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

Owen Campbell Marc Harper Vincent Knight Karol Langner

January 30, 2016

1 Introduction

Several iterated prisoner's dilemma tournaments have generated much interest, including Axelrod's original tournaments, two 2004 anniversary tournaments, and the Stewart and Plotkin 2012 tournament 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, 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 [29] being a notable counter-example. In some cases strategies are revised without updates to their names or published implementations [19, 20]. As a result, some of the results related to these strategies and tournaments have not been reliably replicated, and have not met the basic scientific criterion of falsifiability.

This paper introduces a software package: the Axelrod python library [32]. 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

This library is motivated by an ongoing discussion in the academic community about reproducible research [8, 14, 27, 28], 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 99.59% coverage
- Well-documented: all features of the library are documented for ease of use and modification
- Extensive: over 100 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.

1.1 Review of the literature

As stated in [5]: "*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: [1, 2] which described a computer tournament that has been at the origin of a majority of game theoretic work [4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 17, 18, 21, 23, 24, 25, 26, 30, 31]. As described in [5] this work has not only had mathematical impact but has also led to insights in biology (for example in [30], a real tournament where Blue Jays are the participants is described) and in particular to the study of evolution.

The tournament is based on an iterated game (see [22] 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 [12] 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**punishment);
- 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).

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

The game of (1) is called the Prisoner's Dilemma. Numerical values of $(R, S, T, P) = (3, 0, 5, 1)$ are often used in the literature. Axelrod's tournaments (and further implementations of these) are sometimes referred to as Iterated Prisoner's Dilemma (IPD) tournaments. An overview of published tournaments is given in Table 1.

Year	Reference	Number of Strategies	Type	Source Code
1979	[1]	13	Standard	Not immediately available
1979	[2]	64	Standard	Available in FORTRAN
1991	[5]	13	Noisy	Not immediately available
2002	[30]	16	Wildlife	Not applicable
2005	[16]	223	Varied	Not available
2012	[31]	13	Standard	Not fully available

Table 1: An overview of published tournaments

[23] describes how incomplete information can be used to enhance cooperation, in a similar approach to the proof of the Folk theorem for repeated games [22]. This aspect of incomplete information is also considered in [24, 5, 18] where "noisy" tournaments randomly flip the choice made by a given strategy. In [25], 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: [11, 18, 26, 31] consider this in a traditional evolutionary game theory context. These works investigate particular evolutionary contexts within which cooperation can emerge as stable. Often these works consider direct opposition to another strategy and disregard the population dynamics, for example in [18] a simple machine learning algorithm outperforms a strategy found in [26].

Further to these evolutionary ideas, [7, 9] are examples of using machine learning techniques to evolve particular strategies. In [3], 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 [32] now boasts over 90 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 (`http://scikit-learn.org/`);
- Visualisation (`http://matplotlib.org/`);
- Mathematics (`http://www.sagemath.org/`);
- Astrophysics (`http://www.astropy.org/`);
- Data manipulation (`http://pandas.pydata.org/`);
- Algorithmic Game Theory (`http://gambit.sourceforge.net/)`).

Furthermore, [15] 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 popular version control system which is one of the recommended aspects of reproducible research [8, 28].

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 2.

```
>>> 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 2.

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

- Noisy tournaments
- Ecological tournaments
- Full tournament history
- Creation of plots of distributions of scores and wins

As stated in Section 1 one of the main goals of the library is to allow for the easy contribution of strategies. To do this requires the writing of a simple python class (which can inherit from other predefined classes).

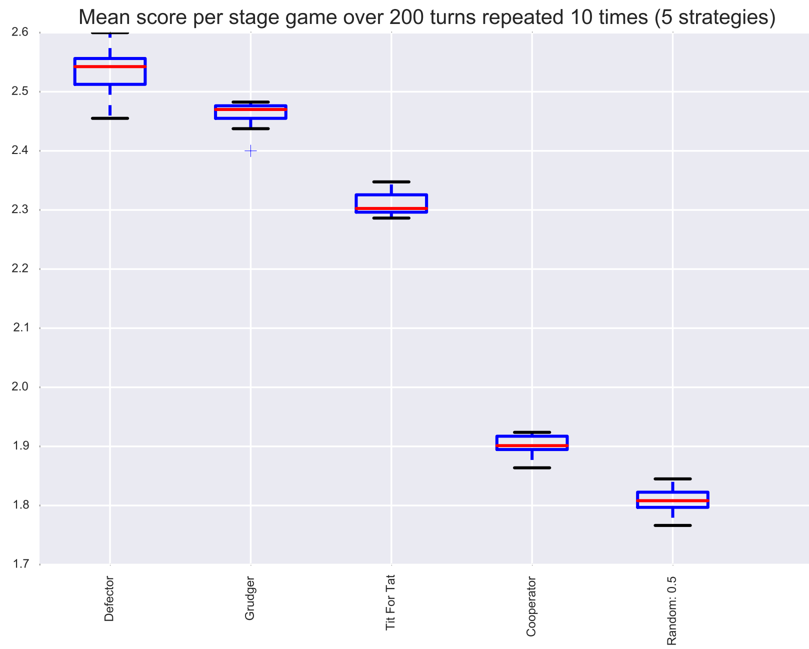

Figure 2: The results from a simple tournament.

Full contribution guidelines can be found in the documentation. Figures 3 and 4 show the source code for the Grudger strategy as well as its corresponding test.

```
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.

You can see an overview of some of the source code in Figure 5.

```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.

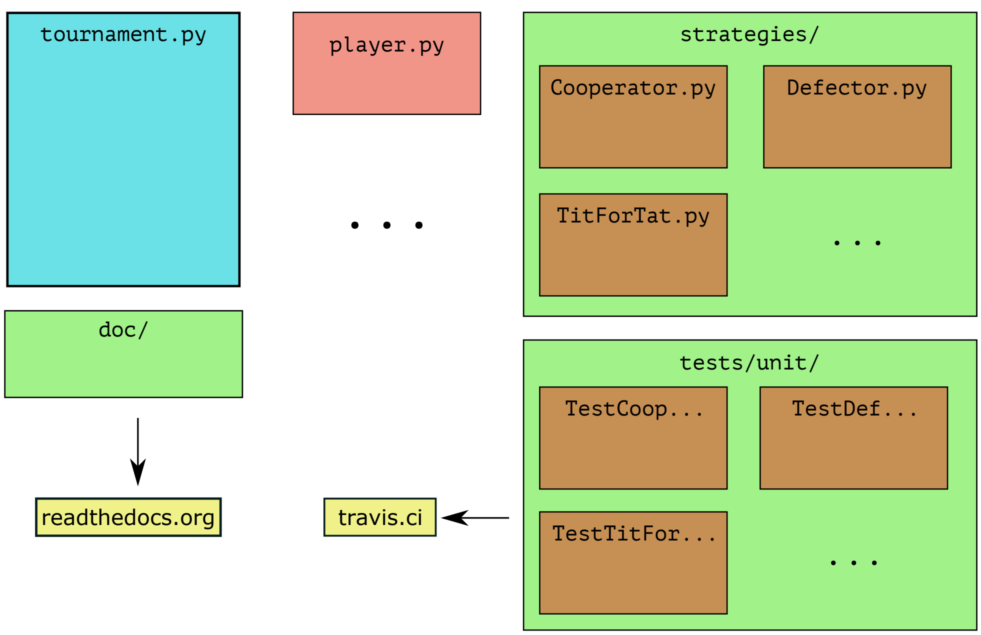

Figure 5: An overview of the source code.

To date the library has had contributions from 19 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.

Before discussing the novel insights obtained from the study of this library in Section 3 an overview of some tournaments that have been reproduced will be given in Section 2.

2 Reproducing previous tournaments

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 6. Despite this, due to previous research being done in an irreproducible way due to, for example no source code, and/or vaguely described strategies; not all previous tournaments are yet to be reproduced.

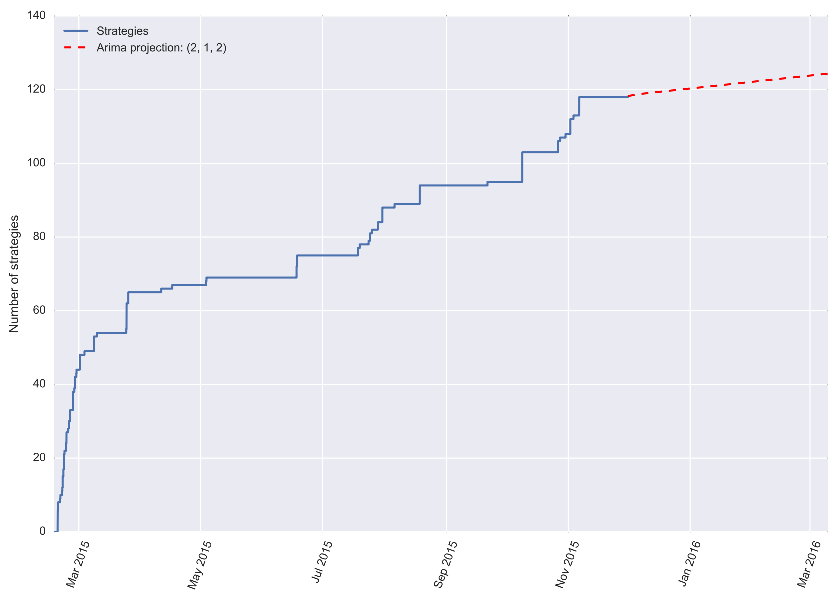

Figure 6: The number of strategies included in the library

One tournament that is possible to reproduce is that of [31]. The strategies used in that tournament are the following:

1. Cooperator
2. Defector
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 7, which gives the plot of Figure 8.

```
>>> 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 7: Source code for reproducing the tournament of [31]

Work is actively ongoing to include more strategies. In the next Section, an overview of some of the early and novel research insights made possible by the library.

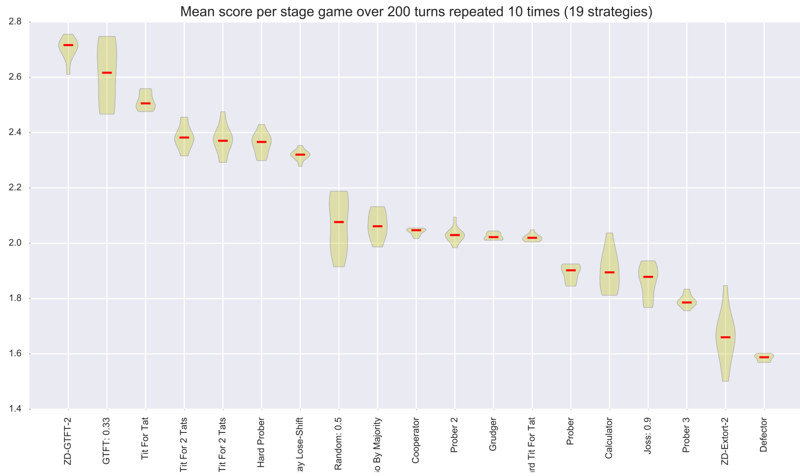

Figure 8: The results from [31].

3 New strategies, tournaments and implications

In parallel to the Python library, a tournament is being kept up to date that pits all strategies against each other. Figure 9 shows the results from the full tournament.

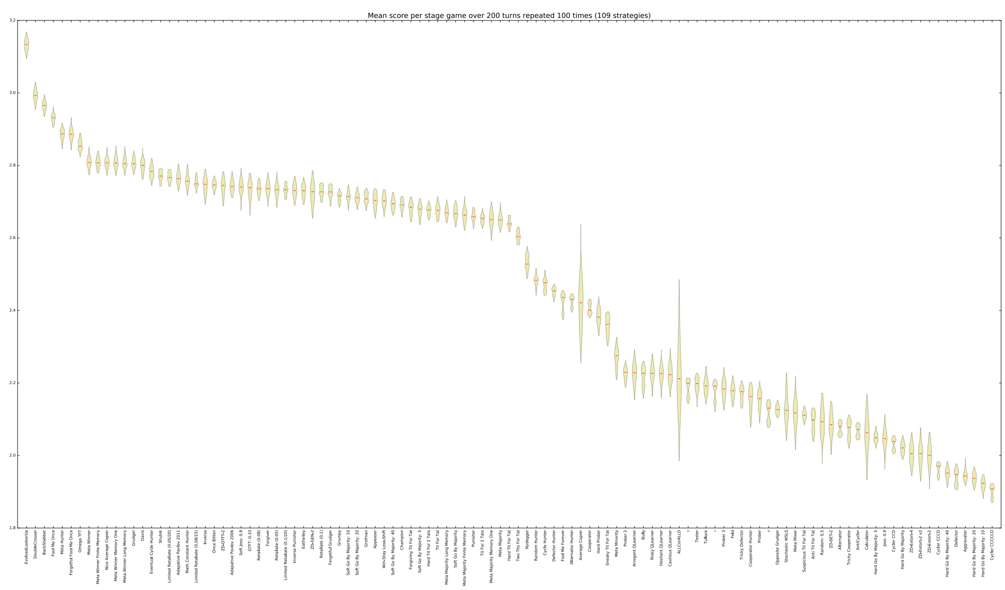

Figure 9: Results from the library tournament (2015-12-19)

The current winning strategy is a novel strategy of type: **LookerUp**. 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 in the following way:

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

The above example 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 the intentions of it whilst still taking in to account the recent play. The actual winning strategy an instance of the this framework for $m = n = 2$ for which an evolutionary algorithm has been used to train it. More details of this can be found in [**mojones**].

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 [31] in which zero determinant strategies are shown to be able to perform better than a given 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 and Fool Me Once. These strategies continue to perform well in noisy tournaments.
- Strategies like LookerUp and MetaHunter 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.

4 Conclusion

References

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

[2] 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 p. 2).

[3] R. Axelrod. *The Evolution of Cooperation* (cit. on p. 2).

[4] 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).

[5] 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 p. 2).

[6] 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).

[7] 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 p. 2).

[8] 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).

[9] 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 p. 2).

[10] 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).

[11] 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).

[12] 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).

[13] 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).

[14] 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).

[15] a.G. 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).

[16] G. Kendall, X. Yao, and S. Y. Chong. *The iterated prisoners’ dilemma: 20 years on.* World Scientific Publishing Co., Inc., 2007 (cit. on p. 2).

[17] 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).

[18] 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 p. 2).

[19] 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).

[20] 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).

[21] 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).

[22] 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).

[23] 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).

[24] 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).

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

[26] 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 p. 2).

[27] 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).

[28] 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).

[29] 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).

[30] 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 p. 2).

[31] 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. 2, 6–9).

[32] A.-P. project team. *Axelrod-Python v0.0.15*. 2015. URL: `http://axelrod.readthedocs.org/` (visited on 06/30/2015) (cit. on pp. 1, 3).