Thesis Title

Thesis Subtitle

James Campbell

B.Sc. Final Year Dissertation

Cardiff School of Mathematics


CARDIFF
UNIVERSITY
PRIFYSGOL
CAERDYDD

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Contents

List of Figures

Chapter 1

Introduction

The Prisoner's Dilemma (PD) is a classical model in Game Theory.d It is a simple two player game where the agents must make a choice of two options without communicating. This is often presented as two suspects who have been arrested and are being interrogated separately. They have the option of whether to cooperate with each other or defect and each player receives an individual payoff that depends on the actions that have been taken.

Consider the situation that both prisoners return to their cells each night, to be presented with the same choice the next day. Furthermore, allow this process to continue repeatedly. This is called the Iterated Prisoner's Dilemma (IPD) and has been an object of interest ever since the 1950's but became more popular after Robert Axelrod's work in the 1980's [5]. Every player in IPD will have a strategy. A strategy is a predetermined program of play that tells the player what actions to take in response to every possible strategy other players might use.

Several computer based IPD tournaments have been held over the years, but the first were held by Axelrod in 1980 (use our paper for refs). Others worth noting are the two anniversary tournaments held in 2004 and the Stewart and Plotkin 2012 tournament. The original source code of these is only available for Axelrod's second tournament (in FORTRAN) and it is not well documented, tested or easily re-usable.

Recently, a group of scientists have been working to improve this situation by producing an open source version of Axelrod's original tournament [29]. Referred to as the Axelrod-Python library, it aims to provide a resource for the design of new strategies and interactions between them, as well as conducting tournaments and ecological simulations for populations of strategies. The version of the library used in this work will be [14], the library currently has 139 strategies implemented with the capability to run evolutionary tournaments and tournaments on different topologies.

1.1 Prisoner's Dilemma

The first formal definition of PD was presented by Albert W. Tucker during a seminar at Stanford University [17]. However, the idea was first formulated by the Merrils [15] in 1950 whilst working for the RAND cooperation.

A description of the PD - Two players simultaneously decide whether to Cooperate (C) or Defect (D), without exchanging information. They receive payoffs as follows:

- They both choose C (mutual cooperation) and receive a payoff R (Reward)
- They both choose D (mutual defection) and receive a payoff P (Punish)
- One player chooses C and the other chooses D. The cooperator receives a payoff S (Sucker) and the defector receives a payoff T (Temptation).

Figure 1.1 shows the payoff matrix.

$$P = \begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{pmatrix} (R,R) & (T,S) \\ (S,T) & (P,P) \end{pmatrix} \end{array} \tag{1.1}$$

There are also two assumptions that need to be stated. Firstly, both players are rational. Secondly, there is no communication between them. It is then easy to see that regardless of the choice of one player, the other will always obtain a higher payoff by defecting instead of cooperating. Therefore we have a pure Nash Equilibrium where both players defect, despite the fact that both players would do better if they were to cooperate with each other. Additionally, to ensure this behaviour occurs, the payoffs must satisfy the following inequalities:

$$S < D < C < T \tag{1.2}$$

and

$$(S + T) < 2C \tag{1.3}$$

Equation 1.2 merely fixes the payoffs in their intuitive order. Equation 1.3 ensures that alternating between cooperating and defecting (players take it in turns to stab each other in the back) performs no better than mutual cooperation. These inequalities allow for many different payoff matrices to be formulated, but values of $(R, S, T, P) = (3, 0, 5, 1)$ are commonly used in literature (find references). We can now formulate the new payoff matrix, as shown in Figure 1.4 A more detailed explanation of the Prisoner's Dilemma is given

in [18].

$$P = \begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C & D \end{array} \\ \begin{pmatrix} (3,3) & (5,0) \\ (0,5) & (1,1) \end{pmatrix} \end{array} \tag{1.4}$$

1.2 Problem Description

As previously mentioned, the Axelrod-Python library contains 139 strategies, significantly more than the 13 and 64 strategies submitted to Axelrod's first and second tournaments. This now raises the issue of duplication, and subsequently, how to differentiate between strategies. Currently the only known approach to this problem in literature is Fingerprinting which produces a visual representation of a strategy, an example is shown in Figure.

The issue with this method is that it relies on knowledge of the underlying Markov chain of the strategy which cannot be easily constructed from the source code. In this paper an algorithm is presented that allows the construction of a fingerprint for any strategy. Example code is also given that shows how this has been implemented within the Axelrod library.

1.3 Structure

Throughout this paper examples and demonstrations of code will be given. Some of this code will be directly copied from the Axelrod-Python source code, at other times it will have been written specifically for this paper. In order to help the reader differentiate between each scenario, two different colour schemes have been used. The first, for Axelrod-Python source code is shown in listing 1 and the second, for any bespoke code, is shown in listing 2.

```
def function_name(parameters):
    """
    multiline comment
    """
    function_code = 4
    return some_output
```

Listing 1: An example of how Axelrod-Python source code will be displayed

This report is organized into several chapters. Continuing from this introduction:

- Chapter 2 An overview of previous literature is given
- Chapter 3 an example analytical fingerprint is constructed
- Chapter 5 example code, the algorithm and comparison of fingerprints

```
def function_name(parameters):
    """
    multiline comment
    """
    function_code = 4
    return some_output
```

Listing 2: An example of how demonstrative code will be displayed

- Chapter 6

Chapter 2

Literature Review

The Prisoner's Dilemma is a very popular model in game theory and there have been many papers written about the subject. The game has been applied to many different research areas is often used to model systems in biology [45], sociology [16], psychology [24], and economics [10]. The start of this chapter will give a brief overview of the literature and particularly relevant work will be highlighted. This is followed by an outline of how Axelrod's work is currently being reproduced by an open-source community. Finally, an introduction to fingerprinting and some necessary definitions and theorems are given at the end of the chapter.

2.1 Background

The political scientist Robert Axelrod held the first IPD tournament in 1980 [4]. Many well-known game theorists were invited to submit strategies that would compete against each other in a round robin style format. All strategies also competed against a random strategy (that would randomly choose between 'C' and 'D') and a copy of themselves. All strategies knew that the length of each game was 200 moves, and the whole tournament was repeated 5 times for reliability. Out of the 13 strategies that were entered, TitForTat was announced as the winner and was submitted by Professor Anatol Rapoport from the Department of Psychology of the University of Toronto.

TitForTat is a very simple strategy (see Section 2.2.1) and as explained in [1], it won because of three defining characteristics:

- 'Niceness' - A strategy is said to be nice if it is not the first to defect.
- 'Provocability' - Immediately after an opponent defects, the strategy should defect in retaliation.
- 'Forgiveness' - The strategy is willing to continue with mutual cooperation even after some defections.

Axelrod's second tournament [1] saw a dramatic increase in terms of size, with 62 strategies being entered from 6 different countries. The contestants ranged from a 10-year-old computer hobbyist to professors of

computer science, economics, psychology, mathematics, sociology, political science and evolutionary biology. The countries represented were the United States, Canada, Great Britain, Norway, Switzerland, and New Zealand. Despite the fact that all contestants had full knowledge of the previous tournament, TitForTat was the overall winner once again. One large difference in the mechanics of the first and second tournament was that the second tournament did not specify how many moves a game would last. Instead, the game ended probabilistically with a 0.00346 chance of finishing on any given move. This parameter was chosen so that the median length of a game would be 200 moves (in line with the first tournament).

Year	Reference	Number of Strategies	Type
1979	[4]	13	Standard
1979	[1]	64	Standard
1984	[7]	64	Evolutionary
1991	[9]	13	Noisy
2005	[11]	223	Varied
2012	[47]	13	Standard

Table 2.1: An overview of published tournaments

Axelrod continued to extend his work by considering an evolutionary version of the tournament [7, 6]. In this case, the proportion of the population playing a certain strategy depends on how the strategy performed on the previous round. A strategy is evolutionarily stable if a population of individuals using that strategy cannot be invaded by a rare mutant adopting a different strategy [7].

In [35, 36, 37], Nowak extends this further by studying Spatial Games. In his variation, the game is played on a 2 dimensional square lattice where the payoff for an individual is the sum over all interactions with its 8 nearest neighbours (see Figure 2.1) and itself. In the next generation, an individual cell is occupied with the strategy that received the highest payoff among all 8 nearest neighbours and itself.

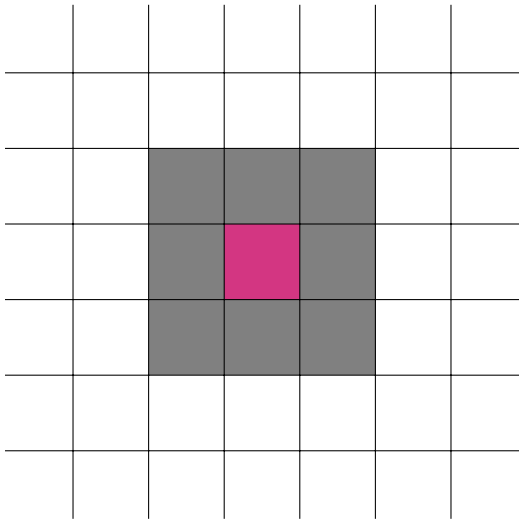


Figure 2.1: The topology of Nowak's spatial variation

To simplify things, Nowak limited each cell to be either Cooperator or Defector. Thus the game is com-

pletely deterministic and the outcome depends only on the initial configuration and the payoff matrix. One particularly interesting result is that if a single Defector invades a world of Cooperators, long (but finite) sequences of patterns emerge. Due to the symmetry of the game rules, all the patterns are highly symmetric and have the appearance of kaleidoscopes.

The primary purpose of [9] was to discover the effect of noise on TitForTat's performance in IPD. As had been previously claimed its performance was significantly reduced. The authors of [9] suggest that this is due to the provocability mentioned earlier, which in the presence of noise, can causes it to fall into unintended vendettas with other nice, but provocable strategies. Strategies that outperformed TitForTat were far more forgiving, which enabled them to get back to mutual cooperation much faster after an accidental defection.

2.2 Strategies of Particular Interest

Throughout this report many different strategies will be discussed and used in small examples. This section will provide a description of some of these strategies in order to aid the reader, as a strategy's name is not always particularly descriptive.

2.2.1 TitForTat

TitForTat is the most well known strategy for playing Prisoner's Dilemma due to it winning both of Axelrod's first two tournaments. It starts with a cooperative move and proceeds to play the same as the opponent did on the previous move [1, 19].

2.2.2 Pavlov

In [36], the strategy Pavlov is introduced (sometimes referred to as Win-Stay Lose-Shift). Pavlov plays by repeating its previous move if it was successful (received payoff T or R), and swapping the move if it was unsuccessful (received payoff P or S). The paper explains how this allows it take advantage of strategies that cooperate unconditionally and can also correct occasional mistakes.

2.2.3 Gradual

The strategy Gradual is present in [8]. It begins the game by cooperating, then after the first defection of the other player, it defects one time and cooperates twice. After the second defection of the opponent, it defects two times and cooperates twice. After the n^{th} defection it reacts with n consecutive defections and then two cooperations. In [8] it is claimed that Gradual outperforms TitForTat and this has been confirmed by work done through the Axelrod-Python library [29].

2.2.4 Random

Random plays exactly as expected, by choosing randomly between whether to cooperate or defect. The probabilities of choosing cooperate or defect are not necessarily even, instead they could rely on some distribution. We will denote the strategy that chooses to cooperate with probability p as Random(p). For example, the strategy that randomly chooses to cooperate 20% of the time and defect 80% of the time would be Random(0.2).

2.2.5 Cooperator/Defector

Cooperator and Defector are very simple. Cooperator will choose to cooperate at every turn, and similarly Defector will choose to defect every turn.

2.2.6 Cycler

The strategy Cycler(s) will repeat a given sequence of moves s. For example Cycler(CD) would play CD-CDCD... and Cycler(CDDDC) would play CDDDCCDDDCCDDDC...

2.2.7 Evolved Looker-Up

The winning strategy in the version of Axelrod-Python used throughout this paper is Evolved Looker-Up. The author, Martin Jones, has produced a thorough explanation of how the strategy was written in [25], and a brief outline will be given here. Quite simply, the strategy uses a lookup table to determine the action it should take. A lookup table is similar to a dictionary when programming, where the keys could be the opponents previous moves and the values the next action the strategy should take. The length of the key could be extended to include the opponents two previous moves (or more). It is clear that for a key length l, there are $n = 2^l$ keys. For each key the next action could be Cooperate or Defect, and so there are a total of 2^n possible lookup tables. Evolved Looker-up uses as its keys: the opponents first two moves, the opponents previous two moves, and the strategies own previous two moves. So we have $n = 2^6 = 64$ keys, and therefore $2^{64} \simeq 10^{18}$ possible lookup tables.

An evolutionary algorithm is then used to find the best possible table. The operates by initially creating a population of many different lookup tables. Then a new lookup table is created by combining two pre existing ones together (similar to how DNA is joined in reproduction). There is a probability (normally low) of some of the new lookup table mutating. Next, several more random lookup tables are produced. Finally, all lookup tables are scored and poor performers are discarded. Then the whole process is repeated with the new generation of lookup tables.

Evolved Looker-Up was produced after 200 generations, and the final lookup table can be found in the Axelrod-Python source code. Alternatively, the code used to produce the lookup table is given in [32].

2.3 Finite State Machines and Automaton

An Automaton is an abstract model of a machine that can perform operations on an input. A more general (and powerful) abstraction of an automaton is the famous Turing machine, which can perform any computational process carried out by present day computers [28]. In the case where an automaton has a finite set of set of states, it is referred to as a Finite State Machine. If this FSM has outputs it is called a Moore Machine [34]. A more detailed of Finite State Machines is given in Section 3.5.

Finite state machines have become important in game theory for several reasons. In the late fifties the idea of "bounded rationality" was explored in economics by Simon [46]. This was then extended in [41] where strategies that played the IPD were implemented as Moore Machines with an extra rule that added a cost associated with the complexity of the Moore Machine. Complexity was defined to be the number of states in the machine and the author states that this was a "fairly naive" approach.

In [26, 27], proofs are given that show that every strategy can be represented by an automaton. However, in Section 3.6 it is shown that if the length of a game in IPD is known, every strategy can be represented as a FSM.

2.4 Axelrod-Python Library

The Axelrod-Python library [14] is an open source Python package for creating reproducible research into the Prisoner's Dilemma. The original aim was to recreate Axelrod's tournaments as described in section 2.1 and verify their results. As the library has grown, so has the overall aim. The goal now, is "to provide a resource, with facilities for the design of new strategies and interactions between them, as well as conducting tournaments and ecological simulations for populations of strategies" [29].

For many of the tournaments that have been described the original source code is not available, and in the few cases where access was available there were no tests and minimal documentation. The library is partly motivated by a desire to improve this situation, and there is much discussion within academia currently regarding reproducible research [13, 20, 38, 42]. Some key characteristics are that the library is:

- Open: all code is released under an MIT license [40]
- Reproducible and well-tested: at the time of writing there is an excellent level of integrated tests with 99.73% coverage (including property based tests: [30])
- Well-documented: all features of the library are documented for ease of use and modification
- Extensive: 139 strategies are included, with infinitely many available in the case of parametrised strategies
- Extensible: easy to modify to include new strategies and to run new tournaments

Each of these items will be discussed in more detail in Section 4.5.

Figure 2.2: Ranked violin plot of the mean payoff for each player

Figure 2.3: Matrix plot of pair wise payoffs for each player

In listing 3 an example of how to produce a simple tournament is shown. Lines 7 - 10 create two plots. The first is a ranked violin plot of the mean payoff for each player and the second is a matrix plot of pair wise payoffs for each player. These plots can be seen in figures 2.2 and 2.3.

```
1  >>> import axelrod as axl
2  >>> axl.seed(0)  # Set a seed
3  >>> players = [s() for s in axl.strategies]  # Create players
4  >>> tournament = axl.Tournament(players)  # Create a tournament
5  >>> results = tournament.play()  # Play the tournament
6  >>> plot = axl.Plot(results)
7  >>> p = plot.boxplot()
8  >>> p.show()
9  >>> q = plot.payoff()
10 >>> q.show()
```

Listing 3: Example code to produce a simple tournament

2.5 Fingerprinting

It is easy to write a genetic algorithm to produce large numbers of strategies, however their analysis can be time consuming. Even the simple question 'Are these two strategies the same?' does not always have an obvious answer. This is especially true when considering source code, as two identical strategies can be coded in different ways. For example, differentiating between Random(0.5) and Cycler(CD) (see Section 2.2) isn't trivial unless you have access to their source code. The results of a game they play isn't necessarily enough.

A method for comparing strategies is first given in [3]. Ashlock outlines several definitions, theorems and proofs concerning the construction of a fingerprint. These are then followed by some examples, however they are of low quality and the only probe (see definition 4) used is TitForTat.

Ashlock then extends his fingerprinting work further in [2]. More examples are presented, and many different fingerprint functions are listed. Also, a large number of the analytical fingerprint functions use a probe that is not TitForTat. Fingerprinting is then used to to assess how three evolutionary algorithms produce different populations. The evolutionary methods involved are finite-state machines, lookup tables and feed forward neural nets.

Chapter 3

Theory

Many definitions will be now be presented, with the overall aim being to have a rigorous definition for a fingerprint. Definitions for the Dual, Joss-Ann and Fingerprint were first presented by Ashlock in [3] as mentioned in Section 2.5. Then a definition of Finite State Machines will be given, and an explanation of how they relate to fingerprinting. Finally, an example of how to construct a fingerprint will be shown in Section 3.7.

3.1 The Joss-Ann

The Joss-Ann is a basic transformation that can be applied to a strategy. It operates by making a probabilistic choice of cooperation, defection or the original move. More formally:

Definition 1 *If A is a strategy for playing the iterated prisoner's dilemma, then the* ***Joss-Anne of A****, $JA(A, x, y)$ is a transformation of that strategy. Instead of the original behaviour, it makes move 'C' with probability x, move 'D' with probability y, and otherwise uses the response appropriate to strategy A (if $x + y < 1$).*

The notation JA comes from the initials of the names Joss and Anne. Joss was a strategy submitted to one of Axelrods original tournaments and it would occasionally defect without provocation in the hopes of a slight improvement in score. Anne is the first name of A. Stanley who suggested the addition of random cooperation instead of random defection [2]. When $x + y = 1$, the original strategy is not used, and the resulting behaviour is a random strategy with probabilities (x, y). In more general terms, a JA strategy is an alteration of a strategy A that causes the strategy to be played with random noise inserted into the responses.

3.2 The Dual

The Dual is another, more complex transformation of a strategy (note that transformations can be applied over each other). Given a history for an opponent, the responses of the original strategy and the dual would be opposite.

Definition 2 *Strategy A' is said to be the **Dual** of strategy A if A and A' can be written as finite-state machines that are identical except that their responses are reversed.*

It is important to note that this is different to taking a strategy and flipping it's responses. The dual relies on knowledge of the underlying state of the original strategy, whereas the flip does not. This is shown in Table 3.1. For an outline of how Pavlov plays, see Section 2.2.2.

3.3 The Flip

The Flip is a simple transformation that returns the opposite of the strategy. This is subtly different from the Dual as demonstrated below.

Definition 3 *Strategy A' is said to be the **Flip** of strategy A if, A and A' return different actions for the same game history (includes opponent and self).*

Opponent	Pavlov	Dual	Flip
C	C	D	D
D	C	D	C
D	D	C	C
C	C	D	C
C	C	D	D
D	C	D	C
C	D	C	C
D	D	C	D
	C	D	D

Table 3.1: The different responses of Pavlov, Pavlov's Dual and Flipped Pavlov

The subtle difference between Dual and Flip can be highlighted further by inspecting each row individually.

Row 1 - Pavlov always plays 'C' on the first go. Flip will change this to 'D'. Dual knows that Pavlov always plays 'C' regardless of the opponents history and so swaps to 'D'.

Row 2 - In the previous round for Pavlov the strategies played (C, C), and so Pavlov plays 'C' again. For Flip, the preceding interaction was (D, C), in this instance Pavlov would play 'D' again, so this gets flipped to 'C'. The previous turn for Dual was (D, C) so it infers that Pavlov had (C, C). It knows that Pavlov would play 'C' and so plays 'D'.

Row 3 - In the previous round for Pavlov the strategies played (C, D), and so Pavlov would change to play 'D'. For Flip, the preceding interaction was (C, D), in this instance Pavlov would change to 'D', so this gets flipped to play 'C' again. The previous turn for Dual was (D, D) so it infers that Pavlov had (C, D). It knows that Pavlov would play 'D' in this instance and so plays 'C'.

By following this procedure we can see that the Dual responses and Pavlov's responses are always opposite. This is achieved by the Dual inferring what the original strategy must have played and then reacting appropriately. In the above example, Pavlov's memory is only one turn deep, so the Dual only has to infer the preceding turn, however, for more complex strategies, it may need to infer the entire history of the original strategy in order to proceed correctly.

3.4 Fingerprint and Double Fingerprint

The fingerprint function returns the expected score of a strategy when it plays against the Joss-Ann with varying (x, y). The double fingerprint extends this idea to $x + y \geq 1$.

Definition 4 *A **Fingerprint** $F_A(S, x, y)$ with $0 \leq x, y \leq 1$, $x + y \leq 1$ for strategy S and **probe** A, is the function that returns the expected score of strategy S against $JA(A, x, y)$ for each possible (x, y).*

Definition 5 *The **Double Fingerprint** $F_{AB}(S, x, y)$ with $0 \leq x, y \leq 1$ returns the expected score of strategy S against $JA(A, x, y)$ if $x + y \leq 1$, and $JA(B, 1 - y, 1 - x)$ if $x + y \geq 1$.*

We now show how the double fingerprint, with an appropriate choice of probe, is equivalent to the fingerprint function over the unit square.

Theorem 1 *If A and A' are dual strategies, then $F_{AA'}(S, x, y)$ is identical to the function $F_A(S, x, y)$ extended over the unit square.*

A proof for this theorem will now be given which was first presented in [3]:

Proof 1 *The Markov chain for the dual strategy A' will have the same transitions as the Markov chain for the strategy A. However, each entry for x corresponds to the probability that the strategy $JA(A, x, y)$ will randomly choose 'C' when it would not normally do so. For strategy A', this will occur whenever $JA(A', x, y)$ does not randomly respond 'D', which has probability $1 - y$.*

Similarly, each y corresponds to the probability that the strategy $JA(A, x, y)$ will randomly choose 'D' when it would usually respond 'C'. For strategy A', this will occur whenever $JA(A', x, y)$ does not randomly respond 'C', which has probability $1 - x$.

Thus the Markov chain for $JA(A', x, y)$ is the Markov chain for $JA(A, x, y)$ with the mapping $(x, y) \to (1 - y, 1 - x)$. Therefore $F_{AA'}(S, x, y)$ extends to the remainder of the unit square the function given by $F_A(S, x, y)$. ■

Theorem 1 allows the fingerprint function to naturally extend over the unit square. Figure 3.1 shows that in the bottom left region of the plot, the strategy plays against $\text{JA}(A, x, y)$ and in the top right region it plays against $\text{JA}(A', 1 - y, 1 - x)$.

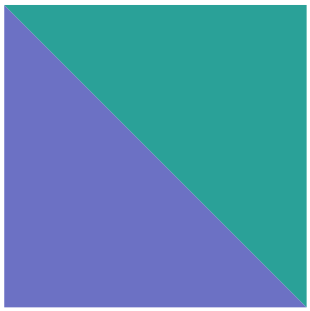


Figure 3.1: Where the Strategy plays against the Joss-Ann of the probe or the Joss-Ann of the dual of the probe

3.5 Finite State Machines

A formal definition of a Finite State Machine is given by Definition 6 but first we will outline some motivating key characteristics of a system that can be modelled with a FSM:

- The system must be describable by a finite set of states.
- The system must have a finite set of inputs that can trigger transitions between states.
- The behaviour of the system at a given point in time depends upon the current state and the input that occurs at that time.
- For each state the system may be in, behaviour is defined for each possible input.
- The system has a particular initial state.

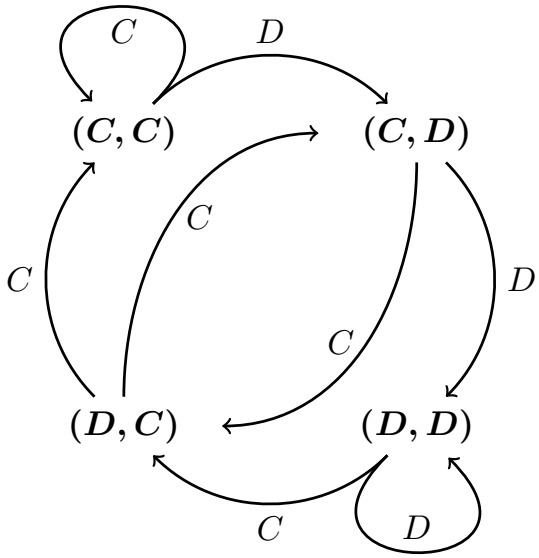
We can make the above bullet points rigorous for deterministic cases with the following definition:

Definition 6 *A **Deterministic Finite State Machine** M is a tuple $(S, \sigma, \delta, s_0, F)$ where*

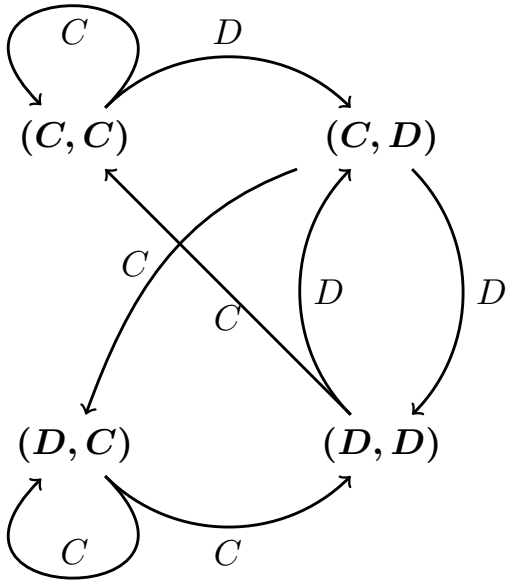
- *σ is the set of symbols representing the input of M.*
- *S is the set of states of M.*
- *$s_0 \in S$ is the starting state.*
- *$F \subseteq S$ is the set of final states of M.*
- *$\delta : S \times \sigma \to S$ is the transition function.*

Figure 3.2a and figure 3.2b show FSM representations for TitForTat and Pavlov respectively (see Sections 2.2.1 and 2.2.2 for an explanation of how these strategies operate). Here nodes represent the previous action taken by the strategy and the opponent, ie. node (D, C) implies that on the preceding turn, the strategy chose to Defect and the opponent chose to Co-operate. Arcs represent the choice made by the opponent at the current turn, and lead us to the state for the next turn.

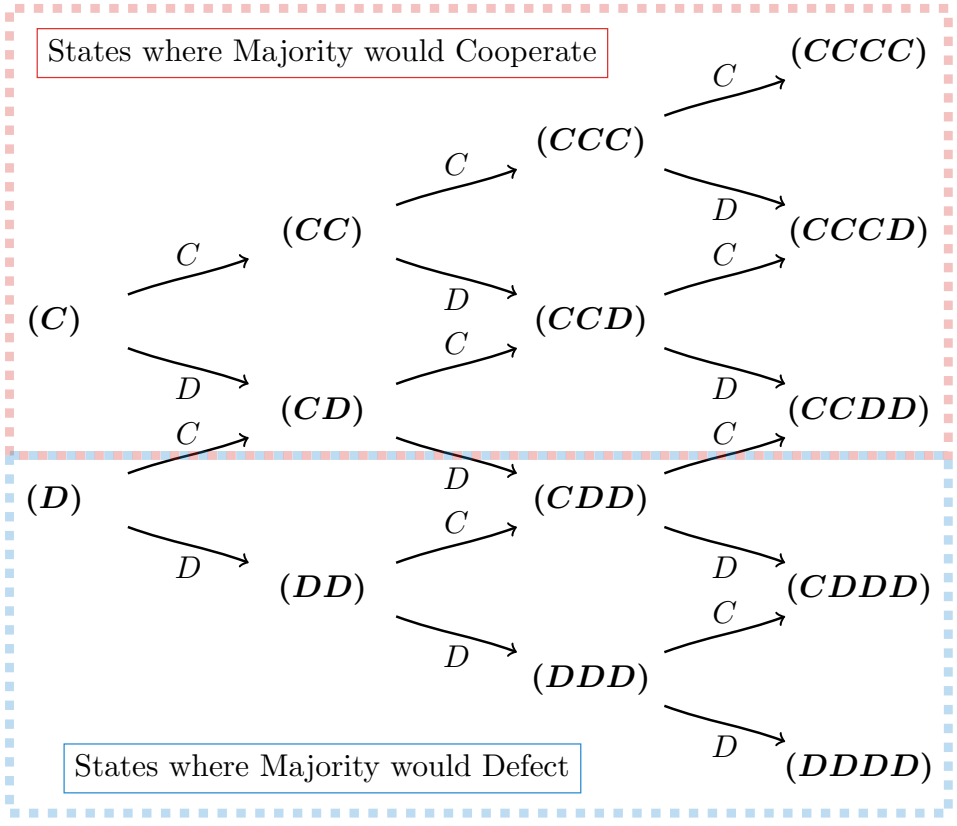
These are not necessarily the simplest FSM representation of the strategies. For example, TitForTat requires no knowledge of its own previous moves, but they have been included for completeness.



(a) FSM for TitForTat



(b) FSM for Pavlov (Win-Stay Lose-Shift)



(c) FSM for Majority in a game with 4 Turns

Figure 3.2: Finite State Machine representations for TitForTat, Pavlov and Majority

In figure 3.2c we have a more complex FSM for the strategy Majority for a game with 4 turns. Majority plays in the following way:

- If the opponent has cooperated the majority of the time, Majority will cooperate
- If the opponent has defected the majority of the time, Majority will defect
- Note - the strategy shown is technically Soft Majority, if the opponents cooperations and defections are equal it will cooperate. Hard Majority would defect in this situation.

Clearly this means that the strategy Majority requires knowledge of all previous states. In general this implies that Majority could not be represented as a FSM. However if the number of turns in a game is known, any strategy can be represented as a FSM.

3.6 Proof of FSM for every strategy

3.7 Analytical Fingerprints

There are several steps to constructing the Fingerprint of a strategy and basic knowledge of Markov Chains is required. An outline of the steps is as follows:

1. Build a Markov chain model of an IPD between the strategy and probe strategy.
2. Construct the corresponding transition matrix.
3. Find the steady state distribution.
4. Calculate the overall expected score by taking the dot product of the steady state distribution with the payoff vector given in Section 1.1 to obtain the fingerprint function.
5. This can then be plotted as a heat map to make it easier to visualize.

As an example, this process will now be applied to obtain a fingerprint for the strategy Win-Stay-Lose-Shift (sometimes referred to as Pavlov) when probed by Tit-For-Tat.

Step 1 - Build the Markov Chain, shown in Figure 3.3.

Step 2 - Construct the transition matrix.

$$T = \begin{array}{c} \\ (C,C) \\ (C,D) \\ (D,C) \\ (D,D) \end{array} \begin{array}{c} \begin{array}{cccc} (C,C) & (C,D) & (D,C) & (D,D) \end{array} \\ \begin{pmatrix} 1-y & 0 & 0 & x \\ y & 0 & 0 & 1-x \\ 0 & 1-y & x & 0 \\ 0 & y & 1-x & 0 \end{pmatrix} \end{array} \tag{3.1}$$

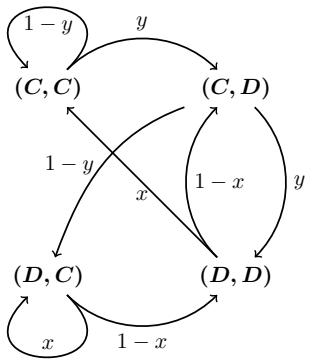


Figure 3.3: Markov chain for Pavlov

Step 3 - Find the steady state distribution.

$$\pi = \begin{bmatrix} \frac{x(1-x)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-y)}{2y(1-x)+x(1-x)+y(1-y)}, \\ \frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)} \end{bmatrix} \tag{3.2}$$

Step 4 - Calculate the expected score.

$$F = \pi \cdot \begin{bmatrix} 3 \\ 0 \\ 5 \\ 1 \end{bmatrix} = \frac{3x(1-x)+y(1-x)+5y(1-y)}{2y(1-x)+x(1-x)+y(1-y)} \tag{3.3}$$

Step 5 - Plot the resulting function, shown in Figure 3.4

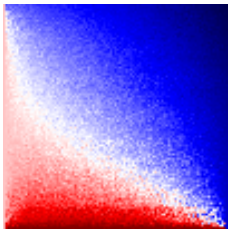


Figure 3.4: Fingerprint example for Pavlov

Theorem 2 *Given a deterministic strategy α and 2 histories h_1, h_2, then for all games of length $n \in \mathbb{N}$ there*

exists a FSM such that $\alpha(h_1, h_2)$ can be obtained from the FSM.

Proof 2 *Let $\sigma = \{C, D\}$ and*

$$S = \bigcup_{i=0}^{n+1} \{C, D\}^i \times \{C, D\}^i \delta((h_1, h_2), a) = ()$$

Chapter 4

Implementation

This chapter will explain how the method of fingerprinting was implemented within the Axelrod Python Library. Each of the definitions presented in Chapter 3 directly correspond to functions which are described below. All strategies have an equivalent class within the library and this is demonstrated in Section 4.3.

4.1 The Dual

The dual of a strategy is defined such that when the original strategy and the dual are presented with identical histories they will return opposite actions, as outlined in Definition 2. This means the dual relies on knowledge of how the original strategy would have behaved in a given situation, which is impractical to infer from the source code. However, the required behaviour can be achieved by having the original strategy as an attribute of the dual. Whenever the dual has to submit a move, it can first get the original strategy to suggest what move should it would have made, and then flip that action.

```
1 while Game is being played do
2 |   if First Turn then
3 |   |   create copy of original strategy;
4 |   end
5 |   simulate original strategy;
6 |   update original strategy's history/internal state;
7 end
8 return Flip of original strategy's move
```

Algorithm 1: The Dual of a Strategy

4.2 The Joss Ann

A formal definition of the Joss Ann is given in Section 3.1. Given a probability distribution (x, y) the Joss Ann Cooperates with probability x, Defects with probability y, and plays the original move with probability $1 - x - y$. This can be implemented very simply as seen in Algorithm 2. First, a random number between 0 and 1 is generated. The thresholds for Cooperation, Defection and original move are then $x, x + y$ and 1 respectively.

```
while Game is being played do
    p ← Random number;
    if p ≤ Cooperation Threshold then
        Next Move ← C;
    else if p ≤ Defection Threshold then
        Next Move ← D;
    else
        Next Move ← Original choice of Strategy;
    end
end
return Next Move
```

Algorithm 2: The Joss Ann of a Strategy

4.3 Implementation of Fingerprinting

As defined in section a fingerprint function is merely the expected score of a strategy when played against a Joss-Ann transformer of a probe with varying parameters (see definition 4). As part of this project, a numerical implementation has now been included in the Axelrod-Python library. It begins by taking a sample of the x, y values that may define a Joss-Ann Transformer. The strategy then plays a match against a transformer with each of the sampled values. The average score per turn can be calculated at the end of each match which corresponds to the expected score required by the analytical fingerprint function. The whole process can be repeated for reliability and the resulting scores plotted. The player interactions have been modelled as a spatial tournament within Axelrod-Python, where the strategy plays all of the probes and a probe only plays the strategy. For an example with 9 probes, see Figure 4.1.

Whether the numerical fingerprint matches the analytical one relies heavily on the choice of parameters. Specifically the `turns, repetetitons` and `step` variables. The `step` variable determines the number of x, y values taken. Listing 4 shows how a grid of points is constructed over the unit square where the distance between each point is taken as `step`. Therefore, a smaller `step` value means more points are created and so greater detail is included in the plot (similar to pixels).

The `turns` variable determines how many interactions there will be in a match. Enough turns must be selected to ensure that steady long term behaviour is reached otherwise the average score per turn can be wildly innacurate. However, once this state is reached, extending the number of turns has a minimal effect

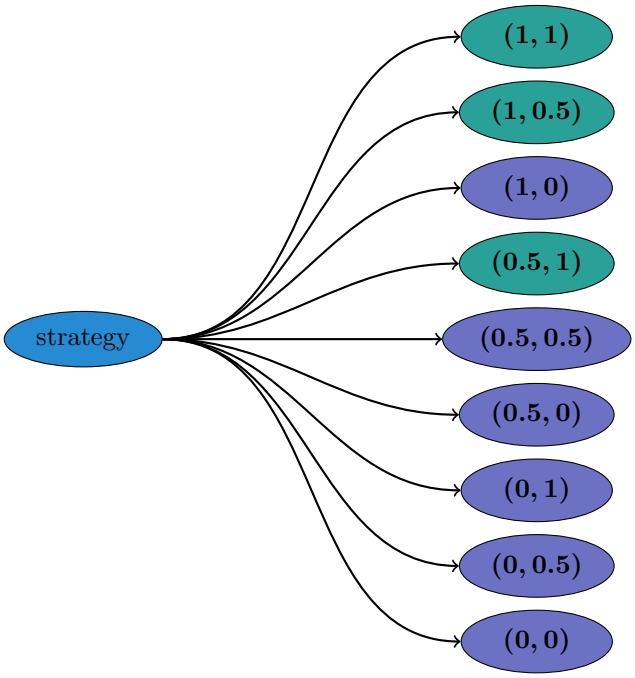


Figure 4.1: A spatial tournament for the strategy against 9 probes

```
def create_points(step):
    """Creates a set of Points over the unit square.
    A Point has coordinates (x, y). This function constructs points that are
    separated by a step equal to `step`. The points are over the unit
    square which implies that the number created will be (1/`step` + 1)^2.
    Parameters
    ----------
    step : float
        The separation between each Point. Smaller steps will produce more
        Points with coordinates that will be closer together.
    Returns
    ----------
    points : list
        of Point objects with coordinates (x, y)
    """
    num = int((1 / step) // 1) + 1
    points = [Point(j, k) for j in np.linspace(0, 1, num)
              for k in np.linspace(0, 1, num)]

    return points
```

Listing 4: Axelrod-Python code to create a sample of x, y points

on the accuracy of the plot. The `repetitions` variable decides how many times the tournament would be repeated. The Axelrod-Python implementation of fingerprinting is a random process (due to the Joss-Ann) and high repetitions helps to reduce the effects of this.

4.4 Comparison of Analytical and Numerical Plots

In figure 4.2, several analytical fingerprints from previous literature are shown [3, 2]. Colourings or shadings are used to make certain features stand out, and an attempt to replicate this behaviour was implemented in Axelrod-Python. The popular plotting library, matplotlib, has many options for different colour maps which are demonstrated in Appendix 7.1.

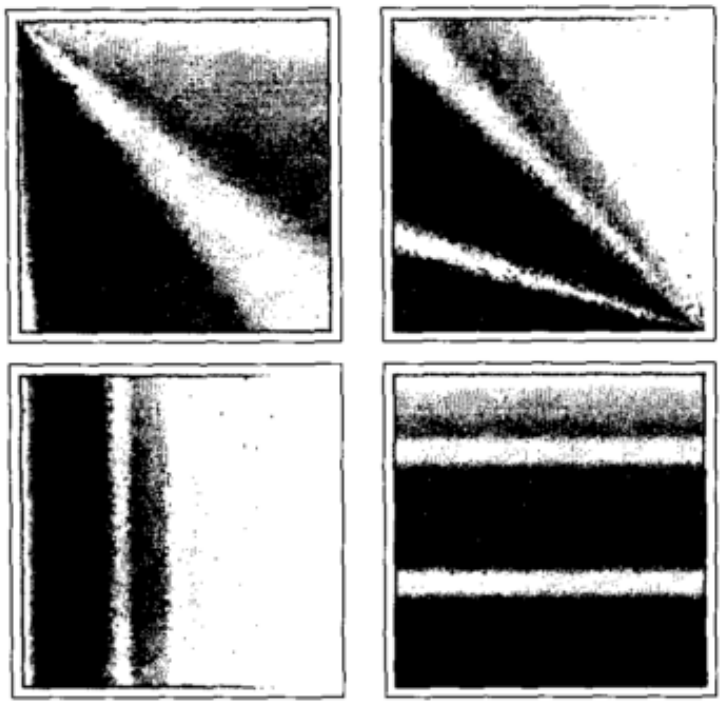
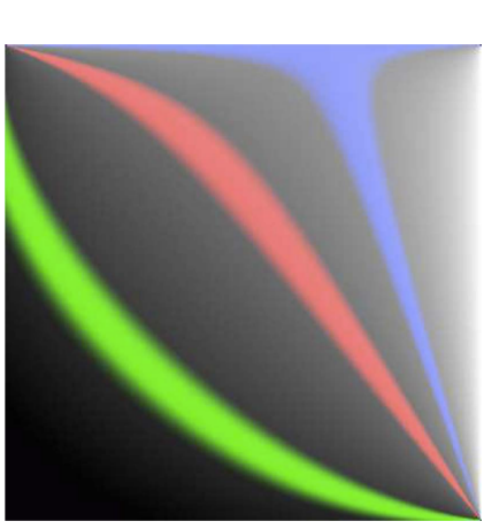


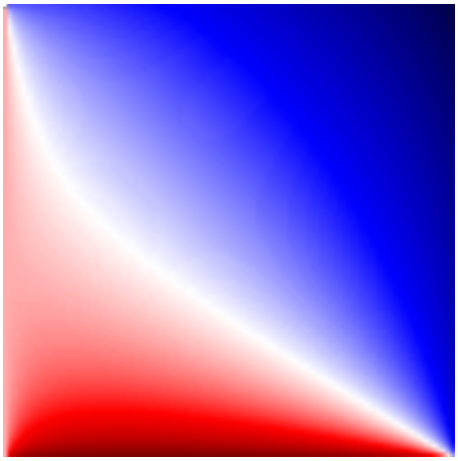
Figure 4.2: Shaded plots of the fingerprint functions for the strategies TitForTat, Psycho, AllD and AllC, in reading order from [3]

Using the analytical fingerprints from previous literature [3, 2], and the fingerprint formulae provided alongside them, the most appropriate colour map was chosen. The colour map Seismic [21] was selected due to its divergent properties (although all colour maps are available within the library). With divergent colour maps, all extreme values (high or low) are coloured, whilst mid range values are left white [33]. This highlights areas of interest, and in Figure 4.3 it can be seen that this matches previous work well.

With the knowledge that the choice of colourmap is appropriate, a comparison can now be made between analytical fingerprints and numerical ones obtained via the Axelrod-Python library. Table 4.1 gives the analytical fingerprint functions of several well known strategies that will then be used to validate the numerical



(a) WSLS fingerprint from previous literature [2]



(b) Analytical WSLS fingerprint demonstrating Seismic colouring

Figure 4.3: A comparison of a fingerprint plot from previous literature to asses the suitability of the Seismic colour map [2]

versions.

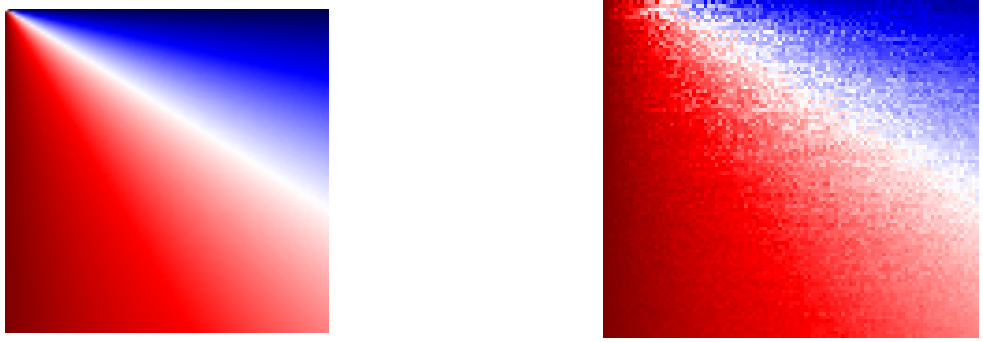
Strategy	Analytical Fingerprint Function
TitForTat	$\frac{y^2+5xy+3x^2}{(x+y)^2}$
Psycho (Anti TitForTat	$\frac{4(y-1)(x-1)+5(y-1)^2}{2(y-1)(x-1)+(x-1)^2+(y-1)^2}$
WinStayLoseShit (Pavlov)	$\frac{(3x+y)(x-1)+5y(y-1)}{(x+2y)(x-1)+y(y-1)}$
AllC (Cooperator)	$3-3y$
AllD (Defector)	$4x+1$

Table 4.1: A selection of analytical fingerprint functions for well known strategies. The probe used is TitForTat.

Figures 4.4 4.5 4.6 4.7 4.8 compare plots of known analytical fingerprint functions with numerical approximations obtained with the Axelrod-Python library. The analytical plots were created with the code seen in listing 5. The parameters `turns=500, repetitions=200, step=0.01` are as described in section 4.3. The parameter `processes=0` ensures that the function will use the maximum number of cores available on the computer.

```
import axelrod as axl
strats = [axl.TitForTat, axl.WinStayLoseShift, axl.AntiTitForTat,
          axl.Cooperator, axl.Defector]
for s in strats:
    probe = axl.TitForTat
    af = axl.AshlockFingerprint(s, probe)
    data = af.fingerprint(turns=500, repetitions=200, step=0.01, processes=0)
    p = af.plot()
    p.savefig('{}-Numerical.pdf'.format(s.name))
```

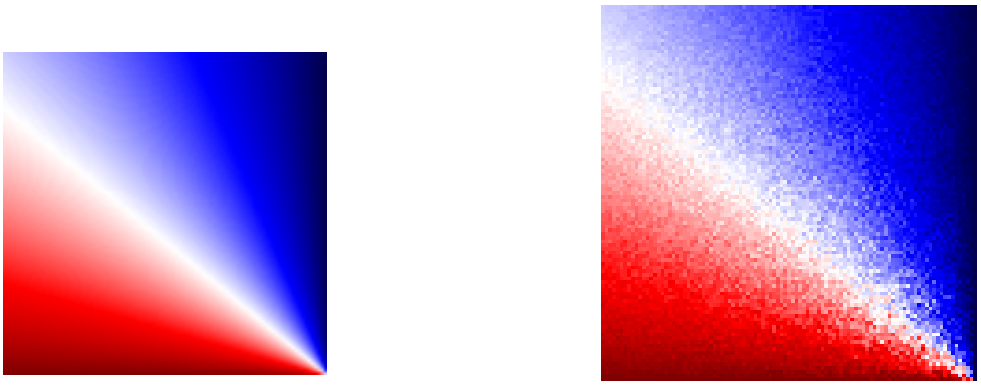
Listing 5: Code to create the numerical plots for several strategies



(a) Exact analytical fingerprint

(b) Numerical Fingerprint

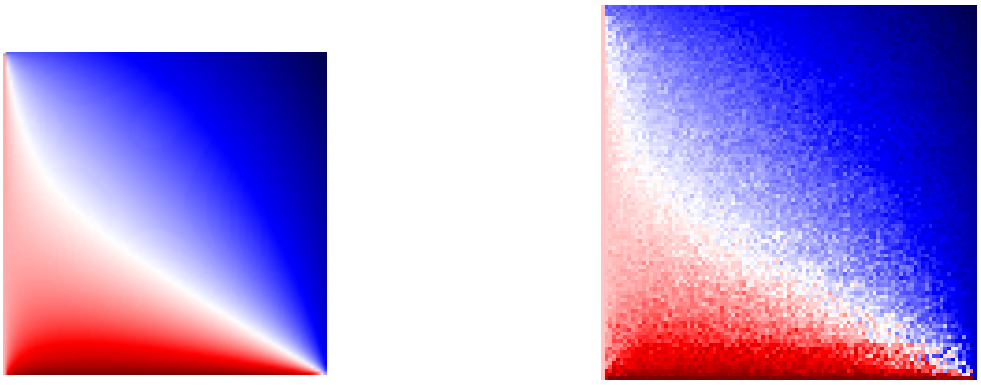
Figure 4.4: A comparison of the analytical fingerprint of TitForTat and the numerical version produced by Axelrod-Python library.



(a) Exact analytical fingerprint

(b) Numerical Fingerprint

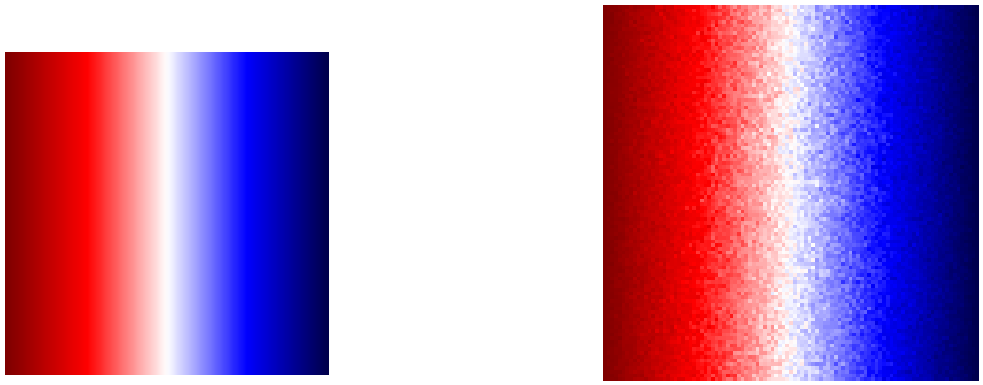
Figure 4.5: A comparison of the analytical fingerprint of Psycho and the numerical version produced by Axelrod-Python library.



(a) Exact analytical fingerprint

(b) Numerical Fingerprint

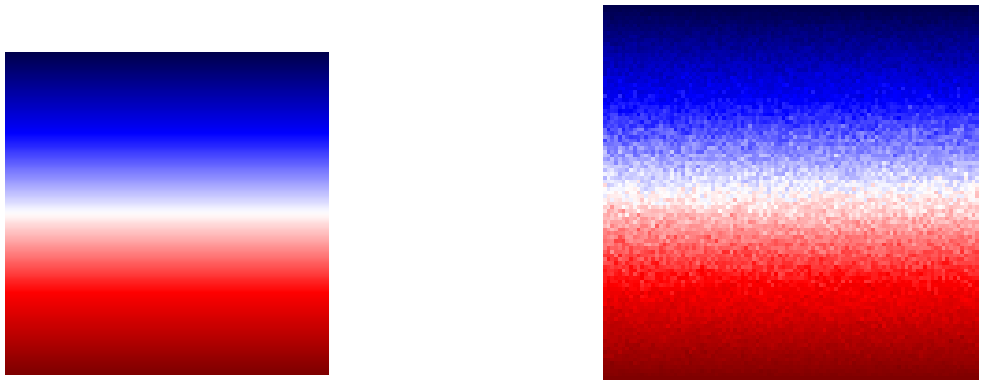
Figure 4.6: A comparison of the analytical fingerprint of WinStayLoseShit and the numerical version produced by Axelrod-Python library.



(a) Exact analytical fingerprint

(b) Numerical Fingerprint

Figure 4.7: A comparison of the analytical fingerprint of Cooperator and the numerical version produced by Axelrod-Python library.



(a) Exact analytical fingerprint

(b) Numerical Fingerprint

Figure 4.8: A comparison of the analytical fingerprint of Defector and the numerical version produced by Axelrod-Python library.

4.5 The Development Process

The Axelrod-Python library aims to follow best practice at all times with regards to development. This section outlines some of these key ideas, and how they were relevant to the implementation of the ability to produce fingerprints within Axelrod-Python.

4.5.1 Version Control

Scientists face many issues when producing research in the modern world. One of the main reasons research can be hard to reproduce is due to the lack of access to underlying data and code, which reduces the opportunity for others to verify findings [44, 23]. Also, two of the biggest challenges scientists and other programmers face when working with code and data are keeping track of changes (and being able to revert them if things go wrong), and collaborating on a program or dataset [51, 31]. Both of these issues can be resolved by ensuring that all code is version controlled.

Version control systems give users the ability to save versions of files during development along with informative comments which are referred to as commit messages. Every change and accompanying notes are stored independent of the files. Commits serve as checkpoints where individual files or an entire project can be safely reverted to when necessary [39]. Version control ensures that all changes to a code base are tracked and can be traced back to a particular author. There are many version control systems available and Axelrod-Python uses Git [48] with all code hosted at Github. One of the major benefits is that every repository (including the one on Github) contains the entire history of all changes, including authorship, and can be viewed by anyone.

4.5.2 Review Process

The library also applies a strict review process, where any code submission is analysed by several members of the organisation before it can be included. This normally involves other developers requesting changes to the submission, this ensure that all source code is of the same high standard. Figures 4.9 to 4.12 shows some screen shots of the discussions, requests and suggestions made during the development process of the fingerprint code.

Figures 4.9 and 4.10 show suggestions that have been made. These suggestions do not highlight errors within the code, but are merely small recommendations from experienced developers for ways to improve.

Figure 4.11 is an example of the power of using git. Code that is being developed elsewhere can still be used as part of this project, even though it is not yet available as part of the standard library. This is a common occurrence when developing software; some code may rely on other parts but they are not being written by the same person.

Figure 4.12 is a request to move unnecessary lines of code. Whilst these would not cause the software to break, it is bad coding practice to have unused imports and redundant lines of code. This is an example of

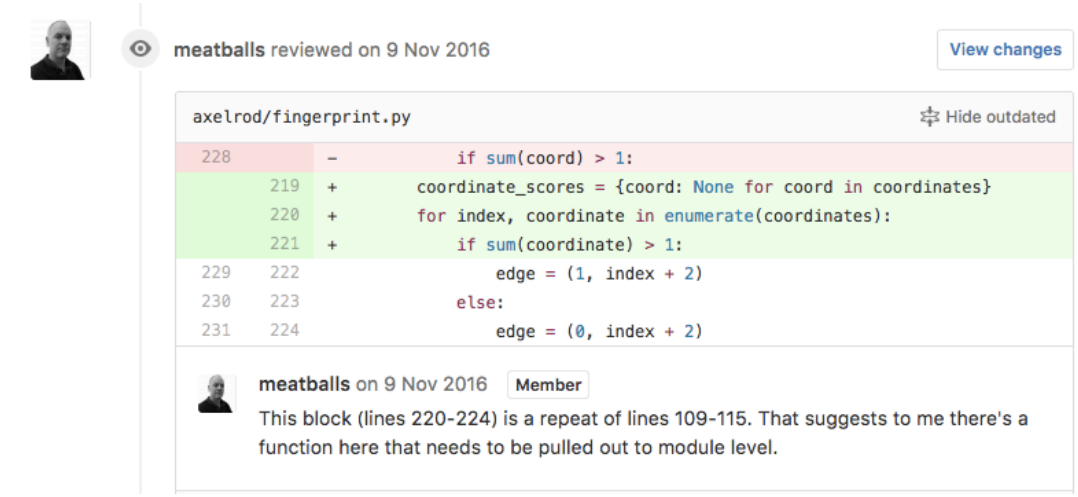


Figure 4.9: A refactoring suggestion to create a new function to avoid repetition

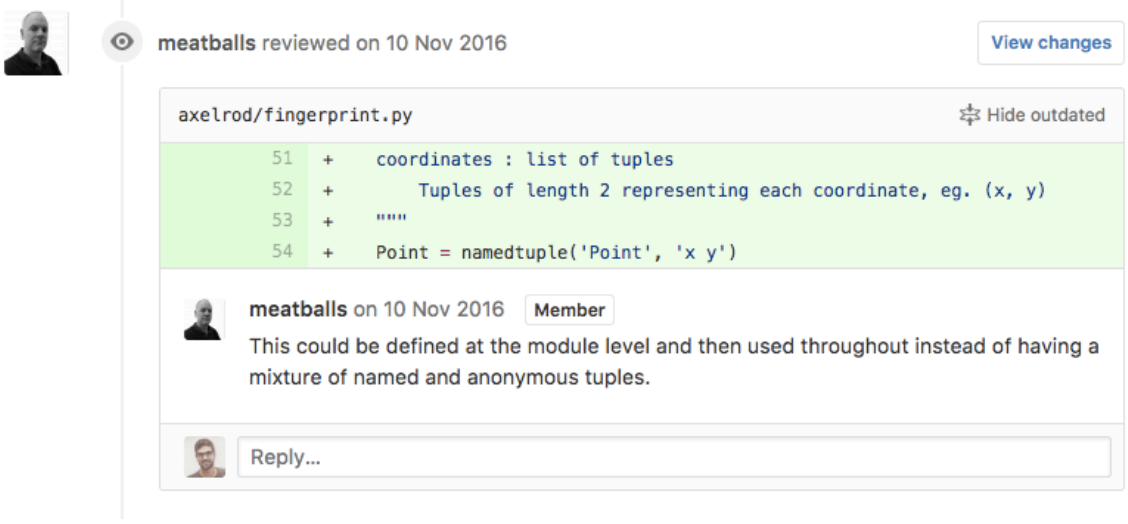


Figure 4.10: Suggestion to move some code to the module level so that it is more accessible

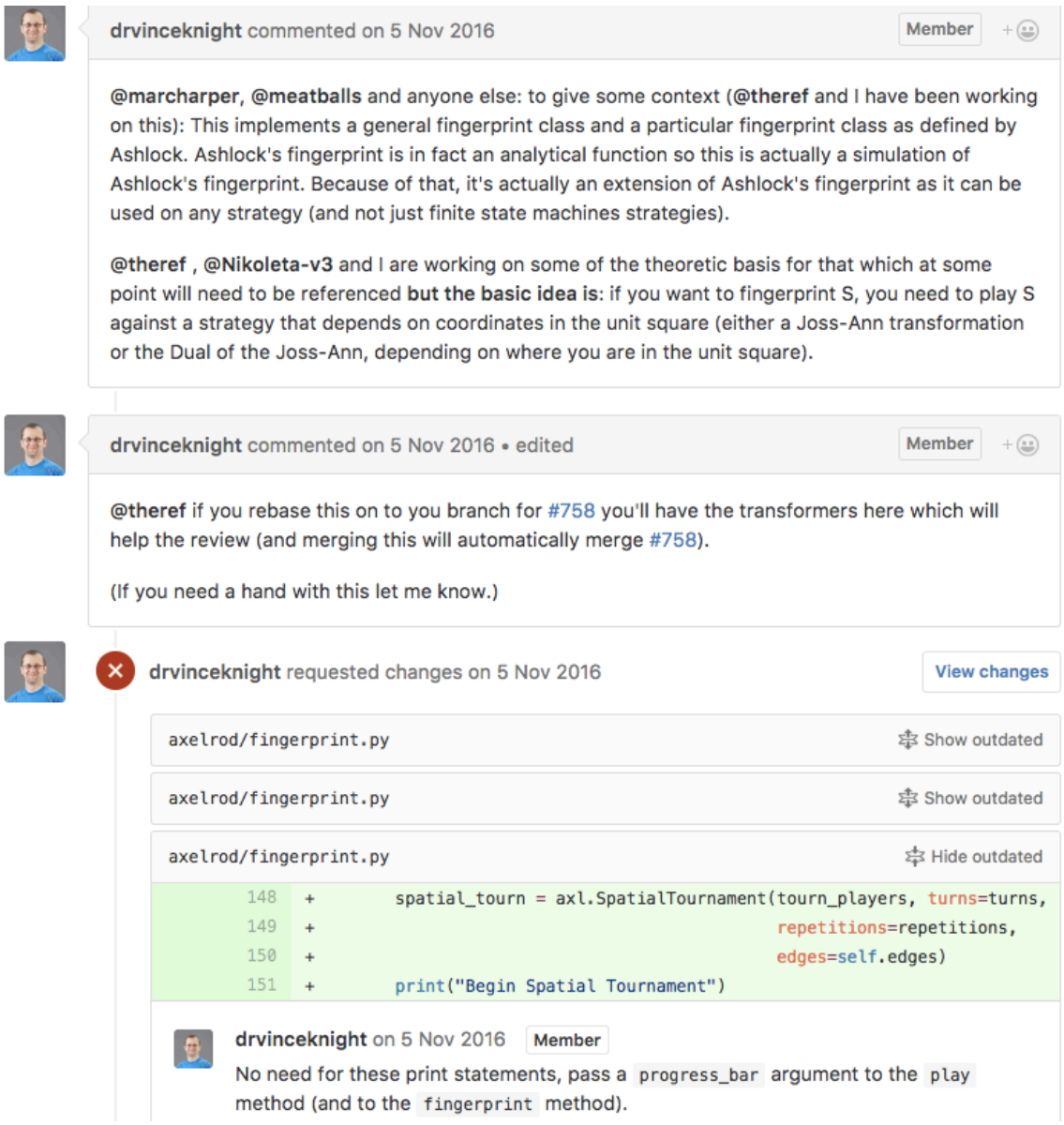


Figure 4.11: Advice to use other code being developed for the library

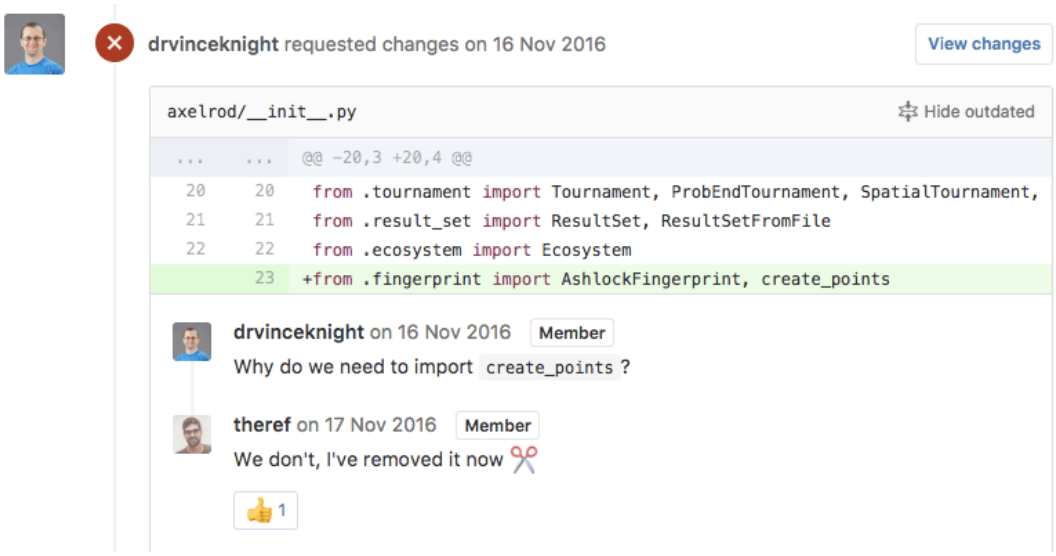


Figure 4.12: Request to remove redundant lines of code

something that would have prevented the code I had written from being included in the production version of the package.

4.5.3 Testing

All code in Axelrod-Python must be tested and the most common example of testing is unittesting. A unit test is a function that tests a small pieces of code and not an entire package/module, although it sometimes can. They operate by observing the result for a specific input and comparing it with a known output, then returning whether they are the same. A result of True indicates that the code is behaving as intended, and a result of False indicates that it is not. Consequently any program relying on that code cannot be trusted to behave as intended [43, 50].

Axelrod-Python uses several external programmes to help with testing. Travis [49] provides continuous integration testing, so all tests are run automatically whenever any changes to code are made. Coveralls [12] calculates what proportion of the code is tested, and can highlight areas that are either untested or need improvement. Finally, Hypothesis [30] is a library for property testing. It works by generating random data matching a specification and checking that a specified guarantee still holds in that case. If it finds an example where it doesnt, it takes that example and cuts it down to size, simplifying it until it finds a much smaller example that still causes the problem [22].

Chapter 5

Results

In this chapter several fingerprints will be examined in detail. In Section 5.1 a breakdown of the fingerprint for TitForTat is given with an explanation of its appearance. Section 5.4 shows several plots for the strategy GoByMajority, with a varying parameter. As the parameter changes, a continuous deformation of the plot can be observed. Finally, in Section 5.6 we compare the underlying data for all fingerprints of strategies within Axelrod-Python. This data is used to produce a heat plot of strategies similarity.

It may be useful to recall that a fingerprint for a strategy is produced by playing it against varying stochastic transformations of a probe. The parameters varied are x and y, as seen on fingerprints in this section. High x values correspond to high cooperation. Conversely, high y values correspond to high defection. The colour at point $\hat{x}, \hat{y}$ on the plot indicates the expected value of the strategy when played against the transformation of the probe with parameters $\hat{x}, \hat{y}$. This expected score has been approximated as outlined in Chapter 4.

5.1 Interpretation of TFT

TitForTat (see Section 2.2.1 for an explanation of its behaviour) it is one of the most well known strategies for Prisoner's Dilemma. Also, it produces a clean and easy to understand fingerprint, shown in Figure 5.1, making it an ideal place to start.

The plot slowly transitions from a strip of blue in the top left, rotating around to a large block of red in the bottom right. The blue area corresponds to low scores and it can be seen that this occurs for small values of x. However, as x increases higher scores (shown in red) quickly take over. This is exactly as expected, TitForTat plays well in highly cooperative environments. An immediate question is; why does the white stripe not follow the diagonal? The main diagonal is where $x = y$ and due to the random nature of the cooperation and defections, the expected average score for TitForTat would be !!!! about half the maximum. Due to the global minimum and maximum that TitForTat achieves, a score of !!!! is not midway, and so the white line showing the middle score is off centre. TitForTat is able to produce better than half scores in this scenario by encouraging the underlying probe strategy to cooperate.

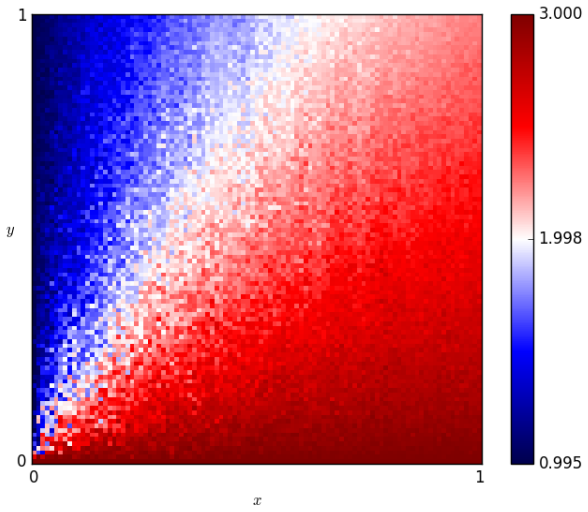


Figure 5.1: Fingerprint for TitForTat, with colour bar to add context

5.2 Varying the parameter for Random

It is possible to observe how changing a parameter for a strategy will affect its behaviour by comparing the fingerprints for each parameter. As described in Section 2.2.4, Random accepts a parameter that determines the probability of cooperation. We can vary this parameter and create a new fingerprint each time.

5.3 Alternator, Cycler(CD), Cooperator, Defector

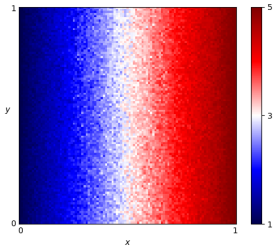
5.4 GoByMajority for different parameters

5.5 Random and Cycler(CDDC) with probes TFT and T42T

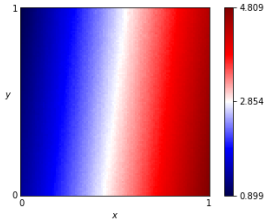
The importance of probe selection will now be demonstrated. Thus far, all fingerprints shown have used TitForTat as a probe, but this is not always enough. For example, Random and Cycler(CDDC) produce identical fingerprints when probed with TitForTat, as shown in

However, by changing the probe to TitForTwoTats, it becomes obvious that the strategies are different.

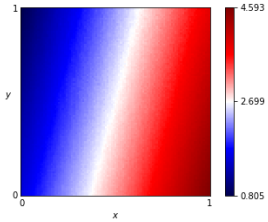
5.6 LSE plot and table



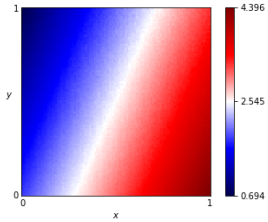
(a) Defector



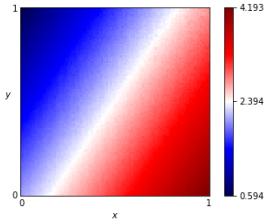
(b) Random(0.1) - will cooperate 10% of the time



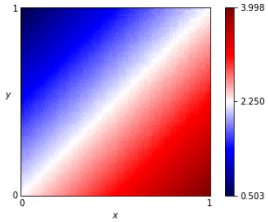
(c) Random(0.2) - will cooperate 20% of the time



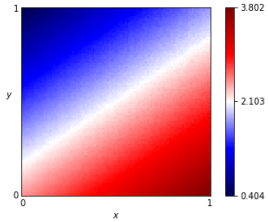
(d) Random(0.3) - will cooperate 30% of the time



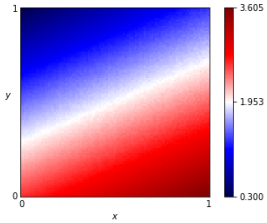
(e) Random(0.4) - will cooperate 40% of the time



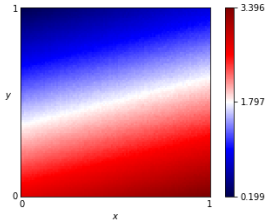
(f) Random(0.5) - will cooperate 50% of the time



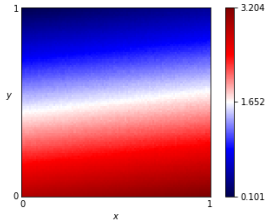
(g) Random(0.6) - will cooperate 60% of the time



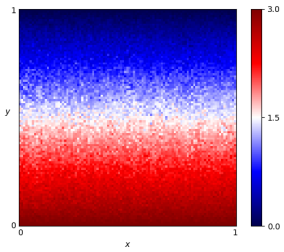
(h) Random(0.7) - will cooperate 70% of the time



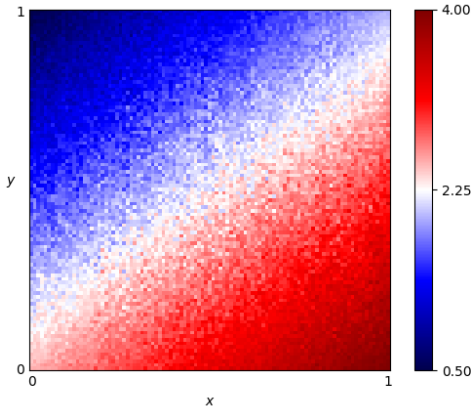
(i) Random(0.8) - will cooperate 80% of the time



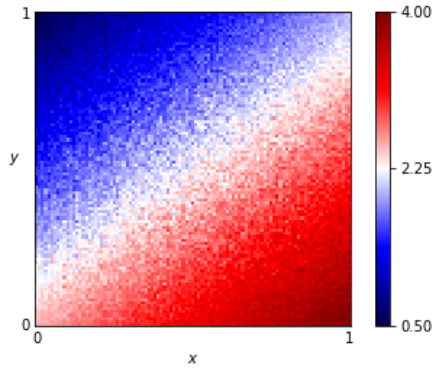
(j) Random(0.8) - will cooperate 90% of the time



(k) Cooperator

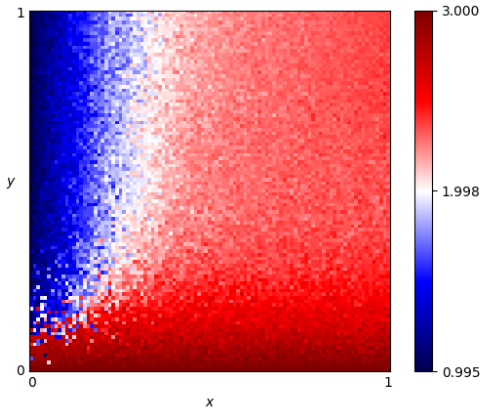


(a) Fingerprint for Alternator

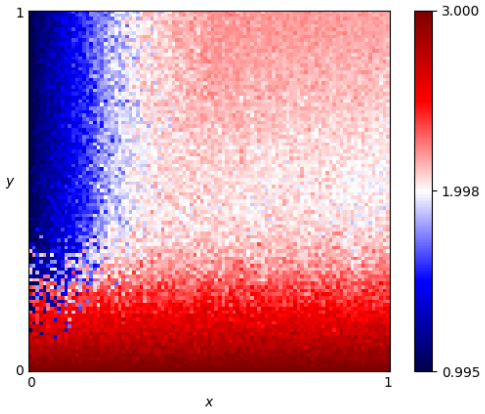


(b) Fingerprint for Cycler(CD)

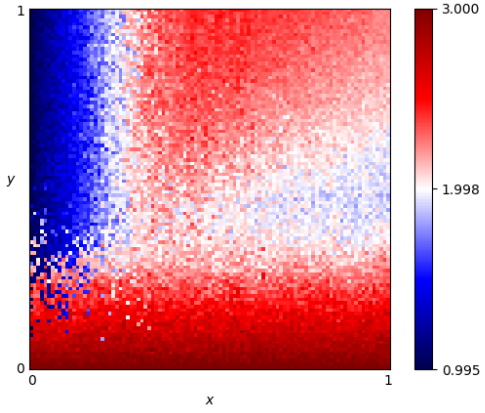
Figure 5.2: Fingerprints for Alternator and Cylcer(CD) when probed by TitForTat



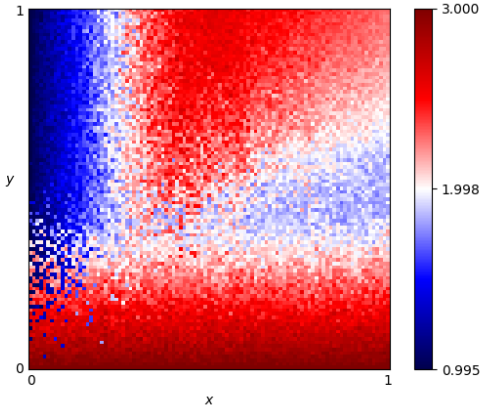
(a) GoByMajority with memory depth 5



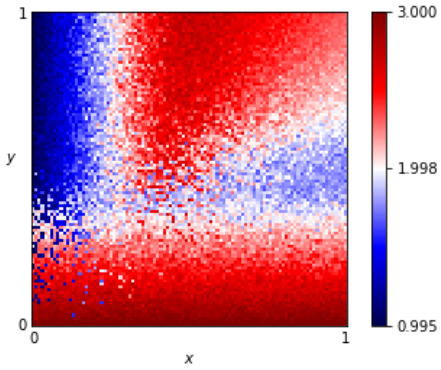
(b) GoByMajority with memory depth 10



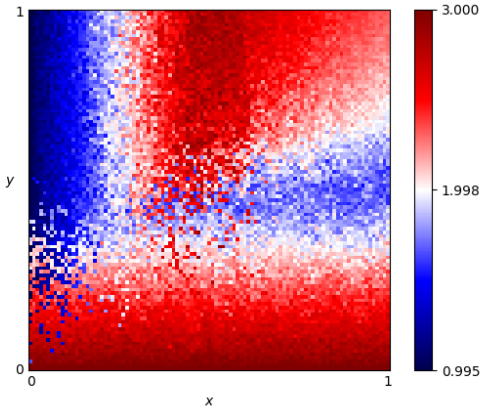
(c) GoByMajority with memory depth 20



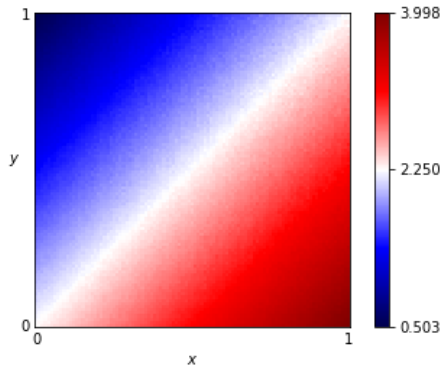
(d) GoByMajority with memory depth 40



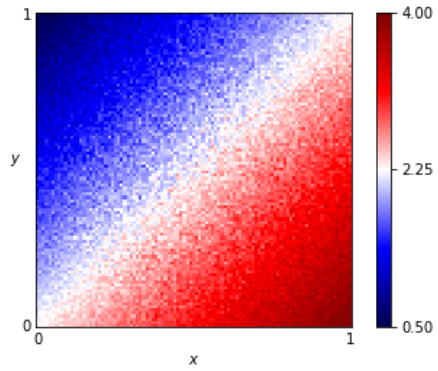
(e) GoByMajority with memory depth 75



(f) GoByMajority with memory depth infinite

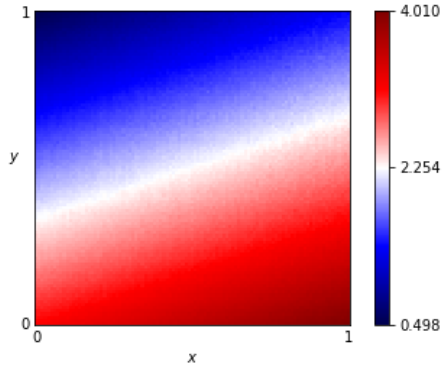


(a) Fingerprint for Random

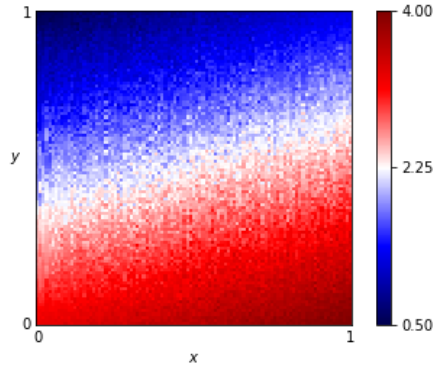


(b) Fingerprint for Cycler(CDDC)

Figure 5.3: Fingerprints for Random and Cylcer(CDDC) when probed by TitForTat

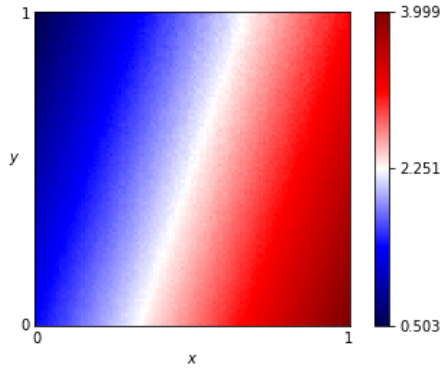


(a) Fingerprint for Random

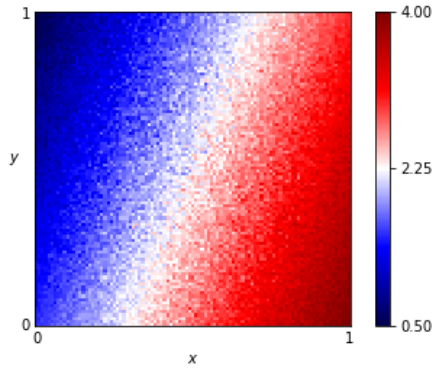


(b) Fingerprint for Cycler(CDDC)

Figure 5.4: Fingerprints for Random and Cylcer(CDDC) when probed by TitForTwoTats



(a) Fingerprint for Random



(b) Fingerprint for Cycler(CDDC)

Figure 5.5: Fingerprints for Random and Cylcer(CDDC) when probed by TwoTitsForTat

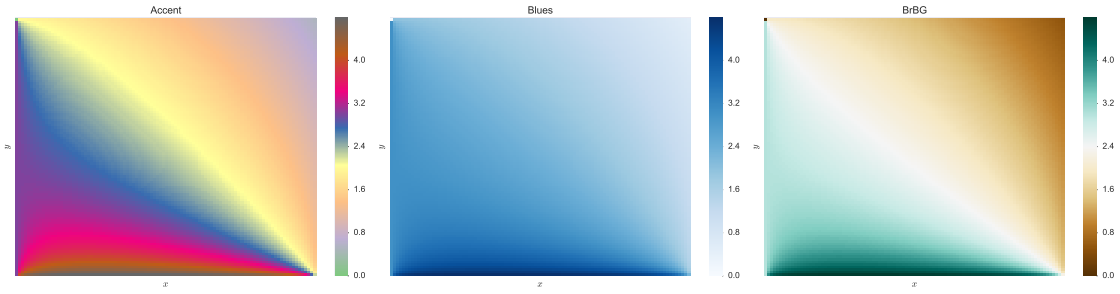
Chapter 6

Conclusion

Chapter 7

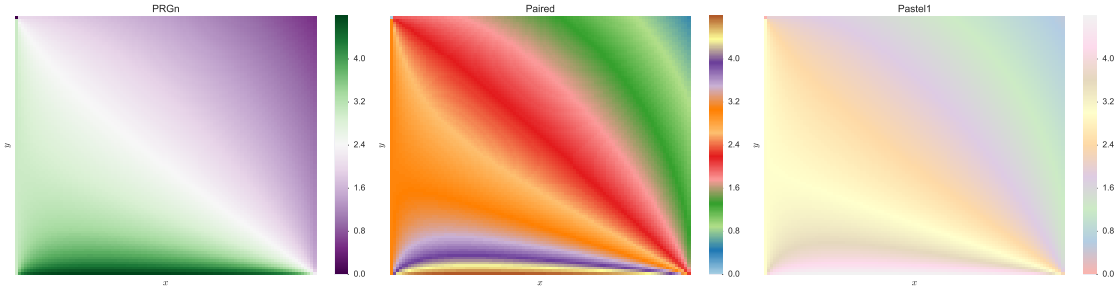
Appendix

7.1 Colour Maps

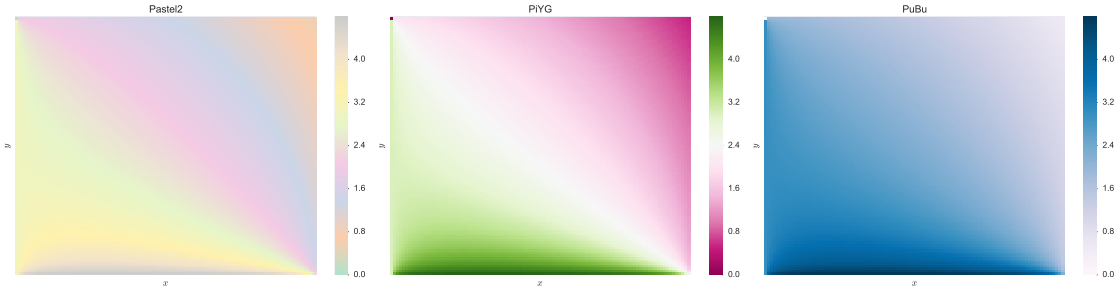


(a) Accent (b) Blues (c) BrBG

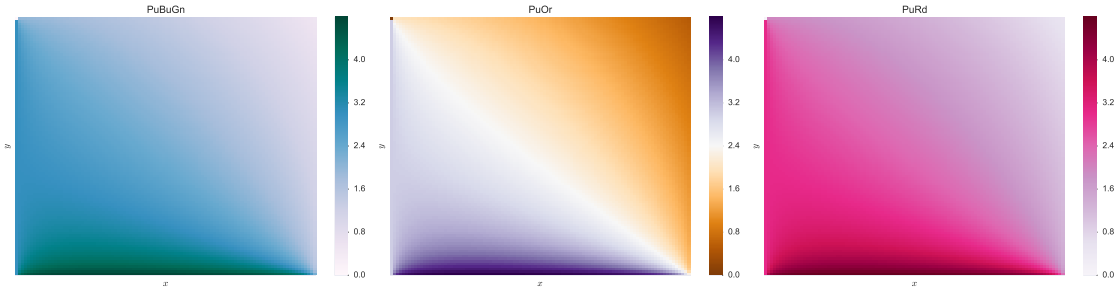
(d) BuGn (e) BuPu (f) CMRmap



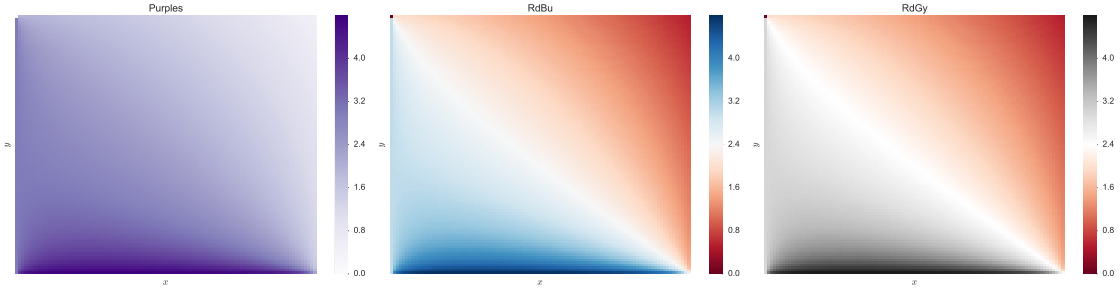
(g) PRGn (h) Paired (i) Pastel1



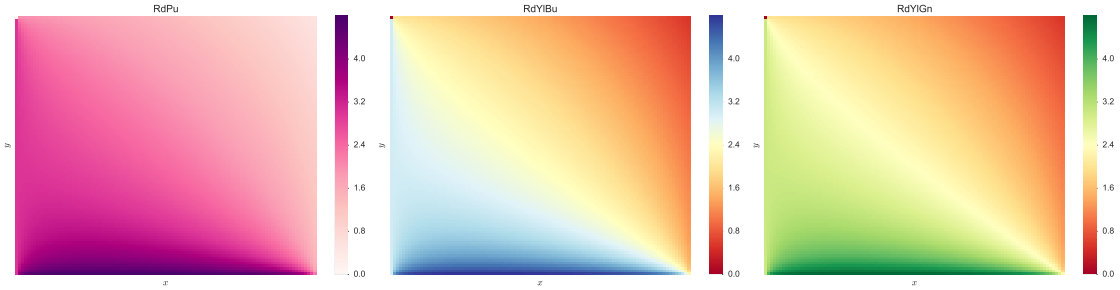
(j) Pastel2 (k) PiYG (l) PuBu



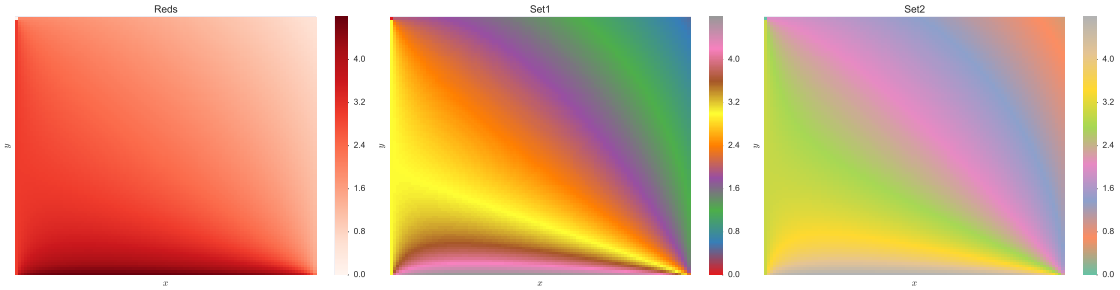
(m) PuBuGn (n) PuOr (o) PuRd



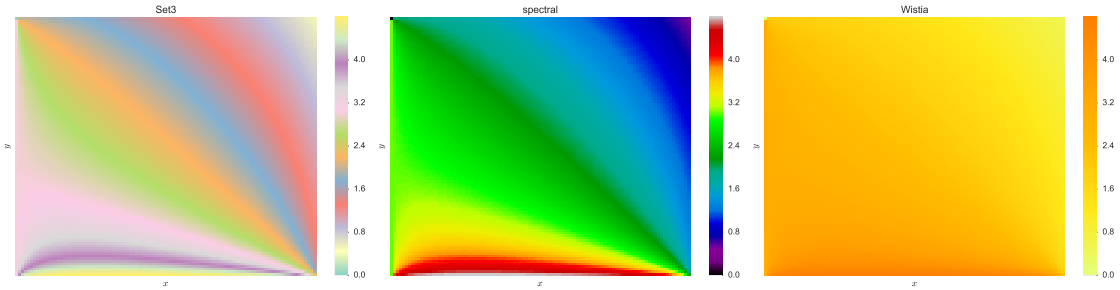
(p) Purples (q) RdBu (r) RdGy



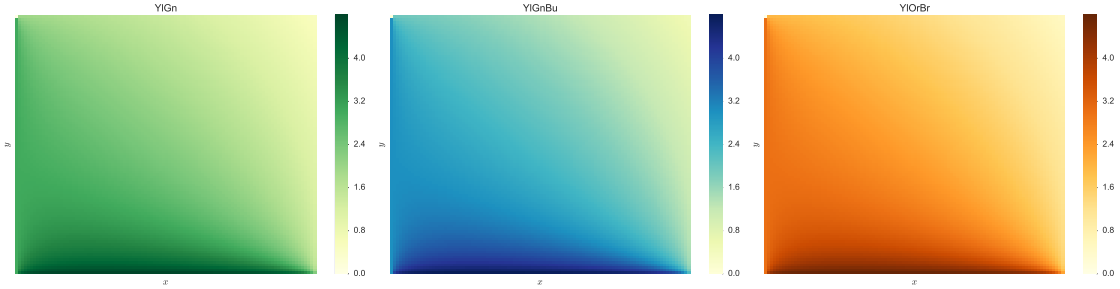
(s) RdPu (t) RdYlBu (u) RdYlGn



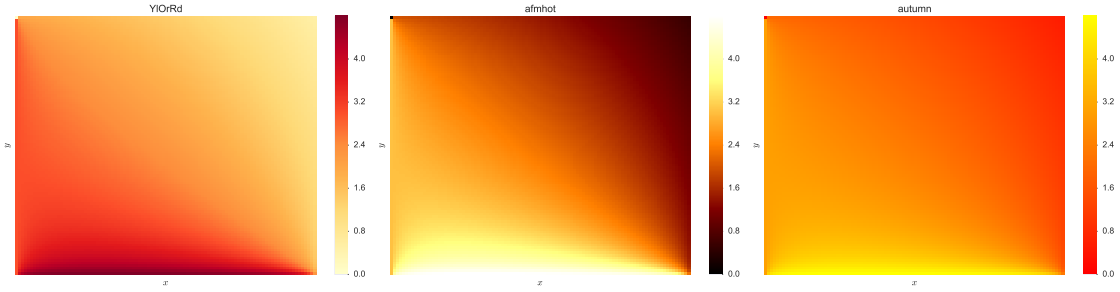
(v) Reds (w) Set1 (x) Set2



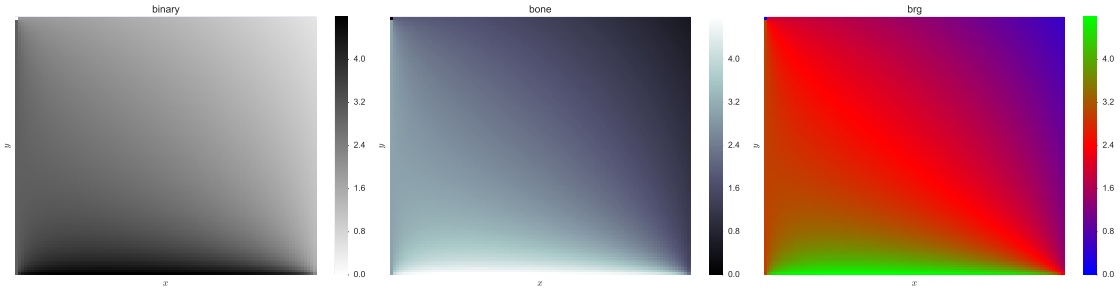
(y) Set3 (z) Spectral (aa) Wistia



(ab) YlGn (ac) YlGnBu (ad) YlOrBr

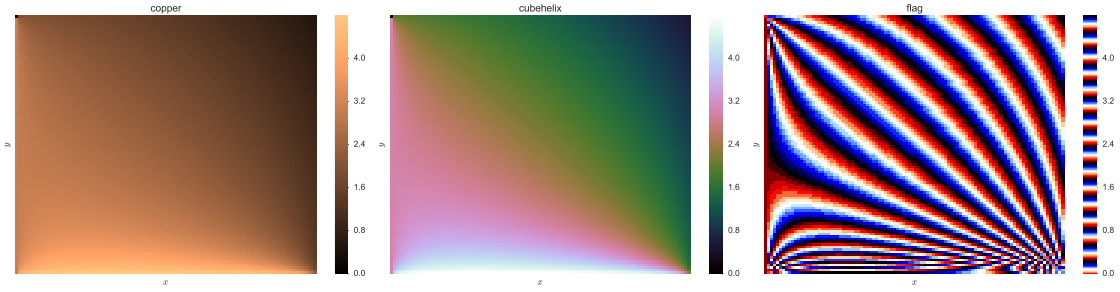


(ae) YlOrRd (af) afmhot (ag) autumn

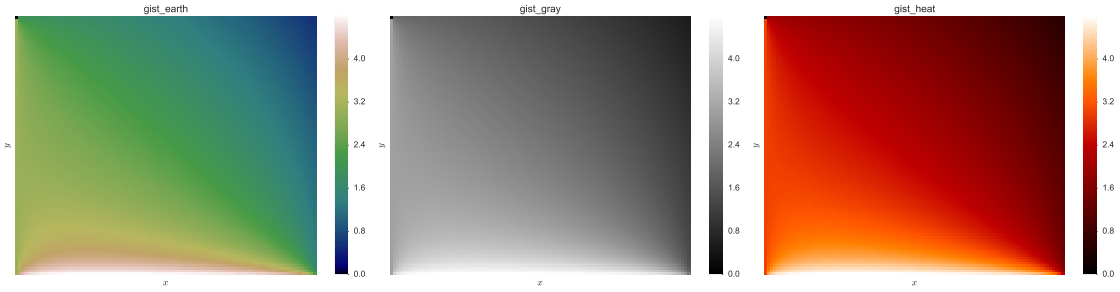


(ah) binary (ai) bone (aj) brg

(ak) bwr (al) cool (am) coolwarm

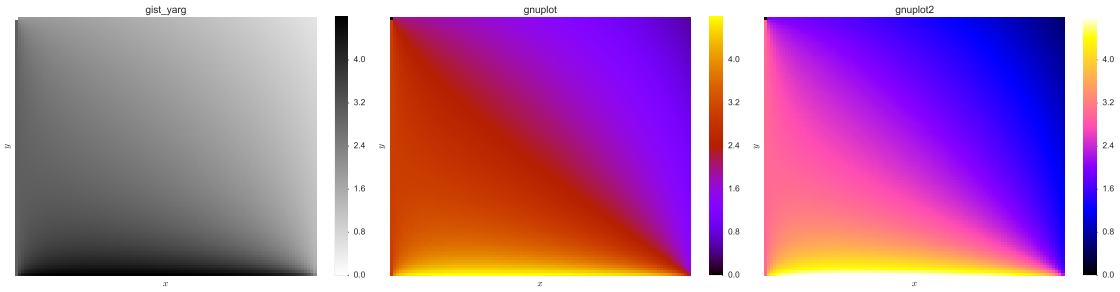


(an) copper (ao) cubehelix (ap) flag

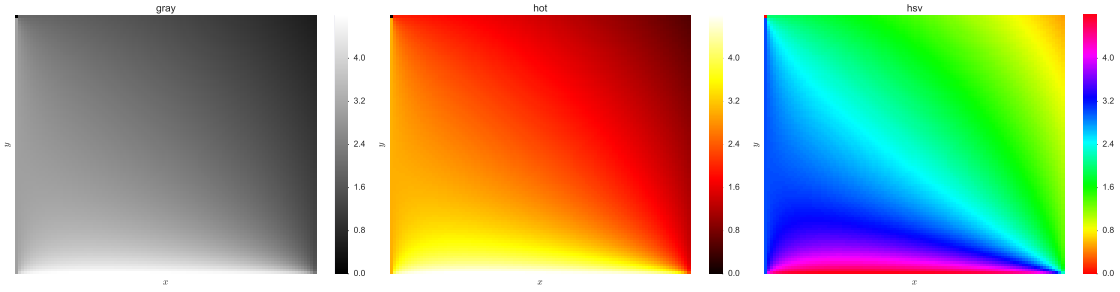


(aq) gist earth (ar) gist gray (as) gist heat

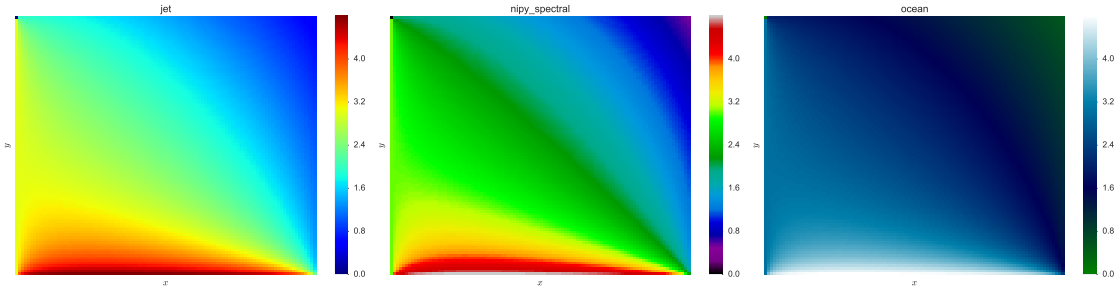
(at) gist ncar (au) gist rainbow (av) gist stern



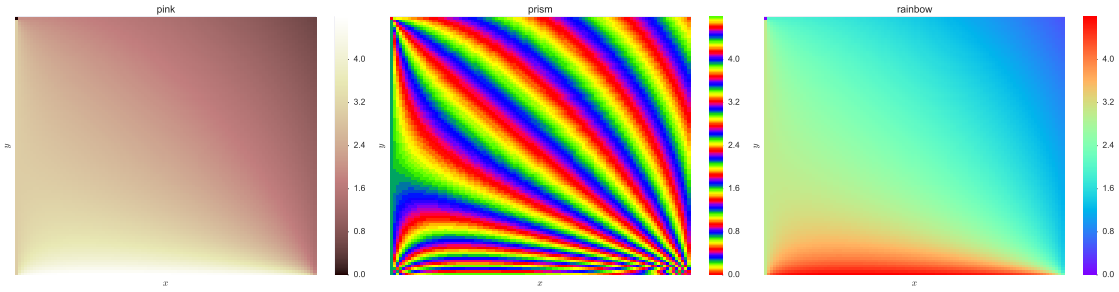
(aw) gist yarg (ax) gnuplot (ay) gnuplot2



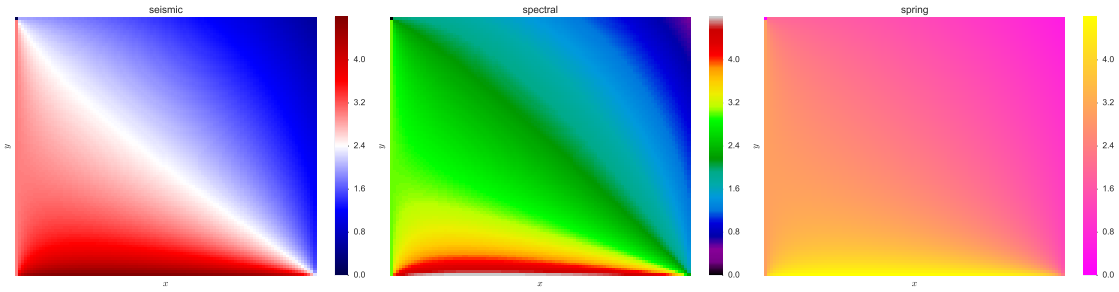
(az) gray (ba) hot (bb) hsv



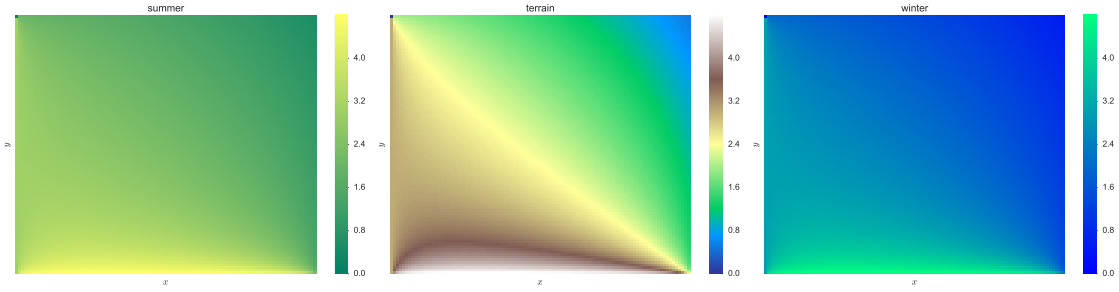
(bc) jet (bd) nipy spectral (be) ocean



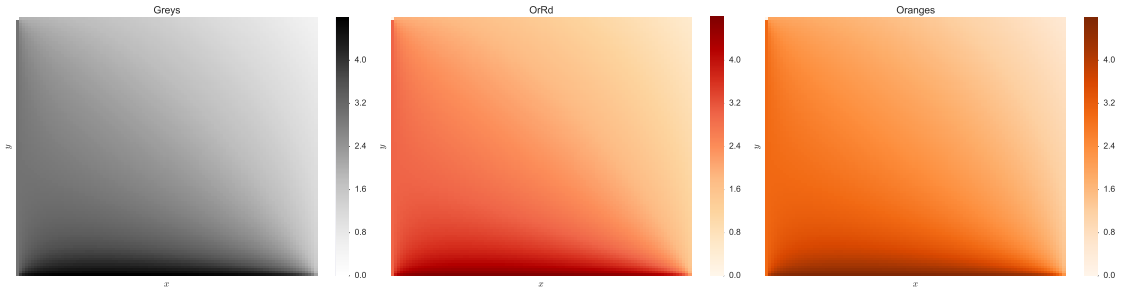
(bf) pink (bg) prism (bh) rainbow



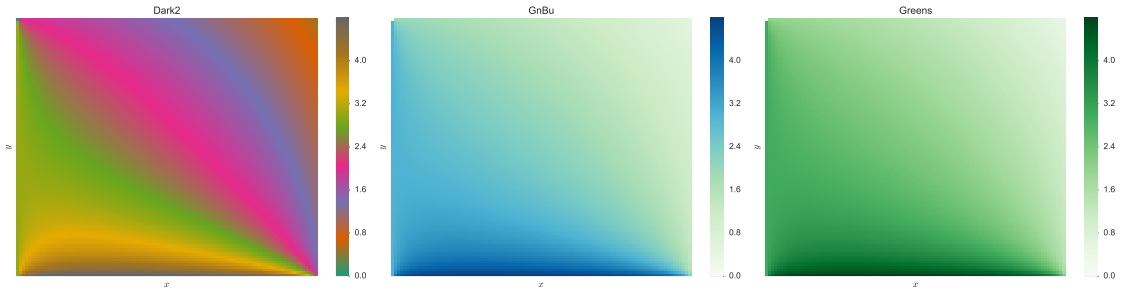
(bi) seismic (bj) spectral (bk) spring



(bl) summer (bm) terrain (bn) winter



(bo) Greys (bp) OrRd (bq) Oranges



(br) Dark2 (bs) GnBu (bt) Greens

Figure 7.-5: All possible colour maps from Matplotlib

Bibliography

[1] "An Open Framework for the Reproducible Study of the Iterated Prisoner ' s Dilemma". In: 4.1 (2016), e35 (cit. on pp. 10–12).

[2] Daniel Ashlock and Eon Youn Kim. "Fingerprinting: Visualization and automatic analysis of prisoner's dilemma strategies". In: *IEEE Transactions on Evolutionary Computation* 12.5 (2008), pp. 647–659. DOI: 10.1109/TEVC.2008.920675 (cit. on pp. 15, 16, 27, 28).

[3] Daniel Ashlock, Eun Youn Kim, and Warren Kurt. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: 56.2 (2004), pp. 397–410 (cit. on pp. 15, 16, 18, 27).

[4] Robert Axelrod. "Effective Choice in the Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 65.2 (1980), pp. 217–233 (cit. on pp. 10, 11).

[5] Robert Axelrod. "Effective choice in the prisone's dilemma". In: *Journal of Conflict Resolution* 24.1 (1980), pp. 3–25. URL: http://jcr.sagepub.com/cgi/content/abstract/24/1/3 (cit. on p. 6).

[6] Robert Axelrod. *The Evolution of Cooperation*. 1984 (cit. on p. 11).

[7] Robert Axelrod and William D. Hamilton. "The Evolution of Cooperation". In: *Science* 211.4489 (1981), pp. 1390–1396. DOI: 10.1126/science.7466396. arXiv: t8jd4qr3m [13960]. URL: http://www.jstor.org/stable/1685895 (cit. on p. 11).

[8] Bruno Beaufils, Jean-Paul Delahaye, and Philippe Mathieu. "Our meeting with gradual, a good strategy for the iterated prisoner's dilemma". In: *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. 1997, pp. 202–209. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3 (cit. on p. 12).

[9] J. Bendor, R. M. Kramer, and S. Stout. "When in Doubt ... Cooperation in a Noisy Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 35.4 (1991), pp. 691–719. URL: http://jcr.sagepub.com/content/35/4/691 (cit. on pp. 11, 12).

[10] Siang Y. Chong and Xin Yao. "Behavioral diversity, choices and noise in the iterated prisoner's dilemma". In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 540–551. DOI: 10.1109/TEVC.2005.856200 (cit. on p. 10).

[11] Siang Yew Chong et al. "Chapter 1 The Iterated Prisoner ' s Dilemma : 20 Years On". In: (2004), pp. 1–21 (cit. on p. 11).

[12] *Coveralls*. https://coveralls.io/ (cit. on p. 35).

[13] Tom Crick. "Share and Enjoy: Publishing Useful and Usable Scientific Models". In: (2014). arXiv: arXiv:1409.0367v1 (cit. on p. 14).

[14] The Axelrod project developers. *Axelrod: v1.18.1*. 2016. DOI: 10.5281/168358. URL: http://dx.doi.org/10.5281/zenodo.168358 (cit. on pp. 6, 14).

[15] Merrill M. Flood and Melvin Dresher. *Some Experimental Games*. 1958. DOI: 10.1287/mnsc.5.1.5 (cit. on p. 7).

[16] N Franken and a P Engelbrecht. "Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma". In: *Evolutionary Computation, IEEE Transactions on* 9.6 (2005), pp. 562–579. DOI: 10.1109/TEVC.2005.856202 (cit. on p. 10).

[17] Saul I. Gass and Arjang a Assad. *an Annotated Timeline of Operations Research*. 2005, p. 125. ISBN: 140208112X. DOI: 1402081138. URL: http://ebooks.kluweronline.com (cit. on p. 7).

[18] N. M. Gotts, J. G. Polhill, and A. N R Law. "Agent-based simulation in the study of social dilemmas". In: *Artificial Intelligence Review* 19.1 (2003), pp. 3–92. ISSN: 02692821. DOI: 10.1023/A:1022120928602 (cit. on p. 8).

[19] Shaun Hargreaves Heap and Yanis Varoufakis. *Game Theory: A Critical Text*. 2nd ed. 2003. ISBN: 0203199278 (cit. on p. 12).

[20] Neil P Chue Hong et al. "Top Tips to Make Your Research Irreproducible". In: (2015), pp. 5–6. arXiv: arXiv:1504.00062v2 (cit. on p. 14).

[21] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55. URL: http://matplotlib.org/examples/color/colormaps_reference.html (cit. on p. 27).

[22] *Hypothesis Documentation*. https://hypothesis.readthedocs.io/en/latest/ (cit. on p. 35).

[23] Darrel C Ince, Leslie Hatton, and John Graham-cumming. "The case for open computer programs". In: *Nature* 482.7386 (2012), pp. 485–488. ISSN: 0028-0836. DOI: 10.1038/nature10836. URL: http://dx.doi.org/10.1038/nature10836 (cit. on p. 32).

[24] Hisao Ishibuchi and Naoki Namikawa. "Evolution of iterated prisoner's dilemma game strategies in structured demes under random pairing in game playing". In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 552–561. DOI: 10.1109/TEVC.2005.856198 (cit. on p. 10).

[25] Martin Jones. *Evolving strategies for an Iterated Prisoner's Dilemma tournament*. 2015. URL: http://mojones.net/evolving-strategies-for-an-iterated-prisoners-dilemma-tournament.html (cit. on p. 13).

[26] Ehud Kalai and William Stanford. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: 679 (1986) (cit. on p. 14).

[27] Ehud Kalai and William Stanford. "Finite Rationality and Interpersonal Complexity in Repeated Games". In: *The Econometric Society* 56.2 (1988), pp. 397–410. URL: http://www.jstor.org/stable/1911078 (cit. on p. 14).

[28] Shyamalendu Kandar. *Introduction to Automata Theory, Formal Languages and Computation*. 1st ed. 2013 (cit. on p. 14).

[29] Vincent Knight et al. "An Open Framework for the Reproducible Study of the Iterated Prisoner's Dilemma". In: *Journal of Open Research Software* 4.1 (2016), e35. DOI: 10.5334/jors.125 (cit. on pp. 6, 12, 14).

[30] David R. MacIver. *Hypothesis 3.6.1*. https://github.com/HypothesisWorks/hypothesis-python. 2016 (cit. on pp. 14, 35).

[31] David Matthews, Greg Wilson, and Steve Easterbrook. "Configuration Management for Large-Scale Scientific Computing at the UK Met Office". In: *Computing in Science and Engineering* () (cit. on p. 32).

[32] Mojones. *axelrod-evolver*. https://github.com/mojones/axelrod-evolver. 2016 (cit. on p. 13).

[33] Kenneth Moreland. "Diverging color maps for scientific visualization". In: 5876 LNCS.PART 2 (2009), pp. 92–103. ISSN: 03029743. DOI: 10.1007/978-3-642-10520-3_9 (cit. on p. 27).

[34] Tertulien Ndjountche. *Digital Electronics, Volume 3*. 1st ed. Wiley, 2016 (cit. on p. 14).

[35] Martin A Nowak and Robert M May. "Evolutionary games and spatial chaos". In: *Letters to Nature* (1992) (cit. on p. 11).

[36] Martin A Nowak and Robert M May. "The Spatial Dilemmas of Evolution". In: *International Journal of Bifurcation and Chaos* 3 (1992), pp. 35–78 (cit. on pp. 11, 12).

[37] Martin A Nowak, Robert M May, and Sebastian Bonhoeffer. "Spatial games and the maintenance of cooperation". In: 91.May (1994), pp. 4877–4881 (cit. on p. 11).

[38] James B Procter and Andreas Prlic. "Ten Simple Rules for the Open Development of Scientific Software". In: 8.12 (2012), pp. 8–10. DOI: 10.1371/journal.pcbi.1002802 (cit. on p. 14).

[39] Karthik Ram. "Git can facilitate greater reproducibility and increased transparency in science". In: *Source Code for Biology and Medicine* (2013), pp. 1–8 (cit. on p. 32).

[40] Rosen and Lawrence. *Open Source Licensing*. 2004, p. 85. ISBN: 0131487876 (cit. on p. 14).

[41] Ariel Rubinstein. "Finite Automata Play the Repeated Prisoner's Dilemma". In: *Journal of Economic Theory* 96 (1986), pp. 83–96 (cit. on p. 14).

[42] Geir Kjetil Sandve et al. "Ten Simple Rules for Reproducible Computational". In: 9.10 (2013), pp. 1–4. DOI: 10.1371/journal.pcbi.1003285 (cit. on p. 14).

[43] Gopal P Sarma et al. "Unit testing , model validation , and biological simulation". In: (2016), pp. 1–13. URL: https://peerj.com/preprints/1315.pdf (cit. on p. 35).

[44] Matthias Schwab, Martin Karrenbach, and Jon Claerbout. "Making Scientific Computations Reproducible". In: *Scientific Computations* (2000), pp. 61–67 (cit. on p. 32).

[45] Karl Sigmund and Martin A Nowak. "Evolutionary game theory". In: *Current Biology* 9.14 (1999), pp. 503–505. DOI: 10.1007/978-1-4614-1800-9_63 (cit. on p. 10).

[46] Herbert A. Simon. *Theories of Bounded Rationality*. 1972 (cit. on p. 14).

[47] a. J. Stewart and J. B. Plotkin. "Extortion and cooperation in the Prisoner's Dilemma". In: *Proceedings of the National Academy of Sciences* 109.26 (2012), pp. 10134–10135. ISSN: 0027-8424. DOI: 10.1073/pnas.1208087109 (cit. on p. 11).

[48] Linus Torvald. *Git*. https://github.com/git/git (cit. on p. 32).

[49] *Travis*. https://travis-ci.org/ (cit. on p. 35).

[50] Laurie Williams, Gunnar Kudrjavets, and Nachiappan Nagappan. "On the Effectiveness of Unit Test Automation at Microsoft". In: (). URL: https://collaboration.csc.ncsu.edu/laurie/Papers/Unit%7B%5C_%7Dtesting%7B%5C_%7DcameraReady.pdf (cit. on p. 35).

[51] Greg Wilson et al. "Best Practices for Scientific Computing". In: (2013), pp. 1–18. arXiv: arXiv:1210.0530v4 (cit. on p. 32).