An exploration of cooperation and defection in iterated Prisoner's Dilemma

Afsana Ferdous, Roni Herschmann, Andrew Villeneuve

Abstract

This report seeks to analyze the Prisoner's Dilemma with variables applied independently to determine using various principles of linear algebra point of stability, instability, and dominant strategies. Furthermore, we intend to analyze how these same principles may be proven through real world comparisons with game theory.

Introduction:

The Prisoner's Dilemma is a useful tool for understanding game theory, as it represents a simple example of the push and pull between cooperation and competition. The setup for the game is simple, two prisoners with no contact with each other are given the opportunity to either cooperate and both confess to a crime for a greatly commuted sentence, defect and get a lower sentence, or one can defect while the other cooperates, but the one who defects gets the lowest sentence of all while the other player gets a harsher sentence (Le & Boyd, 2007). In the case of the standard Prisoner's Dilemma, the best option is for both prisoners to cooperate, this will give both prisoners the least amount of prison time (Nalebuff et al., 2018). However, cooperation is a state of instability (Arigapudi et al., 2021). This may seem counterintuitive at first, but if we look at a table representing the game:

Outcomes in years	Prisoner 1 Cooperates	Prisoner 1 Defects
Prisoner 2 cooperates	(1, 1)	(5, 0)
Prisoner 2 Defects	(0, 5)	(3, 3)

cooperation does in fact net the lowest sentencing but it is not guaranteed. It is unstable because it requires the other player to also cooperate to achieve a better outcome. This makes defection the true Nash Equilibrium, where you get the best deal regardless of the opponent's decision. However, does defection remain the best strategy in an Iterated Prisoner's Dilemma? In an Iterated Prisoner's dilemma, two players earn points based on the following scenarios.

Points	Player 1 Cooperates	Player 1 Defects
Player 2 cooperates	(3, 3)	(5, 0)

Player 2 Defects	(0, 5)	(1, 1)

All the points earned by the players are added up at the end of the game and the player with the most points wins. Here, defection becomes an unstable strategy as it introduces retaliation that may lead to suboptimal results.

Through simulations of Iterated Prisoner's Dilemma with multiple strategies, we seek to identify the nash equilibrium, and qualities of winning strategies.

Methodology:

2.1: Iterated Prisoner's Dilemma Program

In order to create the simulation of Iterated Prisoner's Dilemma, we used the Axelrod library in Python. It is a library containing many strategies used in Axelrod's first and second tournaments of the Iterated Prisoner's Dilemma game. Axelrod is a political scientist who ran a nationwide tournament to discover the best strategies in Iterated Prisoner's Dilemma (Axelrod, 2006). These tournaments were at the basis of how we ran these simulations and the personalities we chose. For our first three rounds, we chose five very basic personalities to see how they interact with each other.

The Cooperator: It will always cooperate.
The Alternator: It starts by cooperating and will alternate every turn.
The Tit for Tat: It starts by cooperating and will reciprocate the opponents move every round.
The Defector: It will always defect.
The Grudger: It will cooperate until the opponent's first defection. It will defect every round after.

We will analyze the outcomes of these tournaments to understand Axelrod's tournaments and the landslide victory of Tit for Tat. We are also interested in learning how the dominant strategy changes due to the numerous iterations, and the tournament environment. Axelrod describes that Tit for Tat isn't the best strategy overall, but a good strategy must have the following qualities: niceness (do not defect first), provocabilty (reciprocate defection timely), forgiveness (return to cooperation) (Axelrod, 2006). We chose to use three different strategies for the fourth round that plays around with this strategy.

The Two Tits for Tat: Starts by cooperating and reciprocates opponent's every defection with two defections.
The Tit for Two Tats: Starts by cooperating and reciprocates opponent's every two defections with a defection.
The Adaptive: Starts by cooperating, tests defection, and continues to choose the action with the best payoff.

Each of this strategy plays with Axelrod's definition of best strategy and is very reactive to the opponent. This tests Axelrod's theory and the quality that emerges most important in our tournament environment.

2.2: Data

With the Prisoner's Dilemma program, simulations could begin to be run with the outlined variables. The simulations were run in three different amounts, twenty five, fifty, and one thousand games. These numbers of games were chosen as they would provide enough data to be statistically significant.

2.3: Algorithms and simulation results

Payoff Matrices in NumPy and Game Theory Strategic Games Implementation

The algorithm was created to analyze the Prisoner's Dilemma, exploring various strategies. The variety of strategies ultimately allowed for a more broad examination of how different strategies can adapt over more game iterations.

Match 1: Strategies in play: Cooperator, Alternator, Tit for Tat, Defector, Grudger

```
[2]: #Three Different Prisoners Dilemma Tournaments
players =  (axl.Cooperator(), axl.Alternator(), axl.TitForTat(), axl.Defector(), axl.Grudger())
matchSetGame = axl.Tournament(players=players, turns=50)
results =  matchSetGame.play()

Playing matches: 100%|██████████| 15/15 [00:00<00:00, 451.06it/s]
Analysing: 100%|██████████| 25/25 [00:00<00:00, 203.25it/s]

[3]: #The Payoff Matrix
payoff_matrix = np.array(results.payoff_matrix)
print("Payoffs:")
print(payoff_matrix)

Payoffs:
[[3.   1.5  3.   0.   3.  ]
 [4.   2.   2.56 0.5  0.64]
 [3.   2.46 3.   0.98 3.  ]
 [5.   3.   1.08 1.   1.08]
 [3.   2.94 3.   0.98 3.  ]]
```

The first match includes all strategies, representing the most justified prisoner (Cooperator) against its polar opposite (Defector). Moreover, including the Alternator and Grudger allowed for another level of strategic complexity alongside conditional responses. This setup allowed us to simulate an environment incorporating extreme/balanced strategies, allowing for a deeper analysis of the most adaptable and successful strategy in the Prisoner's Dilemma.

The payoff matrix represents the average score of each strategy against other strategies. For example, (1,1) represents the average score of the Cooperator against the Cooperator, (2,4) represents the average score of the Alternator against the Defector.

Match 2: Strategies in play: Cooperator, Alternator

```
[4]: #Three Different Prisoners Dilemma Tournaments
players =  (axl.Cooperator(), axl.Alternator())
matchSetGameTwo = axl.Tournament(players=players, turns=25)
results =  matchSetGameTwo.play()

Playing matches: 100%|████████████████████████| 3/3 [00:00<00:00, 560.44it/s]
Analysing: 100%|████████████████████████| 25/25 [00:00<00:00, 433.58it/s]

[5]: #The Payoff Matrix
payoff_matrix = np.array(results.payoff_matrix)
print("Payoffs:")
print(payoff_matrix)

Payoffs:
[[3.   1.56]
 [3.96 2.04]]
```

This match represented a more straightforward scenario than the first match, which was created to analyze the outcomes of two highly predictable behaviors within the scope of the prisoner's dilemma. The Cooperator and Alternator do not have reactive mechanics, so this was a baseline for understanding the predictability of the base strategies without interference from exploitative strategies that tend to benefit when this behavior is placed alongside them.

Match 3: Strategies in play: Tit for Tat, Defector, Grudger

```
[28] #Three Different Prisoners Dilemma Tournaments
     players =  (axl.TitForTat(), axl.Defector(), axl.Grudger())
     matchSetGameThree = axl.Tournament(players=players, turns=25)
     results =  matchSetGameThree.play()

     Playing matches: 100%|████████| 6/6 [00:00<00:00, 45.93it/s]
     Analysing: 100%|████████| 25/25 [00:00<00:00, 41.71it/s]

[29] #The Payoff Matrix
     payoff_matrix = np.array(results.payoff_matrix)
     print("Payoffs:")
     print(payoff_matrix)

     Payoffs:
     [[3.   0.96 3.  ]
      [1.16 1.   1.16]
      [3.   0.96 3.  ]]

[30] results.wins

     [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

In the third match, the algorithm focuses on the strategies that had the most reactive elements. The Grudger and Tit for Tat strategies both utilize retaliation and forgiveness, as opposed to the Defector, which exploits others with no desire to cooperate. This match allowed us to explore how two strategies, conditional on their surroundings, can deal with a selfish approach to the match.

The variation in matches played allowed us to understand better how performance-based strategies evolved. The longer matches stabilized patterns, which allowed the cooperative strategies to build trust, while the shorter matches rewarded dishonesty and selfish approaches.

The wins matrix represents the number of games won by each strategy. This matrix reveals that the Defector won against both the Cooperator and the Grudger every round.

Nash Equilibrium Code:

Utilizing the Nash equilibrium to deduce quantitative analysis

Following the implementation of each match, the algorithm utilizes the Nashpy import to compute the Nash equilibrium for each respective match played. The use of Nash equilibrium in the simulations provided above allowed for a basis for analyzing the strategies deployed in each match of the Prisoner's Dilemma. Identifying equilibria for each match allowed for the evaluation of the strategic yield that resulted in the most consistent outcomes.

```
[19]: #Nashpy for Nash Equilibrium
playerOnePayoff = payoff_matrix[:2,:2]
playerTwoPayoff = payoff_matrix[:2, :2]
nashTest = nash.Game(playerOnePayoff, playerTwoPayoff)
print("Nash Equilibrium")
for eq in nashTest.support_enumeration():
    print(eq)

Nash Equilibrium
(array([1., 0.]), array([1., 0.]))
```

The nash equilibrium matrix identifies the winning strategy for the players.

Data Visualization and Analysis of Strategy Trends Over Games:

Analyzing the most successful Payoff in correlation to Prisoner-Strategy

Using the Seaborn Python library, the algorithm generates a graph outlining the average payoff per strategy over the games played. The graph shows that the cooperative strategy fell short in comparison to some of its competitors, as the game rewarded dishonesty.

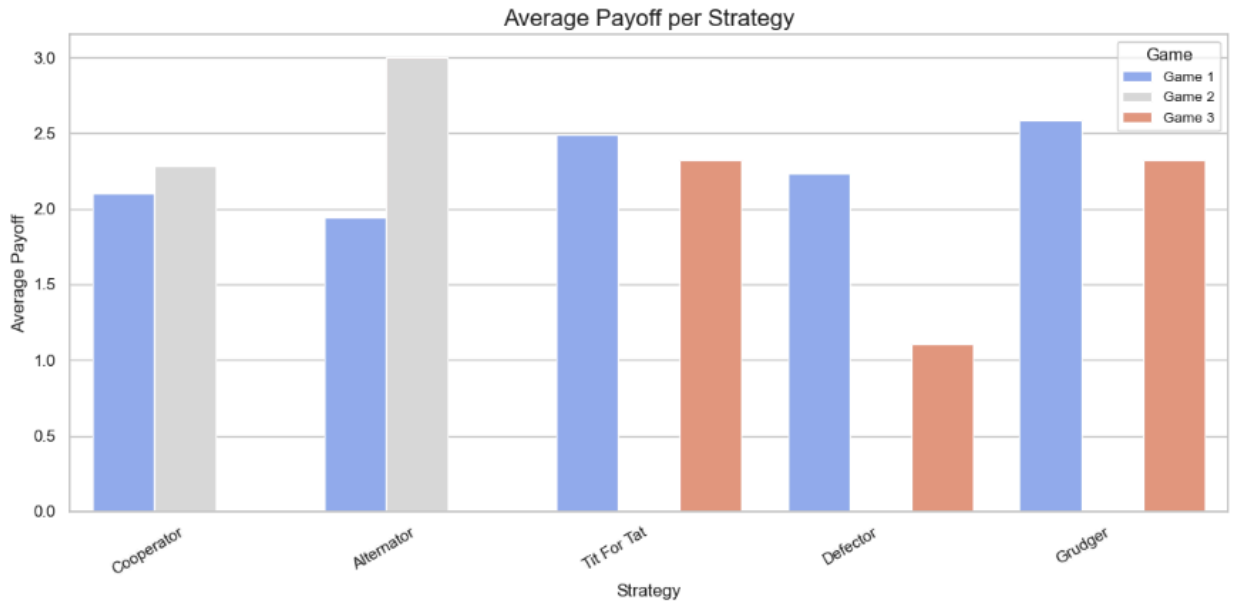

Figure 1. The average payoff of each strategy in the first three tournaments

Deep Dive into the Visualization of the Data

The graph represents the average payoffs of strategies across all the matches. There are four main aspects to this that we noticed in our analysis:

1. Trends in Payoffs
    a. On the graph, the defector received higher payoffs in the early rounds, though through the progression of each game, the average payoffs in strategies Tit for Tat and Grudger increased, beating the defectors. This showcases that cooperation and retaliation paid off more over time.
2. Defector's Dominance
    a. The defector's payoff advantage starkly increases early in games. This strategy is more dominant the shorter the game because it can exploit others for its own gain.
3. Stability Over Time
    a. Payoffs that incorporate strategies that tend to cooperate/retaliate have a stable equilibrium over time. Constant exploitation leads to more retaliation, which reduces the payoffs for defectors in longer matches.

Match 4: Strategies in Play: Two Tits for Tat, Tit for Two Tats, Adapter

```
[26] players =  (axl.TwoTitsForTat(), axl.TitFor2Tats(), axl.Adaptive())
     matchSetGameFour = axl.Tournament(players=players, turns=1000)
     results1 =  matchSetGameFour.play()
```

```
Playing matches: 100%|██████████| 6/6 [00:03<00:00,  1.90it/s]
Analysing: 100%|██████████| 25/25 [00:00<00:00, 116.60it/s]
```

```
[27] #The Payoff Matrix
     payoff_matrix = np.array(results1.payoff_matrix)
     print("Payoffs:")
     print(payoff_matrix)
```

```
Payoffs:
[[3.    3.    2.993]
 [3.    3.    2.99 ]
 [2.988 2.995 2.99 ]]
```

```
[5]  results1.normalised_state_to_action_distribution
```

```
[[Counter(),
  Counter({(((C, C), C): 1.0}),
  Counter({(((C, C), C): 1.0,
            ((C, D), D): 1.0,
            ((D, C), C): 0.5,
            ((D, C), D): 0.5,
            ((D, D), D): 1.0})],
 [Counter({(((C, C), C): 1.0}),
  Counter(),
  Counter({(((C, C), C): 1.0,
            ((C, D), C): 0.5,
            ((C, D), D): 0.5,
            ((D, C), C): 1.0,
            ((D, D), D): 1.0})],
 [Counter({(((C, C), C): 0.998991935483871,
            ((C, C), D): 0.0010080645161290322,
            ((C, D), C): 1.0,
            ((D, C), D): 1.0,
            ((D, D), C): 0.25,
            ((D, D), D): 0.75}),
  Counter({(((C, C), C): 0.998992950654582,
            ((C, C), D): 0.0010070493454179255,
            ((C, D), C): 1.0,
            ((D, C), D): 1.0,
            ((D, D), C): 0.3333333333333333,
            ((D, D), D): 0.6666666666666666}),
  Counter()]]
```

```
[18] payoff_matrix_two_tits = np.array([[1, 0], [0.5, 0]])
     payoff_matrix_two_tats = np.array([[1, 0.5], [1, 0]])
     payoff_matrix_adaptive = np.array([[.99, 1], [0, .25]])

     nashTest = nash.Game(payoff_matrix_two_tits, payoff_matrix_two_tats)
     print("Nash Equilibrium!")
     for eq in nashTest.support_enumeration():
         print(eq)

Nash Equilibria!
(array([1., 0.]), array([1., 0.]))
```

```
[32] nashTest1 = nash.Game(payoff_matrix_adaptive, payoff_matrix_two_tits)
     print("Nash Equilibria!")
     for eq in nashTest1.support_enumeration():
         print(eq)

Nash Equilibria!
(array([1., 0.]), array([1., 0.]))
```

```
[20] nashTest2 = nash.Game(payoff_matrix_two_tats, payoff_matrix_adaptive)
     print("Nash Equilibrium!")
     for eq in nashTest2.support_enumeration():
         print(eq)

Nash Equilibria!
(array([1., 0.]), array([0., 1.]))
```

This match allowed us to explore Axelrod's understanding of a good strategy. By using highly reactive strategies, we compared which aspects of a win strategy lead to better results. We put forgiveness (Tit for Two Tats), reciprocation (Two Tits for Tat) and provocation (Adaptive) into the test to better understand how they might play against each other.

The normalised state to action distribution matrix identifies how each strategy reacts to the previous round. For example, (1, 3) shows the probability of Two Tits for Tat reaction to the previous round with Adaptive. This matrix allowed us to manipulate the data to create our own payoff matrix to find nash equilibrium for each two competing strategies.

3: Analysis

Match 1: Strategies in play: Cooperator, Alternator, Tit for Tat, Defector, Grudger

This reflected the strength of defection. The Cooperator and Alternator failed to retaliate due to their strategies, allowing the Defector to exploit and gain higher payoffs. This highlights the Cooperator's inability to thrive in an environment that lacks mutual trust.

Match 2: Strategies in play: Cooperator, Alternator

Similar to what is discussed above, neither of these strategies involves retaliation or adaptation, so the scenario is straightforward to analyze. The payoffs alternate based on the Alternate's behavior. The Alternator can gain higher payoffs, while the Cooperator has lower payoffs due to the inherent cost of constant cooperation.

Match 3: Strategies in play: Tit for Tat, Defector, Grudger

The equilibrium shifted as retaliation was introduced. This discourages defection through the overall decreasing payoff. The cooperation resulted in high-payoff equilibria where there were repeating interactions.

Match 4: Strategies in play: Two Tits for Tat, Tit for Two Tats, Adaptive

The Nash equilibrium reveals the weakness in Tit for Two Tats where it is easy to take advantage of its likeliness to cooperate. The win of Two Tits for Tat suggests that reciprocation is the strongest aspect of a winning strategy, while forgiveness is the weakest.

While Axelrod tournaments revealed that Tit for Tat was the best strategy, our simulations reveal a weakness in it. Since Tit for Tat is only reactionary, it can not take advantage of strategies that are more likely to cooperate. For example, in our first tournament, the winner is the Defector. Even as we repeat this tournament 10 times, the winner remains the Grudger. If we look at the payoff matrix for tournament 1, we can see that the total payoff for the Defector is higher than every other strategy.

$$\begin{bmatrix} 3 & 1.5 & 3 & 0 & 3 \\ 4 & 2 & 2.56 & 0.5 & 0.64 \\ 3 & 2.46 & 3 & 0.98 & 3 \\ 5 & 3 & 1.08 & 1 & 1.08 \\ 3 & 2.94 & 3 & 0.98 & 3 \end{bmatrix}$$

The payoff of the Defector (fourth row) added together is 11.16, higher than other rows

This is due to the fact that defecting is the dominant strategy in a single round game. Unlike all the other strategies, the Defector does not play nice and therefore, takes advantage of all of the other strategies' initial cooperation. Additionally, what makes the defector the strongest strategy in this tournament is it competes against strategies that can be taken advantage of such as the Cooperator. Since the Cooperator will always cooperate regardless of the actions of its opponent, the Defectors continuous defection is the winning strategy against it.

The dominant strategy of defection continues in the second round. The Alternator defects and takes advantage of the Cooperator to come out with the highest pay off. This shows that constant cooperation is the weakest strategy. Since it isn't the dominant strategy in a single round game, it continues to fail in iterated prisoner's dilemma.

Surprisingly, in the third tournament, the Defector came in last. The Grudger and Tit for Tat were able to increase their payoff through mutual cooperation. As we can see from the matrix of wins, the Defector has continuously won against both strategies. However, the key to iterated prisoner's dilemma is not to win every match, but rather to maximize one's payoff.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The Defector (second row) wins every round against the other two strategies

The Grudger and Tit for Tat are tied in this tournament. Since neither strategy deflected first, their highest payoff is 3. In this tournament, since there were more nice strategies, they were easily able to win.

In the fourth tournament, we use our findings in the previous tournaments to better understand the Axelrod tournaments and its results. Based on the three qualities of a good strategy by Axelrod, we picked Two Tits for Tat, Tit for Two Tats, and Adaptive. Two Tits for Tat does not forgive easily, choosing to punish the opponent twice for each time they defect. Tat for Two Tits doesn't reciprocate defections enough, choosing to defect once for every two times the opponent defects. Finally, the Adaptive is a mostly nice strategy, but chooses to start conflicts randomly to test the probability of a higher payoff from defection.

Two Tits for Tat won the fourth tournament. From our previous simulations, this is unsurprising as Tit for Two Tats and Adaptive are nice strategies, so defection by Two Tit for Tats emerge as the dominant strategy. In this tournament, we analyzed the consequences of choosing to cooperate or defect with each strategy. The normalized state to action distribution matrix analyzes how each strategy reacts to the previous turn. Therefore, we can analyze the payoff matrix through each action pattern. The Nash Equilibrium of playing with each strategy revealed that while the best strategy is to cooperate with Two Tits for Tat and Adaptive, it is better to defect with Tit for Two Tats. Since Tit for Two Tats does not reciprocate every defection, it can increase payoff. This can clearly seen in the payoff for Adaptive against Tit for Two Tats being the highest amongst all three.

However, Adaptive had the lowest payoff out of all three strategies. This emphasizes playing nice in a tournament of nice strategies. Through the comparison between the three qualities of good strategies, reciprocation comes out top and provocation comes out last. This re-emphasizes the power of cooperation.

4: Real-world Comparison of data

As the Prisoner's Dilemma is merely a conceptualization of basic principles of game theory, it would be prudent to also look at real examples of what the Prisoner's Dilemma represents to see how the data found from the simulations compares to that of the real world. In essence, since the data shows that, no matter what additional variables applied, defection will remain the dominant strategy, exploring if this also holds true in the practical application of the Prisoner's Dilemma is a worthwhile exercise.

The iterated Prisoner's Dilemma is interesting, as when placed into the real world, facets of interaction begin to emerge that our simulations cannot model, like reputation and trust. The iterated Prisoner's Dilemma is incredibly common, in a normal week, many people are met with decisions that can inconvenience themselves or others in some ways. These are decisions we tend to make flippantly, as the outcomes are things we do not have to feel. However, there are also common situations that more accurately resemble an iterated Prisoner's Dilemma, where you must continue interactions and thus feel the consequences of your outcomes. This shift in the dynamic of having to deal with the fallout of your actions can shift one's thinking on how they

interact with a situation. (Geher, 2021) Take, for example, you work on a project at a company with a team. Everyone has their assignments and is doing their best to meet deadlines. You have the choice to continue working with them and towards your joint goal, or alternatively, you could reason out that the project has to be completed by a certain deadline, and if you do not do your part, someone will pick up your slack to finish the job. Now, in this case, defecting means less work, but also a worsening of relationships with your colleagues and bosses. You could also gain a reputation for not meeting deadlines which could inhibit your chances of promotion or finding other jobs. So, in the iterated game of the Prisoner's Dilemma, it is not necessarily the dominant strategy to defect, as you will face the consequences of your actions. This does not hold true to our data, as it is difficult to represent such elements within the confinements of the simulations, that being said, there are many other examples of iterated Prisoner's Dilemmas that do lead towards defecting.

5: Conclusion

In conclusion, even with variables applied to the Prisoner's Dilemma, defecting remains the strongest strategy. There are additional complexities added by the variables, however, even with those complexities, the Nash Equilibrium, eigenvalue and eigenvector analysis, and payoff matrices all point towards the strength of the defecting strategy. While there are noteworthy examples of shortcomings of this research when applied to the real world, more often than not it remains the dominant strategy.

6: Improvements

While the simulations and the math presented here are strong, there are a few shortcomings of the simulations that we ran that could be tweaked with further work. Firstly, finding a way to more adequately represent retaliatory gameplay within the program could be an interesting dynamic to see represented and how that would change outcomes. Secondly, incorporating a person to play against computer opponents to see how the interaction between set personalities, and a personality that can change in a moment could net different data that would be useful to analyze further to see if we could model the real world dynamics of iterated Prisoner's Dilemma. Finally, running the simulations again, but strictly with people. This is potentially the most interesting, as people are not coded to act certain ways, and could lead to interesting dynamics to be analyzed. This could go further, with having players separated at first, but then letting them see, but not interact with one another, thus increasing the connection without spoiling the decisions of other players. This seems the most interesting, as people tend to act better when they have to continually interact with the same players repeatedly. (Geher, 2021)

References

Arigapudi, S., Heller, Y., & Milchtaich, I. (2021). Instability of defection in the prisoner's dilemma under best experienced payoff dynamics. *Journal of Economic Theory*, *197*, 105174. https://doi.org/10.1016/j.jet.2020.105174

Axelrod, R. M. (2006). *The Evolution of Cooperation*. Basic Books.

Geher, G. (2021, December 14). *The prisoner's dilemma in everyday life*. Psychology Today. https://www.psychologytoday.com/us/blog/darwins-subterranean-world/202112/the-prisoners-dilemma-in-everyday-life

Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007, June 18). *Matplotlib: A 2D graphics environment | IEEE Journals & Magazine | IEEE Xplore*. IEEExplore. https://ieeexplore.ieee.org/document/4160265/

Knight, V., Campbell, O., Marc, Gaffney, T. J., Shaw, E., Reddy Janga, V., Glynatsi, N., Campbell, J., Langner, K. M., Singh, S., Rymer, J., Campbell, T., Young, J., Hakem, M., Palmer, G., Glass, K., Mancia, D., Argenson, E., Martin, J., … Areeb, A. (2023). Axelrod-Python/Axelrod: v4.12.0 (v4.13.0). Zenodo. https://doi.org/10.5281/zenodo.7861907

Le, S., & Boyd, R. (2007). Evolutionary Dynamics of the continuous iterated prisoner's dilemma. *Journal of Theoretical Biology*, *245*(2), 258–267. https://doi.org/10.1016/j.jtbi.2006.09.016

Nalebuff, A. D. and B., Munger, M., Moore, G., Heyne, P., & Anderson, L. (2018, June 27). *Prisoners' dilemma*. Econlib. https://www.econlib.org/library/Enc/PrisonersDilemma.html

The pandas development team. (2024). pandas-dev/pandas: Pandas (v2.2.3). Zenodo. https://doi.org/10.5281/zenodo.13819579

Ralf Gommers, Pauli Virtanen, Matt Haberland, Evgeni Burovski, Warren Weckesser, Tyler Reddy, Travis E. Oliphant, David Cournapeau, Andrew Nelson, alexbrc, Pamphile Roy, Pearu Peterson, Ilhan Polat, Josh Wilson, endolith, Nikolay Mayorov, Stefan van der Walt, Matthew Brett, Denis Laxalde, … Kai Striega. (2024). scipy/scipy: SciPy 1.14.1 (v1.14.1). Zenodo. https://doi.org/10.5281/zenodo.13352243

Szilagyi, M. N. (2024). An investigation of n-person prisoners' dilemmas. *Complex Systems*, *14*(2), 155–174. https://doi.org/10.25088/complexsystems.14.2.155

Vince Knight. (2016). drvinceknight/Nashpy: v0.0.1 (v0.0.1). Zenodo. https://doi.org/10.5281/zenodo.164954

Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.