An open reproducible framework for the study of the iterated prisoner's dilemma

Owen Campbell Marc Harper Vincent Knight Karol Langner

March 9, 2016

Abstract

The Axelrod library is an open source Python package that allows for reproducible game theoretic research into the iterated Prisoner's Dilemma. This area of research began in the 1980s but suffers from a lack of documented and test code. The goal of the library is to provide such a resource, with facilities the design new strategies and interactions between them, as well as conducting tournaments and ecological simulations for population of strategies.

With a growing collection of 120 strategies the library is a also a platform for an original tournament that in itself is of interest to the game theoretic community.

This paper describes the iterated Prisoner's Dilemma, the Axelrod library and its development, and insights gained from some novel research.

1 Introduction

Several iterated prisoner's dilemma tournaments have generated much interest, including Axelrod's original tournaments [2, 3], two 2004 anniversary tournaments (ref?), and the Stewart and Plotkin 2012 tournament [42] following the discovery of zero-determinant strategies. Subsequent research has spawned an enormous number of papers, but rarely are the results reproducible. Among well-known tournaments, in only one case is the full original source code available (Axelrod's second tournament [3], in FORTRAN); in no cases is the available code well-documented, easily modifiable, or released with significant test suites.

To make matters more complicated, often a newly-created strategy is studied in isolation by opponents chosen by the strategy's creators, and often such strategies are not sufficiently described to enable reliable recreation (in the absense of source code), with [40] being a notable counter-example. In some cases strategies are revised without updates to their names or published implementations [25, 26]. As a result, some of the results related to these strategies and tournaments cannot be reliably replicated, and therefore have not met the basic scientific criterion of falsifiability.

This paper introduces a software package: the Axelrod Python library [22]. The Axelrod-Python project has the following stated goals:

- To enable the reproduction of Iterated Prisoner's Dilemma research as easily as possible
- To produce the de-facto tool for any future Iterated Prisoner's Dilemma research
- To provide as simple a means as possible for anyone to define and contribute new and original Iterated Prisoner's Dilemma strategies

The presented library is partly motivated by an ongoing discussion in the academic community about reproducible research [9, 16, 36, 38], and is:

- Open: all code is released under an MIT license;

- Reproducible and well-tested: at the time of writing there is an excellent level of integrated tests with 99.37% coverage
- Well-documented: all features of the library are documented for ease of use and modification
- Extensive: 120 strategies are included, with infinitely-many strategies available in the case of parameterized strategies
- Extensible: easy to modify to include new strategies and to run new tournaments

Before describing the package in more detail in Section 1.2, an overview of some previous Iterated Prisoner's Dilemma research will be given in Section 1.1. Section 2 will describe some early research and a conclusion is offered in Section 3.

1.1 Review of the literature

As stated in [6]: "*few works in social science have had the general impact of [Axelrod's study of the evolution of cooperation]*". In 1980, Axelrod wrote two papers: [2, 3] which describe a computer tournament that has been a imajor influence on subsequent game theoretic work [5, 6, 7, 8, 10, 11, 12, 13, 15, 18, 23, 24, 27, 31, 32, 33, 35, 41, 42]. As described in [6] this work has not only had impact in mathematics but has also led to insights in biology (for example in [41], a real tournament where Blue Jays are the participants is described) and in particular in the study of evolution.

The tournament is based on an iterated game (see [28] or similar for details) where two players repeatedly play the normal form game of (1) in full knowledge of each others playing history to date. An excellent description of the *one shot* game is given in [13] which is paraphrased below:

Two players must choose between *Cooperate* (C) and *Defect* (D):

- If both choose C, they receive a payoff of R (**R**eward);
- If both choose D, they receive a payoff of P (**P**unishment);
- If one chooses C and the other D, the defector receives a payoff of T (**T**emptation) and the cooperator a payoff of S (**S**ucker).

and the following reward matrix results from the Cartesian product of two decision vectors $\langle C, D\rangle$,

$$\begin{pmatrix} R,R & S,T \\ S,S & P,P \end{pmatrix} \quad \text{such that } T > R > P > S \text{ and } 2R > T + S \tag{1}$$

The game of (1) is called the Prisoner's Dilemma. Specific numerical values of $(R, S, T, P) = (3, 0, 5, 1)$ are often used in the literature, although any satisfying the conditions in 1 will yield similar results. Axelrod's tournaments (and further implementations of these) are sometimes referred to as Iterated Prisoner's Dilemma (IPD) tournaments. An incomplete overview of published tournaments is given in Table 1.

In [31] a description is given of how incomplete information can be used to enhance cooperation, in a similar approach to the proof of the Folk theorem for repeated games [28]. This aspect of incomplete information is also considered in [32, 6, 24] where "noisy" tournaments randomly flip the choice made by a given strategy. In [33], incomplete information is considered in the sense of a probabilistic termination of each round of the tournament.

As mentioned before, IPD tournaments have been studied in an evolutionary context: [12, 24, 35, 42] consider this in a traditional evolutionary game theory context. These works investigate particular evolutionary contexts within which cooperation can evolve and persist. This can be in the context of direct interactions between strategies or population dynamics for populations of many players using a variety of strategies,

Year	Reference	Number of Strategies	Type	Source Code
1979	[2]	13	Standard	Not immediately available
1979	[3]	64	Standard	Available in FORTRAN
1991	[6]	13	Noisy	Not immediately available
2002	[41]	16	Wildlife	Not applicable
2005	[20]	223	Varied	Not available
2012	[42]	13	Standard	Not fully available

Table 1: An overview of a selection of published tournaments. Not all tournaments were 'standard' round robins; for more details see the indicated references.

which can lead to very different results. For example, in [24] a machine learning algorithm in a population context outperforms strategies described in [35] and [42] that are claimed to dominate any evolutionary opponent in head-to-head interactions.

Further to these evolutionary ideas, [8, 10] are examples of using machine learning techniques to evolve particular strategies. In [4], Axelrod describes how similar techniques are used to genetically evolve a high performing strategy from a given set of strategies. Note that in his original work, Axelrod only used a base strategy set of 12 strategies for this evolutionary study. This is noteworthy as [22] now boasts over 120 strategies that are readily available for a similar analysis.

1.2 Description of the Axelrod Python package

The library is written in Python (`http://www.python.org/`) which is a popular language in the academic community with libraries developed for a variety of uses including:

- Machine learning [34] (`http://scikit-learn.org/`);
- Visualisation [17] (`http://matplotlib.org/`);
- Mathematics [37] (`http://www.sagemath.org/`);
- Astrophysics [1] (`http://www.astropy.org/`);
- Data manipulation [30] (`http://pandas.pydata.org/`);
- Algorithmic Game Theory [29] (`http://gambit.sourceforge.net/`)).

Furthermore, in [18] Python is described as an appropriate language for the reproduction of Iterated Prisoner's dilemma tournaments due to its object oriented nature and readability.

The library itself is available at `https://github.com/Axelrod-Python/Axelrod`. This is a hosted git repository. Git is a version control system which is one of the recommended aspects of reproducible research [9, 38].

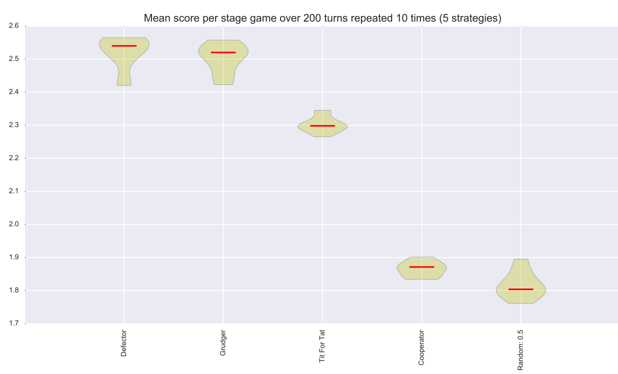
Installation of the library is straightforward as it is available via the standard Python installation package: 'pip' (`https://pypi.python.org/pypi`). This ensures it can be used on all major operating systems (Windows, OS X and Linux).

Figure 1 shows a very simple example of using the library to create a basic tournament giving the graphical output shown in Figure 2a.

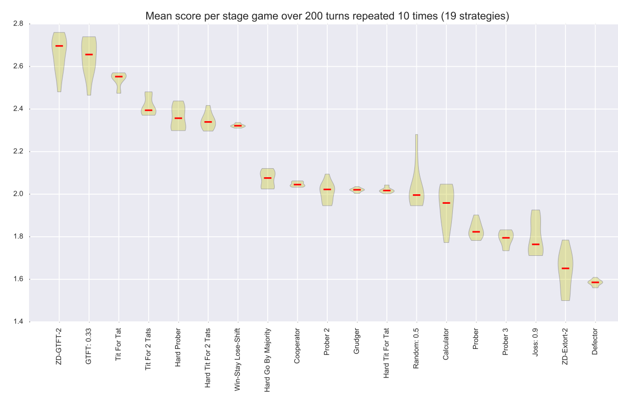
Figure 1 shows the very basic utilisation of the library and further details can be found at the online documentation: `http://axelrod.readthedocs.org`.

```
>>> import axelrod
>>> strategies = [s() for s in axelrod.demo_strategies]
>>> tournament = axelrod.Tournament(strategies)
>>> results = tournament.play()
>>> plot = axelrod.Plot(results)
>>> plot.boxplot()  # doctest:+SKIP
```

Figure 1: A simple set of commands to create a demonstration tournament. The output is shown in Figure 2a.



(a) The results from a simple tournament.



(b) The results from [42].

Figure 2: Summary plots produced by the library.

As stated in Section 1 one of the main goals of the library is to allow for the easy contribution of strategies. Doing this requires the writing of a simple Python class (which can inherit from other predefined classes). Full contribution guidelines can be found in the documentation. As an example, Figures 3 and 4 show the source code for the Grudger strategy as well as its corresponding test.

To date the library has had contributions from 24 contributors from a variety of backgrounds. These contributions have both been in terms of strategies (one strategy is the creation of an undergraduate mathematics student with little prior knowledge of programming) as well as the architecture of the library itself. You can see an overview of the structure of the source code in Figure 5a.

Section 2 will describe in more detail some of research capabilities of the library.

2 New strategies, tournaments and implications

Due to the open nature of the library the number of strategies included has grown at a fast pace, as can be seen in Figure 5b. Despite this, due to previous research being done in an irreproducible manner, with for example no source code, and/or vaguely described strategies, not all previous tournaments are yet to be reproduced. In fact, some of the early tournaments might be impossible to reproduce as the source code is forever lost. This library aims to prevent this from happening again in the future.

One tournament that is possible to reproduce is that of [42]. The strategies used in that tournament are the following:

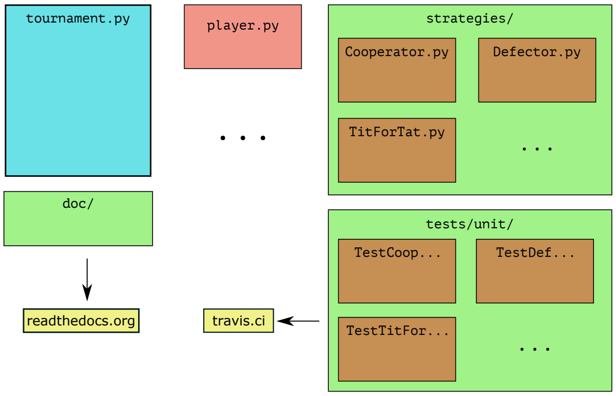
1. Cooperator
2. Defector

```python
class Grudger(Player):
    """A player starts by cooperating however will defect if
       at any point the opponent has defected."""

    name = 'Grudger'
    classifier = {
        'memory_depth': float('inf'),  # Long memory
        'stochastic': False,
        'inspects_source': False,
        'manipulates_source': False,
        'manipulates_state': False
    }

    def strategy(self, opponent):
        """Begins by playing C, then plays D for the remaining
           rounds if the opponent ever plays D."""
        if opponent.defections:
             return D
        return C
```

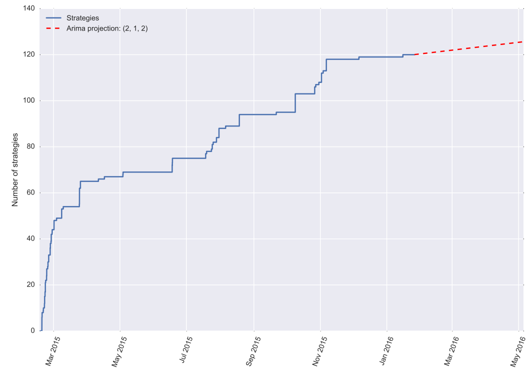
Figure 3: Source code for the Grudger strategy.

```python
class TestGrudger(TestPlayer):

    name = "Grudger"
    player = axelrod.Grudger
    expected_classifier = {
        'memory_depth': float('inf'),  # Long memory
        'stochastic': False,
        'inspects_source': False,
        'manipulates_source': False,
        'manipulates_state': False
    }

    def test_initial_strategy(self):
        """
        Starts by cooperating
        """
        self.first_play_test(C)

    def test_strategy(self):
        """
        If opponent defects at any point then the player will defect forever
        """
        self.responses_test([C, D, D, D], [C, C, C, C], [C])
        self.responses_test([C, C, D, D, D], [C, D, C, C, C], [D])
```

Figure 4: Test code for the Grudger strategy.



(a) An overview of the source code.



(b) The number of strategies included in the library

Figure 5: An overview of the library.

3. ZD-Extort-2
4. Joss: 0.9
5. Hard Tit For Tat
6. Hard Tit For 2 Tats
7. Tit For Tat
8. Grudger
9. Tit For 2 Tats
10. Win-Stay Lose-Shift
11. Random: 0.5
12. ZD-GTFT-2
13. GTFT: 0.33
14. Hard Prober
15. Prober
16. Prober 2
17. Prober 3
18. Calculator
19. Hard Go By Majority

This can be reproduced as shown in Figure 6, which gives the plot of Figure 2b.

In parallel to the Python library, a tournament is being kept up to date that pits all available strategies against each other. Figure 7 shows the results from the full tournament which can also be seen (in full detail) here: `http://axelrod-tournament.readthedocs.org/`.

The current winning strategy is a novel strategy called: Looker Up. This is a strategy that maps a given set of states to actions. The state space is defined generically by m, n so as to map states to actions as shown in (2).

```
>>> import axelrod

>>> strategies = [axelrod.Cooperator(),
...               axelrod.Defector(),
...               axelrod.ZDExtort2(),
...               axelrod.Joss(),
...               axelrod.HardTitForTat(),
...               axelrod.HardTitFor2Tats(),
...               axelrod.TitForTat(),
...               axelrod.Grudger(),
...               axelrod.TitFor2Tats(),
...               axelrod.WinStayLoseShift(),
...               axelrod.Random(),
...               axelrod.ZDGTFT2(),
...               axelrod.GTFT(),
...               axelrod.HardProber(),
...               axelrod.Prober(),
...               axelrod.Prober2(),
...               axelrod.Prober3(),
...               axelrod.Calculator(),
...               axelrod.HardGoByMajority()]
>>> tournament = axelrod.Tournament(strategies)
>>> results = tournament.play()
>>> plot = axelrod.Plot(results)
>>> plot.boxplot()  # doctest:+SKIP
```

Figure 6: Source code for reproducing the tournament of [42]

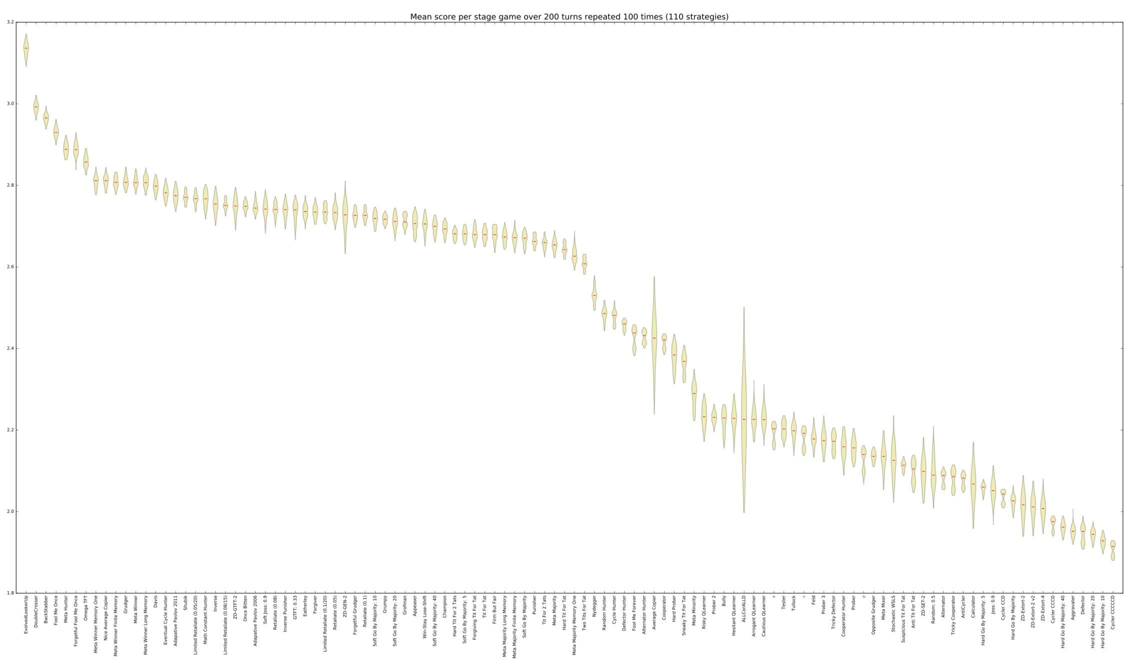


Figure 7: Results from the library tournament (2016-02-10)

$$(\underbrace{(C,D,D,D,C,D,D,C)}_{m \text{ first actions by opponent}}, \overbrace{((C,C),(C,C))}^{n \text{ last pairs of actions}}) \to D \tag{2}$$

The example of (2) is an incomplete illustration of the mapping for $m = 8, n = 2$. Intuitively, this state space uses the initial plays of the opponent to gain some information about its intentions whilst still taking in to account the recent play. The actual winning strategy is an instance of the framework for $m = n = 2$ for which an evolutionary algorithm has been used to train it. More details of this can be found in [19]. In [21] experiments are described that evaluate how this particular strategy behaves in environments other than those in which it was trained and it continues to perform strongly.

There are various other insights that have been gained from ongoing open research on the library. These include:

- A closer look at zero determinant strategies, showing that extortionate strategies obtain a large number of wins *but* do not perform particularly well. This is relevant given the findings of [42] in which zero determinant strategies are shown to be able to perform better than any other strategy. This finding extends to noisy tournaments (which are also implemented in the library).
- This negative relationship between wins and performance does not generalise. There are some strategies that perform well, both in terms of matches won and overall performance: Back stabber, Double crosser, Looker Up, and Fool Me Once. These strategies continue to perform well in noisy tournaments, however some of these have knowledge of the length of the game (Back stabber and Double crosser). This is not necessary to rank well in both wins and score as demonstrated by Looker Up and Fool Me Once.
- Strategies like Looker Up and Meta Hunter seem to be generally cooperative yet still exploit naive strategies. The MetaHunter strategy is a particular type of Meta strategy which uses a variety of other strategy behaviours to choose a best action. These strategies perform very well in general and continue to do so in noisy tournaments.

Details for the above can be found in [14].

3 Conclusion

This paper has presented a game theoretic software package that aims to address reproducibility of research into the iterated prisoner's dilemma. The open nature of the development of the library has lead rapidly to the inclusion of many well known strategies, many novel strategies, and new and recapitulated insights.

The capabilities of the library mentioned above are not at all comprehensive, a list of the current abilities include:

- Noisy tournaments;
- Ecological analysis of tournaments;
- Morality metrics based on [39];
- Transformation of strategies (in effect giving an infinite number of strategies);
- Classification of strategies according to multiple dimensions.
- Gathering of full interaction history for all interactions.

These capabilities are constantly being updated. Some immediate further work includes allowing for tournaments of probabilistic length as well as further implementation of known strategies.

The ability to readily have available a large number of strategies makes this tool an excellent and obvious example of the benefits of open research which should positively impact the game theoretic community.

References

[1] Astropy Collaboration et al. "Astropy: A community Python package for astronomy". In: *Astronomy and Astrophysics* 558, A33 (Oct. 2013), A33. DOI: 10.1051/0004-6361/201322068. arXiv: 1307.6212 [astro-ph.IM] (cit. on p. 3).

[2] R. Axelrod. "Effective Choice in the Prisoner's Dilemma". In: *Journal of Conflict Resolution* 24.1 (1980), pp. 3–25 (cit. on pp. 1–3).

[3] R. Axelrod. "More Effective Choice in the Prisoner's Dilemma". In: *Journal of Conflict Resolution* 24.3 (1980), pp. 379–403. ISSN: 0022-0027. DOI: 10.1177/002200278002400301 (cit. on pp. 1–3).

[4] R. Axelrod. *The Evolution of Cooperation* (cit. on p. 3).

[5] J. S. Banks and R. K. Sundaram. "Repeated games, finite automata, and complexity". In: *Games and Economic Behavior* 2.2 (1990), pp. 97–117. ISSN: 08998256. DOI: 10.1016/0899-8256(90)90024-O (cit. on p. 2).

[6] J. Bendor, R. M. Kramer, and S. Stout. "When in doubt . . .: Cooperation in a noisy prisoner's dilemma". In: *Journal of Conflict Resolution* 35.4 (1991), pp. 691–719. ISSN: 0022-0027. DOI: 10.1177/0022002791035004007 (cit. on pp. 2, 3).

[7] R. Boyd and J. P. Lorberbaum. "No pure strategy is evolutionarily stable in the repeated Prisoner's Dilemma game". In: *Nature* 327 (1987), pp. 58–59. ISSN: 0028-0836. DOI: 10.1006/jtbi.1994.1092 (cit. on p. 2).

[8] K. Chellapilla and D. B. Fogel. "Evolution, neural networks, games, and intelligence". In: *Proceedings of the Ieee* 87.9 (1999), pp. 1471–1496. ISSN: 00189219. DOI: Doi10.1109/5.784222. URL: %3CGo%20to%20ISI%3E://WOS:000082176700004 (cit. on pp. 2, 3).

[9] T. Crick et al. ""Share and Enjoy": Publishing Useful and Usable Scientific Models". In: (2014). arXiv: 1409.0367. URL: http://arxiv.org/abs/1409.0367 (cit. on pp. 1, 3).

[10] F. David B. "Evolving Behaviors in the Iterated Prisoner's Dilemma". In: *Evol. Comput.* 1.1 (1993), pp. 77–97. ISSN: 1063-6560. DOI: 10.1162/evco.1993.1.1.77. URL: http://dx.doi.org/10.1162/evco.1993.1.1.77$%5Cbackslash$nhttp://dl.acm.org/ft%5C_gateway.cfm?id=1326628%5C&type=pdf$%5Cbackslash$nhttp://www.mitpressjournals.org/action/cookieAbsent (cit. on pp. 2, 3).

[11] M. Doebeli and C. Hauert. "Models of cooperation based on the Prisoner's Dilemma and the Snowdrift game". In: *Ecology Letters* 8.7 (2005), pp. 748–766. ISSN: 1461023X. DOI: 10.1111/j.1461-0248.2005.00773.x (cit. on p. 2).

[12] G. Ellison. "Cooperation in the prisoner's dilemma with anonymous random matching". In: *Review of Economic Studies* 61.3 (1994), pp. 567–588. ISSN: 00346527. DOI: 10.2307/2297904 (cit. on p. 2).

[13] N. Gotts, J. Polhill, and A. Law. "Agent-based simulation in the study of social dilemmas". In: *Artificial Intelligence Review* 19 (2003), pp. 3–92. ISSN: 0269-2821. DOI: 10.1023/A:1022120928602. URL: http://dl.acm.org/citation.cfm?id=608970 (cit. on p. 2).

[14] M. Harper. *Marc Harper Codes*. 2015. URL: http://marcharper.codes/ (visited on 02/10/2016) (cit. on p. 8).

[15] C. Hilbe, M. a. Nowak, and A. Traulsen. "Adaptive Dynamics of Extortion and Compliance". In: *PLoS ONE* 8.11 (2013), e77886. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0077886. URL: http://dx.plos.org/10.1371/journal.pone.0077886 (cit. on p. 2).

[16] N. P. C. Hong et al. “Top Tips to Make Your Research Irreproducible”. In: (2015), pp. 5–6. arXiv: 1504.00062. URL: http://arxiv.org/abs/1504.00062 (cit. on p. 1).

[17] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95 (cit. on p. 3).

[18] A. Isaac. “Simulating Evolutionary Games: A Python-Based Introduction”. In: *Journal of Artificial Societies and Social Simulation* 11.3 (2008), p. 8. ISSN: 14607425. URL: http://jasss.soc.surrey.ac.uk/11/3/8.html (cit. on pp. 2, 3).

[19] M. Jones. *Evolving strategies for an Iterated Prisoner’s Dilemma tournament.* 2015. URL: http://mojones.net/evolving-strategies-for-an-iterated-prisoners-dilemma-tournament.html (visited on 02/09/2016) (cit. on p. 8).

[20] G. Kendall, X. Yao, and S. Y. Chong. *The iterated prisoners’ dilemma: 20 years on.* World Scientific Publishing Co., Inc., 2007 (cit. on p. 3).

[21] V. Knight. *http://vknight.org/unpeudemath/gametheory/2015/11/28/Experimenting-with-a-high-performing-evolved-strategy-in-other-environments/.* 2015. URL: http://vknight.org/unpeudemath/gametheory/2015/11/28/Experimenting-with-a-high-performing-evolved-strategy-in-other-environments/ (visited on 02/10/2016) (cit. on p. 8).

[22] V. Knight et al. *Axelrod: Alternator Release.* Jan. 2016. DOI: 10.5281/zenodo.45187. URL: http://dx.doi.org/10.5281/zenodo.45187 (cit. on pp. 1, 3).

[23] D. Kraines and V. Kraines. “Pavlov and the prisoner’s dilemma”. In: *Theory and Decision* 26.1 (1989), pp. 47–79. ISSN: 00405833. DOI: 10.1007/BF00134056 (cit. on p. 2).

[24] C. Lee, M. Harper, and D. Fryer. “The Art of War: Beyond Memory-one Strategies in Population Games”. In: *Plos One* 10.3 (2015), e0120625. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0120625. URL: http://dx.plos.org/10.1371/journal.pone.0120625 (cit. on pp. 2, 3).

[25] J. Li. “How to design a strategy to win an IPD tournament”. In: *The iterated prisoners dilemma* 20 (2007), pp. 89–104 (cit. on p. 1).

[26] J. Li, P. Hingston, and G. Kendall. “Engineering design of strategies for winning iterated prisoner’s dilemma competitions”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 3.4 (2011), pp. 348–360 (cit. on p. 1).

[27] J. P. Lorberbaum. “No strategy is evolutionarily stable in the repeated Prisoner’s Dilemma game”. In: *Journal of Theoretical Biology* 168.2 (1994), pp. 117–130 (cit. on p. 2).

[28] M. Maschler, E. Solan, and S. Zamir. *Game theory.* Cambridge University Press, 2013, p. 1003. ISBN: 9781107005488. DOI: http://dx.doi.org/10.1017/CBO9780511794216. URL: http://www.cambridge.org/gb/academic/subjects/economics/economics-general-interest/game-theory (cit. on p. 2).

[29] R. Mckelvey et al. *Gambit: Software tools for game theory.* Tech. rep. 2006 (cit. on p. 3).

[30] W. McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference.* Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56 (cit. on p. 3).

[31] P. Milgrom, J. Roberts, and R. Wilson. “Rational Cooperation in the Finitely Repeated Prisoners’ Dilemma”. In: *Journal of Economic Theory* 252 (1982), pp. 245–252 (cit. on p. 2).

[32] P. Molander. “The optimal level of generosity in a selfish, uncertain environment”. In: *The Journal of Conflict Resolution* 29.4 (1985), pp. 611–618. ISSN: 0022-0027. DOI: 10.1177/0022002785029004004 (cit. on p. 2).

[33] J. K. Murnighan et al. “Expecting Continued Play in Prisoner ’ s Dilemma Games”. In: 27.2 (1983), pp. 279–300 (cit. on p. 2).

[34] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 3).

[35] W. H. Press and F. J. Dyson. "Iterated Prisoner's Dilemma contains strategies that dominate any evolutionary opponent". In: *Proceedings of the National Academy of Sciences* 109.26 (2012), pp. 10409–10413. ISSN: 0027-8424. DOI: 10.1073/pnas.1206569109 (cit. on pp. 2, 3).

[36] A. Prli and J. B. Procter. "Ten Simple Rules for the Open Development of Scientific Software". In: *PLoS Computational Biology* 8.12 (2012), e1002802. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1002802. URL: http://dx.plos.org/10.1371/journal.pcbi.1002802 (cit. on p. 1).

[37] T. S. Developers. *Sage Mathematics Software (Version 7.0)*. http://www.sagemath.org. 7.0 (cit. on p. 3).

[38] G. K. Sandve et al. "Ten Simple Rules for Reproducible Computational Research". In: *PLoS Computational Biology* 9.10 (2013), pp. 1–4. ISSN: 1553734X. DOI: 10.1371/journal.pcbi.1003285 (cit. on pp. 1, 3).

[39] T. Singer-Clark. "Morality Metrics On Iterated Prisoners Dilemma Players". In: (2014) (cit. on p. 8).

[40] W. Slany and W. Kienreich. "On some winning strategies for the iterated prisoners dilemma". In: *The iterated prisoners dilemma* (2007), pp. 171–204 (cit. on p. 1).

[41] D. W. Stephens, C. M. McLinn, and J. R. Stevens. "Discounting and reciprocity in an Iterated Prisoner's Dilemma." In: *Science (New York, N.Y.)* 298.5601 (2002), pp. 2216–2218. ISSN: 00368075. DOI: 10.1126/science.1078498 (cit. on pp. 2, 3).

[42] 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 pp. 1–4, 7, 8).