Package ‘binaryRL’

June 10, 2025

Version 0.8.7

Title Reinforcement Learning Tools for Two-Alternative Forced Choice Tasks

Description Tools for building reinforcement learning (RL) models specifically tailored for Two-Alternative Forced Choice (TAFC) tasks, commonly employed in psychological research. These models build upon the foundational principles of model-free reinforcement learning detailed in Sutton and Barto (1998) <ISBN:0262039249>. The package allows for the intuitive definition of RL models using simple if-else statements. Our approach to constructing and evaluating these computational models is informed by the guidelines proposed in Wilson & Collins (2019) <doi:10.7554/eLife.49547>. Example datasets included with the package are sourced from the work of Mason et al. (2024) <doi:10.3758/s13423-023-02415-x>.

Maintainer YuKi <hmz1969a@gmail.com>

URL https://github.com/yuki-961004/binaryRL

BugReports https://github.com/yuki-961004/binaryRL/issues

License GPL-3

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.3.2

Depends R (>= 4.0.0)

Imports future, doFuture, foreach, doRNG, progressr

Suggests stats, GenSA, GA, DEoptim, pso, mlrMBO, mlr, ParamHelpers, smoof, lhs, DiceKriging, rgenoud, cmaes, nloptr

NeedsCompilation no

Author YuKi [aut, cre] (ORCID: <https://orcid.org/0009-0000-1378-1318>)

Repository CRAN

Date/Publication 2025-06-10 12:10:07 UTC

Contents

`fit_p` *Step 3: Optimizing parameters to fit real data*

Description

This function optimizes free parameters of reinforcement learning models built with the ‘run_m‘ function. After constructing a reinforcement learning model (a function with only ONE argument, ‘params‘), the ‘fit_p‘ function searches for the optimal values of these free parameters.

The function provides several optimization algorithms:

- 1. L-BFGS-B (from ‘stats::optim‘);
- 2. Simulated Annealing (‘GenSA‘);
- 3. Genetic Algorithm (‘GA‘);
- 4. Differential Evolution (‘DEoptim‘);
- 5. Particle Swarm Optimization (‘pso‘);
- 6. Bayesian Optimization (‘mlrMBO‘);
- 7. Covariance Matrix Adapting Evolutionary Strategy (‘cmaes‘);
- 8. Nonlinear Optimization (‘nloptr‘)

For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
fit_p(
  data,
  id = NULL,
  n_trials = NULL,
  fit_model = list(TD, RSTD, Utility),
  funcs = NULL,
  model_name = c("TD", "RSTD", "Utility"),
  lower = list(c(0, 0), c(0, 0, 0), c(0, 0, 0)),
  upper = list(c(1, 1), c(1, 1, 1), c(1, 1, 1)),
  initial_params = NA,
  initial_size = 50,
  iteration = 10,
  seed = 123,
  nc = 1,
  algorithm
)
```

Arguments

`data` [data.frame] raw data. This data should include the following mandatory columns:

- "sub"
- "time_line" (e.g., "Block", "Trial")
- "L_choice"
- "R_choice"
- "L_reward"
- "R_reward"
- "sub_choose"

`id` [vector] which subject is going to be analyzed. is being analyzed. The value should correspond to an entry in the "sub" column, which must contain the subject IDs. e.g., ‘id = unique(data$Subject)‘

`n_trials` [integer] number of total trials

`fit_model` [list] A collection of functions applied to fit models to the data.

`funcs` [vector] A character vector containing the names of all user-defined functions required for the computation.

`model_name` [list] the name of fit modals

`lower` [list] The lower bounds for model fit models

`upper` [list] The upper bounds for model fit models

`initial_params` [vector] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘L-BFGS-B‘, ‘GenSA‘, set ‘initial = c(0, 0, ...)‘

`initial_size` [integer] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘Bayesian‘, ‘GA‘, set ‘initial = 50‘

`iteration`	[integer] the number of iteration
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘
`nc`	[integer] Number of CPU cores to use for parallel computation.
`algorithm`	[character] Choose an algorithm package from ‘L-BFGS-B‘, ‘GenSA‘, ‘GA‘, ‘DEoptim‘, ‘PSO‘, ‘Bayesian‘, ‘CMA-ES‘. In addition, any algorithm from the ‘nloptr‘ package is also supported. If your chosen ‘nloptr‘ algorithm requires a local search, you need to input a character vector. The first element represents the algorithm used for global search, and the second element represents the algorithm used for local search.

Value

The optimal parameters found by the algorithm for each subject, along with the model fit calculated using these parameters. This is returned as an object of class `binaryRL` containing results for all subjects with all models.

`func_epsilon` *Function: Epsilon Greedy*

Description

Function: Epsilon Greedy

Usage

```
func_epsilon(i, var1 = NA, var2 = NA, threshold = 1, epsilon = NA, lambda)
```

Arguments

`i`	The current row number. The ‘threshold‘ for random selection, which is used to explore the value of different options, will be determined based on this row number. This is because I believe that in the early stages of an experiment, participants will choose options completely at random to explore the reward value associated with each option.
`var1`	[character] column name of extra variable 1. If your model uses more than just reward and expected value, and you need other information, such as whether the choice frame is Gain or Loss, then you can input the ’Frame’ column as var1 into the model. e.g., ‘var1 = "Extra_Var1"‘
`var2`	[character] column name of extra variable 2. If one additional variable, var1, does not meet your needs, you can add another additional variable, var2, into your model. e.g., ‘var2 = "Extra_Var2"‘
`threshold`	[integer] the number of initial trials during which the subject makes random choices rather than choosing based on the values of the options. This occurs because the subject has not yet learned the values of the options. For example, ‘threshold = 20‘ means the subject will make completely random choices for the first 20 trials. default: ‘threshold = 1‘

`epsilon`	[vector] Parameters used in the Exploration Function ‘expl_func‘ determining whether the subject makes decisions based on the relative values of the left and right options, or chooses completely randomly. For example, when epsilon = 0.1, it means the subject has a 10 chance of making a completely random choice and a 90 based on the values of the options. e.g., ‘epsilon = c(0.1)‘
`lambda`	[vector] Extra parameters that may be used in functions. e.g., ‘lambda = c(0.4, 0.7, 20, 60)‘

Value

explore or not

Note

When customizing these functions, please ensure that you do not modify the arguments. Instead, only modify the ‘if-else‘ statements or the internal logic to adapt the function to your needs.

`func_eta` *Function: Learning Rate*

Description

Function: Learning Rate

Usage

```
func_eta(value, utility, reward, occurrence, var1 = NA, var2 = NA, eta, lambda)
```

Arguments

`value`	The expected value of the stimulus in the subject’s mind at this point in time.
`utility`	The subjective value that the subject assigns to the objective reward.
`reward`	The objective reward received by the subject after selecting a stimulus.
`occurrence`	The number of times the same stimulus has appeared.
`var1`	[character] column name of extra variable 1. If your model uses more than just reward and expected value, and you need other information, such as whether the choice frame is Gain or Loss, then you can input the ’Frame’ column as var1 into the model. e.g., ‘var1 = "Extra_Var1"‘
`var2`	[character] column name of extra variable 2. If one additional variable, var1, does not meet your needs, you can add another additional variable, var2, into your model. e.g., ‘var2 = "Extra_Var2"‘
`eta`	[vector] Parameters used in the Learning Rate Function ‘rate_func‘ representing the rate at which the subject updates the difference (prediction error) between the reward and the expected value in the subject’s mind. In the TD model, there is a single learning rate throughout the experiment. In the RSTD model, two different learning rates are used when the reward is higher or lower than the expected value. e.g., ‘eta = c(0.3, 0.7)‘

`lambda`	[vector] Extra parameters that may be used in functions. e.g., ‘lambda = c(0.4, 0.7, 20, 60)‘

Value

learning rate eta

Note

When customizing these functions, please ensure that you do not modify the arguments. Instead, only modify the ‘if-else‘ statements or the internal logic to adapt the function to your needs.

`func_gamma` *Function: Utility Function*

Description

Function: Utility Function

Usage

```
func_gamma(
  value,
  utility,
  reward,
  occurrence,
  var1 = NA,
  var2 = NA,
  gamma = 1,
  lambda
)
```

Arguments

`value`	The expected value of the stimulus in the subject’s mind at this point in time.
`utility`	The subjective value that the subject assigns to the objective reward.
`reward`	The objective reward received by the subject after selecting a stimulus.
`occurrence`	The number of times the same stimulus has appeared.
`var1`	[character] column name of extra variable 1. If your model uses more than just reward and expected value, and you need other information, such as whether the choice frame is Gain or Loss, then you can input the ’Frame’ column as var1 into the model. e.g., ‘var1 = "Extra_Var1"‘
`var2`	[character] column name of extra variable 2. If one additional variable, var1, does not meet your needs, you can add another additional variable, var2, into your model. e.g., ‘var2 = "Extra_Var2"‘

`gamma`	[vector] Parameters used in the Utility Function ‘util_func‘, often referred to as the discount rate. For example, ‘utility = reward^gamma‘. If ‘gamma < 1‘, it indicates that people tend to discount the objective reward. This equation is very similar to the Stevens’ power function, reflecting humans’ nonlinear perception of physical quantities. e.g., ‘gamma = c(0.7)‘.
`lambda`	[vector] Extra parameters that may be used in functions. e.g., ‘lambda = c(0.4, 0.7, 20, 60)‘

Value

Discount rate and utility

Note

When customizing these functions, please ensure that you do not modify the arguments. Instead, only modify the ‘if-else‘ statements or the internal logic to adapt the function to your needs.

`func_tau` *Function: Soft-Max Function*

Description

Function: Soft-Max Function

Usage

```
func_tau(LR, try, L_value, R_value, var1 = NA, var2 = NA, tau = 1, lambda)
```

Arguments

`LR`	Are you calculating the probability for the left option or the right option?
`try`	If the choice was random, the value is 1; if the choice was based on value, the value is 0.
`L_value`	The value of the left option
`R_value`	The value of the right option
`var1`	[character] column name of extra variable 1. If your model uses more than just reward and expected value, and you need other information, such as whether the choice frame is Gain or Loss, then you can input the ’Frame’ column as var1 into the model. e.g., ‘var1 = "Extra_Var1"‘
`var2`	[character] column name of extra variable 2. If one additional variable, var1, does not meet your needs, you can add another additional variable, var2, into your model. e.g., ‘var2 = "Extra_Var2"‘

`tau`	[vector] Parameters used in the Soft-Max Function ‘prob_func‘ representing the sensitivity of the subject to the value difference when making decisions. It determines the probability of selecting the left option versus the right option based on their values. A larger value of tau indicates greater sensitivity to the value difference between the options. In other words, even a small difference in value will make the subject more likely to choose the higher-value option. e.g., ‘tau = c(0.5)‘
`lambda`	[vector] Extra parameters that may be used in functions. e.g., ‘lambda = c(0.4, 0.7, 20, 60)‘

Value

The probability of choosing this option

Note

When customizing these functions, please ensure that you do not modify the arguments. Instead, only modify the ‘if-else‘ statements or the internal logic to adapt the function to your needs.

`Mason_2024_Exp1` *Experiment 1 from Mason et al. (2024)*

Description

This dataset is from Experiment 1 of Mason et al. (2024). (Rare and extreme outcomes in risky choice). Data is publicly available on OSF: https://osf.io/hy3q4/. We performed basic cleaning to meet our package needs.

Format

A data frame with 45000 rows and 11 columns:

Subject Subject ID, an integer (16 to 144)

Block Block number, an integer (1 to 6)

Trial Trial number, an integer (1 to 60)

L_choice Left choice, A = 100% gain 4, B = 90% gain 0 and 10% gain 40, C = 100% lose 4, D = 90% lose 0 and 10% lose 40.

R_choice Right choice, A = 100% gain 4, B = 90% gain 0 and 10% gain 40, C = 100% lose 4, D = 90% lose 0 and 10% lose 40.

L_reward Reward associated with the left choice.

R_reward Reward associated with the right choice.

Sub_Choose The chosen option, either L_choice or R_choice.

Frame Type of frame, "Gain", "Loss", "Catch".

NetWorth The participant’s net worth at the end of each trial.

RT The participant’s reaction time (in milliseconds) for each trial.

Examples

```
# Load the Mason_2024_Exp1 dataset
data(Mason_2024_Exp1)
head(Mason_2024_Exp1)
```

`Mason_2024_Exp2` *Experiment 2 from Mason et al. (2024)*

Description

This dataset is from Experiment 1 of Mason et al. (2024). (Rare and extreme outcomes in risky choice). Data is publicly available on OSF: https://osf.io/hy3q4/. We performed basic cleaning to meet our package needs.

Format

A data frame with 45000 rows and 11 columns:

Subject Subject ID, an integer (16 to 144)

Block Block number, an integer (1 to 6)

Trial Trial number, an integer (1 to 60)

L_choice Left choice, A = 100% gain 36, B = 90% gain 40 and 10% gain 0, C = 100% lose 36, D = 90% lose 40 and 10% lose 0.

R_choice Right choice, A = 100% gain 36, B = 90% gain 40 and 10% gain 0, C = 100% lose 36, D = 90% lose 40 and 10% lose 0.

L_reward Reward associated with the left choice.

R_reward Reward associated with the right choice.

Sub_Choose The chosen option, either L_choice or R_choice.

Frame Type of frame, "Gain", "Loss", "Catch".

NetWorth The participant's net worth at the end of each trial.

RT The participant's reaction time (in milliseconds) for each trial.

Examples

```
# Load the Mason_2024_Exp2 dataset
data(Mason_2024_Exp2)
head(Mason_2024_Exp2)
```

`optimize_para` *Process: Optimizing Parameters*

Description

This function is an internal function of ‘fit_p‘. We isolate it from direct use by capable users.

The function provides several optimization algorithms:

- 1. L-BFGS-B (from ‘stats::optim‘);
- 2. Simulated Annealing (‘GenSA‘);
- 3. Genetic Algorithm (‘GA‘);
- 4. Differential Evolution (‘DEoptim‘);
- 5. Particle Swarm Optimization (‘pso‘);
- 6. Bayesian Optimization (‘mlrMBO‘);
- 7. Covariance Matrix Adapting Evolutionary Strategy (‘cmaes‘);
- 8. Nonlinear Optimization (‘nloptr‘)

For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
optimize_para(
  data,
  id,
  obj_func,
  n_params,
  n_trials,
  lower,
  upper,
  initial_params = NA,
  initial_size = 50,
  iteration = 10,
  seed = 123,
  algorithm
)
```

Arguments

`data`	[data.frame] raw data. This data should include the following mandatory columns: • "sub" • "time_line" (e.g., "Block", "Trial") • "L_choice" • "R_choice" • "L_reward"

- "R_reward"
- "sub_choose"

`id`	[integer] which subject is going to be analyzed. is being analyzed. The value should correspond to an entry in the "sub" column, which must contain the subject IDs. e.g., ‘id = 18‘
`obj_func`	[function] A function with only ONE argument ‘params‘. Refer to ‘binaryRL::TD‘ to mimic the establishment of an objective function.
`n_params`	[integer] The number of free parameters in your model.
`n_trials`	[integer] The total number of trials in your experiment.
`lower`	[vector] lower bounds of free parameters
`upper`	[vector] upper bounds of free parameters
`initial_params`	[vector] Initial values for the free parameters. automatically generate initial values. for ‘L-BFGS-B‘, ‘GenSA‘, set ‘initial = c(0, 0, ...)‘
`initial_size`	[integer] Initial population size for the free parameters. automatically generate initial values. for ‘Bayesian‘, ‘GA‘, set ‘initial = 50‘
`iteration`	[integer] the number of iteration
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘
`algorithm`	[character] Choose an algorithm package from ‘L-BFGS-B‘, ‘GenSA‘, ‘GA‘, ‘DEoptim‘, ‘PSO‘, ‘Bayesian‘, ‘CMA-ES‘. In addition, any algorithm from the ‘nloptr‘ package is also supported. If your chosen ‘nloptr‘ algorithm requires a local search, you need to input a character vector. The first element represents the algorithm used for global search, and the second element represents the algorithm used for local search.

Value

the result of binaryRL with optimal parameters

`rcv_d` *Step 2: Generating fake data for parameter and model recovery*

Description

This function fits multiple sets of simulated data using a loop. You need to provide a list of simulation functions, fitting functions, and parameter bounds. If you prefer to handle the process manually, you can use the internal functions ‘simulate_list‘ and ‘recovery_data‘.

For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
rcv_d(
  data,
  id = NULL,
  n_trials = NULL,
  simulate_models = list(TD, RSTD, Utility),
  simulate_lower = list(c(0, 0), c(0, 0, 0), c(0, 0, 0)),
  simulate_upper = list(c(1, 1), c(1, 1, 1), c(1, 1, 1)),
  fit_models = list(TD, RSTD, Utility),
  fit_lower = list(c(0, 0), c(0, 0, 0), c(0, 0, 0)),
  fit_upper = list(c(1, 1), c(1, 1, 1), c(1, 1, 1)),
  model_names = c("TD", "RSTD", "Utility"),
  funcs = NULL,
  initial_params = NA,
  initial_size = 50,
  iteration_s = 10,
  iteration_f = 10,
  seed = 1,
  nc = 1,
  algorithm
)
```

Arguments

`data`	[data.frame] raw data. This data should include the following mandatory columns: • 1. L-BFGS-B (from ‘stats::optim‘); • 2. Simulated Annealing (‘GenSA‘); • 3. Genetic Algorithm (‘GA‘); • 4. Differential Evolution (‘DEoptim‘); • 5. Particle Swarm Optimization (‘pso‘); • 6. Bayesian Optimization (‘mlrMBO‘); • 7. Covariance Matrix Adapting Evolutionary Strategy (‘cmaes‘); • 8. Nonlinear Optimization (‘nloptr‘)
`id`	[vector] Specifies which subject is being analyzed. In recovery analyses, individual subject information is not needed; trials from the same experimental procedure can be used across all "subjects". Therefore, ‘id‘ can be set to ‘1‘. For example, ‘id = 1‘.
`n_trials`	[integer] number of total trials
`simulate_models`	[list] A collection of functions used to generate simulated data.
`simulate_lower`	[list] The lower bounds for simulate models
`simulate_upper`	[list] The upper bounds for simulate models
`fit_models`	[list] A collection of functions applied to fit models to the data.
`fit_lower`	[list] The lower bounds for model fit models
`fit_upper`	[list] The upper bounds for model fit models

`model_names`	[list] the names of fit modals
`funcs`	[vector] A character vector containing the names of all user-defined functions required for the computation.
`initial_params`	[vector] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘L-BFGS-B‘, ‘GenSA‘, set ‘initial = c(0, 0, ...)‘
`initial_size`	[integer] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘Bayesian‘, ‘GA‘, set ‘initial = 50‘
`iteration_s`	[integer] the number of iteration in simulation (simulate)
`iteration_f`	[integer] the number of iteration in algorithm (fit)
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘
`nc`	[integer] Number of CPU cores to use for parallel computation.
`algorithm`	[character] Choose an algorithm package from ‘L-BFGS-B‘, ‘GenSA‘, ‘GA‘, ‘DEoptim‘, ‘PSO‘, ‘Bayesian‘, ‘CMA-ES‘. In addition, any algorithm from the ‘nloptr‘ package is also supported. If your chosen ‘nloptr‘ algorithm requires a local search, you need to input a character vector. The first element represents the algorithm used for global search, and the second element represents the algorithm used for local search.

Value

A list where each element is a data.frame. Each data.frame within this list records the results of fitting synthetic data (generated by Model A) with Model B.

`recovery_data` *Process: Recovering Fake Data*

Description

This function applies ‘optimize_para‘ to each fake data in the list generated by ‘simulate_list‘. The results can be used for parameter recovery and model recovery, helping evaluate the consistency and validity of the reinforcement learning model.

For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
recovery_data(
  list,
  id = 1,
  fit_model,
  funcs = NULL,
  model_name,
  n_params,
  n_trials,
  lower,
  upper,
  initial_params = NA,
  initial_size = 50,
  iteration = 10,
  seed = 123,
  nc = 1,
  algorithm
)
```

Arguments

`list`	[list] a list generated by function ‘simulate_list‘
`id`	[integer] default = 1
`fit_model`	[function] fit model
`funcs`	[vector] A character vector containing the names of all user-defined functions required for the computation.
`model_name`	[character] the name of your modal
`n_params`	[integer] The number of free parameters in your model.
`n_trials`	[integer] The total number of trials in your experiment.
`lower`	[vector] lower bounds of free parameters
`upper`	[vector] upper bounds of free parameters
`initial_params`	[vector] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘L-BFGS-B‘, ‘GenSA‘, set ‘initial = c(0, 0, ...)‘
`initial_size`	[integer] Initial values for the free parameters. These need to be set only when using L-BFGS-B. Other algorithms automatically generate initial values. for ‘Bayesian‘, ‘GA‘, set ‘initial = 50‘
`iteration`	[integer] the number of iteration
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘
`nc`	[integer] Number of CPU cores to use for parallel computation.
`algorithm`	[character] Choose an algorithm package from ‘L-BFGS-B‘, ‘GenSA‘, ‘GA‘, ‘DEoptim‘, ‘PSO‘, ‘Bayesian‘, ‘CMA-ES‘. In addition, any algorithm from the ‘nloptr‘ package is also supported. If your chosen ‘nloptr‘ algorithm requires a local search, you need to input a character vector. The first element represents the algorithm used for global search, and the second element represents the algorithm used for local search.

Value

a data frame for parameter recovery and model recovery

rpl_e *Step 4: Replaying the experiment with optimal parameters*

Description

This function takes the optimal parameters generated by 'fit_p' and applies them back to each subject's data to generate a new column, 'Rob_Choose'. This allows users to analyze whether the reinforcement learning model can reproduce the original experimental effects observed in the data.

Usage

```
rpl_e(data, result, model, model_name, param_prefix, n_trials = NA)
```

Arguments

`data`	[data.frame] This data should include the following mandatory columns: • "sub" • "time_line" (e.g., "Block", "Trial") • "L_choice" • "R_choice" • "L_reward" • "R_reward" • "sub_choose"
`result`	[data.frame] Output data generated by the 'fit_p()' function. Each row represents model fit results for a subject.
`model`	[function] A model function to be applied in evaluating the experimental effect.
`model_name`	[character] A character string specifying the name of the model to extract from the result.
`param_prefix`	[character] A prefix string used to identify parameter columns in the 'result' data (e.g., "param_").
`n_trials`	[integer] Number of total trials in the experimental task.

Value

A list, where each element is a data.frame representing one subject's results. Each data.frame includes the value update history for each option, the learning rate (`eta`), discount rate (`gamma`), and other relevant information used in each update.

RSTD *Model: RSTD*

Description

Model: RSTD

Usage

```
RSTD(params)
```

Arguments

`params`	[vector] algorithm packages accept only one argument

Value

negative log likelihood

run_m *Step 1: Building reinforcement learning model*

Description

This function requires the optimal parameter values obtained through the ‘algorithm‘ package. Once the best parameter values are solved for, they are incorporated into the reinforcement learning model, allowing the model to simulate human-like decision-making. The function leverages these optimized parameters to generate choices that mimic the decision-making process of subjects, enabling the study of behavior under varying conditions. By integrating the best-fit parameters from the ‘algorithm‘ package, this function offers a powerful tool for simulating human choices in reinforcement learning contexts.

For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
run_m(
  data,
  id,
  mode = "fit",
  initial_value = NA,
  softmax = TRUE,
  threshold = 1,
  seed = 123,
  n_params,
  n_trials,
  gamma = 1,
  eta,
  epsilon = NA,
  tau = 1,
  lambda = NA,
  util_func = func_gamma,
  rate_func = func_eta,
  expl_func = func_epsilon,
  prob_func = func_tau,
  sub = "Subject",
  time_line = c("Block", "Trial"),
  L_choice = "L_choice",
  R_choice = "R_choice",
  L_reward = "L_reward",
  R_reward = "R_reward",
  sub_choose = "Sub_Choose",
  rob_choose = "Rob_Choose",
  raw_cols = NULL,
  var1 = NA,
  var2 = NA,
  digits_1 = 2,
  digits_2 = 5
)
```

Arguments

`data`	[data.frame] raw data. This data should include the following mandatory columns: • "sub" • "time_line" (e.g., "Block", "Trial") • "L_choice" • "R_choice" • "L_reward" • "R_reward" • "sub_choose"
`id`	[integer] which subject is going to be analyzed. is being analyzed. The value should correspond to an entry in the "sub" column, which must contain the subject IDs. e.g., ‘id = 18‘
`mode`	[character] This parameter has three possible values: ‘simulate‘, ‘fit‘, and ‘review‘. These correspond to their use in ‘rcv_d‘, ‘fit_p‘, and ‘rev_e‘ respectively. In most cases, you won’t need to modify this, as suitable default values are set for different contexts.
`initial_value`	[numeric] subject’s initial expected value for each stimulus’s reward. If this value is not set (‘initial_value = NA‘), the subject will use the reward received after the first trial as the initial value for that stimulus. In other words, the learning rate for the first trial is 100 e.g., ‘initial_value = 0‘
`softmax`	[logical] whether to use the softmax function. When ‘softmax = TRUE‘, the value of each option influences the probability of selecting that option. Higher

	values increase the probability of selecting that option. When ‘softmax = FALSE‘, the subject will always choose the option with the higher value, with no possibility of selecting the lower-value option. default: ‘softmax = TRUE‘
`threshold`	[integer] the number of initial trials during which the subject makes random choices rather than choosing based on the values of the options. This occurs because the subject has not yet learned the values of the options. For example, ‘threshold = 20‘ means the subject will make completely random choices for the first 20 trials. default: ‘threshold = 1‘
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘
`n_params`	[integer] The number of free parameters in your model.
`n_trials`	[integer] The total number of trials in your experiment.
`gamma`	[vector] Parameters used in the Utility Function ‘util_func‘, often referred to as the discount rate. For example, ‘utility = reward^gamma‘. If ‘gamma < 1‘, it indicates that people tend to discount the objective reward. This equation is very similar to the Stevens’ power function, reflecting humans’ nonlinear perception of physical quantities. e.g., ‘gamma = c(0.7)‘.
`eta`	[vector] Parameters used in the Learning Rate Function ‘rate_func‘ representing the rate at which the subject updates the difference (prediction error) between the reward and the expected value in the subject’s mind. In the TD model, there is a single learning rate throughout the experiment. In the RSTD model, two different learning rates are used when the reward is higher or lower than the expected value. e.g., ‘eta = c(0.3, 0.7)‘
`epsilon`	[vector] Parameters used in the Exploration Function ‘expl_func‘ determining whether the subject makes decisions based on the relative values of the left and right options, or chooses completely randomly. For example, when epsilon = 0.1, it means the subject has a 10 chance of making a completely random choice and a 90 based on the values of the options. e.g., ‘epsilon = c(0.1)‘
`tau`	[vector] Parameters used in the Soft-Max Function ‘prob_func‘ representing the sensitivity of the subject to the value difference when making decisions. It determines the probability of selecting the left option versus the right option based on their values. A larger value of tau indicates greater sensitivity to the value difference between the options. In other words, even a small difference in value will make the subject more likely to choose the higher-value option. e.g., ‘tau = c(0.5)‘
`lambda`	[vector] Extra parameters that may be used in functions. e.g., ‘lambda = c(0.4, 0.7, 20, 60)‘
`util_func`	[function] Utility Function.
`rate_func`	[function] Learning Rate Function.
`expl_func`	[function] Exploration Function.
`prob_func`	[function] Soft-Max Function.
`sub`	[character] column name of subject ID e.g., ‘sub = "Subject"‘
`time_line`	[vector] A vector specifying the name of the column that the sequence of the experiment. This argument defines how the experiment is structured, such as

	whether it is organized by "Block" with breaks in between, and multiple trials within each block. e.g., ‘time_line = c("Block", "Trial")‘
`L_choice`	[character] column name of left choice. e.g., ‘L_choice = "Left_Choice"‘
`R_choice`	[character] column name of right choice. e.g., ‘R_choice = "Right_Choice"‘
`L_reward`	[character] column name of the reward of left choice e.g., ‘L_reward = "Left_reward"‘
`R_reward`	[character] column name of the reward of right choice e.g., ‘R_reward = "Right_reward"‘
`sub_choose`	[character] column name of choices made by the subject. e.g., ‘sub_choose = "Choose"‘
`rob_choose`	[character] column name of choices made by the model. e.g., ‘rob_choose = "Rob_Choose"‘ you should ignore this argument
`raw_cols`	[vector] Defaults to ‘NULL‘. If left as ‘NULL‘, it will directly capture all column names from the raw data.
`var1`	[character] column name of extra variable 1. If your model uses more than just reward and expected value, and you need other information, such as whether the choice frame is Gain or Loss, then you can input the ’Frame’ column as var1 into the model. e.g., ‘var1 = "Extra_Var1"‘
`var2`	[character] column name of extra variable 2. If one additional variable, var1, does not meet your needs, you can add another additional variable, var2, into your model. e.g., ‘var2 = "Extra_Var2"‘
`digits_1`	[integer] The number of decimal places to retain for columns related to the value function The default is 2.
`digits_2`	[integer] The number of decimal places to retain for columns related to the select function. The default is 5.

Value

A list of class `binaryRL` containing the results of the model fitting.

Examples

```
data <- binaryRL::Mason_2024_Exp1

test <- binaryRL::run_m(
  data = data,
  id = 18,
  eta = c(0.321, 0.765),
  n_params = 2,
  n_trials = 360
)

summary(test)
```

simulate_list *Process: Simulating Fake Data*

Description

This function generates simulated data using a user-defined objective function. You can specify the number of iterations to control how many data are generated. These datasets can be used for parameter recovery and model recovery. For more information, please refer to the GitHub repository: https://github.com/yuki-961004/binaryRL

Usage

```
simulate_list(
  data,
  id = 1,
  obj_func,
  n_params,
  n_trials,
  lower,
  upper,
  iteration = 10,
  seed = 123
)
```

Arguments

`data`	[data.frame] raw data. This data should include the following mandatory columns: - "sub", "time_line", "L_choice", "R_choice", "L_reward", "R_reward".
`id`	[vector] which subject is going to be analyzed. is being analyzed. The value should correspond to an entry in the "sub" column, which must contain the subject IDs. e.g., ‘id = c(1:40)‘
`obj_func`	[function] a function with only ONE argument ‘params‘. Additionally, it is important to note that the data needs to be retrieved from parent.frame(). This function returns the binaryRL.res(res).
`n_params`	[integer] The number of free parameters in your model.
`n_trials`	[integer] The total number of trials in your experiment.
`lower`	[vector] lower bounds of free parameters
`upper`	[vector] upper bounds of free parameters
`iteration`	[integer] the number of iteration
`seed`	[integer] random seed. This ensures that the results are reproducible and remain the same each time the function is run. default: ‘seed = 123‘

Value

a list with fake data generated by random free parameters `summary.binaryRL` *S3method summary*

Description

S3method summary

Usage

```
## S3 method for class 'binaryRL'
summary(object, ...)
```

Arguments

`object`	binaryRL result
`...`	others

Value

summary

`TD` *Model: TD*

Description

Model: TD

Usage

```
TD(params)
```

Arguments

`params`	[vector] algorithm packages accept only one argument

Value

negative log likelihood

`Utility` *Model: Utility*

Description

Model: Utility

Usage

```
Utility(params)
```

Arguments

`params`	[vector] algorithm packages accept only one argument

Value

negative log likelihood

Index