Package: tempodisco (via r-universe)

May 14, 2026

Type Package

Title Temporal Discounting Models

Description Tools for working with temporal discounting data, designed for behavioural researchers to simplify data cleaning/scoring and model fitting. The package implements widely used methods such as computing indifference points from adjusting amount task (Frye et al., 2016, <doi:10.3791/53584>), testing for non-systematic discounting per the criteria of Johnson & Bickel (2008, <doi:10.1037/1064-1297.16.3.264>), scoring questionnaires according to the methods of Kirby et al. (1999, <doi:10.1037//0096-3445.128.1.78>) and Wileyto et al (2004, <doi:10.3758/BF03195548>), Bayesian model selection using a range of discount functions (Franck et al., 2015, <doi:10.1002/jeab.128>), drift diffusion models of discounting (Peters & D'Esposito, 2020, <doi:10.1371/journal.pcbi.1007615>), and model-agnostic measures of discounting such as area under the curve (Myerson et al., 2001, <doi:10.1901/jeab.2001.76-235>) and ED50 (Yoon & Higgins, 2008, <doi:10.1016/j.drugalcdep.2007.12.011>).

Author Isaac Kinley [aut, cre]

Maintainer Isaac Kinley <isaac.kinley@gmail.com>

License GPL-3

Version 2.2.0

RoxygenNote 7.3.3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports stats, graphics, methods, grDevices, RWiener

Suggests knitr, rmarkdown, testthat

Config/testthat/edition 3

VignetteBuilder knitr

URL https://kinleyid.github.io/tempodisco/

BugReports https://github.com/kinleyid/tempodisco/issues

Repository https://kinleyid.r-universe.dev

Date/Publication 2026-03-15 03:14:24 UTC

RemoteUrl https://github.com/kinleyid/tempodisco

RemoteRef HEAD

RemoteSha 8faa10e14730b0f5bb1afb0a527f344d97e72dbb

Contents

`adj_amt_indiffs` *Indifference points from adjusting amount procedure*

Description

Compute indifference points for data from an adjusting amount procedure (also called a "titrating procedure").

Usage

```
adj_amt_indiffs(data, block_indic = "del", order_indic = NULL)
```

Arguments

`data`	A dataframe where each row corresponds to a binary choice, with at least columns `val_imm`, `val_del`, and `imm_chosen`, along with a block indicator and (if applicable) an order indicator.
`block_indic`	Column name of the block indicator—i.e., the column that will identify a block of trials for which an indifference point should be computed. If unspecified, defaults to `'del'`, which assumes that each block corresponds to a different delay.
`order_indic`	Column name of the order indicator—i.e., the column that specifies the order in which trials were completed. Sorting by this column within a block should sort the rows in chronological order. If unspecified, the rows are assumed to already be in chronological order.

Value

A dataframe with two columns: one for the block indicator and another for the corresponding indifference point.

Examples

```
data("adj_amt_sim")
adj_amt_indiffs(adj_amt_sim)
adj_amt_indiffs(adj_amt_sim, block_indic = 'del', order_indic = 'trial_idx')
```

adj_amt_sim *Simulated adjusting amount procedure*

Description

A minimal example of data from a single participant for an adjusting amount procedure.

Author(s)

Isaac Kinley <isaac.kinley@gmail.com>

attention_checks *Test for failed attention checks*

Description

Check whether participants failed attention checks, either choosing an immediate reward of 0 or choosing a delayed reward equal in face value to an immediate reward. If the participant was never offered either choice, a warning is given.

Usage

```
attention_checks(data, warn = FALSE, ppn = FALSE)
```

Arguments

data	A data.frame with columns val_imm, val_del and del_chosen, representing data from a single participant.
warn	Logical: give a warning for failed attention checks?
ppn	Logical: return proportions of attention checks participant failed, versus absolute numbers?

Value

Named vector counting the number of times the participant chose an immediate reward of 0 (imm_0) or chose a delayed reward equal in face value to an immediate reward (del_eq_imm).

Examples

```
# On a model
data("td_bc_single_ptpt")
attention_checks(td_bc_single_ptpt)
```

AUC *Area under the curve (AUC)*

Description

Compute either the model-based or model-free area under the curve.

Usage

```
AUC(
  obj,
  min_del = 0,
  max_del = NULL,
  val_del = NULL,
  del_transform = c("none", "log", "ordinal-scaling"),
  ...
)
```

Arguments

`obj`	A temporal discounting model or a dataframe with columns `indiff` (indifference point) and `del` (delay).
`min_del`	Lower limit to use for integration. Defaults to 0.
`max_del`	Upper limit to use for integration. Defaults to the maximum delay in the data.
`val_del`	Delayed value to use for computing the indifference curve, if applicable. Defaults to the average `del_val` in the data.
`del_transform`	String specifying transformation to apply to the delays (e.g., log10 + 1 transform or ordinal scaling transform; Borges et al., 2016, doi:10.1002/jeab.219). Default is no transform.
`...`	Further arguments passed to 'integrate()'.

Value

AUC value.

Note

An indifference point of 1 is assumed at delay 0.

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = "exponential")
print(AUC(mod))
```

coef.td_bclm *Extract model coefficients*

Description

Get coefficients of a temporal discounting binary choice linear model.

Usage

```
## S3 method for class 'td_bclm'
coef(object, df_par = TRUE, ...)
```

Arguments

object	An object of class td_bcnm.
df_par	Boolean specifying whether the coefficients returned should be the parameters of a discount function (versus the beta parameters from the regression).
...	Additional arguments currently not used.

Value

A named vector of coefficients.

coef.td_bcnm *Extract model coefficients*

Description

Get coefficients of a temporal discounting binary choice nonlinear model.

Usage

```
## S3 method for class 'td_bcnm'
coef(object, ...)
```

Arguments

object	An object of class td_bcnm.
...	Additional arguments currently not used.

Value

A named vector of coefficients.

See Also

Other nonlinear binary choice model functions: deviance.td_bcnm(), fitted.td_bcnm(), logLik.td_bcnm(), predict.td_bcnm(), residuals.td_bcnm(), td_bcnm()

`coef.td_ddm` *Extract model coefficients*

Description

Get coefficients of a temporal discounting drift diffusion model.

Usage

```
## S3 method for class 'td_ddm'
coef(object, type = "all", ...)
```

Arguments

`object`	An object of class td_ddm.
`type`	A string specifying which coefficients to extract. `'all'` extracts them all, `'ddm'` extracts only DDM-specific parameters, and `'df'` extracts only discount function parameters.
`...`	Additional arguments currently not used.

Value

A named vector of coefficients.

See Also

Other drift diffusion model functions: deviance.td_ddm(), fitted.td_ddm(), logLik.td_ddm(), predict.td_ddm(), td_ddm()

`coef.td_ipm` *Extract model coefficients*

Description

Get coefficients of a temporal discounting indifference point model.

Usage

```
## S3 method for class 'td_ipm'
coef(object, ...)
```

Arguments

object	An object of class td_ipm
...	Additional arguments currently not used.

Value

A named vector of coefficients.

See Also

Other indifference point model functions: fitted.td_ipm(), logLik.td_ipm(), predict.td_ipm(), residuals.td_ipm()

---

deviance.td_bcnm *Model deviance*

Description

Compute deviance for a temporal discounting binary choice model.

Usage

```
## S3 method for class 'td_bcnm'
deviance(object, ...)
```

Arguments

object	An object of class td_bcnm.
...	Additional arguments currently not used.

Value

The value of the deviance extracted from the model

See Also

Other nonlinear binary choice model functions: coef.td_bcnm(), fitted.td_bcnm(), logLik.td_bcnm(), predict.td_bcnm(), residuals.td_bcnm(), td_bcnm()

deviance.td_ddm *Model deviance*

Description

Compute deviance for a temporal discounting drift diffusion model.

Usage

```
## S3 method for class 'td_ddm'
deviance(object, ...)
```

Arguments

object	An object of class td_ddm.
...	Additional arguments currently not used.

Value

The value of the deviance extracted from the model

See Also

Other drift diffusion model functions: coef.td_ddm(), fitted.td_ddm(), logLik.td_ddm(), predict.td_ddm(), td_ddm()

discount_function *Get discount function from model*

Description

Access the name of the discount function of a model.

Usage

```
discount_function(mod)
```

Arguments

mod	A temporal discounting model.

Value

The name of the discount function.

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = "exponential")
discount_function(mod)
```

`ED50` *Median effective delay*

Description

Compute the median effective delay.

Usage

```
ED50(mod, val_del = NULL)
```

Arguments

`mod`	A temporal discounting model.
`val_del`	Delayed value, if applicable (i.e., if magnitude effects are accounted for).

Value

Median effective delay value.

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = "exponential")
print(ED50(mod))
```

`fitted.td_bcnm` *Get fitted values*

Description

Get fitted values of a temporal discounting binary choice model.

Usage

```
## S3 method for class 'td_bcnm'
fitted(object, ...)
```

Arguments

object	An object of class td_bcnm.
...	Additional arguments currently not used.

Value

A named vector of fitted values.

See Also

Other nonlinear binary choice model functions: coef.td_bcnm(), deviance.td_bcnm(), logLik.td_bcnm(), predict.td_bcnm(), residuals.td_bcnm(), td_bcnm()

---

fitted.td_ddm *Get fitted values*

Description

Get fitted values of a temporal discounting drift diffusion model.

Usage

```
## S3 method for class 'td_ddm'
fitted(object, ...)
```

Arguments

object	An object of class td_ddm.
...	Additional arguments currently not used.

Value

A named vector of fitted values.

See Also

Other drift diffusion model functions: coef.td_ddm(), deviance.td_ddm(), logLik.td_ddm(), predict.td_ddm(), td_ddm()

fitted.td_ipm *Get fitted values*

Description

Get fitted values of a temporal discounting indifference point model.

Usage

```
## S3 method for class 'td_ipm'
fitted(object, ...)
```

Arguments

object	An object of class td_ipm.
...	Additional arguments currently not used.

Value

A named vector of fitted values.

See Also

Other indifference point model functions: coef.td_ipm(), logLik.td_ipm(), predict.td_ipm(), residuals.td_ipm()

get_available_discount_functions *Get all available pre-defined discount functions*

Description

Get a list of all of the available pre-defined discount functions.

Usage

```
get_available_discount_functions()
```

Value

A character vector containing the names of the avialable pre-defined discount functions.

Examples

```
get_available_discount_functions()
```

indiffs *Get model-free indifference points*

Description

Create a dataframe of delays and the corresponding indifference points predicted by a model.

Usage

```
indiffs(mod)
```

Arguments

mod	A model of class td_bcnm, td_ipm, or td_ddm.

Value

A dataframe with columns del (delay) and indiff (indifference point).

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = 'model-free')
indiff_data <- indiffs(mod)
```

invariance_checks *Check for invariant responding*

Description

Check whether participants always chose the immediate reward or always chose the delayed reward.

Usage

```
invariance_checks(data, warn = FALSE)
```

Arguments

data	A data.frame with columns val_imm, val_del and del_chosen, representing data from a single participant.
warn	Logical: give a warning for invariant responding?

Value

Named vector specifying whether the participant chose only immediate rewards (all_imm) or chose all delayed rewards (all_del).

Examples

```
# On a model
data("td_bc_single_ptpt")
attention_checks(td_bc_single_ptpt)
```

`kirby_consistency` *Compute consistency score*

Description

Compute the consistency score per the method of Kirby et al. (1999, doi:10.1037//00963445.128.1.78). This is described in detail in Kaplan et al. (2016, doi:10.1007/s4061401600709), where it's suggested that a consistency score below 0.75 might be a sign of inattentive responding.

Usage

```
kirby_consistency(
  data,
  discount_function = c("hyperbolic", "exponential", "power", "arithmetic")
)
```

Arguments

`data`	Responses to score.
`discount_function`	Should k values be computed according to the hyperbolic, exponential, power, or arithmetic discount function? The original method uses the hyperbolic, but in principle any single-parameter discount function can also be used (though these should be considered experimental features).

Value

A consistency score between 0 and 1.

Examples

```
data("td_bc_single_ptpt")
mod <- kirby_consistency(td_bc_single_ptpt)
```

kirby_score *Kirby MCQ-style scoring*

Description

Score a set of responses according to the method of Kirby et al. (1999, doi:10.1037//00963445.128.1.78). This is described in detail in Kaplan et al. (2016, doi:10.1007/s4061401600709).

Usage

```
kirby_score(
  data,
  discount_function = c("hyperbolic", "exponential", "power", "arithmetic")
)
```

Arguments

`data`	Responses to score.
`discount_function`	Should k values be computed according to the hyperbolic, exponential, or power discount function? The original method uses the hyperbolic, but in principle the exponential and power are also possible (though these should be considered experimental features).

Value

An object of class `td_ipm`.

Examples

```
data("td_bc_single_ptpt")
mod <- kirby_score(td_bc_single_ptpt)
```

logLik.td_bcnm *Extract log-likelihood*

Description

Compute log-likelihood for a temporal discounting binary choice nonlinear model.

Usage

```
## S3 method for class 'td_bcnm'
logLik(object, ...)
```

Arguments

object	An object of class td_bcnm
...	Additional arguments currently not used.

Value

Returns an object of class logLik with attributed df and nobs

See Also

Other nonlinear binary choice model functions: coef.td_bcnm(), deviance.td_bcnm(), fitted.td_bcnm(), predict.td_bcnm(), residuals.td_bcnm(), td_bcnm()

---

logLik.td_ddm *Extract log-likelihood*

Description

Compute log-likelihood for a temporal discounting drift diffusion model.

Usage

```
## S3 method for class 'td_ddm'
logLik(object, type = c("resp_rt", "resp", "rt"), ...)
```

Arguments

object	An object of class td_bcnm.
type	Should probabilities /probability densities be computed for responses and RTs ('resp_rt', default) or responses only ('resp')?
...	Additional arguments currently not used.

Value

Returns an object of class logLik with attributed df and nobs

See Also

Other drift diffusion model functions: coef.td_ddm(), deviance.td_ddm(), fitted.td_ddm(), predict.td_ddm(), td_ddm()

logLik.td_ipm *Extract log-likelihood*

Description

Compute log-likelihood for a temporal discounting indifference point model.

Usage

```
## S3 method for class 'td_ipm'
logLik(object, ...)
```

Arguments

object	An object of class td_ipm
...	Additional arguments currently not used.

Value

Returns an object of class logLik with attributed df and nobs

See Also

Other indifference point model functions: coef.td_ipm(), fitted.td_ipm(), predict.td_ipm(), residuals.td_ipm()

most_consistent_indiffs *Experimental method for computing indifference points*

Description

Experimental method for computing indifference points

Usage

```
most_consistent_indiffs(data)
```

Arguments

data	Responses to score.

Value

A dataframe with two columns: one for the block indicator and another for the corresponding indifference point.

nonsys *Check for non-systematic discounting*

Description

Check for non-systematic discounting, per the Johnson & Bickel (2008) criteria. These are:

- C1: No indifference point can exceed the previous by more than 0.2
- C2: Last indifference point must be lower than first by at least 0.1

Usage

```
nonsys(obj)
```

Arguments

obj	Either a `data.frame` with columns `indiff` and `del`, or a discounting model of class `td_bcnm` or `td_ipm`, fit using the `"model-free"` discount function.

Value

Named logical vector specifying whether nonsystematic discounting is exhibited according to C1/C2.

Examples

```
# On a model
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = 'model-free')
any(nonsys(mod))

# On a dataframe
data("td_ip_simulated_ptpt")
any(nonsys(td_ip_simulated_ptpt))

# Artificial case of nonsystematic discounting
nonsys(data.frame(del = 1:3, indiff = c(0.5, 0.8, 0.6))) # Both TRUE
```

plot.td_um *Plot models*

Description

Plot delay discounting models.

Usage

```
## S3 method for class 'td_um'
plot(
  x,
  type = c("summary", "endpoints", "link", "rt"),
  legend = TRUE,
  p_lines = NULL,
  p_tol = 0.001,
  verbose = TRUE,
  del = NULL,
  val_del = NULL,
  q_lines = c(0.025, 0.975),
  ...
)
```

Arguments

x	A delay discounting model. See dd_prob_model and dd_det_model.
type	Type of plot to generate.
legend	Logical: display a legend? Only relevant for type = 'summary' and type = 'rt'.
p_lines	Numerical vector. When type = 'summary' the discount curve, where the probability of selecting the immediate reward is 0.5, is plotted. p_lines allows you to specify other probabilities for which similar curves should be plotted (only applicable for probabilistic models, e.g. td_bcnm, td_bclm and td_ddm).
p_tol	If p_lines is not NULL, what is the maximum distance that estimated probabilities can be from their true values? Smaller values results in slower plot generation.
verbose	Whether to print info about, e.g., setting del = ED50 when type = 'endpoints'.
del	Plots data for a particular delay.
val_del	Plots data for a particular delayed value.
q_lines	When type = 'rt', plot dashed lines for which quantiles of the predicted RT distribution? Default is 0.025 and 0.975 (i.e., a quantile-based 95% confidence interval).
...	Additional arguments to plot().

Value

No return value (called to produce a plot)

Examples

```
data("td_bc_single_ptpt")
mod <- td_bclm(td_bc_single_ptpt, model = 'hyperbolic.1')
plot(mod, type = 'summary', p_lines = c(0.25, 0.75), log = 'x')
plot(mod, type = 'endpoints')
```

plot_choices *Plot choices*

Description

Create a plot that displays binary choices at each delay

Usage

```
plot_choices(data, legend = TRUE, ...)
```

Arguments

data	A data frame with columns val_imm and val_del for the values of the immediate and delayed rewards, del for the delay, and imm_chosen (Boolean) for whether the immediate reward was chosen.
legend	Logical: display a legend?
...	Additional arguments to plot().

Value

No return value (called to produce a plot)

Examples

```
data('td_bc_single_ptpt')
plot_choices(td_bc_single_ptpt)
```

predict.td_bclm *Model Predictions*

Description

Generate predictions from a temporal discounting binary choice linear model.

Usage

```
## S3 method for class 'td_bclm'
predict(
  object,
  newdata = NULL,
  type = c("indiff", "link", "response", "terms"),
  ...
)
```

Arguments

object	A temporal discounting binary choice linear model. See td_bclm.
newdata	Optionally, a data frame to use for prediction. If omitted, the data used to fit the model will be used for prediction.
type	The type of prediction required. For 'indiff' (default) gives predicted indifference points. In this case, newdata needs only a del column. For all other values (e.g. "link", "response"), this function is just a wrapper to predict.glm().
...	Additional arguments passed to predict.glm if type != 'indiff'.

Value

A vector of predictions.

See Also

Other linear binary choice model functions: td_bclm()

Examples

```
data("td_bc_single_ptpt")
mod <- td_bclm(td_bc_single_ptpt, model = 'hyperbolic.1')
indiffs <- predict(mod, newdata = data.frame(del = 1:100), type = 'indiff')
```

---

predict.td_bcnm *Model Predictions*

Description

Generate predictions from a temporal discounting binary choice model.

Usage

```
## S3 method for class 'td_bcnm'
predict(object, newdata = NULL, type = c("link", "response", "indiff"), ...)
```

Arguments

object	A temporal discounting binary choice model. See td_bcnm.
newdata	Optionally, a data frame to use for prediction. If omitted, the data used to fit the model will be used for prediction.
type	The type of prediction required. As in predict.glm, "link" (default) and "response" give predictions on the scales of the linear predictors and response variable, respectively. "indiff" gives predicted indifference points. For predicting indifference points, newdata needs only a del column.
...	Additional arguments currently not used.

Value

A vector of predictions.

See Also

Other nonlinear binary choice model functions: `coef.td_bcnm()`, `deviance.td_bcnm()`, `fitted.td_bcnm()`, `logLik.td_bcnm()`, `residuals.td_bcnm()`, `td_bcnm()`

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = 'hyperbolic')
indiffs <- predict(mod, newdata = data.frame(del = 1:100), type = 'indiff')
```

---

predict.td_ddm *Model Predictions*

Description

Generate predictions from a temporal discounting drift diffusion model.

Usage

```
## S3 method for class 'td_ddm'
predict(
  object,
  newdata = NULL,
  type = c("indiff", "link", "response", "rt"),
  ...
)
```

Arguments

`object`	A temporal discounting drift diffusion model. See `td_ddm`.
`newdata`	Optionally, a data frame to use for prediction. If omitted, the data used to fit the model will be used for prediction.
`type`	The type of prediction required. As in predict.glm, `"link"` (default) and `"response"` give predictions on the scales of the linear predictors and response variable, respectively. `"indiff"` gives predicted indifference points. For predicting indifference points, `newdata` needs only a `del` column. `"rt"` gives predicted reaction times.
`...`	Additional arguments currently not used.

Value

A vector of predictions.

Note

When `type = 'rt'`, expected RTs are computed irrespective of which reward was selected, per equation 5 in Grasman, Wagenmakers, & van der Maas (2009, doi:10.1016/j.jmp.2009.01.006).

See Also

Other drift diffusion model functions: `coef.td_ddm()`, `deviance.td_ddm()`, `fitted.td_ddm()`, `logLik.td_ddm()`, `td_ddm()`

Examples

```
data("td_bc_single_ptpt")
ddm <- td_ddm(td_bc_single_ptpt, discount_function = 'exponential',
              gamma_par_starts = 0.01,
              beta_par_starts = 0.5,
              alpha_par_starts = 3.5,
              tau_par_starts = 0.9)
pred_rts <- predict(ddm, type = 'rt')
```

`predict.td_ipm` *Model Predictions*

Description

Generate predictions from a temporal discounting indifference point model

Usage

```
## S3 method for class 'td_ipm'
predict(object, newdata = NULL, type = c("indiff", "response"), ...)
```

Arguments

`object`	A temporal discounting indifference point model. See `td_ipm`.
`newdata`	A data frame to use for prediction. If omitted, the data used to fit the model will be used for prediction.
`type`	Type of prediction, either `'indiff'` (indifference points) or `'response'` (whether the participants would is predicted to choose the immediate (1) or delayed reward (0)).
`...`	Additional arguments currently not used.

Value

A vector of predictions.

See Also

Other indifference point model functions: `coef.td_ipm()`, `fitted.td_ipm()`, `logLik.td_ipm()`, `residuals.td_ipm()`

Examples

```
data("td_ip_simulated_ptpt")
mod <- td_ipm(td_ip_simulated_ptpt, discount_function = 'hyperbolic')
indiffs <- predict(mod, del = 1:100)
indiffs <- predict(mod, newdata = data.frame(del = 1:100))
```

---

residuals.td_bcnm *Residuals from temporal discounting model*

Description

Get residuals from a temporal discounting binary choice nonlinear model.

Usage

```
## S3 method for class 'td_bcnm'
residuals(object, type = c("deviance", "pearson", "response"), ...)
```

Arguments

`object`	A temporal discounting binary choice model. See `td_bcnm`.
`type`	The type of residuals to be returned. See `residuals.glm`.
`...`	Additional arguments currently not used.

Value

A vector of residuals.

See Also

Other nonlinear binary choice model functions: `coef.td_bcnm()`, `deviance.td_bcnm()`, `fitted.td_bcnm()`, `logLik.td_bcnm()`, `predict.td_bcnm()`, `td_bcnm()`

residuals.td_ipm *Residuals from temporal discounting model*

Description

Get residuals from a temporal discounting indifference point model.

Usage

```
## S3 method for class 'td_ipm'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

object	A temporal discounting model. See td_bcnm.
type	The type of residuals to be returned. See residuals.nls.
...	Additional arguments currently not used.

Value

A vector of residuals.

See Also

Other indifference point model functions: coef.td_ipm(), fitted.td_ipm(), logLik.td_ipm(), predict.td_ipm()

td_bc_single_ptpt *Binary choice data for a single participant*

Description

70 binary choices made by a single participant. Along with the columns required by td_bcnm, the reaction time (rt) is recorded.

Author(s)

Isaac Kinley <isaac.kinley@gmail.com>

References

doi:10.31234/osf.io/qjcwn

td_bc_study *Binary choice data for a study*

Description

Data from 421 participants, who each made 70 binary choices. Along with the columns required by `td_bcnm`, the reaction time (`rt`) in seconds is recorded. Participants are identified by the alphnumeric code in the `id` column.

Author(s)

Isaac Kinley <isaac.kinley@gmail.com>

References

doi:10.31234/osf.io/qjcwn

td_bclm *Temporal discounting binary choice linear model*

Description

Compute a binary choice linear model for a single subject. In these models, we can recover the parameters of a discount function from the weights of a standard logistic regression. β_1

Usage

```
td_bclm(
  data,
 model = c("all", "hyperbolic.1", "hyperbolic.2", "exponential.1", "exponential.2",
    "scaled-exponential", "nonlinear-time-hyperbolic", "power",
    "nonlinear-time-exponential", "arithmetic.1", "arithmetic.2"),
  ...
)
```

Arguments

`data`	A data frame with columns `val_imm` and `val_del` for the values of the immediate and delayed rewards, `del` for the delay, and `imm_chosen` (Boolean) for whether the immediate reward was chosen. Other columns can also be present but will be ignored.

`model` A string specifying which model to use. Below is a list of these models' linear predictors and the means by which we can recover discount function parameters.
`'hyperbolic.1'`: $\beta_1(1 - v_D/v_I) + \beta_2 t$; $k = \beta_2/\beta_1$
`'hyperbolic.2'`: $\beta_1(\sigma^{-1}[v_I/v_D] + \log t) + \beta_2$; $k = \exp[\beta_2/\beta_1]$
`'exponential.1'`: $\beta_1 \log(v_I/v_D) + \beta_2 t$; $k = \beta_2/\beta_1$
`'exponential.2'`: $\beta_1(G^{-1}[v_I/v_D] + \log t) + \beta_2$; $k = \exp[\beta_2/\beta_1]$
`'scaled-exponential'`: $\beta_1 \log(v_I/v_D) + \beta_2 t + \beta_3$; $k = \beta_2/\beta_1$, $w = \exp[-\beta_3/\beta_1]$
`'nonlinear-time-hyperbolic'`: $\beta_1(\sigma^{-1}[v_I/v_D]) + \beta_2 \log t + \beta_3$; $k = \exp[\beta_3/\beta_1]$, $s = \beta_2/\beta_1$
`'nonlinear-time-hyperbolic'`: $\beta_1(G^{-1}[v_I/v_D]) + \beta_2 \log t + \beta_3$; $k = \exp[\beta_3/\beta_1]$, $s = \beta_2/\beta_1$
where $\sigma^{-1}[\cdot]$ is the quantile function of the standard logistic distribution $G^{-1}[\cdot]$ is the quantile function of the standard Gumbel distribution.

`...` Additional arguments passed to `glm`.

Value

An object of class `td_bclm`, nearly identical to a `glm` but with an additional `config` component.

See Also

Other linear binary choice model functions: `predict.td_bclm()`

Examples

```
data("td_bc_single_ptpt")
mod <- td_bclm(td_bc_single_ptpt, model = 'hyperbolic.1')
print(coef(mod))
```

---

td_bcnm *Temporal discounting binary choice nonlinear model*

Description

Compute a binary choice model for a single subject

Usage

```
td_bcnm(
  data,
  discount_function = "franck-2015",
  choice_rule = c("logistic", "probit", "power"),
  fixed_ends = FALSE,
  fit_err_rate = FALSE,
  gamma_par_starts = c(0.01, 1, 100),
  eps_par_starts = c(0.01, 0.25),
  silent = TRUE,
```

```
  optim_args = list(),
  ...
)
```

Arguments

data	A data frame with columns val_imm and val_del for the values of the immediate and delayed rewards, del for the delay, and imm_chosen (Boolean) for whether the immediate reward was chosen. Other columns can also be present but will be ignored.
discount_function	A string specifying the name of the discount functions to use, or an object of class td_fn (used for creating custom discount functions), or a list of objects of class td_fn. Default is 'franck-2015', which is the set of widely used discount functions from Franck et al., 2015: https://doi.org/10.1002/jeab.128.
choice_rule	A string specifying whether the 'logistic' (default), 'probit', or 'power' choice rule should be used.
fixed_ends	A Boolean (false by default) specifying whether the model should satisfy the desiderata that subjects should always prefer something over nothing (i.e., nonzero delayed reward over nothing) and the same reward sooner rather than later. See here: https://doi.org/10.1016/j.jmp.2025.102902
fit_err_rate	A Boolean (false by default) specifying whether the model should include an error rate (parameterized by "eps"). See Eq. 5 here: https://doi.org/10.3758/s13428-015-0672-2.
gamma_par_starts	A vector of starting values to try for the "gamma" parameter (which controls the steepness of the choice rule) during optimization.
eps_par_starts	A vector of starting values to try for the "eps" parameter (which controls the error rate) during optimization. Ignored if 'fit_err_rate = FALSE'.
silent	Boolean (true by default). The call to optim() occurs within a try() wrapper. The value of silent is passed along to try().
optim_args	Additional arguments to pass to optim(). Default is list(silent = TRUE).
...	Additional arguments to provide finer-grained control over the choice rule. Note that using a custom choice rule causes the choice_rule and fixed_ends arguments to be ignored.

Value

An object of class td_bcnm with components data (containing the data used for fitting), config (containing the internal configuration of the model, including the discount_function), and optim (the output of optim()).

See Also

Other nonlinear binary choice model functions: coef.td_bcnm(), deviance.td_bcnm(), fitted.td_bcnm(), logLik.td_bcnm(), predict.td_bcnm(), residuals.td_bcnm()

Examples

```
data("td_bc_single_ptpt")
mod <- td_bcnm(td_bc_single_ptpt, discount_function = "hyperbolic", fixed_ends = TRUE)
# Custom discount function
custom_discount_function <- td_fn(
  name = 'custom',
  fn = function(data, p) (1 - p['b'])*exp(-p['k']*data$del) + p['b'],
  par_starts = list(k = c(0.001, 0.1), b = c(0.001, 0.1)),
  par_lims = list(k = c(0, Inf), b = c(0, 1)),
  ED50 = function(...) 'non-analytic'
)
mod <- td_bcnm(td_bc_single_ptpt, discount_function = custom_discount_function, fit_err_rate = TRUE)
```

---

td_ddm *Temporal discounting drift diffusion model*

Description

Fit a drift diffusion model for a single subject using maximum likelihood estimation.

Usage

```
td_ddm(
  data,
  discount_function,
  gamma_par_starts = c(0.01, 0.1, 1),
  beta_par_starts = c(0.25, 0.5, 0.75),
  alpha_par_starts = c(0.5, 1, 10),
  tau_par_starts = c(0.2, 0.8),
  drift_transform = c("none", "logis"),
  bias_adjust = FALSE,
  silent = TRUE,
  optim_args = list()
)
```

Arguments

`data`	A data frame with columns `val_imm` and `val_del` for the values of the immediate and delayed rewards, `del` for the delays, `imm_chosen` (Boolean) for whether the immediate rewards were chosen, and `rt` for the reaction times (in seconds). Other columns can also be present but will be ignored.
`discount_function`	A string specifying the name of the discount functions to use, or an object of class `td_fn` (used for creating custom discount functions), or a list of objects of class `td_fn`.

`gamma_par_starts`
: A vector of starting values to try for the "gamma" parameter (drift rate multiplier or sharpness parameter) during optimization.

`beta_par_starts`
: A vector of starting values to try for the "beta" parameter (bias) during optimization.

`alpha_par_starts`
: A vector of starting values to try for the "alpha" parameter (boundary separation) during optimization.

`tau_par_starts`
: A vector of starting values to try for the "tau" parameter (non-decision time) during optimization.

`drift_transform`
: A transform to apply to drift rates. Either `"none"` (no transform), `"logis"` (sigmoidal transform described by Peters & D'Esposito, 2020, doi:10.1371/journal.pcbi.1007615, and Fontanesi et al., 2019, doi:10.3758/s134230181554-2).

`bias_adjust`
: Experimental feature. See not below.

`silent`
: Boolean (true by default). The call to `optim()` occurs within a `try()` wrapper. The value of `silent` is passed along to `try()`.

`optim_args`
: Additional arguments to pass to `optim()`. Default is `list(silent = TRUE)`.

Value

An object of class `td_bcnm` with components `data` (containing the data used for fitting), `config` (containing the internal configuration of the model, including the `discount_function`), and `optim` (the output of `optim()`).

Note

Drift rates are computed based on the difference in subjective values between the immediate and delayed rewards. In theory, when they are equally valued, they should have equal probability of being chosen. However, this is only true when the bias parameter of the drift diffusion model (`beta`) is 0.5 (i.e., no bias). To make sure the immediate and delayed reward have equal probability of being chosen when they are equally valued, we can set `bias_adjust = TRUE` to add a bias correction factor to the drift rate. However, this feature is experimental and its effects on model fit etc. have not been tested.

See Also

Other drift diffusion model functions: `coef.td_ddm()`, `deviance.td_ddm()`, `fitted.td_ddm()`, `logLik.td_ddm()`, `predict.td_ddm()`

Examples

```
data("td_bc_single_ptpt")
ddm <- td_ddm(td_bc_single_ptpt, discount_function = 'exponential',
              gamma_par_starts = 0.01,
              beta_par_starts = 0.5,
```

```
                alpha_par_starts = 3.5,
                tau_par_starts = 0.9)
```

`td_fn` *Predefined or custom discount function*

Description

Get a predefined discount function or create a custom discount function.

Usage

```
td_fn(
  predefined = c("hyperbolic", "nonlinear-time-hyperbolic", "exponential",
   "nonlinear-time-exponential", "absolute-stationarity", "relative-stationarity",
    "power", "nonlinear-time-power", "arithmetic", "nonlinear-time-arithmetic",
   "inverse-q-exponential", "scaled-exponential", "scaled-hyperbolic", "fixed-cost",
    "dual-systems-exponential", "additive-utility", "model-free", "constant"),
  name = "unnamed",
  fn = NULL,
  par_starts = NULL,
  par_lims = NULL,
  init = NULL,
  ED50 = NULL,
  par_chk = NULL
)
```

Arguments

`predefined`	A string specifying one of the pre-defined discount functions.
`name`	Name of custom discount function.
`fn`	Function that takes a data.frame called `data` (expected to contain the column `del` for delays) and a vector of named parameters called `p`, and returns a vector of values between 0 and 1 representing the indifference points computed at the given delay.
`par_starts`	A named list of vectors, each specifying possible starting values for a parameter to try when running optimization.
`par_lims`	A named list of vectors, each specifying the bounds to impose of a parameters. Any parameter for which bounds are unspecified are assumed to be unbounded.
`init`	A function to initialize the td_fn object. It should take 2 arguments: "self" (the td_fn object being initialized) and "data" (the data used for initialization).
`ED50`	A function which, given a named vector of parameters `p` and optionally a value of `del_val`, computes the ED50. If there is no closed-form solution, this should return the string "non-analytic". If the ED50 is not well-defined, this should return the string "none". As a shortcut for these latter 2 cases, the strings "non-analytic" and "none" can be directly supplied as arguments.

par_chk Optionally, this is a function that checks the parameters to ensure they meet some criteria. E.g., for the dual-systems-exponential discount function, we require k1 < k2.

Value

An object of class td_fn.

Examples

```
data("td_bc_single_ptpt")
# Custom discount function
custom_discount_function <- td_fn(
  name = 'custom',
  fn = function(data, p) (1 - p['b'])*exp(-p['k']*data$del) + p['b'],
  par_starts = list(k = c(0.001, 0.1), b = c(0.001, 0.1)),
  par_lims = list(k = c(0, Inf), b = c(0, 1)),
  ED50 = 'non-analytic'
)
mod <- td_bcnm(td_bc_single_ptpt, discount_function = custom_discount_function, fit_err_rate = TRUE)
```

---

td_ip_simulated_ptpt *Simulated indifference point data for a single participant*

Description

A dataframe containing simulated indifference points from a single participant exhibiting approximately hyperbolic discounting.

Author(s)

Isaac Kinley <isaac.kinley@gmail.com>

---

td_ipm *Temporal discounting indifference point model*

Description

Compute a model of a single subject's indifference points.

Usage

```
td_ipm(
  data,
  discount_function = "franck-2015",
  optim_args = list(),
  silent = TRUE
)
```

Arguments

data	A data frame with columns indiff for the pre-computed indifference points and del for the delay.
discount_function	A vector of strings specifying the name of the discount functions to use, or an object of class td_fn (used for creating custom discount functions), or a list of objects of class td_fn. Default is 'franck-2015', which is the set of widely used discount functions from Franck et al., 2015: https://doi.org/10.1002/jeab.128.
optim_args	A list of additional args to pass to optim.
silent	A Boolean specifying whether the call to optim (which occurs in a try block) should be silent on error.

Value

An object of class td_ipm with components data (containing the data used for fitting), config (containing the internal configuration of the model, including the discount_function), and optim (the output of optim()).

Examples

```
# Basic usage
data("td_ip_simulated_ptpt")
mod <- td_ipm(td_ip_simulated_ptpt, discount_function = "hyperbolic")
# Custom discount function
custom_discount_function <- td_fn(
  name = 'custom',
  fn = function(data, p) (1 - p['b'])*exp(-p['k']*data$del) + p['b'],
  par_starts = list(k = c(0.001, 0.1), b = c(0.001, 0.1)),
  par_lims = list(k = c(0, Inf), b = c(0, 1)),
  ED50 = function(p) 'non-analytic'
)
mod <- td_ipm(td_ip_simulated_ptpt, discount_function = custom_discount_function)
```

---

wileyto_score *Wileyto score a questionnaire*

Description

Score a set of responses according to the method of Wileyto et al. (2004, doi:10.3758/BF03195548). This function is a thin wrapper to td_bclm.

Usage

```
wileyto_score(data)
```

Arguments

data	Responses to score.

Value

An object of class td_bclm.

Examples

```
data("td_bc_single_ptpt")
mod <- wileyto_score(td_bc_single_ptpt)
```

Index