Working paper
XX(X):1–12
©The Author(s) 2024
DOI: To Be Assigned
DRAFT

IPDToolkit: An R package for simulation and Bayesian analysis of iterated prisoner's dilemma game-play under third-party arbitration

Cody T. Ross[1], Thomas Fikes[2], Hillary Lenfesty[3,4], Richard McElreath[1]

Abstract

Recently, researchers have begun studying the role that third-party arbitration may play in the evolution of cooperation. Using the iterated prisoner's dilemma (IPD), they show that arbitration can mitigate the negative effects of perception errors on the stability of cooperative strategies. Open questions, both theoretical and empirical, however, remain. To promote research on the role of third-party arbitration, we introduce an R package, `IPDToolkit`, which facilitates both simulation of synthetic data and Bayesian analysis of empirical data. To address theoretical questions, `IPDToolkit` provides a Monte Carlo simulation engine that can be used to generate play between arbitrary strategies in the IPD with arbitration and assess expected pay-offs. To address empirical questions, `IPDToolkit` provides customizable, Bayesian finite-mixture models that can be used to identify the strategies responsible for generating empirical game-play data. We present a complete workflow using `IPDToolkit` to teach end-users its functionality.

Keywords

prisoner's dilemma, arbitration, game theory, behavioral economics, Bayesian analysis

Recently, Boyd and Mathew (2021) introduced a theoretical examination of the use of third-party arbitration in the iterated prisoner's dilemma (IPD). They show that arbitration can mitigate the negative effects of perception errors on the stability of cooperative strategies. Perception errors normally create a serious obstacle to the maintenance of reciprocity, because they cause interacting individuals to disagree about past behavior and thus cause cooperation to unravel (Sugden et al. 2004; Boyd 1989; Boerlijst et al. 1997; Nowak and Sigmund 1990, 2005). Third-party arbitration provides a public signal that essentially converts perception errors into execution errors, which are more easily resolved (Sugden et al. 2004; Boyd 1989; Boerlijst et al. 1997).

Important theoretical and empirical questions about this topic remain open. For example, one may wish to examine how Arbitration-Tit-for-Tat (ATFT) fares against other arbitration-based strategy variants using Monte Carlo simulation. Or, perhaps more fundamentally, one may wish to identify which strategies are actually used by human players in experimental IPD setups when arbitration is possible. Here, we introduce an R package, `IPDToolkit`, that can help to address both kinds of questions. `IPDToolkit` provides a Monte Carlo simulation engine that can be used to simulate play between arbitrary strategies in the IPD with arbitration. More substantially, `IPDToolkit` provides customizable, Bayesian models that can be used to identify the strategies responsible for generating empirical game-play data. These functions can be used both to classify the strategies used by human players in experimental setups, and study how covariates influence strategy choice.

In analyzing IPD game-play data, researchers often aim to identify the probability that a given player, i, uses a given strategy, s—e.g., ATFT—out of a set of S possible strategies. They may even wish to investigate the effects of various individual-level predictor variables on the strategies that individual players chose to employ. These kinds of classification problems are commonly addressed using finite mixture models (McNicholas 2017; McLachlan et al. 2019). In such models, the probability of the observed data under each of a finite number of probability mass functions is calculated, and then used to estimate a vector of weights that gives the relative probability that each individual's sequence of data was generated by each specific strategy in the considered set (McNicholas 2017; Nasserinejad et al. 2017). In this paper, we provide a Bayesian implementation of such a finite mixture model specifically tailored to game-play data realized under the IPD where third-party arbitration is possible (Boyd and Mathew 2021).

We begin the paper by outlining the installation and set-up of `IPDToolkit`. We then introduce our framework for the forward simulation of data. We describe each key function, and provide example code. We also review the data structure, describe the set of included strategies, and teach

[1]Department of Human Behavior, Ecology, and Culture, Max Planck Institute for Evolutionary Anthropology. Leipzig, Germany. [2]EdPlus Action Lab, Arizona State University. [3]School of Human Evolution and Social Change, Arizona State University. [4]Institute of Human Origins, Arizona State University.

Corresponding author:
Cody T. Ross
Email: cody_ross@eva.mpg.de

users how additional strategy files can be integrated into our simulation engine. Next, we present the mathematical formalism describing our Bayesian finite mixture models. We then describe the R functions used to implement these models, and provide example code, analyses, and visualizations. We conclude the paper by discussing some of the inferential challenges involved in analysis of empirical IPD game-play data.

Installation and setup

The `IPDToolkit` package is accessible at: https://github.com/ctross/IPDToolkit. This page contains additional annotated R code and example workflows. Bug-reports, feature requests, and other relevant comments should be made through GitHub, where the package will be maintained.

Much of the functionality of `IPDToolkit` is made possible by R (R Core Team 2019) and the `rstan` package (Stan Development Team 2019) and its dependencies. The user must install these programs in order to use our software. Installation and loading of `IPDToolkit` is then simple. Just run three lines of code from R:

```
library(devtools)
install_github("ctross/IPDToolkit")
library(IPDToolkit)
```

Next, the user sets a path, and uses the `setup_folders` function to create a directory structure that will be used to add custom strategy functions and save results. This directory will be named "PrisonersDilema".

```
path = "C:/Maynard/Desktop"
setup_folders(path,import_code=TRUE,
   overwrite=FALSE)
```

The `import_code=TRUE` argument will clone the Stan code provided by `IPDToolkit` into the new directory; set `overwrite=TRUE` to overwrite any previously customized code. At this point, `IPDToolkit` has full functionality, but its set of strategy functions is rather limited. Additional strategy functions, however, can be easily supplied by the user, and integrated into our simulation and analysis engines.

Forward data simulation

Included strategy functions

In the forward data simulation code, we provide a suite of 12 strategies (see Table 1). By placing the names listed in the "Strategy name" column of Table 1 into any of the simulation functions, data will be simulated under that strategy.

Additional strategy functions used for forward data simulation must be written in R code, and should be saved in the `StrategiesR` subdirectory. Additional strategy functions used for data analysis must be written in Stan code, and should be saved in the `StrategiesStan` subdirectory. Once added to the directory, these new functions can be integrated into the R environment by running `integrate_new_functions`:

```
integrate_new_functions(path)
```

User-supplied functions must accept the same inputs as our standard functions, and must return output vectors of the same form as well. We provide a template strategy file in the supplementary materials. This strategy file, which we call GLUM, behaves opposite of TFT, cooperating only after its partner defects, and defecting after its partner cooperates.

Simulation functions

We provide two key functions for forward simulation: `simulate_sequence` and `simulate_round_robin`.

The first function, `simulate_sequence`, simulates an iterated prisoners dilemma game between two strategies:

```
simulate_sequence(n_rounds=40,
   strategies=c("ATFT","ATFT"),
   error_rate=0.05,
   arb_error_rate_type_1=0.1,
   arb_error_rate_type_2=0.1)
```

The argument `n_rounds` controls the length of the game. The argument `strategies` determines which two strategies will play; these names must match the names of valid R functions designed to implement the logic of a strategy. The argument `error_rate` controls the frequency of computer introduced errors that transform intended cooperation events by player i into apparent defections as observed by player j. The argument `arb_error_rate_type_1` controls the rate at which the arbitrator *fails to correctly classify errors* introduced by the computer. The argument `arb_error_rate_type_2` controls the rate at which the arbitrator *incorrectly classifies true defections as errors* introduced by the computer. The `simulate_sequence` function then returns a list of data vectors as described in the next sub-section.

The second function, `simulate_round_robin`, is a wrapper function for `simulate_sequence`, and simulates an iterated prisoners dilemma game between all pairwise combinations of a list of players:

```
simulate_round_robin(n_rounds=40,
   mode="pairwise", players=c("ATFT"
   ,"GLUM","TFTA","WSLS","ALLC","
   RANDY"),  error_rate=0.05,
   arb_error_rate_type_1=0.1,
   arb_error_rate_type_2=0.1,
   n_games=NULL, matchups=NULL)
```

The argument `players` determines which strategies will be included in the round-robin tournament. The argument `mode` determines how players will be paired. The default, "pairwise", creates a full round-robin tournament. Other options are: "random", which will create `n_games` matchups at random from the supplied set of players, and "specified", which allows the user to hard-code the matchups, by supplying an `n_games` by `2` matrix of such player ID pairings via the `matchups` argument. The `simulate_round_robin` function returns a relational data-base, which includes the list of data vectors as described in the next sub-section, along with game-specific, and player-specific data lists.

Table 1. The strategies considered in our model, and some of their properties. The algorithmic definitions of these functions are included in the supplementary Stan code.

Function ID	Strategy name	Deterministic moves	Sneaky	Arbitration-based	Standing-based	Citation
Λ_1	ALLD	✓				
Λ_2	ALLC	✓				
Λ_3	RANDY					
Λ_4	TFT	✓	✓			(Axelrod and Hamilton 1981)
Λ_5	TF2T	✓	✓			
Λ_6	GTFT		✓			(Molander 1985)
Λ_7	WSLS	✓	✓			(Nowak and Sigmund 1993)
Λ_8	TFTA	✓	✓	✓		
Λ_9	TF2TA	✓	✓	✓		
Λ_{10}	GTFTA		✓	✓		
Λ_{11}	WSLSA	✓	✓	✓		
Λ_{12}	ATFT	✓	✓	✓	✓	(Boyd and Mathew 2021)

Simulation visualization

The function `sequence_plot` is also a wrapper function for `simulate_sequence`, but is designed specifically to visualize move sequence data:

```
sequence_plot(n_rounds=20, focal="
    ATFT", partner="ATFT", seed
    =1234,  error_rate=0.1,
    arb_error_rate_type_1=0.5,
    arb_error_rate_type_2=0.5)
```

The arguments `focal` and `partner` determine which two strategies will play, with `focal` being the first to move. The `sequence_plot` function can be used to study the interaction dynamics between strategies visually. For example, Figure 1 shows how sensitive various strategy pairings are to computer introduced errors. Perception errors can mute cooperation between TFT players (McElreath and Boyd 2008), but ATFT players rapidly re-establish cooperation through the use of contrition (Boyd 1989) and third party arbitration to resolve perceived slights (Boyd and Mathew 2021). The `sequence_plot` function is also essential in debugging new strategy files, and validating that their simulation behavior works as expected.

Data structure

In the forward simulation model of the IPD with arbitration, we produce several key variables: (1) whether the arbitrator was called at the start of the half-round, $A \in \{0, 1\}$, which takes a value of 1 if the arbitrator was called and a value of 0 otherwise; (2) the arbitrator's ruling, $E \in \{0, 1\}$, which takes a value of 1 if the arbitrator was called *and* ruled that a defection was introduced by the computer, but a value of 0 otherwise; (3) the focal player's intended move, $\bar{Y} \in \{0, 1\}$, where cooperation is a 1 and defection is a 0; (4) the focal player's observable move (after the computer may have changed $1 \to 0$), $Y \in \{0, 1\}$; (5) the identity of the focal player in each half-round, $I \in \mathbb{N}$; (6) the half-round time-step identifier, $T \in \mathbb{N}$; and, finally, (7) the game identifier, $G \in \mathbb{N}$.

These variables are represented as vectors in long-form: Eqs. 1–7 provide an example of the data structure.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & \dots \end{bmatrix} \quad (1)$$

$$E = \begin{bmatrix} 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & \dots \end{bmatrix} \quad (2)$$

$$\bar{Y} = \begin{bmatrix} 1 & 1 & 0 & 1 & \dots & 1 & 1 & 0 & 1 & \dots \end{bmatrix} \quad (3)$$

$$Y = \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 1 & 1 & 0 & 0 & \dots \end{bmatrix} \quad (4)$$

$$I = \begin{bmatrix} 1 & 2 & 1 & 2 & \dots & 7 & 8 & 7 & 8 & \dots \end{bmatrix} \quad (5)$$

$$T = \begin{bmatrix} 1 & 2 & 3 & 4 & \dots & 1 & 2 & 3 & 4 & \dots \end{bmatrix} \quad (6)$$

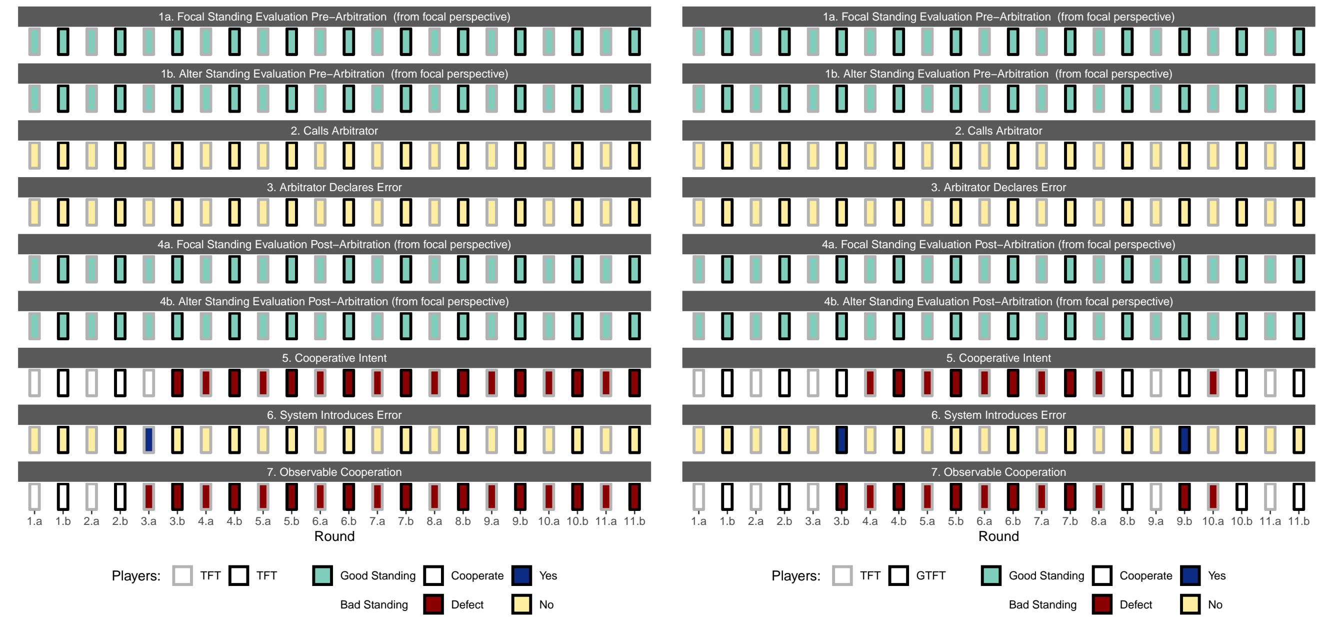
$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 4 & 4 & 4 & 4 & \dots \end{bmatrix} \quad (7)$$

Bayesian data analysis

The simulation engine introduced above is helpful for theoretical analysis concerning the performance of specific strategies against one another. However, researchers often have empirical questions: which strategies do human players use in an IPD game when arbitration is possible? How does the rate of perception errors affect strategy use? Are individuals with specific psychological characteristics more likely to use aggressive strategies like SneakyTFT than forgiving strategies like TF2T? These kinds of questions can be addressed by using finite mixture models (McNicholas 2017; McLachlan et al. 2019) to evaluate the weight of evidence that a given player used a given strategy, conditional on some observed sequence of game play data, some set of individual-level covariate data, and some set of person-specific parameters.

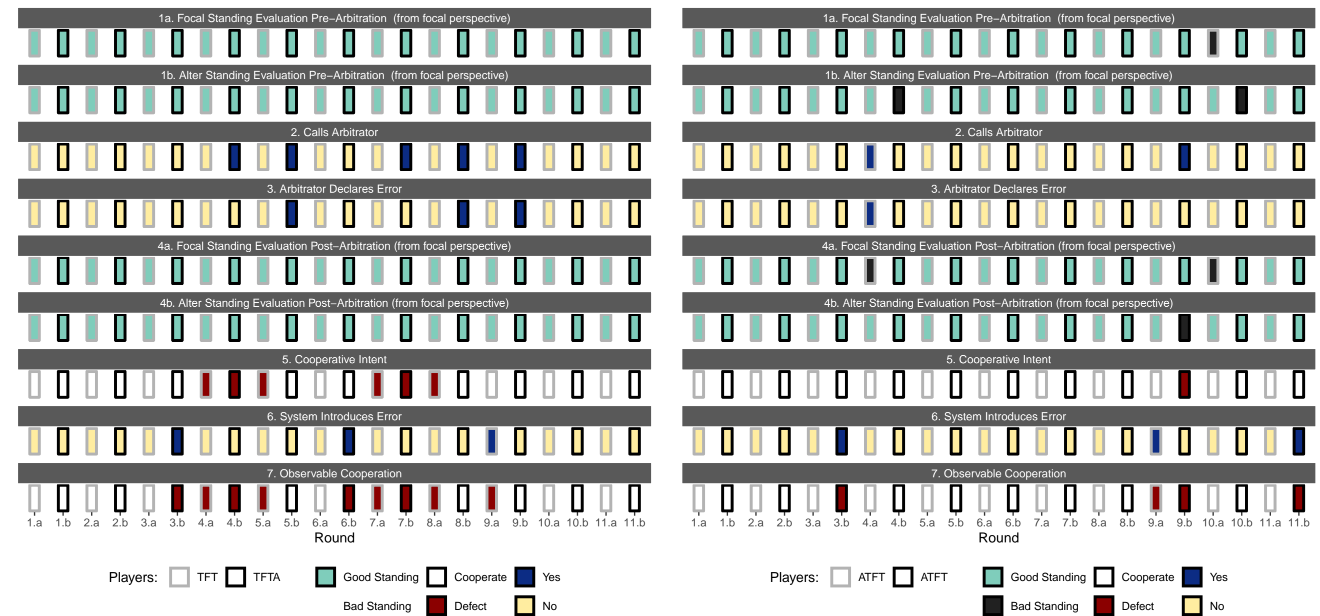
We begin this section by describing the notation we use to represent strategies. We then detail the individual-level parameters used by our strategy functions. Following this, we provide our methodology for determining the probability of a sequence of game-play moves under a specific strategy function. Next, we detail the construction of

Figure 1. Examples of plots produced by `sequence_plot`. From left to right within a plot, the game progresses round by round. In the first half-round, the grey player is making move decisions. In the second half-round, the black player is making move decisions. Within half-rounds, time proceeds top to bottom.



(a) TFT versus TFT

(b) TFT versus GTFT



(c) TFT versus TFTA

(d) ATFT versus ATFT

In Figure 1a, TFT and TFT begin with mutual cooperation. However, once the computer introduces an error, cooperation breaks down, and an unending sequence of mutual defection ensues. In Figure 1b, TFT and GTFT begin with mutual cooperation. Once the computer introduces an error, cooperation breaks down, but GTFT eventually forgives TFT and cooperation resumes. In Figure 1c, TFT and TFTA begin with mutual cooperation. Once the computer introduces an error, cooperation breaks down. TFTA calls the arbitrator to figure out if the observed defection was the result of an error. If the arbitrator rules that no error occurred, then the cycle of defection continues. But, when the arbitrator rules that the observed defection was the result of an error, TFTA forgives, and cooperation is reestablished. In Figure 1d, ATFT and ATFT begin with mutual cooperation. Once the computer introduces an error, cooperation is rapidly restored. Through the use of a *standing* indicator, ATFT can restore cooperation within a single round, regardless of the ruling of the arbitrator.

the strategy weight parameters, and outline how individual-level covariate data can be integrated into the model. Finally, we implement the mixture model and then use Bayes' rule to determine the posterior probability that each individual used each given strategy, conditional on their game-play data.

To test our statistical models, we use the forward data simulator to produce a round-robin data-set. We then analyze this simulated data-set and ensure that our model can recover the true strategy used by each simulated player.

Notation for the strategy functions

By default, we consider a suite of $S = 12$ total strategies (see Table 1). Each is denoted as a strategy-specific function, $\Lambda_s(Q, \Phi)$, of data and parameters. Each Λ_s function takes in a list of data, $Q = \{A, E, \bar{Y}, Y, I, T, G, i, t, g\}$ and a list of parameters, Φ, and returns a 2-vector, λ, as an output. The first output element, $\lambda_{[1]} \in (0, 1)$, gives the strategy's probability of calling the arbitrator in a given round conditional on the values of the function's inputs. The second element, $\lambda_{[2]} \in (0, 1)$, gives the strategy's probability of cooperating in a given round conditional on the values of the function's inputs.

For example, the 4th strategy in our set is SneakyTFT. As such, for a given player, i, with a parameter set $\Phi_i = \{\xi_{[i]}\}$, at a given time-point, t, in a given game, g, we could write $\Lambda_4(Q_g, \Phi_i)$ explicitly. We indicate that we sub-set all data vectors in Q to include only the data where $G = g$ by writing Q_g. Then we can write:

$$\Lambda_4(Q_g, \Phi_i) := \begin{cases} (\,0 \;\; 1\,), & \text{if } T_{[t]} = 1 \\ (\,0 \;\; 0\,), & \text{if } T_{[t]} > 1 \text{ and } Y_{[t-1]} = 0 \\ (\,0 \;\; 1-\xi_{[i]}\,), & \text{if } T_{[t]} > 1 \text{ and } Y_{[t-1]} = 1 \end{cases} \tag{8}$$

First, note that SneakyTFT is not an arbitration strategy, so the first element of its output vector is always 0—it never calls the arbitrator. Also, since SneakyTFT is programmed here to be a 'nice' strategy, it always cooperates on the first half-round. Thus, in the first time-step of a game, SneakyTFT returns the vector $(\,0 \;\; 1\,)$. For all other rounds, SneakyTFT first evaluates its partner's observed move from the last half-round, $Y_{[t-1]}$, and defects if the other player appears to have defected (i.e., if $Y_{[t-1]} = 0$). If the other player cooperated (i.e., if $Y_{[t-1]} = 1$), then SneakyTFT will cooperate with probability $1 - \xi_{[i]}$, where $\xi_{[i]}$ is the 'sneaky defection rate' of individual i. If the value of $\xi_{[i]}$ in individual i approaches zero, then SneakyTFT behaves equivalently to TFT.

Other strategies can be more complicated, and involve more parameters. Consider our 10th strategy, SneakyGTFTA. For a given player, i, with a parameter set $\Phi_i = \{\xi_{[i]}, \psi_{[i]}\}$, at a given time-point, t, in a given game, g, we can write $\Lambda_{10}(Q_g, \Phi_i)$ explicitly as:

$$\Lambda_{10}(Q_g, \Phi_i) := \begin{cases} (\,0 \;\; 1\,), & \text{if } T_{[t]} = 1 \\ (\,0 \;\; 1-\xi_{[i]}\,), & \text{if } T_{[t]} > 1 \text{ and } Y_{[t-1]} = 1 \\ (\,1 \;\; \psi_{[i]}\,), & \text{if } T_{[t]} > 1 \text{ and } Y_{[t-1]} = 0 \\ & \quad \text{and } E_{[t]} = 0 \\ (\,1 \;\; 1-\xi_{[i]}\,), & \text{if } T_{[t]} > 1 \text{ and } Y_{[t-1]} = 0 \\ & \quad \text{and } E_{[t]} = 1 \end{cases} \tag{9}$$

Since SneakyGTFTA is also programmed to be a 'nice' strategy, it cooperates on the first half-round, returning the vector $(\,0 \;\; 1\,)$ when $T_{[t]} = 1$. In other time-points, it cooperates with probability $1 - \xi_{[i]}$, as long as its partner cooperated in the prior round (i.e., when $Y_{[t-1]} = 1$). However, if the partner did not cooperate in the prior round, then SneakyGTFTA calls the arbitrator. If the arbitrator rules that the computer did not introduce an error ($E_{[t]} = 0$; i.e., that the observed defection was 'real'), then SneakyGTFTA nevertheless cooperates with probability, $\psi_{[i]}$, a person-specific generosity/forgiveness rate. If the arbitrator rules that the computer did introduce an error ($E_{[t]} = 1$; i.e., that the observed defection was 'a mistake'), then SneakyGTFTA cooperates with probability, $1 - \xi_{[i]}$, behaving *as if* the other player cooperated.

For similar details on every other strategy file, see Supplementary Stan code.

Individual-level game-play parameters

Each individual in the data-set has a unique set of parameters that controls the behavior of that individual's strategy functions. The probability of calling the arbitrator after an observed defection is controlled by $\alpha_{[i]}$. The 'sneaky defection' rate—or the probability that an individual playing a sneaky strategy variant (e.g., SneakyTFT) defects in cases where they would not if they were playing the corresponding non-sneaky strategy variant (i.e., TFT)—is controlled by $\xi_{[i]}$. The generosity rate—or the probability that an individual playing a generous strategy variant (e.g., GTFT) cooperates in cases where they would not if they were playing the corresponding non-generous strategy variant (i.e., TFT)—is controlled by $\psi_{[i]}$. The cooperation rate for the RANDY strategy—which cooperates completely at random—is given by $\pi_{[i]}$. These parameters have weak priors in our most basic models:

$$\alpha_{[i]} \sim \text{Beta}(11, 1) \tag{10}$$

$$\xi_{[i]} \sim \text{Beta}(1, 11) \tag{11}$$

$$\psi_{[i]} \sim \text{Beta}(4, 8) \tag{12}$$

$$\pi_{[i]} \sim \text{Beta}(6, 6) \tag{13}$$

However, if any of these parameters are key targets of theoretical interest, they can instead be modeled as a function

of some covariate vector $Z_{[i]}$, as:

$$\alpha_{[i]} = \text{logistic}(\hat{\alpha}_{[i]} + \nu_{\alpha[1]} Z_{[i,1]} + \ldots) \quad (14)$$

$$\xi_{[i]} = \text{logistic}(\hat{\xi}_{[i]} + \nu_{\xi[1]} Z_{[i,1]} + \ldots) \quad (15)$$

$$\psi_{[i]} = \text{logistic}(\hat{\psi}_{[i]} + \nu_{\psi[1]} Z_{[i,1]} + \ldots) \quad (16)$$

$$\pi_{[i]} = \text{logistic}(\hat{\pi}_{[i]} + \nu_{\pi[1]} Z_{[i,1]} + \ldots) \quad (17)$$

where individual-level random effects are given by: $\hat{\alpha}_{[i]}, \hat{\xi}_{[i]}, \hat{\psi}_{[i]}, \hat{\pi}_{[i]} \sim \text{Normal}(0, 5)$. The suite of ν parameters above are standard regression parameters with weak priors of the form: $\nu \sim \text{Normal}(0, 5)$. The symbol $logistic$ represents the inverse logit function, and maps the linear model to the unit interval.

Likelihood of game-play data

To estimate the probability of a given strategy conditional on a long sequences of moves, we will build up a finite mixture model as indicated previously. But first, we need to introduce a couple key functions. The first is a modified Bernoulli log probability mass function. Let $X \in \{0, 1\}$ represent a move decision (either A or $\bar{Y}$) in game, g, at a given time-point, t. Let, υ represent the probability of a move decision as generated by some strategy function conditional on some set of observed data and person-specific parameters. Finally, let η be a constant that provides some implementation error allowance. Note that in the model for arbitration calls, we let $\upsilon = 0$ if the strategy does not call the arbitrator, but set $\upsilon = \alpha_{[i]}$ if the strategy does call the arbitrator; this allows individuals to call the arbitrator less frequently than predicted under the pure arbitration strategy. The log probability of observing the outcome X conditional on υ and η is given by:

$$P(X|\, \upsilon, \eta) := \begin{cases} \log\big(\upsilon(1-\eta) + (1-\upsilon)\eta\big), & \text{if } X = 1 \\ \log\big(\upsilon\eta + (1-\upsilon)(1-\eta)\big), & \text{if } X = 0 \end{cases} \quad (18)$$

This function, P, is a generalization of the Bernoulli log probability mass function that uses a Dirichlet mixture for the Bernoulli's mean parameter. Note that as $\eta \to 0$, $P(X|\, \upsilon, \eta)$ converges to the Bernoulli log probability mass function (Blitzstein and Hwang 2019). The $P(X|\, \upsilon, \eta)$ function is more general, in that, for $\eta > 0$, it allows multiple causal paths to produce a given outcome. For example, if $X = 1$, then the first term in the mixture gives the probability of the outcome given some strategy function's output, υ, in the circumstance that no implementation error occurs, $(1 - \eta)$. The second term in the mixture is the probability that an implementation error occurs multiplied by the probability of a positive outcome when such an error occurs. Similar logic holds when $X = 0$.

Next, because there are two move decisions made by the player in each half-round, t, of a given game, g—an arbitration call, $A_{[t]}$, and a cooperation/defection move, $\bar{Y}_{[t]}$—that both inform our estimation of strategies, we need a log probability function that jointly accounts for both outcomes. We first note that we let Φ_i represent the set of all move parameters for individual i: i.e., $\Phi_i = \{\alpha_{[i]}, \xi_{[i]}, \psi_{[i]}, \pi_{[i]}\}$. Then, we can write our main log probability function as: $F(A_{[t]}, \bar{Y}_{[t]}|\, \Lambda_s, Q_g, \Phi_i, \eta)$. For clarity, we will denote the eth element from the output vector of a strategy function as: $\Lambda_s(Q_g, \Phi_i)_{[e]}$.

Then, in a given game, g:

$$F(A_{[t]}, \bar{Y}_{[t]}|\, \Lambda_s, Q_g, \Phi_i, \eta) := P(A_{[t]}|\, \Lambda_s(Q_g, \Phi_i)_{[1]}, \eta) + P(\bar{Y}_{[t]}|\, \Lambda_s(Q_g, \Phi_i)_{[2]}, \eta) \quad (19)$$

This function gives a log probability value for the data in a given round of a given game, for a given respondent, under a given strategy. However, our target of inference here is the person-specific probability of the observed *sequence* of game-play data under a given strategy. The probability of the full move sequence is equal to the product of the per-round probabilities. As such, the next step in our model is to sum the log probability of the observed data conditional on each strategy, s, over all of the games played by individual i. We will let $\Upsilon_{[i,s]}$ store these values.

To calculate $\Upsilon_{[i,s]}$, we sum over all games, and all moves within each game, where individual i was the player making the move:

$$\Upsilon_{[i,s]} = \sum_{g=1}^{\tilde{G}} \sum_{t=1}^{\tilde{T}_g} \begin{cases} F(A_{[t]}, \bar{Y}_{[t]}|\, \Lambda_s, Q_g, \Phi_i, \eta), & \text{if } I_{[t]} = i \\ 0, & \text{otherwise} \end{cases} \quad (20)$$

Here $\tilde{G}$ is the total number of games played and $\tilde{T}_g$ is the total number of time-steps in game g. The element, $\Upsilon_{[i,s]}$, now gives the log probability of individual i's entire sequence of data conditional on the strategy, s, and a set of person-specific move parameters, $\Phi_i = \{\alpha_{[i]}, \xi_{[i]}, \psi_{[i]}, \pi_{[i]}\}$.

Individual-level prior strategy probabilities

Each individual is also given a parameter vector, $\Theta_{[i]}$, that controls their probability of playing each strategy. We construct $\Theta_{[i]}$ using a Softmax link function. We first define:

$$\theta_{[i,s]} = \chi_{[i,s]} + \beta_{[1,s]} Z_{[i,1]} + \ldots \quad (21)$$

with $\chi_{[i,s]} \sim \text{Normal}(0, 5)$, and $\beta_{[1,s]} \sim \text{Normal}(0, 5)$, for $s \in \{1, \ldots, S-1\}$. Then we set:

$$\theta_{[i,S]} = 0 \quad (22)$$

Because the Softmax function is invariant to addition of a constant to each component of its input, it is standard to use $S - 1$ free parameters for the intercept and slope coefficients, and fix the last category to 0 (Stan Development Team 2021a). As such, the random intercept parameters, $\chi_{[i,s]}$, will reflect the log-odds of category s relative to the 'base' category, S, for individual i. Likewise, the $\beta_{[1,s]}$ parameters reflect the change in log-odds of category s relative to the 'base' category, S, as a function of some individual-level covariate, $Z_{[i,1]}$. In our simplest model, we include no covariates, and thus omit any β parameters. We use ALLD as the 'base' category.

To yield an individual-level unit S-simplex on the log-scale, we then write:

$$\Theta_{[i]} = \log\left(\text{Softmax}(\theta_{[i]})\right) \tag{23}$$

where the Softmax function is defined as: $\text{Softmax}(X) := \frac{\exp(X)}{\sum_{k=1}^{K}\exp(X_{[k]})}$, where $\exp(X)$ is the elementwise exponentiation of X, and the sum in the denominator runs over all K elements in the input vector. The elements of the parameter vector, $\Theta_{[i]}$, represent the prior log probability that each strategy is played by individual i.

Modeling the finite mixture

The probability mass of a finite mixture (McNicholas 2017; McLachlan et al. 2019; Stan Development Team 2021b) is given by:

$$H(X \mid \kappa, \gamma) = \sum_{j=1}^{J} \kappa_{[j]} H_j(X \mid \gamma_j) \tag{24}$$

Following Nasserinejad et al. (2017), $H(X \mid \kappa, \gamma)$ is the overall probability mass of the observed data and $H_j(X \mid \gamma_j)$ is the probability mass of the observed data under latent class j. J is the true number of latent classes. κ is a vector that represents the class mixing proportions; its elements are non-negative and sum to 1. Finally, γ is a list of parameter sets used to determine the probability mass of the data in class j.

In our individual-level implementation of this model, the class mixing proportions are each given by: $\exp(\Theta_{[i,s]})$. Likewise, the probability mass of the observed data under each latent class s is given by: $\exp(\Upsilon_{[i,s]})$. Accordingly, the log probability mass of the data for individual i is given by: $\log\left(\sum_{s=1}^{S}\exp(\Theta_{[i,s]} + \Upsilon_{[i,s]})\right)$, which we calculate in Stan using the numerically stable `log_sum_exp` function (Stan Development Team 2021b). We iterate this calculation over all individuals in the data set to complete the model. Further details about finite mixture models in Stan can be found in the Stan manual (Stan Development Team 2021b).

Recovering posterior strategy probabilities

To calculate the posterior probability, $\hat{\Theta}_{[i,s]}$, that individual i used each particular strategy, s, conditional on the observed data, we need to use Bayes' rule (Blitzstein and Hwang 2019):

$$\Pr(A \mid B) = \frac{\Pr(A)\Pr(B \mid A)}{\Pr(B)} \tag{25}$$

which we will unpack here as:

$$\underbrace{\hat{\Theta}_{[i,s]}}_{\substack{\text{Probability that}\\ \text{player } i \text{ played}\\ \text{strategy } s \text{ given}\\ \text{the move data}}} = \frac{\overbrace{\exp\left(\Theta_{[i,s]}\right)}^{\substack{\text{Probability that}\\ \text{player } i \text{ plays}\\ \text{strategy } s}} \cdot \overbrace{\exp\left(\Upsilon_{[i,s]}\right)}^{\substack{\text{Probability of } i\text{'s}\\ \text{move data given}\\ \text{strategy } s}}}{\underbrace{\sum_{s=1}^{S}\exp(\Theta_{[i,s]})\exp(\Upsilon_{[i,s]})}_{\substack{\text{Probability of } i\text{'s move data}\\ \text{under any strategy } s \in \{1,\ldots,S\}}}} \tag{26}$$

Running the model

In contrast to simple linear regression models—which can be written in a general form that permits broad use, e.g., with functions like `lm` in R—the potential complexity of strategy functions in the IPD necessitates much more bespoke model specifications. To make the process of defining custom Stan models easier for end users, we use a two-step process. After the user runs the `setup_folders`, all of the Stan code needed to run a finite mixture model is cloned from the `IPDToolkit` package into the "PrisonersDilema" directory. These copies can be modified by the user if needed. If the user wishes to use our standard model, then they can simply run:

```
create_stan_models(path)
```

which will compile all of the different strategy files and associated Stan code into a single Stan model. For now, we will assume the user is using the basic models, but we will review how to modify and expand our code shortly.

Once the Stan model is built, the user has two options for estimating the model: optimization using the L-BFGS algorithm in Stan (see Stan Development Team 2021d, for further details) or Markov Chain Monte Carlo (MCMC) using a variant of Hamiltonian Monte Carlo (see Stan Development Team 2021c, for further details). Optimization is fast, and is useful for the exploratory phase of research. However, it has a few key shortcomings that we review later. MCMC is slow (McElreath 2018, and see Bommarito 2014), but it affords much more robust inference, and should be used for final model fits.

To fit the model using optimization, run:

```
fit_IPD_optim(d, covariates=NULL,
    n_strategies=12)
```

and to fit the model using MCMC, run:

```
fit_IPD_mcmc(d, covariates=NULL,
    n_strategies=12,   n_chains=1,
    n_cores=1, iterations=2000,
    warmup=1000, adapt_delta=0.95,
    max_treedepth=12)
```

The symbol `d` is a data object of the form exported by the `simulate_round_robin` function, `covariates` is a matrix with as many rows as there are individuals in the data-set, and `n_strategies` gives the number of considered game-play strategies. Both functions return a `Stanfit` object which can be processed using the standard tools included in `rstan`.

Modifying the base model

Writing in parable about statistics, McElreath (2018) claims that "mass production has some advantages, but it also makes our clothes fit badly." In order for statistical tools to provide insight, they often need to be tailored to specific scientific problems; they must become bespoke, custom-designed products. In the case of modeling IPD game-play, researchers need to use their knowledge of the data generating process (e.g., as gleaned through qualitative debriefing interviews) to ensure that the set of considered strategies plausibly covers

the set of strategies used to generate the empirical data. Every context is different, and new strategy files may need to be integrated into the `IPDToolkit` data analysis code in order for the model to produce meaningful estimates.

Writing bespoke Stan models, however, is not facile, so we have designed a work-flow to minimize the complexity of the task, while still allowing for customizability. In the "PrisonersDilema" directory, there are two sub-folder which contain Stan code. The "StrategiesStan" folder may contain only strategy files; new strategy files may be added by the user here. All other Stan code is in the "StanCode" folder; these files may be edited by the user.

If the user wishes to simply change some priors, they can open the `Model.R` file, change the priors as desired, and then re-run the `create_stan_models(path)` function to push these changes into the final model.

More substantial changes are also possible. For example, imagine that data was collected on "opposite day", and debriefing interviews indicated that it was common for many respondents to report punishing cooperators and cooperating with defectors. To account for this somewhat peculiar play style, we can add a new strategy file, GLUM, to our model. This is a two-step process. In the first step, we add a function file. Inside of the "StrategiesStan" folder, we can open the file `SneakyTFT.R`, and modify the line containing: $coop[1] == 1$, to read as: $coop[1] == 0$ and save it as `GLUM.R`. The final code will then read as:

```
vector Pred_GLUM(int the_round, int
    [] arb, int[] arb_error, int[]
    coop, int[] coop_intent, int[]
    coop_err, real xi){
  vector[2] pred;
  pred[1] = 0;
  if(the_round==1){
  pred[2] = 1;
  }
  else{
  pred[2] = (coop[1]==0)?(1-xi):0;
  }
  return pred;
}
```

Unpacking the details of the code above is a more advanced topic than we can discuss here, but we refer readers to our GitHub link for feature requests. Additionally, the Stan Manual has excellent tutorials on coding logical functions.

The second step is to edit the `Parameters.R` file. The following can be added on line 96:

```
p = Pred_GLUM(g_round[i], arb[(i-1)
    :i], arb_err[(i-1):i], coop[(i
    -1):i], coop_intent[(i-1):i],
    coop_err[(i-1):i], xi[actor_id[
    i]]);
p = P(p,arb[i],coop_intent[i],G[
    actor_id[i]],H[actor_id[i]],
    alpha[actor_id[i]]);
Upsilon[actor_id[i],13] += sum(p);
```

The first step adds the strategy function to Stan's database, and the second step calls that function in the appropriate place. As before, the user must run the `create_stan_models(path)` function to push these changes into the final model.

Visualizing the results

To visualize the strategy probability results, $\hat{\Theta}_{[i]}$, for each individual, we provide a heatmap that shows the distribution of posterior probability over strategies for each player:

```
visualize_IPD_results(fit, d, mode="
    optim", smart_sort=FALSE, color=
    "darkred", strategy_set=c("ALLD"
    ,"ALLC","RANDY","TFT","TF2T","
    GTFT","WSLS","TFTA","TF2TA","
    GTFTA","WSLSA","ATFT"))
```

The argument `fit` is the results object returned by either `fit_IFD_optim` or `fit_IFD_mcmc`. The argument `d` is the data object passed to either `fit_IFD_optim` or `fit_IFD_mcmc`. The argument `mode` $\in$ {"optim", "mcmc"} indicates which method was used to fit the model. The argument `smart_sort` sorts players so that individuals with similar strategies are clustered on the y-axis; this makes it easier to see the relative frequencies of the inferred strategies in the sample of respondents. The argument `color` gives the color of the 'hot' end of the heatmap. Finally, the argument `strategy_set` gives the list of strategies in the Stan model. The order here must match the order of strategies in the Stan file. Figure 2 plots some example heatmaps, with and without sorting.

Testing the model

The basic model

To validate our statistical model, we first simulate data using our forward simulation code. We then analyze these data to ensure that we recover the correct strategy classifications (and to validate our simulation code, we visually check pairwise matchups using the `sequence_plot` function).

To simulate a full round-robin tournament between all $S = 12$ strategies, we write:

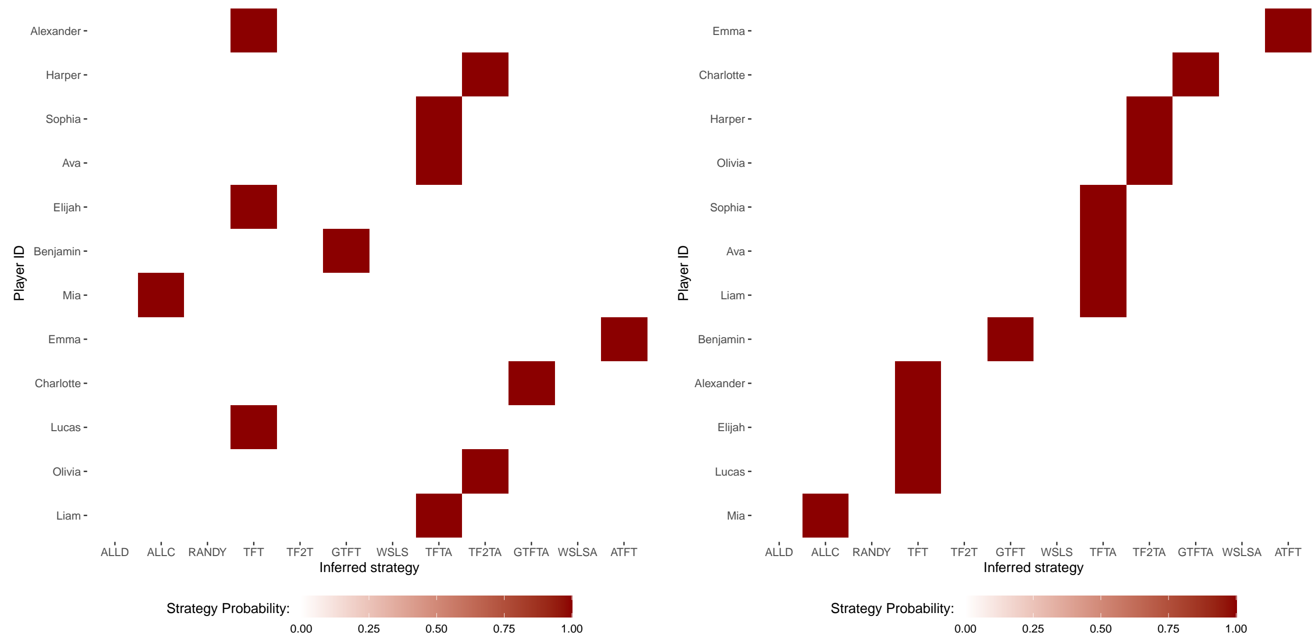
```
  d = simulate_round_robin(players =
      c("ALLD","ALLC","RANDY","TFT","
      TF2T","GTFT","WSLS","TFTA","
      TF2TA","GTFTA","WSLSA","ATFT"),
         n_rounds=40,
         error_rate=0.2,
         arb_error_rate_type_1=0.5,
         arb_error_rate_type_2=0.25)
```

Then, we can fit the optimizer-based and/or MCMC-based models to the data:

```
create_stan_models(path)
f_opt = fit_IPD_optim(d,n_strategies
    =12)
f_mcmc = fit_IPD_mcmc(d,n_strategies
    =12)
```

Finally, we create heat-maps to check classification accuracy:

Figure 2. Example heatmap plots produced by `visualize_IPD_results`. Each row represents a player. Each column represents a strategy. The color intensity of the cell represents the posterior probability that the player indicated by the row-names used the strategy indicated by the column-names. In frame 2a, smart sorting is turned off, so players appear in the order they are supplied. In frame 2b, smart sorting is turned on, so players appear sorted by strategy type; this makes it easier to see the relative frequencies of the inferred strategies in the sample of respondents.



(a) Smart sorting turned off.

(b) Smart sorting turned on.

```
visualize_IPD_results(f_opt, d,
   color="darkred")
visualize_IPD_results(f_mcmc, d,
   color="darkred", mode="mcmc")
```

Figure 3 illustrates that our model can achieve near-perfect strategy classification, at least in ideal circumstance, like here, where each strategy competes with every other strategy. To test the model in more empirically plausible situations, we use a different data simulation set up, and explore the effect of covariates on model performance.

The model with covariates

In most empirical tests of the iterated prisoner's dilemma, a large set of individuals will not compete in a full round-robin tournament. Rather, even if there is a large set of respondents, each will generally only play with a few partners. This complicates the estimation procedure, because, for example, if a TFT player happens to compete only against ALLC players, it will be hard too conclude that this player is actually playing TFT and not ALLC, as the move sequence would be identical under either strategy. Additionally, in empirical settings, individuals often vary in key ways, and researchers often seek to explore the effects of such covariates on strategy use.

Here, we attempt to create a more empirically plausible test case. We create a sample of 60 individuals who vary in some way, Z. We let $Z_{[i]}$ affect both the probability that individual i plays strategy s (i.e., strategy choice), and the value of $\xi_{[i]}$ (i.e., the 'sneaky defection rate'). We then randomly sample players into dyads who play an iterated prisoner's dilemma. In total, we simulate 180 games; so every individual plays only a few games each. Because the code gets a bit lengthy at this point, we direct readers to the supplemental R workflow on the package's GitHub page to follow along.

Figure 4 illustrates the results of strategy classification and Figure 5 illustrates the results of the sneaky defection rate estimation. In both cases, the model performs well.

Possible problems, and solutions

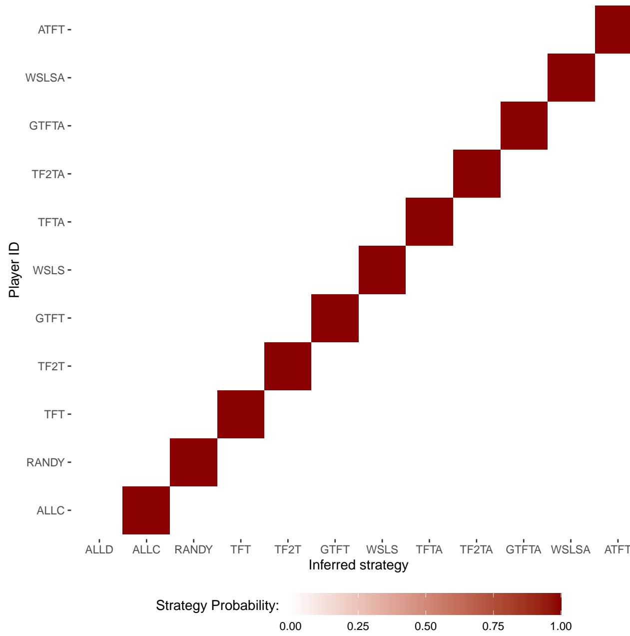
Finite sets and misclassifications

The quality of inferential results generated under the finite mixture modeling approach is highly contingent on the considered strategy set meaningfully capturing the bulk of empirical variation in game-play behavior. If players are empirically using strategies that are not included in the model's strategy set, then model outcomes are not interpretable; put rather anthropomorphically, the model believes that the data *must have been* generated under at least one of the S considered strategies.

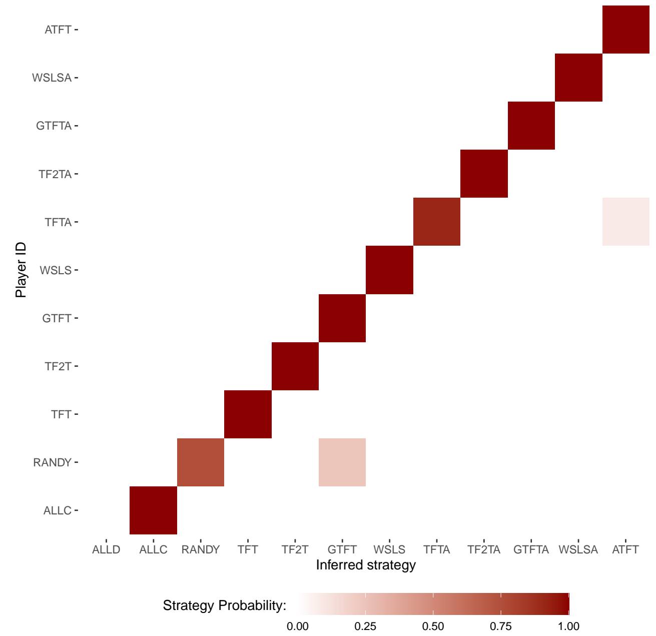
In order to obtain interpretable quantitative results, researchers must first have a good qualitative understanding of the data generating process in their system of interest (McElreath 2018). Debriefing interviews can be used to give respondents the opportunity to describe how they have been playing, which heuristics/strategies they may have been using, and why they have chosen to play in that way.

By integrating domain knowledge, and using custom strategy functions, classification can be improved. For example, see Figure 6, where we show how the frequency distribution of strategies in a sample can be misleading when

Figure 3. Model evaluation with simulated data. As before, each row represents a player. Each column represents a strategy. The color intensity of the cell represents the posterior probability that the player indicated by the row-names used the strategy indicated by the column-names. Here, the player IDs (row names) have been set to show the true strategy that the player on that row used to generate their data. Perfect classification is apparent by all density laying on the diagonal. In frame 3a, we find that optimization yields apparently perfect classification, even with only 30 rounds of play per match-up. In frame 3b, we see that MCMC also yields near-perfect classification. Some advantages between MCMC and optimization are apparent. Optimization will stop at some point on a probability surface; however, many points on that probability surface can be near equally probable. Notice that MCMC indicates that data generated under RANDY look similar to data generated under (sneaky) GTFT. Why? There is a near perfect invariance in the log probability function when: $(1 - \xi_{[i]}) = \psi_{[i]} = \pi_{[i]}$, and this is accurately captured by MCMC. Optimization is overconfident; MCMC better represents our inability to distinguish between RANDY and (sneaky) GTFT. Finally, we note that the excellent classification in these test cases is helped greatly by the fact that each strategy faces off against each other strategy in the data simulation, yielding a large set of diverse data. Typical use cases might not permit such accurate classification.



(a) Model fit with optimization.



(b) Model fit with MCMC.

the full set of generative strategies is not considered in the mixture model. Notably, when a key strategy is missing, we will often observe elevated density on (sneaky) GTFT and GTFTA, as these strategies have independent person-specific cooperation rate parameters after opponent cooperation and defection events. As such, if the true strategy file is not included in the mixture model, these rather flexible strategies will tend to be preferred by the classifier, as they are often probability maximizing in the context of the remaining strategy options.

Invariances between strategies

As we have discussed earlier, and seen in Figure 4, a plethora of invariances can emerge between strategies—limiting the inferences that one can draw from simple visualizations. For example, SneakyGTFT with parameters $\psi_{[i]}$ and $\xi_{[i]}$ behaves identically to RANDY with parameter $\pi_{[i]}$, when $(1 - \xi_{[i]}) = \psi_{[i]} = \pi_{[i]}$. Likewise, SneakyGTFT with parameters $\psi_{[i]}$ and $\xi_{[i]}$ behaves identically to SneakyTFT with parameter $\xi_{[i]}$, when $\psi_{[i]} \to 0$. Additionally, random sampling phenomena can affect classification. For instance, a GTFT player with a low forgiveness rate might be classified as TFT, if during a short game, they happened to not forgive a rare defection.

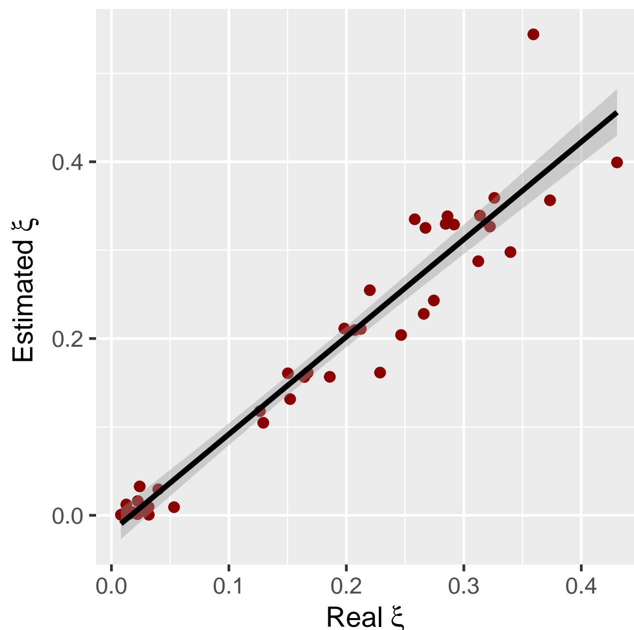
In many cases, invariances can be minimized by running longer games, increasing the number of partners that each focal player is paired with, and increasing the rate of computer introduced errors. These three interventions help to improve classification by providing the additional data needed to distinguish strategies. For example, if pilot debriefing interviews suggest that GTFT players only cooperate 10 percent of the time after observed defections, this suggests that long games with high computer-introduced error rates will be needed to distinguish GTFT players from TFT players; whereas, if forgiveness rates are high, then many fewer rounds may be needed to achieve good classifications. In cases where more data alone cannot improve classification, priors can sometimes be made informative by integrating domain knowledge (McElreath 2018); in some cases, priors can fully resolve an invariance (see Figure 7).

Conclusions

In this paper, we have introduced a new R package that should help facilitate studies of the role that third-party

Figure 4. Here, we have attempted to classify the strategy used by each of 60 players using optimization. The row names give the ID code of each player and the strategy they played. The x-axis location of probability weight is indicative of predicted strategy usage. As before, we note generally good classification. Most players were correctly classified, with the exception of a couple RANDY players who were classified as playing other strategies, and some TFT and GTFT players who were mixed up. One player, Luna, who was programmed to play TFT, was, by chance, never selected to compete in the tournament. As a result, her posterior probability vector over strategies is essentially flat; with no game play data to condition on, the model cannot classify her strategy.

Figure 5. Among the players using strategies with a sneaky defection rate, ξ, we observe a strong correlation $\rho = 0.91$ between the real ξ values used to generate data and the ξ values recovered from our model.



arbitration plays in stabilizing cooperation. Our package provides functionality for both theoretical simulation studies, and empirical investigations of game-play behavior. We have provided some example analyses here, and validated the performance of our Bayesian models by recovering the parameters of the simulation models. The `IPDToolkit` package is openly accessible at: https://github.com/ctross/IPDToolkit. We invite users to submit any questions, bug-reports, feature requests, and other relevant comments through GitHub, where the package will be maintained and improved.

References

Axelrod, R. and W. D. Hamilton
1981. The evolution of cooperation. *Science*, 211(4489):1390–1396.

Blitzstein, J. K. and J. Hwang
2019. *Introduction to probability*. CRC Press.

Boerlijst, M. C., M. A. Nowak, and K. Sigmund
1997. The logic of contrition. *Journal of Theoretical Biology*, 185(3):281–293.

Bommarito, N.
2014. Patience and perspective. *Philosophy East and West*, 64(2):269–286.

Boyd, R.
1989. Mistakes allow evolutionary stability in the repeated prisoner's dilemma game. *Journal of Theoretical Biology*, 136(1):47–56.

Boyd, R. and S. Mathew
2021. Arbitration supports reciprocity when there are frequent perception errors. *Nature Human Behaviour*, Pp. 1–8.

McElreath, R.
2018. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC.

McElreath, R. and R. Boyd
2008. *Mathematical models of social evolution: A guide for the perplexed*. University of Chicago Press.

McLachlan, G. J., S. X. Lee, and S. I. Rathnayake
2019. Finite mixture models. *Annual Review of Statistics and Its Application*, 6:355–378.

McNicholas, P. D.
2017. *Mixture model-based classification*. CRC press.

Molander, P.
1985. The optimal level of generosity in a selfish, uncertain environment. *Journal of Conflict Resolution*, 29(4):611–618.

Nasserinejad, K., J. van Rosmalen, W. de Kort, and E. Lesaffre
2017. Comparison of criteria for choosing the number of classes in Bayesian finite mixture models. *PLoS One*, 12(1):1–23.

Nowak, M. and K. Sigmund
1990. The evolution of stochastic strategies in the prisoner's dilemma. *Acta Applicandae Mathematicae*, 20(3):247–265.

Nowak, M. and K. Sigmund
1993. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364(6432):56.

Nowak, M. A. and K. Sigmund
2005. Evolution of indirect reciprocity. *Nature*, 437(7063):1291.

R Core Team
2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Stan Development Team
2019. RStan: The R interface to Stan. R package version 2.19.2.

Stan Development Team
2021a. Stan modeling language users guide and reference manual, version 2.20.0: Chapter 1.6 multi-logit regression.

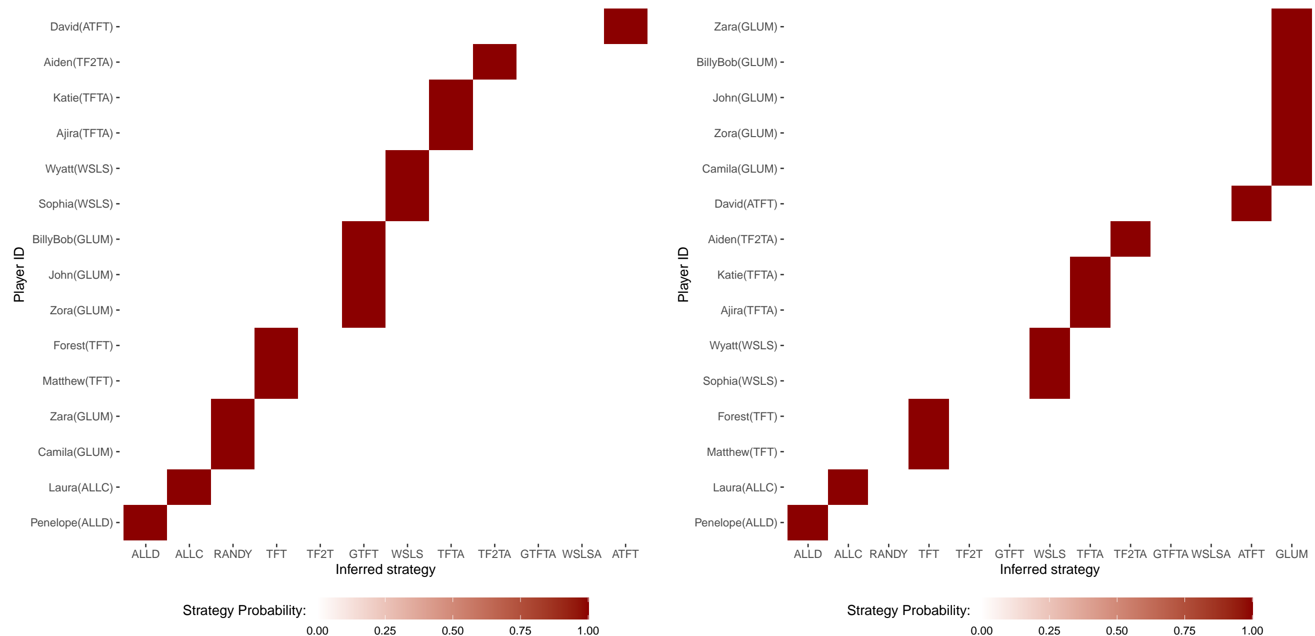
Stan Development Team
2021b. Stan modeling language users guide and reference manual, version 2.20.0: Chapter 5 finite mixtures.

Stan Development Team
2021c. Stan reference manual: Chapter 16 posterior analysis.

Stan Development Team
2021d. Stan reference manual: Chapter 17 optimization.

Sugden, R. et al.
2004. *The economics of rights, co-operation and welfare*. Springer.

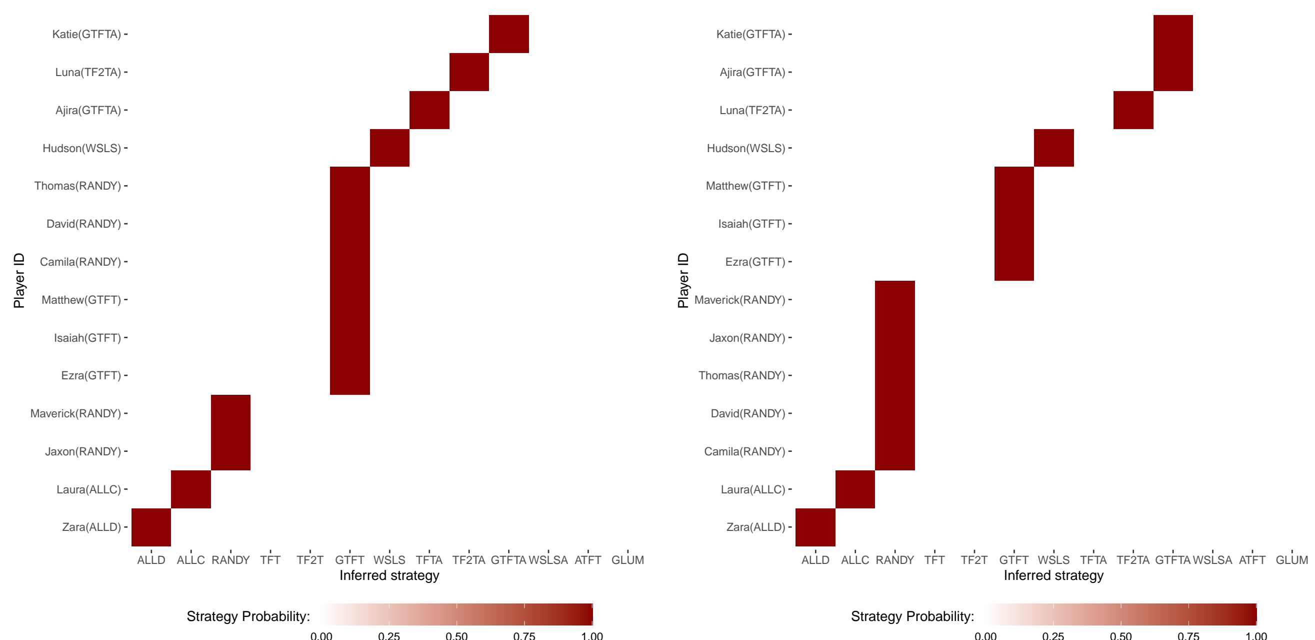
Figure 6. In order for the frequencies of different strategies in a sample of respondents to be accurately estimated, the set of considered strategies must include the bulk of the strategies employed empirically. Consider, for example, frame 6a—here, GLUM is a common strategy in the sample, but it was not a considered strategy in the mixture model. As such, the implied strategy frequencies in frame 6a diverge from the true strategy frequencies in frame 6b. Notably, GTFT seems to be quite common in the naïve model in frame 6a; however, from the improved model based on domain knowledge that GLUM is common (in frame 6b), it is clear that GTFT is never actually played.



(a) Model fit without GLUM strategy file.

(b) Model fit with GLUM strategy file.

Figure 7. The effect of priors on classification. Invariances between some strategies can lead to classification errors, especially when models are fit via optimization. Frame 7a shows that when very flat priors—i.e., Beta(1,1)—are used on ψ and π, RANDY is often misclassified as GTFT. By using moderate priors, perhaps informed by pilot interviews, to specify more reasonable ranges for key parameters, classification is improved. In frame 7b, we set $\psi_{[i]} \sim \text{Beta}(20, 70)$ and $\pi_{[i]} \sim \text{Beta}(45, 45)$, to better reflect our prior knowledge that $\psi_{[i]} \approx 0.2$ and $\pi_{[i]} \approx 0.6$ in the generative model.



(a) Model with flat priors on ψ and π.

(b) Model with moderately informative priors on ψ and π.