントの状態 s_t が所与の下, **二次元格子空間上での囚人のジレンマゲームを実行するアルゴリズム**である.

Algorithm 1: **1 回の学習**

Step 1: $\hat{Q^{\pi^*}}(s,a)$ のコピーである $\hat{Q^{\pi^*}}_{\mathrm{new}}(s,a)$ を作成する.
Step 2: **Algorithm 2** を 10 回繰り返し行う.
Step 3: 現在の $\hat{Q^{\pi^*}}(s,a)$ を $\hat{Q^{\pi^*}}_{\mathrm{new}}(s,a)$ で更新し, さらに現在の方策 π を ϵ-greedy $\hat{Q^{\pi^*}}$ で更新する.

Algorithm 2: **二次元格子空間上での繰り返し囚人のジレンマゲーム**

Step 1: 時刻 t を $t=0$ と初期化する. また, 初期状態の確率分布 $p_s(s_0)$ に従い, エージェントの初期状態 s_0 が確率的に定まる.
Step 2: **Algorithm 3** を 10 回繰り返し行う.

Algorithm 3: **二次元格子空間上での囚人のジレンマゲーム**

Step 1: 現在の方策 $\pi_a(a_t \mid s_t)$ に従い, エージェントの行動 a_t が確率的に定まる.
Step 2: 報酬の確率分布 $p_r(r_t \mid s_t, a_t)$ と状態の確率分布 $p_s(s_{t+1} \mid s_t, a_t)$ に従い, エージェントの報酬 r_t 及び次の時刻でのエージェントの状態 s_{t+1} が確率的に**同時に**定まる.
Step 3: $\hat{Q^{\pi^*}}_{\mathrm{new}}(s_t,a_t)$ を

$$\hat{Q^{\pi^*}}_{\mathrm{new}}(s_t,a_t) \longleftarrow \hat{Q^{\pi^*}}_{\mathrm{new}}(s_t,a_t) + \alpha\Big[r_t + \gamma \max_{a' \in \boldsymbol{A}_{s_{t+1}}} \hat{Q^{\pi^*}}_{\mathrm{new}}(s_{t+1},a') - \hat{Q^{\pi^*}}_{\mathrm{new}}(s_t,a_t)\Big]$$

により更新する.
Step 4: 時刻 t を $t+1$ で更新する.

上で示された, **学習回数が i 回の強化学習を実行する Algorithm** を 100 回実行する. そして, 得られた 100 件の最適行動価値関数の (最終的な) 推定値 $\hat{Q^{\pi^*}}^{i}_{1}(s,a)$, $\hat{Q^{\pi^*}}^{i}_{2}(s,a)$, $\cdots$, $\hat{Q^{\pi^*}}^{i}_{100}(s,a)$ に対する ${\pi^*}^{i}{}_{a}(a_t \mid s_t)$ (5.5 節参照) が最適方策 π^{*i} となる.

5.6.3 パラメータの設定

Q 学習を実行するにあたり, パラメータに設定した具体的な値を以下に示す.

$$(a, b, c, d, \alpha, \gamma, \epsilon) = (3, 4, 1, 2, 0.1, 0.9, 0.3)$$

5.6.4 ソースコード

Q 学習を実行するにあたり実装したソースコードを, ソースコード 5.1 に示す. Python の文法について, 筆者は書籍 [8] を用いて学習した.

ソースコード 5.1 について説明する. このソースコードは, **学習回数が 1 回の強化学習を実行する Algorithm** に対応するものである. したがって, **Algorithm 1** を 1 回終えたとき, 最適行動価値関数の最終的な推定値 $\hat{Q^{\pi^*}}^1(s, a)$ を出力するようにプログラムされている.

次に, 2 次元リスト benefit_list について説明する. benefit_list$[i][j]$ $(i,\ j \in \boldsymbol{A}_s)$ によって参照されるものは利得のパラメータ値であり, benefit_list$[0][0] = d$, benefit_list$[0][1] = b$, benefit_list$[1][0] = c$, benefit_list$[1][1] = a$ となっている.

最後に, **Algorithm 3** の **Step 2**（5.6.2 節参照）の処理について説明する. 5.5 節で定義した $p_r(r_t \mid s_t, a_t)$ や $p_s(s_{t+1} \mid s_t, a_t)$ は, **二次元格子空間上での囚人のジレンマゲーム**における全 8 対戦個体との囚人のジレンマゲームの結果を**一括して**考えた確率分布であった. それに対して, ソースコード 5.1 内での**二次元格子空間上での囚人のジレンマゲーム**の処理は, 各対戦個体と順番に囚人のジレンマゲームを行うような内容となっている. しかし, このように各対戦個体と順番に囚人のジレンマゲームを行うとしても, 全 8 対戦個体と囚人のジレンマゲームを行うことに変わりはないので, 結局, R_t や S_{t+1} は 5.5 節の確率分布 $p_r(r_t \mid s_t, a_t)$ や $p_s(s_{t+1} \mid s_t, a_t)$ に従う. よって, 本質的な違いはない.

ソースコード 5.1: reinforcement learning

```
import numpy as np
import random

gamma = 0.9 # 割引率
alpha = 0.1 # 学習率
epsilon = 0.3
# 協力は 1
# 裏切りは 0
Q = [[0, 0] for i in range(9)]
# 方策の役割を果たす関数
def pi(s):
    action_list = [0, 1]
    if Q[s][0] > Q[s][1]:
        prob_list = [(1 - epsilon) + epsilon / 2, epsilon / 2]
    elif Q[s][0] < Q[s][1]:
        prob_list = [epsilon / 2, (1 - epsilon) + epsilon / 2]
    else:
        prob_list = [1 / 2, 1 / 2]
    return np.random.choice(a=action_list, p=prob_list)


# 利得の設定
benefit_list = [[2, 4], [1, 3]]

# 学習回数 1
for i in range(1):
    # Algorithm 1
    Q_new = Q[:]
    for j in range(10):
        # Algorithm 2
        # 各対戦個体が大きく裏切られたかどうかのフラグ
        greatly_betrayed_flag = [0 for i in range(8)]
        s = 0
        for k in range(10):
            # Algorithm 3
            current_s = s
            current_a = pi(s)
            r = 0
            for f in range(8):
                if greatly_betrayed_flag[f] == 0:
                    opponent_a = np.random.randint(0, 2)
```

```
                    r += (
                        benefit_list[current_a][opponent_a]
                        + benefit_list[opponent_a][current_a]
                    )
                    if current_a == 0 and opponent_a == 1:
                        s += 1
                        greatly_betrayed_flag[f] = 1
                else:
                    opponent_a = 0
                    r += (
                        benefit_list[current_a][opponent_a]
                        + benefit_list[opponent_a][current_a]
                    )
            Q_new[current_s][current_a] += alpha * (
                r + gamma * max(Q_new[s]) - Q_new[current_s][current_a]
            )
    Q = Q_new[:]
print(Q)
```

5.6.5 Q 学習の実行結果から算出した最適方策 π^{*i}

前節のソースコードを用いて Q 学習を実行した結果から,「**本論文における最適方策 π^{*i} の定義**」(5.5 節参照) に基づいて 4 つの最適方策 π^{*1}, π^{*10}, π^{*20}, π^{*30} を算出した. それらを以下に示す. times_1, times_10, times_20, times_30 がそれぞれ, 最適方策 π^{*1}, π^{*10}, π^{*20}, π^{*30} に対応する. 要素 times_$n[s][a]$ は, 状態 $s \in \boldsymbol{S}$ に対して, 行動 $a \in \boldsymbol{A}_s$ を選択する確率を表す.

ソースコード 5.2: result of reinforcement learning

```
times_1=[[0.43999999999999995, 0.56], [0.87, 0.13], [0.74, 0.26],
    [0.5900000000000001, 0.41], [0.45999999999999996, 0.54], [0.49,
    0.51], [0.41000000000000003, 0.59], [0.49, 0.51], [0.48, 0.52]]
times_10=[[0.43000000000000005, 0.57], [0.75, 0.25], [0.63, 0.37],
    [0.5800000000000001, 0.42], [0.47, 0.53], [0.51, 0.49], [0.37,
    0.63], [0.45999999999999996, 0.54], [0.4, 0.6]]
times_20=[[0.39, 0.61], [0.81, 0.19], [0.73, 0.27], [0.62, 0.38], [0.47,
    0.53], [0.51, 0.49], [0.37, 0.63], [0.45999999999999996, 0.54],
    [0.35, 0.65]]
times_30=[[0.21999999999999997, 0.78], [0.86, 0.14], [0.78, 0.22], [0.64,
     0.36], [0.47, 0.53], [0.51, 0.49], [0.37, 0.63],
    [0.42000000000000004, 0.58], [0.25, 0.75]]
```

5.7　シミュレーションの実行

5.7.1　実行環境

実行環境は, 前章での進化論的アプローチのものと同様である.

5.7.2　パラメータの設定

強化学習アプローチの数理モデルをシミュレーションする際, モデルに現れるパラメータに設定した具体的な値を以下に示す.

$$(n, a, b, c, d) = (100, 3, 4, 1, 2)$$

ここで, $(a, b, c, d) = (3, 4, 1, 2)$ は後に示すソースコードに直接は現れない. しかし, **強化学習アプローチの数理モデル**で用いている最適方策 π^* が, **強化学習モデル**において $(a, b, c, d) = (3, 4, 1, 2)$ として強化学習を行った結果から算出したものであることを考慮して, これらもパラメータとして示した.

5.7.3　ソースコード

シミュレーションをするにあたり実装したソースコードを, ソースコード 5.3 に示す. Python の文法について, 筆者は書籍 [8] を用いて学習した.

ソースコード 5.3 について簡単に説明する. このソースコードは, 各個体が最適方策 π^*[1] に従うとしたシミュレーションを行うものである（ソースコード 5.3 の 27 行目にて「times_1」が記載されているため, 確率分布 times_1 に従って行動が選択される）. 基本は**進化論的アプローチ**での説明と同じであるが, 2 次元リスト state によって全個体それぞれの**状態（定義は 5.5 節のものと同様）**が管理されており, それと確率分布 times_1 に従って行動が選択されていることに注意する.

ソースコード 5.3: approach2 prisoner's dilemma simulation after learning

```
1  import numpy as np
2  import random
3  import matplotlib.pyplot as plt
4  from matplotlib.animation import FuncAnimation
5  from matplotlib.animation import PillowWriter
6
7
8  H = 100
9  W = 100
10 time = 500 # 更新回数の上限
```

```
# 協力は 1
# 裏切りは 0


# 学習回数 1
times_1=[[0.43999999999999995, 0.56], [0.87, 0.13], [0.74, 0.26],
    [0.5900000000000001, 0.41], [0.45999999999999996, 0.54], [0.49,
    0.51], [0.41000000000000003, 0.59], [0.49, 0.51], [0.48, 0.52]]
# 学習回数 10
times_10=[[0.43000000000000005, 0.57], [0.75, 0.25], [0.63, 0.37],
    [0.5800000000000001, 0.42], [0.47, 0.53], [0.51, 0.49], [0.37,
    0.63], [0.45999999999999996, 0.54], [0.4, 0.6]]
# 学習回数 20
times_20=[[0.39, 0.61], [0.81, 0.19], [0.73, 0.27], [0.62, 0.38], [0.47,
    0.53], [0.51, 0.49], [0.37, 0.63], [0.45999999999999996, 0.54],
    [0.35, 0.65]]
# 学習回数 30
times_30=[[0.21999999999999997, 0.78], [0.86, 0.14], [0.78, 0.22], [0.64,
     0.36], [0.47, 0.53], [0.51, 0.49], [0.37, 0.63],
    [0.42000000000000004, 0.58], [0.25, 0.75]]

# 最適方策の役割を果たす関数
def pi(s):
    action_list = [0, 1]
    prob_list = times_1[s]
    return np.random.choice(a=action_list, p=prob_list)


action = [[-1] * W for i in range(H)]
greatly_betrayed_flag = [[0] * (H * W) for i in range(H * W)]
state = [[0] * W for i in range(H)]


def update():
    global action
    global state
    for h in range(H):
        for w in range(W):
            action[h][w] = pi(state[h][w])
    dH = [-1, 0, 1]
    dW = [-1, 0, 1]
    for h in range(H):
```

```
45          for w in range(W):
46              for dh in dH:
47                  for dw in dW:
48                      if dh == dw == 0:
49                          continue
50                      if action[h][w] == 0 and action[(h + dh) % H][(w + dw) %
                             W] == 1:
51                          if (
52                              greatly_betrayed_flag[99 * ((h + dh) % H) + ((w +
                                     dw) % W)][
53                                  99 * h + w
54                              ]
55                              == 0
56                          ):
57                              greatly_betrayed_flag[99 * ((h + dh) % H) + ((w +
                                     dw) % W)][
58                                  99 * h + w
59                              ] = 1
60                              state[h][w] += 1
61      return
62
63
64  # アニメーションで可視化
65  fig, ax = plt.subplots()
66  im = ax.imshow(action, cmap="autumn", vmax=1, vmin=0)
67  text = ax.set_title("time={}".format(0))
68  ax.set_aspect("equal")
69
70  # 初期化関数
71  def init():
72      im.set_data(action)
73      text.set_text("time={}".format(0))
74      return (im,)
75
76
77  # 更新関数
78  def update_anim(dt):
79      global action
80      update()
81      im.set_data(action)
82      text.set_text("time={}".format(dt))
83      return (
```

```
84          im,
85          text,
86      )
87
88
89  ani = FuncAnimation(
90      fig,
91      update_anim, # 更新関数
92      init_func=init,
93      frames=np.arange(time),
94      interval=100, # 更新間隔 (ms)
95      repeat=True, # 繰り返し表示させる
96      blit=True,
97  )
98  # アニメーションの保存
99  ani.save("approach2_prisoner's_dilemma_simulation_1.gif", writer="pillow", dpi
        =200)
```

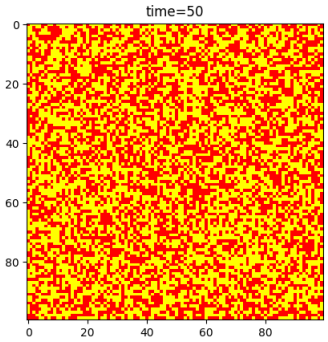
5.8 シミュレーション結果の可視化及び分析

強化学習アプローチの数理モデルのシミュレーション結果を, 次ページ以降に示す. なお, Matplotlib による可視化にあたり, 書籍 [9] を参考にした.

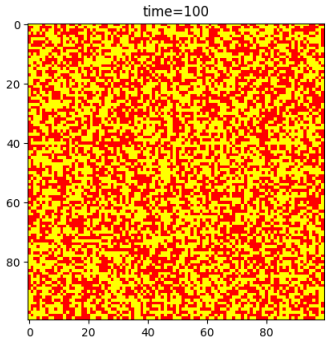
図 5.2, 図 5.3, 図 5.4, 図 5.5 はそれぞれ, 最適方策 π^{*1}, π^{*10}, π^{*20}, π^{*30} に各個体が従うとして**強化学習アプローチの数理モデル**をシミュレーションした場合の, 各個体の行動選択の時間発展を可視化したものである. **各時刻 t で行動「協力」を選択した個体は黄色で, 行動「裏切り」を選択した個体は赤色で示されている**. ここで**協力個体率**とは, その時刻 t において行動「協力」を選択した個体数が全個体数に占める割合を百分率で表したものである. 学習回数の増加に伴って, 協力個体率が上がることがわかる. そして, 時間発展を通してその協力個体率はほぼ一定に保たれるということもわかる.

図 5.6, 図 5.7, 図 5.8, 図 5.9 は, 各行動を選択した個体数の時間変化をグラフで示したものである. 行動「協力」を選択している個体数の時間推移は「cooperation count」で, 行動「裏切り」を選択している個体数の時間推移は「betrayal count」で表されている. 学習回数が増加するほど初めに若干の変動が見られるが, いずれの場合も時刻 $t = 25$ 程度で落ち着いている.

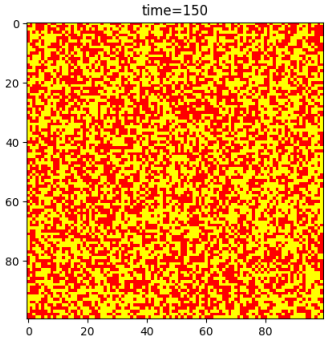
図 5.10, 図 5.11, 図 5.12, 図 5.13 は, **相互協力率**（定義は 4.3 節参照）の時間変化をグラフで示したものである. これも協力個体率と同様に, 学習回数の増加に伴って上昇している.



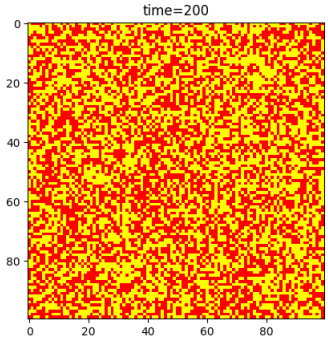
(a) $t = 50$ における行動選択の分布
協力個体率 51.8%



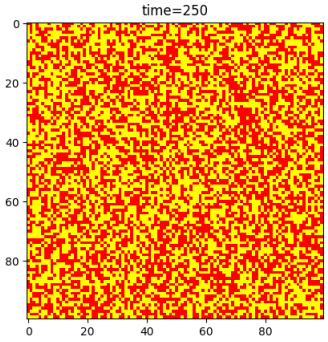
(b) $t = 100$ における行動選択の分布
協力個体率 51.73%



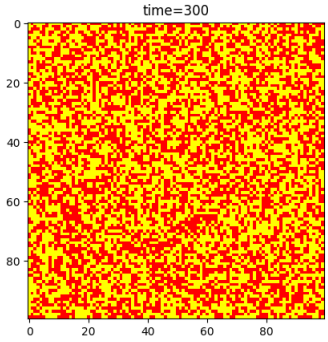
(c) $t = 150$ における行動選択の分布
協力個体率 51.38%



(d) $t = 200$ における行動選択の分布
協力個体率 50.89%

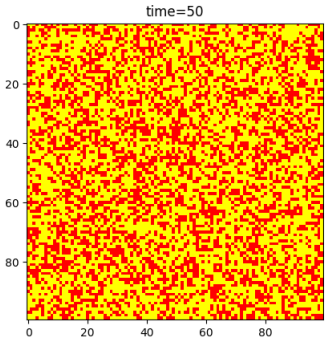


(e) $t = 250$ における行動選択の分布
協力個体率 50.94%

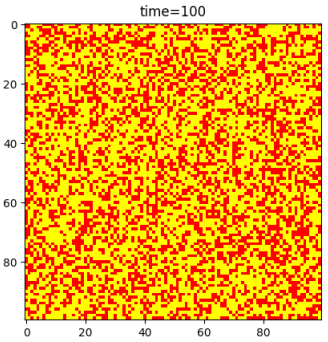


(f) $t = 300$ における行動選択の分布
協力個体率 51.27%

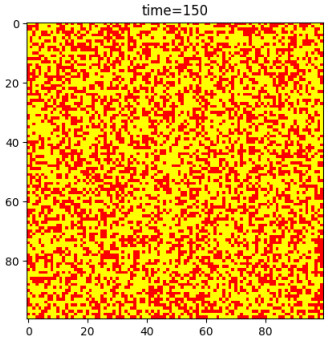
図 5.2: 学習回数が 1 回の場合のシミュレーション結果



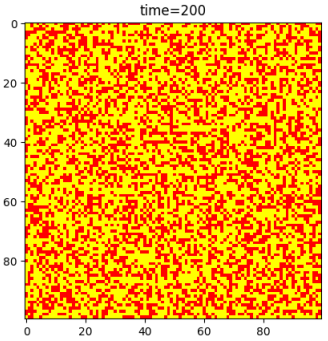
(a) $t = 50$ における行動選択の分布
協力個体率 59.39%



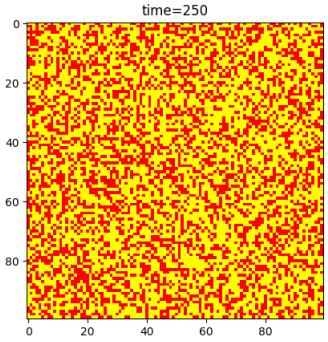
(b) $t = 100$ における行動選択の分布
協力個体率 60.08%



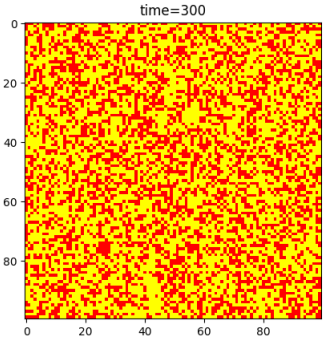
(c) $t = 150$ における行動選択の分布
協力個体率 59.79%



(d) $t = 200$ における行動選択の分布
協力個体率 59.62%

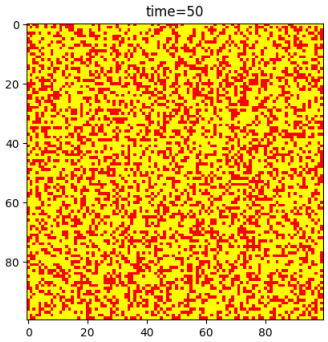


(e) $t = 250$ における行動選択の分布
協力個体率 59.94%

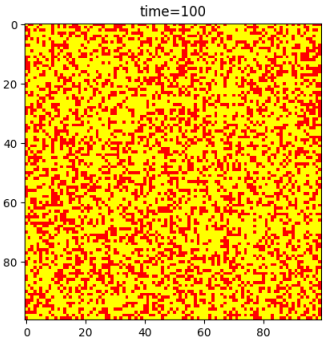


(f) $t = 300$ における行動選択の分布
協力個体率 60.15%

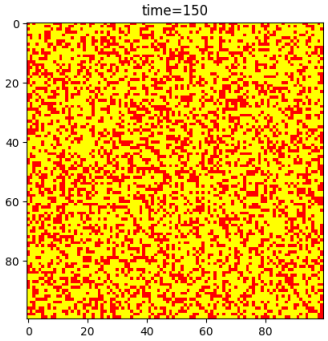
図 5.3: 学習回数が 10 回の場合のシミュレーション結果



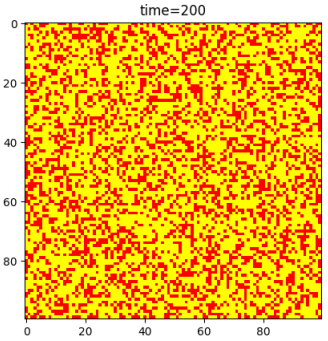
(a) $t = 50$ における行動選択の分布
協力個体率 64.54%



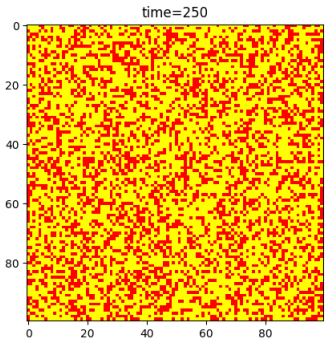
(b) $t = 100$ における行動選択の分布
協力個体率 65.06%



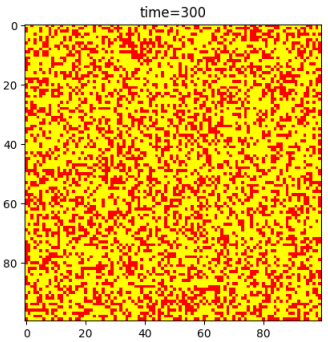
(c) $t = 150$ における行動選択の分布
協力個体率 64.93%



(d) $t = 200$ における行動選択の分布
協力個体率 65.35%

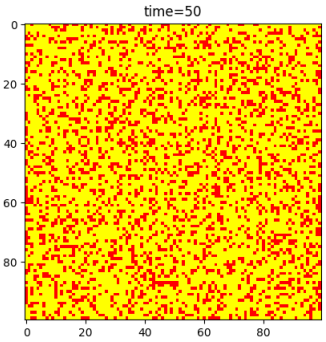


(e) $t = 250$ における行動選択の分布
協力個体率 64.17%

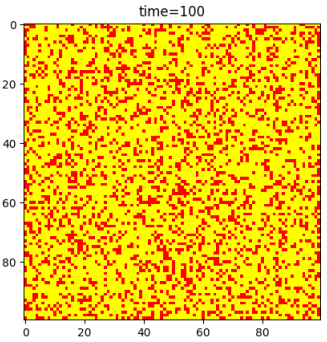


(f) $t = 300$ における行動選択の分布
協力個体率 64.22%

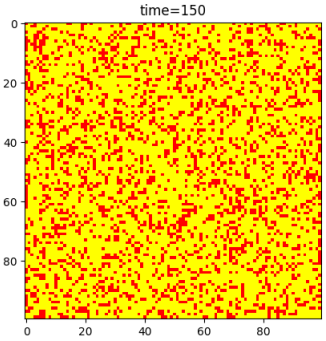
図 5.4: 学習回数が 20 回の場合のシミュレーション結果



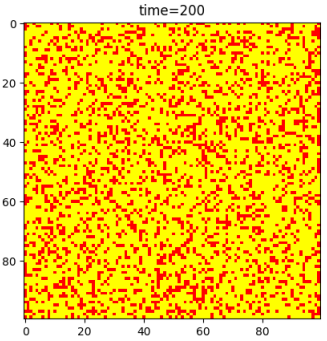
(a) $t = 50$ における行動選択の分布
協力個体率 73.67%



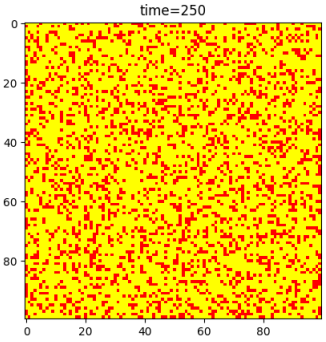
(b) $t = 100$ における行動選択の分布
協力個体率 73.77%



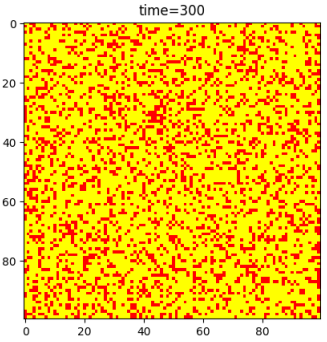
(c) $t = 150$ における行動選択の分布
協力個体率 75.08%



(d) $t = 200$ における行動選択の分布
協力個体率 74.36%



(e) $t = 250$ における行動選択の分布
協力個体率 75.29%



(f) $t = 300$ における行動選択の分布
協力個体率 74.21%

図 5.5: 学習回数が 30 回の場合のシミュレーション結果

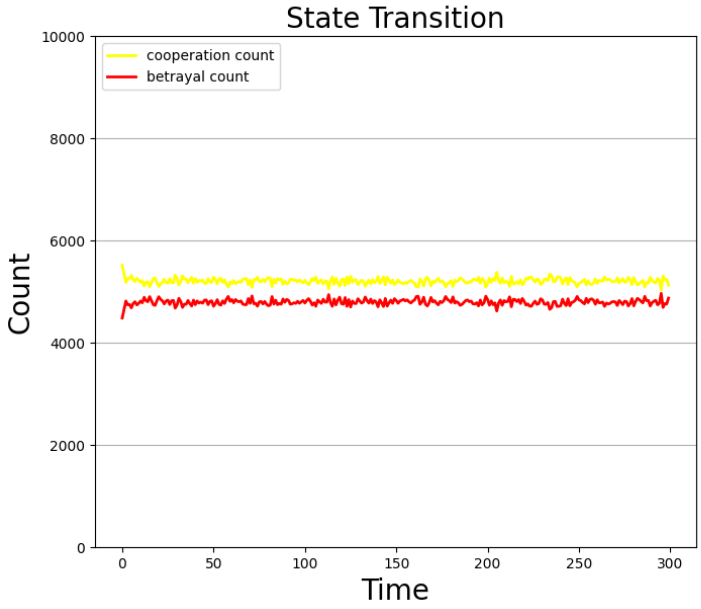


図 5.6: 学習回数が 1 回の場合の, 各行動を選択した個体数の時間変化

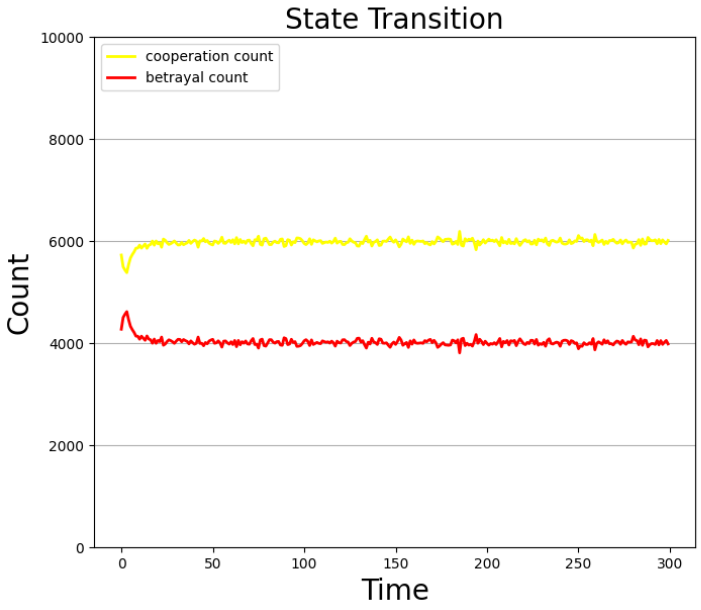


図 5.7: 学習回数が 10 回の場合の, 各行動を選択した個体数の時間変化

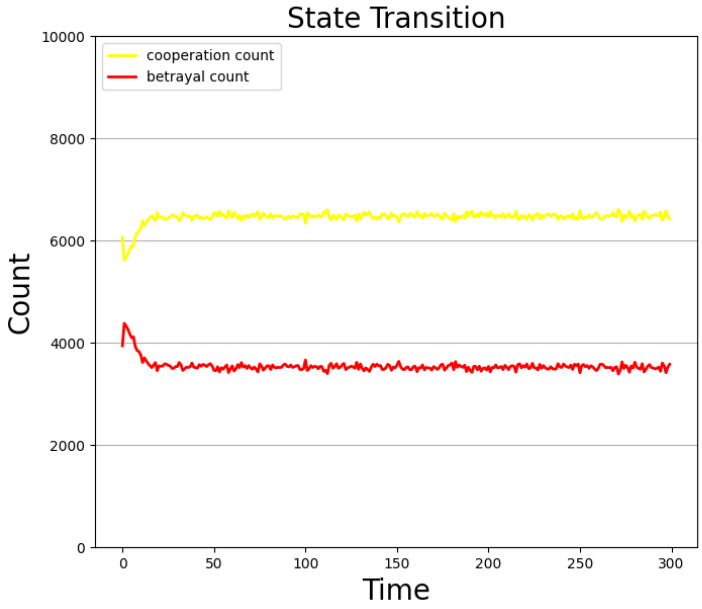


図 5.8: 学習回数が 20 回の場合の, 各行動を選択した個体数の時間変化

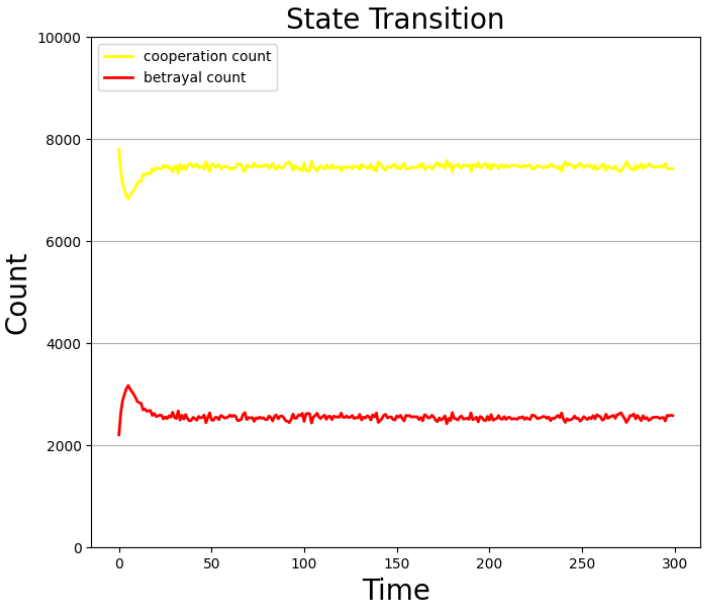


図 5.9: 学習回数が 30 回の場合の, 各行動を選択した個体数の時間変化

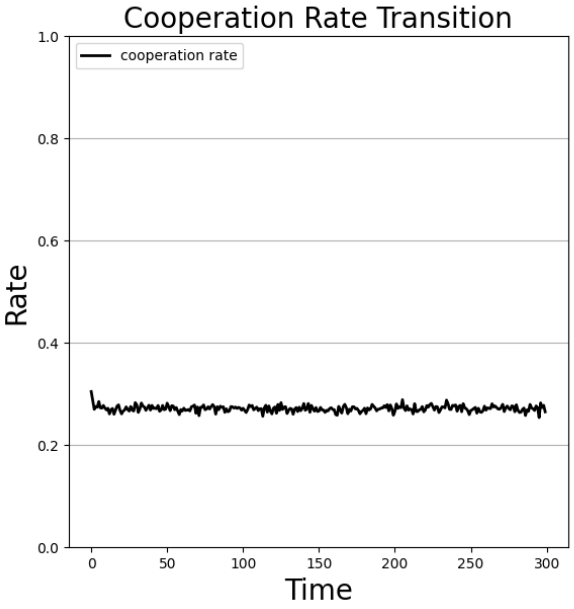


図 5.10: 学習回数が 1 回の場合の, 相互協力率の時間変化

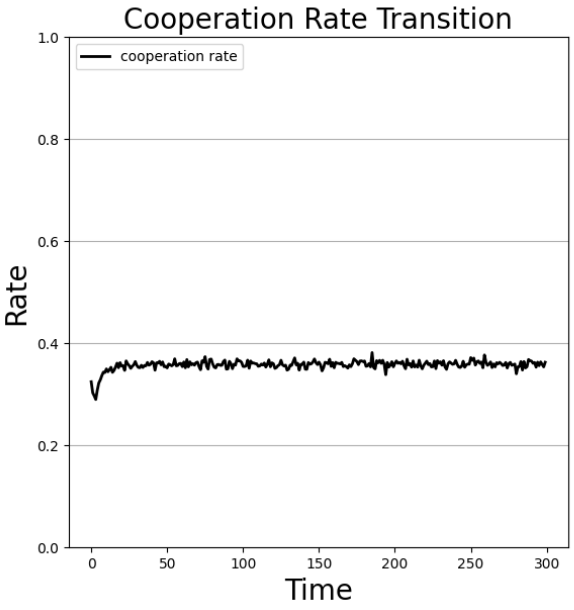


図 5.11: 学習回数が 10 回の場合の, 相互協力率の時間変化

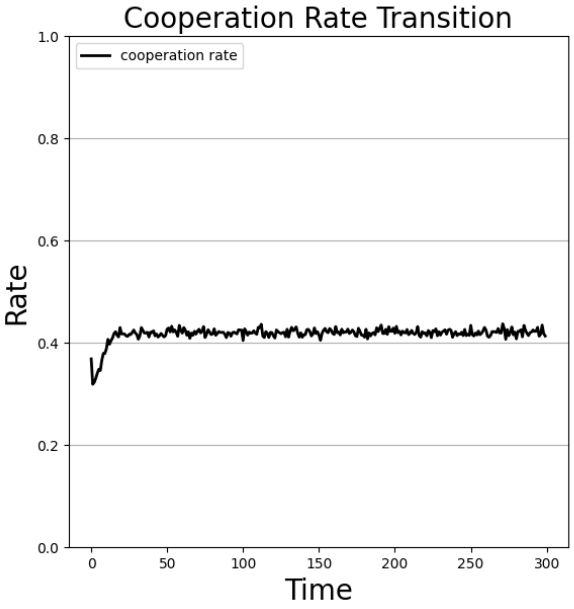


図 5.12: 学習回数が 20 回の場合の, 相互協力率の時間変化

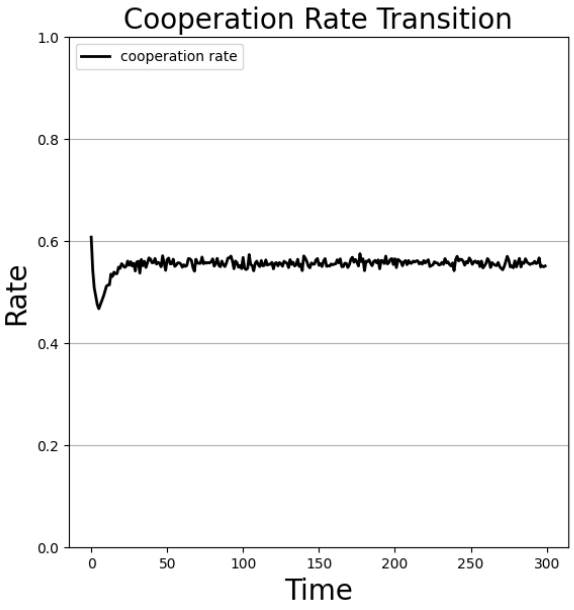


図 5.13: 学習回数が 30 回の場合の, 相互協力率の時間変化

5.9 考察

5.9.1 強化学習モデルの妥当性についての考察

ここでは, **強化学習モデル**の妥当性について考察する. 今回用いた**強化学習モデル**は, **評判**の概念を組み込んで高確率で行動「協力」を選択するようにエージェントに学習させることで, ジレンマが解消する最適方策 π^* を得ることを目的としていた. シミュレーションの結果（5.8 節参照）から,「時間発展を通して協力個体率が（正の値で）ほぼ一定に保たれる」ことが示されたので, 今回用いた**強化学習モデル**はジレンマが解消する最適方策 π^* が得られるようなモデルとして妥当であったと考えられる. さらに「**強化学習の学習回数が増えるほど協力個体率が上がる**」という結果から, 学習回数を増やすほどジレンマ解消の効果は高くなることもいえる. 実際, 学習回数が 1 回だと協力個体率は 50% 程度であり,「ランダムに行動を選択する」場合（5.3 節参照）と大きな差別化はできていない. 学習回数が 10 回を超えてくれば, 協力個体率は 50% を遥かに上回るようになり, 強化学習を用いる意義が生まれてくるといえる.

5.9.2 学習回数によるシミュレーション結果の違いについての考察

最後に, 学習回数によるシミュレーション結果の違いについて考察する. 以下の図 5.14 は, Q 学習の実行結果から算出した最適方策 π^*（ソースコード 5.2 参照）を元に, 各状態 $s_t \in \boldsymbol{S}$ に対して行動が $a_t = 1$（行動「協力」を選択）となる確率 $P(A_t = 1)$ を, 各学習回数について棒グラフで可視化したものである. $s_t = 0$ の場合, 学習回数に関わらず $P(A_t = 1)$ の値は全体的に大きいことがわかる. このことから以下が予想できる. $t = 0$ では全個体が $s_0 = 0$ であることを考慮すると, 多くの個体が $a_0 = 1$ となり少数の個体が $a_0 = 0$ となる. よって, $a_0 = 0$ となったこれら少数の個体は多数の近接個体を大きく裏切ることになり, $t = 1$ での s_1 の値は大きくなる. $s_t = 4, 5, 6$ 程度では $a_t = 0, a_t = 1$ となる確率はどちらも 5 割前後の確率であるので, これらの個体はこれから, ほぼランダムに a_t の値を取り続ける. 任意の s_t の値に対し, $P(A_t = 0) \neq 0 \ \wedge \ P(A_t = 1) \neq 0$ であることから, 以降は全個体が満遍なく s_t の値を伸ばし, 最終的にはほぼ全ての個体が $s_t = 8$ の状態となる.

以上の予想が正しいと仮定すれば,「**$t \to \infty$ での協力個体率は, 最適方策 π^* 中の $s_t = 8$ の場合に対する $P(A_t = 1)$ の値で近似できる**」と考えられる. 実際にシミュレーション結果を見ると, $t = 300$ での協力個体率は $51.27\%, 60.15\%, 64.22\%, 74.21\%$ であり, これらは Q 学習の実行結果から算出した最適方策 π^*（5.6.5 節参照）における値 $0.52(52\%), 0.6(60\%), 0.65(65\%), 0.75(75\%)$ と酷似している. したがって, 本論文の**強化学習アプローチの数理モデル**では,「**算出される最適方策 π^* 中の $s_t = 8$ の場合に対する $P(A_t = 1)$ の値を如何に伸ばすか**」が強化学習の成果を判断する大きな指標であったと考えられる.

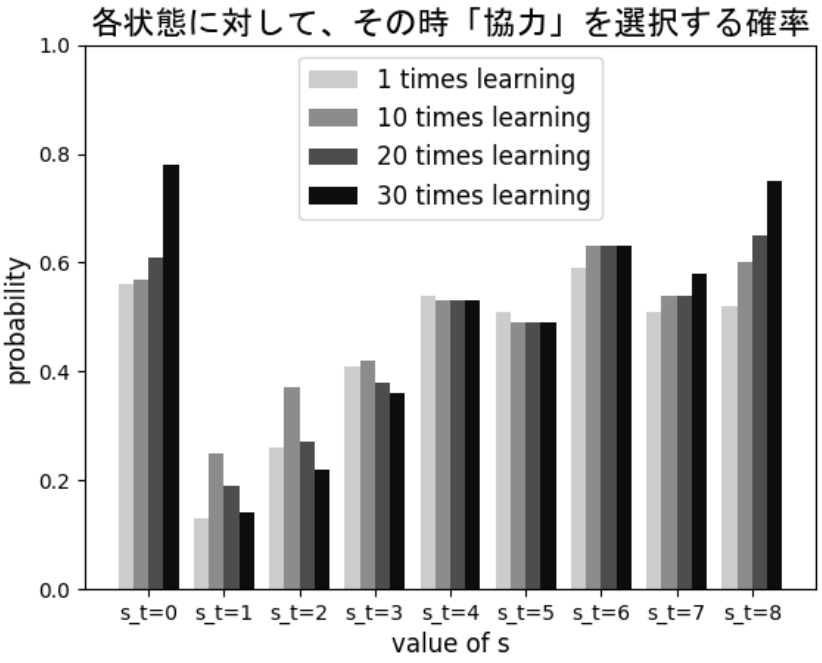

各状態に対して、その時「協力」を選択する確率
1 times learning
10 times learning
20 times learning
30 times learning
probability
1.0
0.8
0.6
0.4
0.2
0.0
s_t=0
s_t=1
s_t=2
s_t=3
s_t=4
s_t=5
s_t=6
s_t=7
s_t=8
value of s

図 5.14

第 6 章

結論

6.1 結論

本論文では, 繰り返し囚人のジレンマゲームをモデルとした新たな数理モデルを複数提案し, それらのモデルに基づいてシミュレーションを行い, 結果を考察した.

第 1 章では, 本論文の背景, 目的について述べた.

第 2 章では, 本シミュレーションを行う上で前提となる知識を基礎理論として紹介した.

第 3 章では, 本シミュレーションに際して登場する各種ゲームの定義を行った.

第 4 章では, 1 つ目のアプローチである**進化論的アプローチ**に基づいたシミュレーション結果をまとめ, 考察を行った.

第 5 章では, 2 つ目のアプローチである**強化学習アプローチ**に基づいたシミュレーション結果をまとめ, 考察を行った.

6.2 今後の課題

まず, **進化論的アプローチ**に関する今後の課題を述べる. **進化論的アプローチ**の考察では, 利得のパラメータから状態遷移の勢いを厳密に評価する定理を導出することができなかった. これを導出することが今後の課題である. また, 考察にて仮定した命題そのものの証明, もしくは, その仮定を多少弱めた場合についても, 今回と同様の考察結果を導くことに挑戦したいと考えている.

次に, **強化学習アプローチ**に関する今後の課題を述べる. **強化学習アプローチ**では 1 体のエージェントに対して強化学習を実行した. しかし, 後の調査により, 複数のエージェントに対して同時に強化学習を行うことができる**マルチエージェント強化学習**という手法が存在することを知った. これを用いれば, 今回のように 1 個体にのみ強化学習を行うのではなく, 全個体に対して同時に強化学習を行うことができると考える. このマルチエージェント強化学習を, 本論文の**強化学習アプローチの数理モデル**に適用し, 新たな発見を探ることが今後の課題である.

第 7 章

謝辞

本研究は, 著者の学部在学中に行われたものであります. 本研究を始めるきっかけを与えて下さり, これを進めるにあたって終始懇切丁寧な御議論, 御指導, 御検討を賜り, 著者の研究生活において, また研究を離れても常に有難い御激励を頂きました宮島信也教授に深く感謝の意を表します. また, 在学中講義などで御指導頂いた岩手大学理工学部物理・材料理工学科の先生方, 事務職員の方に深く感謝の意を表します. 最後に, 著者のこれまでの人生において幾度となく温かい手を差し伸べて下さった多くの方々にこの場を借りて厚く御礼申し上げます.

参考文献

[1] 神戸伸輔. 入門 ゲーム理論と情報の経済学. 日本評論社, (2004)

[2] 江崎貴裕. データ分析のための数理モデル入門 本質をとらえた分析のために. ソシム株式会社, (2020)

[3] 森山甲一. 囚人のジレンマゲームにおける Q 学習による協調の維持. コンピュータ ソフトウェア, 2008, 25.4: 4_145-4_153.

[4] 田口智健, et al. 強化学習エージェントの協調をもたらす N 人囚人のジレンマゲームの利得関数. 第 18 回情報科学技術フォーラム講演論文集, 2019, 2019.2: 317-318.

[5] 大原健; 並川直樹; 石渕久生. 繰り返し囚人のジレンマゲームにおける協調行動の進化と空間構造との関係. システム制御情報学会 研究発表講演会講演論文集, 2006, 0: 133.

[6] 藤原紫王里, et al. 二次元格子上繰り返し囚人のジレンマゲームにおけるフリーライダーの影響. 情報処理学会第 79 回全国大会講演論文集, 2017, 2017.1: 355-356.

[7] 杉山聡. 本質をとらえたデータ分析のための 分析モデル入門 ―統計モデル, 深層学習, 強化学習等 用途・特徴から原理まで一気貫通！―. ソシム株式会社, (2022)

[8] 北村祐稀. Python で問題解決 ―情報オリンピックに出てみよう―. 実教出版, (2022)

[9] Jake VanderPlas, 菊地彰（訳）. Python データサイエンスハンドブック ―Jupyter, NumPy, pandas, Matplotlib, scikit-learn を使ったデータ分析, 機械学習―. 講談社, (2018)