# stratEst: a software package for strategy frequency estimation

Fabian Dvorak

Centre for the Advanced Study of Collective Behaviour, University of Konstanz, Box 146, Konstanz, D-78457, Germany.

Contributing authors: fabian.dvorak@uni-konstanz.de;

**Abstract**

stratEst is a software package for strategy frequency estimation in the freely available statistical computing environment R (R Development Core Team, 2022). The package aims at minimizing the start-up costs of running the modern strategy frequency estimation techniques used in experimental economics. Strategy frequency estimation (Stahl & Wilson, 1994, 1995) models the choices of participants in an economic experiment as a finite mixture of individual decision strategies. The parameters of the model describe the associated behavior of each strategy and its frequency in the data. stratEst provides a convenient and flexible framework for strategy frequency estimation, allowing the user to customize, store and reuse sets of candidate strategies. The package includes useful functions for data processing and simulation, strategy programming, model estimation, parameter testing, model checking, and model selection.

**Keywords:** strategy frequency estimation, software, repeated games

**JEL Classification:** C72 , C87 , C90

## 1 Introduction

The analysis of heterogeneity in decision making has a long tradition in experimental economics. Stahl and Wilson (1994, 1995) pioneered the use of finite mixture models to study the decision strategies of participants in economic experiments. In an influential paper, Dal Bó and Fréchette (2011) estimated the frequencies of a set of candidate strategies to explain participants' choices in a repeated prisoner's dilemma experiment.

In recent years, strategy frequency estimation has become increasingly popular in experimental economics and several model extensions have been proposed.[1]

stratEst is a software package for the freely available statistical computing environment R (R Development Core Team, 2022) that significantly reduces the start-up costs of performing strategy frequency estimation. Programming strategy frequency estimation code from scratch usually requires considerable effort on the part of the analyst. Before model parameter optimization routines can be used, code must be written to compute the probability that a given sequence of decisions is generated by a particular strategy. Since each candidate strategy is different, this task must be performed for each strategy, which can be tedious, especially if the set of candidate strategies is large and the strategies are complex. An additional problem is the difficulty of adapting strategy estimation code to other data; the close correspondence between candidate strategies and data usually requires substantial revision of the code.

The stratEst package allows strategies to be generated, stored, and adjusted without the need for strategy-specific code to calculate the probability of certain decisions. Using the stratEst strategy generation function, the analyst can conveniently create customized strategies with little effort. A guiding principle of the package is that strategies are Markov strategies, represented as finite-state automata, and stored as dataframe-like objects that can be reloaded and adapted for later use. In the automaton representation, choice probabilities are determined by the internal state of the automaton, not by the history of the game. This guarantees that strategies are concisely represented even when the number of game histories is large or potentially infinite.

The simplicity of the automaton representation facilitates the programming and organization of strategies but also makes it easy to adapt existing strategies to other data. At the same time, it does not limit the complexity of strategies. Finite-state automata can mimic complex patterns of behavior based on deterministic sequences of state transitions triggered by inputs from the choice environment. It is important to note that the determinism of the automata concerns only the transitions between states, not the choices of the strategy. This means that it is possible to generate behavior strategies and define (or alternatively estimate) their state-specific choice probabilities.

Another potential obstacle for the analyst who wants to perform strategy estimation is transforming the data into a format suitable for analysis. The package includes a function to create the inputs for the strategies to facilitate the transformation of the

[1] Experimental studies of the repeated prisoner's dilemma that use the strategy frequency estimation method include: Aoyagi, Bhaskar, and Frechette (2019); Arechar, Dreber, Fudenberg, and Rand (2017); Backhaus and Breitmoser (2018); Camera, Casari, and Bigoni (2012); Embrey, Frechette, and Yuksel (2017); Frechette and Yuksel (2017); Kartal and Müller (2022); Kasberger, Martin, Normann, and Werner (2023); Kayaba, Matsushima, and Toyama (2020); Kloosterman (2020); Rand, Fudenberg, and Dreber (2015); Sherstyuk, Tarui, and Saijo (2013). Yaroslav Rosokha maintains an online repository of strategy estimation code for the repeated prisoner's dilemma developed for Romero and Rosokha (2018, 2019), which can be found on GitHub: https://github.com/yaroslavrosokha/sfem. Fudenberg, Rand, and Dreber (2012) introduce lenient and forgiving strategies for analyzing the repeated prisoner's dilemma with imperfect public monitoring. Breitmoser (2015) theoretically derives and estimates the choice probabilities of behavior strategies and Bland (2020) extends the error specification of the strategy frequency estimation model by estimating individual-specific trembles. Dvorak and Fehrler (2018) add individual-level covariates to explain individuals' strategy choices, and Embrey, Frechette, and Stacchetti (2013) use strategy frequency estimation to explain choices in a repeated partnership game with more than two alternatives.

data. This makes it easy to perform strategy estimation on a wide variety of data, and can sometimes be used to analyze new data with just a few lines of code. The package includes a number of helpful functions for data processing and simulation, parameter testing, model checking, and model selection, further reducing the start-up costs of performing strategy estimation.

The estimation function of the package returns the maximum likelihood parameters of a strategy estimation model based on the expectation-maximization algorithm (Dempster, Laird, & Rubin, 1977) and the Newton-Raphson method. The package speeds up the estimation procedure by integrating C++ and R with the help of the R packages Rcpp (Eddelbuettel & François, 2011) and the open-source linear algebra library for the C++ language RppArmadillo (Sanderson & Curtin, 2016). Package development is supported by the R packages devtools (Wickham, Hester, & Chang, 2020), testthat (Wickham, 2011), roxygen2 (Wickham, Danenberg, Csardi, & Eugster, 2020), and Sweave (Leisch, 2002). The strategies are plotted with the packages DiagrammeR (Iannone, 2020) and DiagrammeRsvg (Iannone, 2016).

The purpose of this paper is to introduce the scope and general principles of the stratEst package. The detailed package vignette is available on the author's website.[2] The package is available for download from the Comprehensive R Archive Network and is continuously tested for functionality on Windows, MacOS, and Linux.[3] Non-commercial use of stratEst is free of charge. However, the author kindly asks all users of the package to cite this article in publications or presentations of their research.

## 2 A motivating example

This example illustrates how the package can be used to replicate the results of the influential strategy estimation study by Dal Bó and Fréchette (2011). The study examines the evolution of cooperation in the indefinitely repeated prisoner's dilemma across six experimental treatments. The six treatments differ in the reward offered for mutual cooperation $R$ and the continuation probability $\delta$ of the repeated game. The parameter $R$ is either 32, 40, or 48. For each value of $R$, there are two treatments with continuation probabilities $\delta$ of 1/2 or 3/4, resulting in a $2 \times 3$ between-subjects design with six treatments.

| | c | d |
|---|---|---|
| c | $R$,$R$ | 12,50 |
| d | 50,12 | 25,25 |

**Fig. 1** Stage game of Dal Bó and Fréchette (2011). The stage game features two choices, cooperation (c) and defection (d). $R$ varies across experimental treatments and is either 32, 40, or 48.

[2]http://fdvorak.com/stratEst-vignette.pdf
[3]http://CRAN.R-project.org/package=stratEst

```
R> print(strategies.DF2011$TFT)
  prob.d prob.c tremble tr(cc) tr(cd) tr(dc) tr(dd)
1      0      1      NA      1      2      1      2
2      1      0      NA      1      2      1      2
```

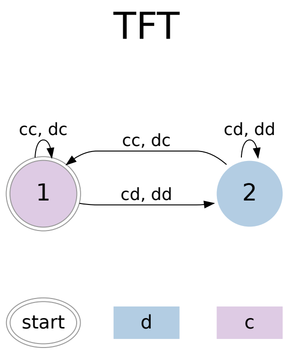

**Fig. 2** The strategy Tit-For-Tat. Left part shows the strategy `TFT` printed to the R console. Rows represent the two states of the automaton. Columns show the defection, cooperation and tremble probabilities in each state, as well as the deterministic state transitions between states. Right part shows the graphical representation of `TFT`. States are depicted as nodes, deterministic state transitions as arrows between nodes. Colors indicate the predicted action in each state.

To follow along in R, all commands in italics after the command prompt `R>` can be executed in the R console. The complete code for generating all the output and figures presented below is also available as supplementary material. The following two commands will install the latest CRAN version of the stratEst package and load it into memory:

```
R> install.packages("stratEst")
R> library(stratEst)
```

***Strategies:***

Dal Bó and Fréchette (2011) fit the same strategy estimation model to the data of each treatment. This model features six strategies: Always Defect (`ALLD`), Always Cooperate (`ALLC`), Tit-For-Tat (`TFT`), Grim-Trigger (`GRIM`), Win-Stay-Lose-Shift (`WSLS`), and a trigger strategy with two punishment periods (`T2`). The package includes the predefined list `strategies.DF2011`, which contains the six strategies used by Dal Bó and Fréchette (2011). Each element of this list represents a strategy, encoded as a finite-state automaton.

The left panel of Figure 2 shows the result of printing the element `TFT` of the list `strategies.DF2011` to the R console. The strategy `TFT` is a finite-state automaton with two states, represented by the two rows of the printed object. In each state, the strategy defects or cooperates, with probabilities defined in the first two columns `prob.d` and `prob.c`. Since these probabilities are either zero or one, there is a column `tremble` for each state. The tremble probabilities define the probability of the action *not* predicted in this state. For the strategy `TFT` (and all other strategies in the list `strategies.DF2011`), the tremble probabilities are not available (`NA`), which tells the estimation function that these probabilities should be estimated from the data.

The remaining four columns define a matrix of deterministic state transitions triggered by four different strategy inputs in the two different states. Since the data come from a repeated prisoner's dilemma, the four inputs `cc`, `cd`, `dc`, and `dd` reflect the four possible combinations of one's own action and the action of the other player in the previous period.

All state transitions must be integers that indicate the future state of the strategy after receiving the input. The interpretation of these transitions is as follows: The value `1` in row one and column `tr(cc)` means that whenever the strategy is in the first state (row one) and the input in the current period is `cc`, the strategy will remain in that first state, and the players will cooperate if no tremble occurs in the period. If the input is `cd` instead, the strategy will transition to the second state and cooperate only if a tremble occurs in that period. By definition, the first row is always the start state of the automaton in the first period; this can also be used to specify a particular behavior in the first period.[4]

Understanding the behavior of complex strategies based on a matrix of deterministic state transitions can be difficult. A more convenient way to examine the behavior associated with the strategy `TFT` is to plot the strategy. The right panel of Figure 2 shows the results of plotting the strategy with the function `plot()`.[5] In the graphical representation, each state is represented by a node, and arrows indicate the deterministic state transitions triggered by the different inputs. Different colors are used to indicate the predicted action in each state. The graphical representation of `TFT` makes it fairly easy to understand the behavior of the strategy.

### *Data:*

To fit the strategies to the data, the stratEst package requires data in the long format with one row for each decision. The object `DF2011` is loaded with the package and contains the data used in Dal Bó and Fréchette (2011). The data set has three columns named `id`, `game`, and `period`, with integers that uniquely identify the subject and indicate the order of the games and the periods in each game. The data set also includes a column named `choice`, which contains the individual's choice encoded as a factor with two levels (`c` and `d`), and a column named `other.choice`, which identifies the choice of their partner in the same period. Readers can also use their own data, in the format discussed, and follow along from here.

What is missing in the data set `DF2011` is a column with the strategy inputs received at the beginning of each period that trigger the deterministic state transitions. The inputs are the crucial information that allows the package's estimation function to determine the current state of each strategy for each observation in the data. The data function of the package can be used to facilitate the generation of the input variable.

```
R> data.DF2011 <- stratEst.data(data = DF2011, choice = "choice",
                                input = c("choice","other.choice"),
                                input.lag = 1)
R> head(data.DF2011)
  treatment id game period choice other.choice input
1     D5R32  1   62      1      d            d  <NA>
2     D5R32  1   63      1      d            d  <NA>
```

[4]For this to be the case, the input must be `NA` in the first period, and all state transitions must be greater than one. This ensures that the strategy starts in the first state and never returns to it.

[5]To plot strategies, the R package DiagrammeR (Iannone, 2020) must be installed with the command `install.packages("DiagrammeR")`.

```
3     D5R32  1   63      2      c            d    dd
4     D5R32  1   63      3      d            c    cd
5     D5R32  1   64      1      d            d  <NA>
6     D5R32  1   64      2      c            d    dd
```

The options `input = c("choice", "other.choice")` and `input.lag = 1` create the input variable by concatenating the players' choices in the preceding period. The generated object `data.DF2011` contains all the information necessary for fitting the strategies. The levels `c` and `d` of the variable `choice` correspond to the choice probabilities `prob.c` and `prob.d` in the first two columns of each strategy object. The levels of the variable input `cc`, `cd`, `dc` and `dd` correspond to columns `tr(cc)`, `tr(cd)`, `tr(dc)`, and `tr(dd)`, respectively. The input is not available (`NA`) in the first period because a lag of one period was used. Whenever the input is unavailable, the strategy will revert to the first state, which is, by definition, the start state of the automaton.

***Model fitting:***

The command below replicates the strategy estimation results reported by Dal Bó and Fréchette (2011):

```
R> model.DF2011 <- stratEst.model(data = data.DF2011,
                                  strategies = strategies.DF2011,
                                  sample.id = "treatment")
```

Choosing the option `sample.id = "treatment"` estimates a model with treatment-specific parameters. This means that, for each treatment in the data, one vector of shares and one tremble parameter is estimated. The command `summary(model.DF2011)` prints a summary of the fitted model to the R console. The estimated shares are the strategy shares reported in Table 7 on page 424 of Dal Bó and Fréchette (2011). The treatment-specific parameters of the fitted model can also be accessed separately. For example, the strategy shares for the data of the treatment with $\delta = 0.5$ and $R = 32$ rounded to two digits are:

```
R> round(model.DF2011$shares$treatment.D5R32, digits = 2)

                ALLD ALLC GRIM  TFT WSLS T2
treatment.D5R32 0.92    0    0 0.08    0  0
```

The fitted strategies can plotted, printed to the console or stored for later use. For example, the TFT strategy fitted to the data of the treatment with $\delta = 0.5$ and $R = 32$ looks like this:

```
R> print(model.DF2011$strategies$treatment.D5R32$TFT)

  prob.d prob.c tremble tr(cc) tr(cd) tr(dc) tr(dd)
1      0      1    0.06      1      2      1      2
2      1      0    0.06      1      2      1      2
```

The maximum likelihood estimate of the treatment-specific tremble probability implies that the fitted TFT strategy randomly selects the action not predicted with a probability of 6%. Accounting for trembles, the effective cooperation probabilities in the two states are 0.94 and 0.06.

***Parameter estimates and standard errors:***

The estimated parameters and standard errors of a fitted model are stored in objects with the extensions `.par` and `.se`. The estimated shares of the fitted model `model.DF2011` can be inspected with the command `print(model.DF2011$shares.par)`. Perhaps somewhat surprisingly, the object `model.DF2011$shares.par` does not indicate which parameter belongs to which strategy. The reason for this is that restricted model specifications can be estimated in which the shares of some strategies are determined by the same share parameter. Another option is to define strategy shares that are not estimated from the data. These possibilities preclude a one-to-one mapping of estimated share parameters and strategies.

The object `shares.indices` can be used to find the share parameter of a certain strategy. For example, the code below retrieves the estimated share of `ALLD` for the data of the first treatment. The same logic can be used to retrieve the estimated parameters and standard errors of trembles and response probabilities.

```
R> index.ALLD.D5R32 <- model.DF2011$shares.indices[,"treatment.D5R32"]["ALLD"]
R> Estimate <- model.DF2011$shares.par[index.ALLD.D5R32]
R> Std.Error <- model.DF2011$shares.se[index.ALLD.D5R32]
R> share.ALLD.D5R32 <- round(cbind(Estimate, Std.Error), 3)
R> print(share.ALLD.D5R32)
```

```
     Estimate Std.Error
[1,]     0.92     0.043
```

By default, the standard errors of the parameters and the quantiles of their sampling distribution are obtained using the empirical observed information matrix (Meilijson, 1989). The estimation based on the empirical observed information matrix creates little computational overhead. However, the method may produce downward biased standard errors for parameters close to the boundary of the parameter space. For statistical testing, it is therefore recommended to estimate standard errors using a nonparameteric block-bootstrap. The bock-bootstrap procedure takes the dependence of choices made by participants with the same `id` into account. The following code illustrates how block-bootstrapped standard errors are obtained. To keep the computation time short, we only use the data from the first treatment.

```
data.DF2011.D5R32 <- data.DF2011[data.DF2011$treatment == "D5R32", ]
model.DF2011.D5R32 <- stratEst.model(data = data.DF2011.D5R32,
                                     strategies = strategies.DF2011,
                                     se = "bootstrap", bs.samples = 1e+4,
                                     quantiles = c(0.05, 0.25, 0.5, 0.75, 0.95))

index.ALLD.D5R32 <- model.DF2011.D5R32$shares.indices["ALLD", ]
Estimate <- model.DF2011.D5R32$shares.par[index.ALLD.D5R32]
Std.Error <- model.DF2011.D5R32$shares.se[index.ALLD.D5R32]
quantiles.ALLD.D5R32 <- model.DF2011.D5R32$shares.quantiles[index.ALLD.D5R32, ]
share.ALLD.D5R32 <- round(cbind(Estimate, Std.Error), 3)
print(share.ALLD.D5R32)
```

```
     Estimate Std.Error
[1,]     0.92     0.043
```

```
R> print(SGRIM)
  prob.d prob.c tremble tr(cc) tr(cd) tr(dc) tr(dd)
1      0      1      NA      1      2      2      3
2     NA     NA      NA      1      2      2      3
3      1      0      NA      1      2      2      3
```

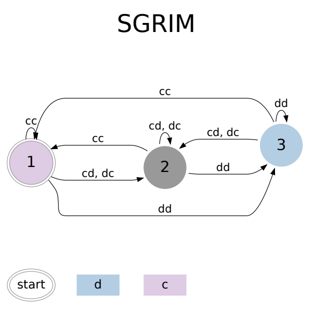

**Fig. 3** The strategy Semi-Grim. Left part shows the strategy SGRIM printed to the R console. Rows represent the three states of the automaton. Columns show the defection, cooperation and tremble probabilities in each state, as well as the deterministic state transitions between states. Right part shows the graphic representation of SGRIM. States are depicted as nodes, deterministic state transitions as arrows between nodes. Colors indicate the predicted action in each state.

Bootstrapping produces the same standard error of the share of ALLD in the first treatment. The estimated quantiles of the sampling distribution are:

```
R> print(quantiles.ALLD.D5R32, digits = 3)

   5%   25%   50%   75%   95%
0.847 0.895 0.921 0.955 0.981
```

***Adaptation:***

A key contribution of the package is that it allows users to perform different variants of strategy estimation with little effort. Some examples are given below. Perhaps most importantly, the package allows users to build and maintain an archive of customized strategies. For example, the following code generates the strategy known as Semi-Grim (Backhaus & Breitmoser, 2018; Breitmoser, 2015):

```
R> SGRIM <- stratEst.strategy(choices= c("d","c"),
                               inputs = c("cc","cd","dc","dd"),
                               prob.choices = c(0,1,NA,NA,1,0),
                               tr.inputs = rep(c(1,2,2,3), 3),
                               num.states = 3)
```

The argument choices made available using the strategy generation function can be used to specify the choice alternatives of the strategy. This creates the columns prob.d and prob.c in the left panel of Figure 3. The specified inputs create the columns with the state transitions tr(cc)-tr(dd). The values passed to prob.choices are filled row by row into the columns prob.d and prob.c. The result is that the strategy will have players cooperating in the first state and defecting in the third state. The use of NA for the choice probabilities after the inputs cd and dc indicates that these are the parameters that should be estimated from the data. The argument tr.inputs is used to specify the deterministic state transitions of SGRIM. The transitions must be integers between one and the total number of states, which is defined by the argument num.states. Since the integers are also filled into the columns row by row, we can replicate the vector c(1,2,2,3) three times to generate transitions that do not depend on the current state. The right panel of Figure 3 shows the plotted results of the strategy SGRIM.

Below are examples of various adaptations of the strategy estimation model available to analysts:

- Adjust the set of candidate strategies, adding the behavior strategy SGRIM, and estimate its cooperation probability after histories cd and dc from the data.

```
R> my.strategies <- c(strategies.DF2011[c("ALLD","ALLC","GRIM","TFT")],
                      list("SGRIM" = SGRIM))
R> my.model <- stratEst.model(data = data.DF2011,
                              strategies = my.strategies,
                              sample.id = "treatment")
R> print(my.model$strategies$treatment.D75R48$SGRIM)
  prob.d prob.c tremble tr(cc) tr(cd) tr(dc) tr(dd)
1  0.000  1.000   0.022      1      2      2      3
2  0.537  0.463      NA      1      2      2      3
3  1.000  0.000   0.022      1      2      2      3
```

- Select a subset of the strategies that provide the best explanation of the data according to the Bayesian information criterion (Schwarz, 1978).

```
R> select.model <- stratEst.model(data = data.DF2011,
                                  strategies = my.strategies,
                                  select = "strategies", crit = "bic",
                                  sample.id = "treatment")
R> select.model.shares <- round(do.call(rbind, select.model$shares), 2)
R> print(select.model.shares)
                 ALLD  TFT SGRIM
treatment.D5R32  0.91 0.07  0.02
treatment.D5R40  0.81 0.15  0.04
treatment.D5R48  0.52 0.21  0.27
treatment.D75R32 0.63 0.15  0.22
treatment.D75R40 0.11 0.24  0.66
treatment.D75R48 0.00 0.30  0.70
```

- Estimate the overall strategy shares by pooling the data of all treatments, keeping the tremble probabilities treatment specific.

```
R> pooled.model <- stratEst.model(data = data.DF2011,
                                  strategies = my.strategies,
                                  sample.id = "treatment",
                                  sample.specific = "trembles")
R> print(round(pooled.model$shares, digits = 2))
      ALLD ALLC GRIM  TFT SGRIM
share 0.51 0.04 0.04 0.19  0.23
```

- Fix selected model parameters, like the tremble probabilities of TFT, mixed cooperation probabilities of SGRIM, and the strategy shares.

```
R> my.strategies$TFT$tremble <- c(0.1,0.2)
R> my.strategies$SGRIM$prob.c <- c(0.95,1/3,0.05)
R> my.strategies$SGRIM$prob.d <- 1 - c(0.95,1/3,0.05)
R> fixed.shares <- c(0.3,0.1,0.1,0.2,0.3)
R> model.fixed <- stratEst.model(data = data.DF2011,
                                 strategies = my.strategies,
                                 shares = fixed.shares)
```

- Transform the data. For example, imagine that the data set DF2011 contains second-mover decisions of a sequential game; second-mover strategies should react to the player's own action in the previous period and the action of the first mover in the current period.

```
R> second.mover.data <- stratEst.data(data = DF2011, choice = "choice",
                                      input = c("choice","other.choice"),
                                      input.lag = c(1,0))
R> second.mover.model <- stratEst.model(data = second.mover.data,
                                        strategies = strategies.DF2011)
```

## 3 Workflow

The core of the stratEst package is a collection of functions for strategy generation, data processing and simulation, model fitting, parameter testing, and model checking. Figure 4 outlines the recommended workflow when using the package and highlights the functions involved in each step. A detailed description of each function can be found in the package's R documentation or, alternatively, in the package vignette.

In the first step, the user collects or creates a set of candidate strategies based on prior knowledge or theoretical considerations. All strategies must assign probabilities to the action space observed in the data and respond to the same set of inputs. Thus, identifying a set of common inputs is often the initial task for the analyst and is ideally guided by theory.

As a second step, it is generally useful to simulate data for the set of candidate strategies. By fitting the correct model to simulated data, the analyst can verify that all model parameters are recovered. These checks are generally recommended because the parameters of the mixture model may not be identified. For example, it is not possible to recover the shares of a mixture of the grim trigger strategy and a strategy that always cooperates in the repeated prisoner's dilemma if both strategies are error free. When working with more complex strategies, identification problems may arise that are much harder to anticipate but easy to detect using simulated data.

If all the model parameters can be recovered from the simulated data, the analyst can proceed to the third step of preparing the experimental data and fitting the model. After the estimation, the fitted model should be tested for misspecification.

## 4 Limitations and future development

The current version of the package has several limitations. First, the action space of all strategies must be discrete. Choices are modeled as independent draws from a

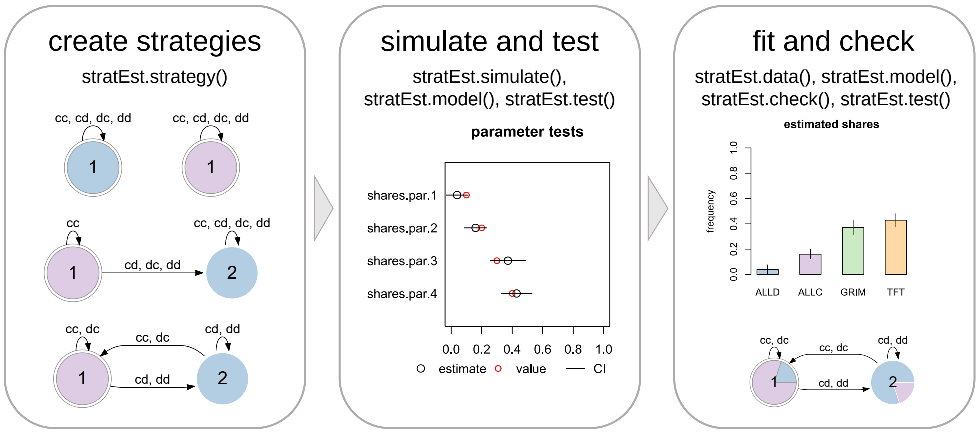

**Fig. 4** stratEst workflow

multinomial distribution defined by the strategy's state-specific choice probabilities; that is, it is not possible to estimate strategies with continuous choices. Another limitation is that the state transitions must be deterministic and specified by the user. This excludes the possibility of fitting Markov strategies with probabilistic state transitions (Hidden Markov Models) or estimating state transitions from data.

The package will be further developed to address its limitations. One restriction that will be relaxed in future versions of the package is that all model strategies must respond to the same set of inputs. For example, for data from the prisoner's dilemma it is typically assumed that all candidate strategies are responses to the players' actions in the previous period. Thus, all strategies are automata reacting to five inputs, the four possible combinations of actions in the last round and the empty history in the first period. In most cases, it will be possible to represent all model strategies as automata that satisfy this restriction. However, the representation of some strategies may be unnecessarily complex in this case, making strategy programming more difficult than it should be.

## 5 Conclusions

This article introduces the R software package stratEst, for strategy frequency estimation. The stratEst package provides a free and easy-to-use framework for performing the modern strategy frequency estimation techniques used in experimental economics. The estimation function of the package fits a finite mixture model of customized individual choice strategies and returns several values, including maximum likelihood estimates and standard errors of all model parameters. The package also includes several helpful functions to facilitate strategy programming, data processing and data simulation, model selection, and model checking, as well as model checks and statistical tests of fitted model parameters.

## Acknowledgments

Support from the German Research Foundation (DFG) through research unit FOR 1882 ”Psychoeconomics” is gratefully acknowledged. I would like to thank Yongping Bao, Yves Breitmoser, Karsten Donnay, Urs Fischbacher, Wieland Müller, Konstantin Käppner, and Susumu Shikano as well as two anonymous reviewers for helpful comments. I am grateful to Zhijian Wang, Bin Xu, and Hai-Jun Zhou for sharing their data and granting permission to include the data in the package. I am especially grateful to Sebastian Fehrler for many stimulating discussions. Replication material for the study is available at https://doi.org/10.17605/OSF.IO/JEH82.

## Conflict of interest

The author declares no conflict of interest.

## References

Aoyagi, M., Bhaskar, V., Frechette, G.R. (2019). The impact of monitoring in infinitely repeated games: Perfect, public, and private. *American Economic Journal: Microeconomics*, *11*(1), 1-43, https://doi.org/10.1257/mic.20160304

Arechar, A.A., Dreber, A., Fudenberg, D., Rand, D.G. (2017). I'm just a soul whose intentions are good?: The role of communication in noisy repeated games. *Games and Economic Behavior*, *104*, 726–743, https://doi.org/10.1016/j.geb.2017.06.013

Backhaus, T., & Breitmoser, Y. (2018). God does not play dice, but do we? on the determinism of choice in long-run interactions [Discussion Paper]. *TRR190 Working Paper*, 96,

Bland, J.R. (2020). Heterogeneous trembles and model selection in the strategy frequency estimation method. *Journal of the Economic Science Association*, *6*(2), 113–124, https://doi.org/https://doi.org/10.1007/s40881-020-00097-y

Breitmoser, Y. (2015). Cooperation, but no reciprocity: Individual strategies in the repeated prisoner's dilemma. *American Economic Review*, *105*(9), 2882–2910, https://doi.org/10.1257/aer.20130675

Camera, G., Casari, M., Bigoni, M. (2012). Cooperative strategies in anonymous economies: An experiment. *Games and Economic Behavior*, *75*(2), 570–586, https://doi.org/10.1016/j.geb.2012.02.009

Dal Bó, P., & Fréchette, G.R. (2011). The evolution of cooperation in infinitely repeated games: Experimental evidence. *American Economic Review*, *101*(1), 411–429, https://doi.org/10.1257/aer.101.1.411

Dempster, A., Laird, N., Rubin, D.B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, *39*(1), 1–38, https://doi.org/10.2307/2984875 arXiv:0710.5696v2

Dvorak, F., & Fehrler, S. (2018). Negotiating cooperation under uncertainty: Communication in noisy, indefinitely repeated interactions. *IZA Working Paper*, 11897, https://doi.org/10.2139/ssrn.3273737

Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, *40*(8), 1–18, https://doi.org/10.18637/jss.v040.i08

Embrey, M., Frechette, G.R., Stacchetti, E. (2013). An experimental study of imperfect public monitoring: Efficiency versus renegotiation-proofness. *SSRN Working Paper*, 2346751, https://doi.org/10.2139/ssrn.2346751

Embrey, M., Frechette, G.R., Yuksel, S. (2017). Cooperation in the finitely repeated prisoner's dilemma. *The Quarterly Journal of Economics*, *133*(1), 509-551, https://doi.org/10.1093/qje/qjx033

Frechette, G.R., & Yuksel, S. (2017). Infinitely repeated games in the laboratory: four perspectives on discounting and random termination. *Experimental Economics*, *20*(2), 279 - 308, https://doi.org/10.1007/s10683-016-9494-z

Fudenberg, D., Rand, D.G., Dreber, A. (2012). Slow to Anger and Fast to Forgive: Cooperation in an Uncertain World. *American Economic Review*, *102*(2), 720–749, https://doi.org/10.1257/aer.102.2.720

Iannone, R. (2016). Diagrammersvg: Export diagrammer graphviz graphs as svg [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=DiagrammeRsvg (R package version 0.1)

Iannone, R. (2020). Diagrammer: Graph/network visualization [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=DiagrammeR (R package version 1.0.6.1)

Kartal, M., & Müller, W. (2022). A new approach to the analysis of cooperation under the shadow of the future: Theory and experimental evidence. *SSRN Working Paper*, 3222964, https://doi.org/10.2139/ssrn.3222964

Kasberger, B., Martin, S., Normann, H.-T., Werner, T. (2023). Algorithmic cooperation. *SSRN Working Paper*, 4389647, https://doi.org/10.2139/ssrn.4389647

Kayaba, Y., Matsushima, H., Toyama, T. (2020). Accuracy and retaliation in repeated games with imperfect private monitoring: Experiments. *Games and Economic Behavior*, *120*, 193-208, https://doi.org/https://doi.org/10.1016/j.geb.2019.12.003

Kloosterman, A. (2020). Cooperation in stochastic games: a prisoner's dilemma experiment. *Experimental Economics*, *23*(2), 447–467, https://doi.org/10.1007/s10683-019-09619-w

Leisch, F. (2002). Compstat. In R.B. Härdle W. (Ed.), (chap. Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis). Physica, Heidelberg.

Meilijson, I. (1989). A fast improvement to the em algorithm on its own terms. *Journal of the Royal Statistical Society B*, *51*(1), 127–138, https://doi.org/10.1111/j.2517-6161.1989.tb01754.x

R Development Core Team (2022). *R: A language and environment for statistical computing* (Tech. Rep.). R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.R-project.org

Rand, D.G., Fudenberg, D., Dreber, A. (2015). It's the thought that counts: The role of intentions in noisy repeated games. *Journal of Economic Behavior & Organization*, *116*, 481-499, https://doi.org/https://doi.org/10.1016/j.jebo.2015.05.013

Romero, J., & Rosokha, Y. (2018). Constructing strategies in the indefinitely repeated prisoner's dilemma game. *European Economic Review*, *104*, 185–219, https://doi.org/https://doi.org/10.1016/j.euroecorev.2018.02.008

Romero, J., & Rosokha, Y. (2019). The evolution of cooperation: The role of costly strategy adjustments. *American Economic Journal: Microeconomics*, *11*(1), 299-328, https://doi.org/10.1257/mic.20160220

Sanderson, C., & Curtin, R. (2016). Armadillo: A template-based c++ library for linear algebra. *Journal of Open Source Software*, *1*, 26, Retrieved from https://CRAN.R-project.org/package=RcppArmadillo

Schwarz, G. (1978). Estimating the dimension of a model. *Ann. Statist.*, *6*(2), 461–464, https://doi.org/10.1214/aos/1176344136

Sherstyuk, K., Tarui, N., Saijo, T. (2013). Payment schemes in infinite-horizon experimental games. *Experimental Economics*, *16*(1), 125–153, https://doi.org/10.1007/s10683-012-9323-y

Stahl, D.O., & Wilson, P.W. (1994). Experimental evidence on players' models of other players. *Journal of Economic Behavior & Organization*, *25*(3), 309 - 327, https://doi.org/10.1016/0167-2681(94)90103-1

Stahl, D.O., & Wilson, P.W. (1995). On players' models of other players: Theory and experimental evidence. *Games and Economic Behavior*, *10*(1), 218 - 254, https://doi.org/10.1006/game.1995.1031

Wickham, H. (2011). testthat: Get started with testing. *The R Journal*, *3*(1), 5–10,

Wickham, H., Danenberg, P., Csardi, G., Eugster, M. (2020). roxygen2: In-line documentation for r [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=roxygen2 (R package version 7.1.0)

Wickham, H., Hester, J., Chang, W. (2020). devtools: Tools to make developing r packages easier [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=devtools (R package version 2.3.0)