Package 'choicedata'

October 9, 2025

Type Package

Title Working with Choice Data

Version 0.1.0

Description Offers a set of objects tailored to simplify working with choice data. It enables the computation of choice probabilities and the likelihood of various types of choice models based on given data.

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/loelschlaeger/choicedata, http://loelschlaeger.de/choicedata/

BugReports https://github.com/loelschlaeger/choicedata/issues

RoxygenNote 7.3.3

Suggests mlogit, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports checkmate, cli, dplyr, Formula, ggplot2, Matrix, mvtnorm, optimizeR, oeli (>= 0.7.5), patchwork, Rdpack, rlang, tibble, tidyr, utils

RdMacros Rdpack

LazyData true

LazyDataCompression xz

Depends R (>= 4.1.0)

NeedsCompilation no

Author Lennart Oelschläger [aut, cre] (ORCID: <https://orcid.org/0000-0001-5421-9313>)

Maintainer Lennart Oelschläger <oelschlaeger.lennart@gmail.com>

Repository CRAN

Date/Publication 2025-10-09 08:00:08 UTC

Contents

`choiceprob_logit` *Calculate logit choice probabilities*

Description

These helper functions compute logit choice probabilities for unordered and ordered outcomes. Panel inputs reuse the observation-level logit formulae, which remain valid because the logit error term is independent across occasions. Latent class models are supported via weighted averages of class-specific probabilities. When `Omega` is supplied, the coefficients are assumed to follow a multivariate normal distribution and the resulting probabilities are evaluated by averaging over simulation draws.

Usage

```
choiceprob_logit(
  X,
  y = NULL,
  Tp = NULL,
  beta,
  Omega = NULL,
  gamma = NULL,
  weights = NULL,
  input_checks = TRUE,
  ordered = !is.null(gamma),
  ranked = !ordered && !is.null(y) && length(y) > 0 && length(y[[1]]) > 1,
  panel = !is.null(Tp) && any(Tp > 1),
  lc = !is.null(weights),
  draws = NULL,
  n_draws = 200
)
```

Arguments

`X`	`[list(N)]` A `list` of length `N` (number observations) of design matrices, each of dimension `J` (number alternatives) times `P` (number effects). In the ordered case (`ordered = TRUE`), the design matrices are of dimension `1` times `P`.
`y`	`[list(N) | NULL]` A `list` of length `N` (number observations) of single integers from `1` to `J` (number alternatives). In the ranked case (`ranked = TRUE`), the list entries each must be a permutation of `1:J`, where the higher-ranked alternatives are in front. In the non-panel case (`panel = FALSE`), `y` can also be `NULL`, in which case probabilities are calculated for all choice alternatives. In the ranked case (`ranked = TRUE`), if `y` is `NULL`, only first place choice probabilities are computed, which is equivalent to computing choice probabilities in the regular (maximum utility) model.
`Tp`	`[NULL | integer(N)]` The panel identifier of length `N` (number observations) for panel data. The number `Tp[1]` indicates, that the first `Tp[1]` observations in `X` and `y` belong to decider 1, the next `Tp[2]` observations belong to decider 2, and so on. Can be `NULL` for no panel data.
`beta`	`[numeric(P) | list]` The coefficient vector of length `P` (number effects) for computing the systematic utility $V = X\beta$. In the latent class case (`lc = TRUE`), `beta` is a `list` of length `C` of such coefficients, where `C` is the number of latent classes.
`Omega`	`[matrix(nrow = P_r, ncol = P_r) | NULL | list]` The covariance matrix of random effects of dimension `P_r` times `P_r`, where `P_r` less than `P` is the number of random effects. Can be `NULL` for no random effects. In the latent class case (`lc = TRUE`), `Omega` is a `list` of length `C` of such covariance matrices, where `C` is the number of latent classes.
`gamma`	`[NULL | numeric(J - 1)]` Only relevant in the ordered case (`ordered = TRUE`). It defines the non-decreasing boundaries of the utility categories. The event $U \leq \gamma_j$ means that alternative j is chosen, while $U > \gamma_{J-1}$ means that alternative J is chosen.
`weights`	`[NULL | numeric()]` Optional class weights for latent class specifications.
`input_checks`	`[logical(1)]` Perform input checks. Set to `FALSE` to skip them.

`ordered`, `ranked`, `panel`, `lc`	
	`[logical(1)]` Flags indicating whether the specification is ordered, ranked, panel, or latent class. These defaults are inferred from the other inputs so callers typically do not need to override them.
`draws`	`[NULL	matrix	list]` Optional simulation draws for the random coefficients when `Omega` is not `NULL`. A matrix provides shared draws for all classes; a list can supply class-specific draw matrices.
`n_draws`	`[integer(1)]` Number of draws to generate when `draws` is `NULL` and `Omega` is provided.

Value

A numeric vector with the choice probabilities for the observed choices when `y` is supplied. If `y` is `NULL`, a matrix with one row per observation and one column per alternative is returned.

`choiceprob_probit` *Calculate probit choice probabilities*

Description

These helper functions calculate probit choice probabilities for various scenarios:

- in the regular (`choiceprob_mnp_*`), ordered (`*_ordered`), and ranked (`ranked = TRUE`) case,
- in the normally mixed (`choiceprob_mmnp_*`) and latent class (`*_lc`) case,
- for panel data (`*_panel`),
- based on the full likelihood (`cml = "no"`), the full pairwise composite marginal likelihood (`cml = "fp"`), and the adjacent pairwise composite marginal likelihood (`cml = "ap"`),
- for the observed choices or for all alternatives (if `y` is `NULL`).

The function `choiceprob_probit()` is the general API which calls the specialized functions and can perform input checks.

Usage

```
choiceprob_probit(
  X,
  y = NULL,
  Tp = NULL,
  cml = "no",
  beta,
  Omega = NULL,
  Sigma,
  gamma = NULL,
  weights = NULL,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  input_checks = TRUE,
  ordered = !is.null(gamma),
  ranked = if (!ordered && !is.null(y) && isTRUE(length(y) > 0)) {
     length(y[[1]]) >
    1
 } else {
     FALSE
 },
  mixed = !is.null(Omega),
  panel = mixed & !is.null(Tp) & any(Tp > 1),
  lc = !is.null(weights)
)

choiceprob_mnp(
  X,
  y,
  beta,
  Sigma,
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  ranked = FALSE
)

choiceprob_mnp_ordered(X, y, beta, Sigma, gamma, lower_bound = 0)

choiceprob_mmnp(
  X,
  y,
  beta,
  Omega,
  Sigma,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  ranked = FALSE
)

choiceprob_mmnp_ordered(
  X,
  y,
  beta,
  Omega,
  Sigma,
  gamma,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  lower_bound = 0
)

choiceprob_mmnp_lc(
  X,
  y,
  beta,
  Omega,
  Sigma,
  weights,
  re_position = utils::tail(seq_along(beta[[1]]), nrow(Omega[[1]])),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  ranked = FALSE
)

choiceprob_mmnp_ordered_lc(
  X,
  y,
  beta,
  Omega,
  Sigma,
  gamma,
  weights,
  re_position = utils::tail(seq_along(beta[[1]]), nrow(Omega[[1]])),
  lower_bound = 0
)

choiceprob_mmnp_panel(
  X,
  y,
  Tp,
  cml,
  beta,
  Omega,
  Sigma,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  ranked = FALSE
)

choiceprob_mmnp_ordered_panel(
  X,
  y,
  Tp,
  cml,
  beta,
  Omega,
  Sigma,
  gamma,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0
)

choiceprob_mmnp_panel_lc(
  X,
  y,
  Tp,
  cml,
  beta,
  Omega,
  Sigma,
  weights,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0,
  ranked = FALSE
)

choiceprob_mmnp_ordered_panel_lc(
  X,
  y,
  Tp,
  cml,
  beta,
  Omega,
  Sigma,
  gamma,
  weights,
  re_position = utils::tail(seq_along(beta), nrow(Omega)),
  gcdf = pmvnorm_cdf_default,
  lower_bound = 0
)
```

Arguments

X	[list(N)] A list of length N (number observations) of design matrices, each of dimension J (number alternatives) times P (number effects). In the ordered case (ordered = TRUE), the design matrices are of dimension 1 times P.
y	[list(N) | NULL] A list of length N (number observations) of single integers from 1 to J (number alternatives).

In the ranked case (`ranked = TRUE`), the list entries each must be a permutation of `1:J`, where the higher-ranked alternatives are in front.

In the non-panel case (`panel = FALSE`), `y` can also be `NULL`, in which case probabilities are calculated for all choice alternatives. In the ranked case (`ranked = TRUE`), if `y` is `NULL`, only first place choice probabilities are computed, which is equivalent to computing choice probabilities in the regular (maximum utility) model.

`Tp`	`[NULL	integer(N)]` The panel identifier of length `N` (number observations) for panel data. The number `Tp[1]` indicates, that the first `Tp[1]` observations in `X` and `y` belong to decider 1, the next `Tp[2]` observations belong to decider 2, and so on. Can be `NULL` for no panel data.
`cml`	`[character(1)]` The composite marginal likelihood (CML) type for panel data. It can be one of `"no"` (full likelihood), `"fp"` (full pairwise), or `"ap"` (adjacent pairwise).
`beta`	`[numeric(P)	list]` The coefficient vector of length `P` (number effects) for computing the systematic utility $V = X\beta$. In the latent class case (`lc = TRUE`), `beta` is a `list` of length `C` of such coefficients, where `C` is the number of latent classes.
`Omega`	`[matrix(nrow = P_r, ncol = P_r)	NULL	list]` The covariance matrix of random effects of dimension `P_r` times `P_r`, where `P_r` less than `P` is the number of random effects. Can be `NULL` for no random effects. In the latent class case (`lc = TRUE`), `Omega` is a `list` of length `C` of such covariance matrices, where `C` is the number of latent classes.
`Sigma`	`[matrix(nrow = J, ncol = J)	numeric(1)]` The covariance matrix of dimension `J` times `J` (number alternatives) for the Gaussian error term $\epsilon = U - V$. In the ordered case (`ordered = TRUE`), `Sigma` is a single, non-negative `numeric`.
`gamma`	`[NULL	numeric(J - 1)]` Only relevant in the ordered case (`ordered = TRUE`). It defines the non-decreasing boundaries of the utility categories. The event $U \leq \gamma_j$ means that alternative j is chosen, while $U > \gamma_{J-1}$ means that alternative J is chosen.
`weights`	`[NULL	numeric(C)]` The weights for the latent classes in the latent class case (`lc = TRUE`).
`re_position`	`[integer(P_r)]` The index positions of the random effects in the coefficient vector `beta`. By default, the last P_r entries of `beta` are considered as random, where P_r is the dimension of Omega.
`gcdf`	`[function(upper, corr)]` A function that computes (or approximates) the centered Gaussian CDF (mean is zero) based on the upper integration limit `upper` and correlation matrix `corr`. The output is expected to be a single `numeric` value between zero and one.

In the no-panel (`panel = FALSE`) ordered case (`ordered = TRUE`), `stats::pnorm()` is used to calculate the one-dimensional Gaussian CDF.

`lower_bound` [numeric(1)]
A lower bound for the probabilities for numerical reasons. Probabilities are returned as `max(prob, lower_bound)`.

`input_checks` [logical(1)]
Perform input checks. Set to `FALSE` to skip them.

`ordered`, `ranked`, `mixed`, `panel`, `lc`
[logical(1)]
Flags indicating the model type. These are determined automatically based on the input arguments.

Value

A `numeric` `vector` of length `N`, the probabilities for the observed choices `y`.

In the panel case (`panel = TRUE`), the probabilities of the observed choice sequence of length `length(Tp)`.

If `y` is `NULL` and in the non-panel case (`panel = FALSE`), a matrix of dimension `N` times `J`, the probabilities for all alternatives. In the ranked case (`ranked = TRUE`), only first place choice probabilities are computed, which is equivalent to computing choice probabilities in the regular (maximum utility) model.

`choice_alternatives` *Define choice alternatives*

Description

The `choice_alternatives` object defines the set of choice alternatives.

Usage

```
choice_alternatives(
  J = 2,
  alternatives = LETTERS[1:J],
  base = NULL,
  ordered = FALSE
)

## S3 method for class 'choice_alternatives'
print(x, ...)
```

Arguments

`J`	`[integer(1)]` The number ≥ 2 of choice alternatives.
`alternatives`	`[character(J)]` Unique labels for the choice alternatives.
`base`	`[character(1)]` The name of the base alternative for alternative-constant covariates, see details. If `NULL` (default), it is set to the first element of the sorted `alternatives`.
`ordered`	`[logical(1)]` Should the supplied order of `alternatives` be preserved and treated as an intrinsic ranking (for ordered response models)? When `TRUE`, the alternatives are kept in the given order. Otherwise, they are sorted alphabetically.
`x`	`[choice_alternatives]` A `choice_alternatives` object.
`...`	Currently not used.

Value

An object of class `choice_alternatives`, i.e. a `character` vector of the choice alternatives with attributes:

`J` The number of choice alternatives.

`base` The name of the base alternative.

`ordered` Do the alternatives encode an inherent ordering?

Base alternative

The full set of coefficients for covariates that are constant across alternatives (including alternative-specific constants) is not identified. To achieve identifiability, the coefficient of alternative `base` is fixed to zero. The other coefficients then have to be interpreted with respect to `base`. The base alternative is marked with a `*` when printing a `choice_alternatives` object.

Examples

```
choice_alternatives(
  J = 3,
  alternatives = c("gas", "electricity", "oil"),
  base = "gas"
)
```

`choice_covariates` *Define choice covariates*

Description

The `choice_covariates` object defines the choice model covariates.

- `generate_choice_covariates()` samples covariates.
- `covariate_names()` gives the covariate names for given `choice_effects`.
- `design_matrices()` builds design matrices.

Usage

```
choice_covariates(
  data_frame,
  format = "wide",
  column_decider = "deciderID",
  column_occasion = NULL,
  column_alternative = NULL,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  delimiter = "_",
  cross_section = is.null(column_occasion)
)

generate_choice_covariates(
  choice_effects = NULL,
  choice_identifiers = generate_choice_identifiers(N = 100),
  labels = covariate_names(choice_effects),
  n = nrow(choice_identifiers),
  marginals = list(),
  correlation = diag(length(labels)),
  verbose = FALSE,
  delimiter = "_"
)

covariate_names(choice_effects)

design_matrices(
  x,
  choice_effects,
  choice_identifiers = extract_choice_identifiers(x)
)
```

Arguments

`data_frame`	`[data.frame]` Contains the choice covariates.

`format`	`[character(1)]` Format of `data_frame`. Use `"wide"` when covariates for all alternatives are stored in a single row per occasion and `"long"` when each alternative forms a separate row.
`column_decider`	`[character(1)]` Column name with decider identifiers.
`column_occasion`	`[character(1) | NULL]` Column name with occasion identifiers. Set to `NULL` for cross-sectional data.
`column_alternative`	`[character(1) | NULL]` Column name with alternative identifiers when `format = "long"`.
`column_ac_covariates`	`[character() | NULL]` Column names with alternative-constant covariates.
`column_as_covariates`	`[character() | NULL]` Column names with alternative-specific covariates in `data_frame`.
`delimiter`	`[character(1)]` Delimiter separating alternative identifiers from covariate names in wide format. May consist of one or more characters.
`cross_section`	`[logical(1)]` Treat choice data as cross-sectional?
`choice_effects`	`[choice_effects | NULL]` Optional `choice_effects` object used to align covariate labels.
`choice_identifiers`	`[choice_identifiers]` A `choice_identifiers` object describing the simulated panel.
`labels`	`[character()]` Unique labels for the regressors.
`n`	`[integer(1)]` The number of values per regressor.
`marginals`	`[list()]` Optionally marginal distributions for regressors. If not specified, standard normal marginal distributions are used. Each list entry must be named according to a regressor label, and the following distributions are currently supported: **discrete distributions** • Poisson: `list(type = "poisson", lambda = ...)` • categorical: `list(type = "categorical", p = c(...))` **continuous distributions** • normal: `list(type = "normal", mean = ..., sd = ...)` • uniform: `list(type = "uniform", min = ..., max = ...)`
`correlation`	`[matrix()]` A correlation matrix of dimension `length(labels)`, where the `(p, q)`-th entry defines the correlation between regressor `labels[p]` and `labels[q]`.

verbose	[logical(1)] Print information about the simulated regressors?
x	A choice_data or choice_covariates object.

Value

A tibble.

Design matrices

A covariate design matrix contains the choice covariates of a decider at a choice occasion. It is of dimension J x P, where J is the number of choice alternatives and P the number of effects. See compute_P to compute the number P.

Examples

```
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | comfort,
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 3)
)

ids <- generate_choice_identifiers(N = 3, Tp = 2)

choice_covariates <- generate_choice_covariates(
  choice_effects = choice_effects,
  choice_identifiers = ids
)
```

---

choice_data *Define choice data*

Description

The choice_data object defines the choice data, it is a combination of choice_responses and choice_covariates.

Usage

```
choice_data(
  data_frame,
  format = "wide",
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = NULL,
  column_alternative = NULL,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  delimiter = "_",
  cross_section = is.null(column_occasion),
  choice_type = c("discrete", "ordered", "ranked")
)

generate_choice_data(
  choice_effects,
  choice_identifiers = generate_choice_identifiers(N = 100),
  choice_covariates = NULL,
  choice_parameters = NULL,
  choice_preferences = NULL,
  column_choice = "choice",
  choice_type = c("auto", "discrete", "ordered", "ranked")
)

long_to_wide(
  data_frame,
  column_ac_covariates = NULL,
  column_as_covariates = NULL,
  column_choice = "choice",
  column_alternative = "alternative",
  column_decider = "deciderID",
  column_occasion = NULL,
  alternatives = unique(data_frame[[column_alternative]]),
  delimiter = "_",
  choice_type = c("discrete", "ordered", "ranked")
)

wide_to_long(
  data_frame,
  column_choice = "choice",
  column_alternative = "alternative",
  alternatives = NULL,
  delimiter = "_",
  choice_type = c("discrete", "ordered", "ranked")
)
```

Arguments

`data_frame`	`[data.frame]` Contains the choice data.
`format`	`[character(1)]` Format of `data_frame`. Use `"wide"` when each row contains all alternatives of

an occasion and `"long"` when each row contains a single alternative.

`column_choice`	`[character(1)]` Column name with the observed choices. In wide layout this column should contain a single value per observation: for discrete data the value is the label of the chosen alternative, for ordered data it is the ordered factor or integer score, and for ranked data it is omitted in favour of one column per alternative (see `choice_type`). In long layout the same column is evaluated once per alternative: discrete data must use a binary indicator (1 for the chosen alternative, 0 otherwise), ordered data repeats the ordinal value for every alternative, and ranked data stores the integer rank `1:J` for each alternative within an observation. Set to `NULL` when no observed choices are available (e.g., for purely covariate tables).
`column_decider`	`[character(1)]` Column name with decider identifiers.
`column_occasion`	`[character(1) | NULL]` Column name with occasion identifiers. Set to `NULL` in cross-sectional data.
`column_alternative`	`[character(1) | NULL]` Column name with alternative identifiers when `format = "long"`.
`column_ac_covariates`	`[character() | NULL]` Column names with alternative-constant covariates.
`column_as_covariates`	`[character()]` Column names of `data_frame` with alternative-specific covariates.
`delimiter`	`[character(1)]` Delimiter separating alternative identifiers from covariate names in wide format. May consist of one or more characters.
`cross_section`	`[logical(1)]` Treat choice data as cross-sectional?
`choice_type`	`[character(1)]` Requested response type. Use `"auto"` (default) to infer the mode from `choice_alternatives()`, or explicitly simulate `"discrete"`, `"ordered"`, or `"ranked"` outcomes.
`choice_effects`	`[choice_effects]` A `choice_effects` object describing the model.
`choice_identifiers`	`[choice_identifiers]` A `choice_identifiers` object that provides the decider and occasion identifiers.
`choice_covariates`	`[choice_covariates]` Covariates to include in the generated data.
`choice_parameters`	`[choice_parameters]` Model parameters supplying utilities and covariance structures.

`choice_preferences`	`[choice_preferences]` Decider-specific preference draws used for simulation.
`alternatives`	`[character(J)]` Unique labels for the choice alternatives.

Details

`choice_data()` acts as the main entry point for observed data. It accepts either long or wide layouts and performs validation before returning a tidy tibble with consistent identifiers. Columns that refer to the same alternative are aligned using `delimiter` so that downstream helpers can detect them automatically. When used with ranked or ordered choices the function checks that rankings are complete and warns about inconsistencies.

Internally the helper converts long inputs to wide format. This guarantees that subsequent steps (such as computing probabilities) receive the same structure regardless of the original layout and keeps the workflow concise.

- `generate_choice_data()` simulates choice data.
- `wide_to_long()` and `long_to_wide()` transform to wide and long format.

The generated `choice_data` object inherits a `choice_type` attribute for the requested simulation mode. Ordered alternatives (`ordered = TRUE`) yield ordered responses, unordered alternatives default to discrete multinomial outcomes, and ranked simulations return complete rankings for every observation.

Value

A `tibble` that inherits from `choice_data`.

See Also

`choice_responses()`, `choice_covariates()`, and `choice_identifiers()` for the helper objects that feed into `choice_data()`.

Examples

```
### simulate data from a multinomial probit model
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ A | B, error_term = "probit",
    random_effects = c("A" = "cn")
  ),
  choice_alternatives = choice_alternatives(J = 3)
)
generate_choice_data(choice_effects)

### transform between long/wide format
long_to_wide(
  data_frame = travel_mode_choice,
  column_alternative = "mode",
  column_decider = "individual"
)
wide_to_long(
  data_frame = train_choice
)
```

`choice_effects` *Define choice model effects*

Description

This function constructs an object of class `choice_effects`, which defines the effects of a choice model.

Usage

```
choice_effects(
  choice_formula,
  choice_alternatives,
  choice_data = NULL,
  delimiter = "_"
)

## S3 method for class 'choice_effects'
print(x, ...)
```

Arguments

`choice_formula`	[choice_formula] A `choice_formula` object.
`choice_alternatives`	[choice_alternatives] A `choice_alternatives` object.
`choice_data`	[NULL | choice_data] A `choice_data` object. Required to resolve data-dependent elements in `choice_formula` (if any).
`delimiter`	[character(1)] The delimiter between covariate and alternative name.
`x`	[choice_effects] The `choice_effects` object to be printed.
`...`	Currently not used.

Value

A `choice_effects` object, which is a `data.frame`, where each row is a model effect, and columns are

1. `"effect_name"`, the name for the effect which is composed of covariate and alternative name,
2. `"generic_name"`, the generic effect name `"beta_<effect number>"`,
3. `"covariate"`, the (transformed) covariate name connected to the effect,
4. `"alternative"`, the alternative name connected to the effect (only if the effect is alternative-specific),
5. `"as_covariate"`, indicator whether the covariate is alternative-specific,
6. `"as_effect"`, indicator whether the effect is alternative-specific,
7. `"mixing"`, a factor with levels in the order
   (a) `"cn"` (correlated normal distribution),

   indicating the type of random effect.

For identification, the choice effects are ordered according to the following rules:

1. Non-random effects come before random effects.
2. According to the ordering of the factor `mixing`.
3. Otherwise, the order is determined by occurrence in `formula`.

It contains the arguments `choice_formula`, `choice_alternatives`, and `delimiter` as attributes.

Examples

```
choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | I(comfort == 1),
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 3)
)
```

`choice_formula` *Define choice model formula*

Description

The `choice_formula` object defines the choice model equation.

Usage

```
choice_formula(formula, error_term = "probit", random_effects = character())

## S3 method for class 'choice_formula'
print(x, ...)
```

Arguments

`formula`	`[formula]` A symbolic description of the choice model, see details.
`error_term`	`[character(1)]` Defines the model's error term. Current options are: • `"probit"` (default): errors are multivariate normally distributed • `"logit"`: errors follow a type-I extreme value distribution
`random_effects`	`[character()]` Defines the random effects in the model. The expected format of elements in `random_effects` is `"<covariate>" = "<distribution>"`, where `"<covariate>"` is the name of a variable on the `formula` right-hand side. Every random effect must reference an explicit covariate (or `"ASC"` for alternative-specific constants) that appears in the supplied model formula. Current options for `"<distribution>"` are: • `"cn"`: correlated (with other `"cn"` random effects) normal distribution To have random effects for the ASCs, use `"ASC"` for `"<covariate>"`.
`x`	`[choice_formula]` A `choice_formula` object.
`...`	Currently not used.

Value

An object of class `choice_formula`, which is a `list` of the elements:

`formula` The model formula.

`error_term` The name of the model's error term specification.

`choice` The name of the response variable.

`covariate_types` The (up to) three different types of covariates.

`ASC` Does the model have ASCs?

`random_effects` The names of covariates with random effects.

Specifying the model formula

The structure of `formula` is `choice ~ A | B | C`, i.e., a standard `formula` object but with three parts on the right-hand side, separated by `|`, where

- `choice` is the name of the discrete response variable,

- `A` are names of **alternative-specific covariates** with **a coefficient that is constant across alternatives**,
- `B` are names of **covariates that are constant across alternatives**,
- and `C` are names of **alternative-specific covariates** with **alternative-specific coefficients**.

The following rules apply:

1. By default, intercepts (referred to as alternative-specific constants, ASCs) are added to the model. They can be removed by adding `+ 0` in the second part, e.g., `choice ~ A | B + 0 | C`. To not include any covariates of the second type but to estimate ASCs, add `1` in the second part, e.g., `choice ~ A | 1 | C`. The expression `choice ~ A | 0 | C` is interpreted as no covariates of the second type and no ASCs.
2. To not include covariates of any type, add `0` in the respective part, e.g., `choice ~ 0 | B | C`.
3. Some parts of the formula can be omitted when there is no ambiguity. For example, `choice ~ A` is equivalent to `choice ~ A | 1 | 0`.
4. Multiple covariates in one part are separated by a `+` sign, e.g., `choice ~ A1 + A2`.
5. Arithmetic transformations of covariates in all three parts of the right-hand side are possible via the function `I()`, e.g., `choice ~ I(A1^2 + A2 * 2)`. In this case, a random effect can be defined for the transformed covariate, e.g., `random_effects = c("I(A1^2 + A2 * 2)" = "cn")`.

Examples

```
choice_formula(
  formula = choice ~ I(A^2 + 1) | B | I(log(C)),
  error_term = "probit",
  random_effects = c("I(A^2+1)" = "cn", "B" = "cn")
)
```

`choice_identifiers` *Define choice identifiers*

Description

The `choice_identifiers` object defines identifiers for the deciders and choice occasions.

- `generate_choice_identifiers()` generates identifiers.
- `extract_choice_identifiers()` extracts choice identifiers.

Usage

```
choice_identifiers(
  data_frame,
  format = "wide",
  column_decider = "deciderID",
  column_occasion = "occasionID",
  cross_section = FALSE
)

generate_choice_identifiers(
  N = length(Tp),
  Tp = 1,
  column_decider = "deciderID",
  column_occasion = "occasionID"
)

extract_choice_identifiers(
  x,
  format = attr(x, "format"),
  column_decider = attr(x, "column_decider"),
  column_occasion = attr(x, "column_occasion"),
  cross_section = attr(x, "cross_section")
)
```

Arguments

`data_frame`	`[data.frame]` Contains the choice identifiers.
`format`	`[character(1)]` Either `"wide"` or `"long"`. In the long case, unique combinations of `column_decider` and `column_occasion` (if present) are used.
`column_decider`	`[character(1)]` The name of the identifier column for deciders.
`column_occasion`	`[character(1) | NULL]` The name of the identifier column for choice occasions (panel data). Can be `NULL` for the cross-sectional case.
`cross_section`	`[logical(1)]` Treat choice data as cross-sectional?
`N`	`[integer(1)]` The number of deciders.
`Tp`	`[integer(1) | integer(N)]` The number of choice occasions per decider. Can also be of length `N` for a variable number of choice occasions per decider.
`x`	An object of class • `choice_data`, • `choice_covariates`, containing the identifiers and covariate values for each decider, occasion, and alternative,

Value

An object of class `choice_identifiers`, which is a `tibble` with columns:

1. column_decider contains the decider identifiers,
2. column_occasion contains the choice occasion identifiers (only if column_occasion is not NULL and cross_section = FALSE).

Examples

```
### panel case
generate_choice_identifiers(N = 2, Tp = 2)

### cross-sectional case
generate_choice_identifiers(N = 5, column_occasion = NULL)

### read choice identifiers
choice_identifiers(
  data_frame = travel_mode_choice,
  format = "long",
  column_decider = "individual",
  column_occasion = NULL,
  cross_section = TRUE
)
```

---

choice_likelihood *Define and compute choice likelihood*

Description

These functions prepare and evaluate the likelihood contribution of observed choices for a given choice model.

- choice_likelihood() pre-computes the design matrices and choice indices implied by choice_data and choice_effects. The returned object stores these quantities so that repeated likelihood evaluations during maximum likelihood estimation avoid redundant work.
- compute_choice_likelihood() evaluates the (log-)likelihood for given choice_parameters. It can either take the original choice objects or a pre-computed choice_likelihood object.

Usage

```
choice_likelihood(
  choice_data,
  choice_effects,
  choice_identifiers = extract_choice_identifiers(choice_data),
  input_checks = TRUE,
  lower_bound = 1e-10,
  ...
)

compute_choice_likelihood(
  choice_parameters,
  choice_data,
  choice_effects,
  logarithm = TRUE,
  negative = FALSE,
  lower_bound = 1e-10,
  input_checks = TRUE,
  ...
)
```

Arguments

`choice_data`	`[choice_data]` A `choice_data` object with the observed choices.
`choice_effects`	`[choice_effects]` A `choice_effects` object that determines the model effects.
`choice_identifiers`	`[choice_identifiers]` A `choice_identifiers` object. The default extracts identifiers from `choice_data`.
`input_checks`	`[logical(1)]` Forwarded to `choiceprob_probit` or `choiceprob_logit` to control additional input validation.
`lower_bound`	`[numeric(1)]` The minimum probability used when computing the log-likelihood. Values below this bound are truncated to avoid taking the logarithm of zero.
`...`	Additional arguments passed to `choiceprob_probit` or `choiceprob_logit`.
`choice_parameters`	`[choice_parameters | list]` A `choice_parameters` object or a list as returned by `switch_parameter_space`. When a numeric vector in optimization space is supplied, it is converted via `switch_parameter_space()`.
`logarithm`	`[logical(1)]` Return the log-likelihood? If `FALSE`, the likelihood is returned.
`negative`	`[logical(1)]` Return the negative (log-)likelihood? Useful for minimization routines.

Value

`choice_likelihood()` returns an object of class `choice_likelihood`, which is a `list` containing the design matrices, the choice indices, and the identifiers. `compute_choice_likelihood()` returns a single numeric value with the (negative) log-likelihood or likelihood, depending on `logarithm` and `negative`.

Examples

```
data(train_choice)

choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | time,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(
    J = 2, alternatives = c("A", "B")
  )
)

choice_data <- choice_data(
  data_frame = train_choice,
  format = "wide",
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = "occasionID"
)

likelihood <- choice_likelihood(
  choice_data = choice_data,
  choice_effects = choice_effects
)

choice_parameters <- generate_choice_parameters(choice_effects)

compute_choice_likelihood(
  choice_parameters = choice_parameters,
  choice_data = likelihood,
  choice_effects = choice_effects,
  logarithm = TRUE
)
```

`choice_parameters` *Define choice model parameters*

Description

These functions construct, validate, and transform an object of class `choice_parameters`, which defines the parameters of a choice model.

- `choice_parameters()` constructs a `choice_parameters` object.
- `generate_choice_parameters()` samples parameters at random, see details.
- `validate_choice_parameters()` validates a `choice_parameters` object.
- `switch_parameter_space()` transforms a `choice_parameters` object between the interpretation and optimization space, see details.

Usage

```
choice_parameters(beta = NULL, Omega = NULL, Sigma = NULL, gamma = NULL)
```

```
generate_choice_parameters(
  choice_effects,
  fixed_parameters = choice_parameters()
)

validate_choice_parameters(
  choice_parameters,
  choice_effects,
  allow_missing = FALSE
)

switch_parameter_space(choice_parameters, choice_effects)
```

Arguments

`beta`	`[numeric(P)]` The coefficient vector of length `P` (number of effects) for computing the linear-in-parameters systematic utility $V = X\beta$.
`Omega`	`[matrix(nrow = P_r, ncol = P_r) | NULL]` The covariance matrix of random effects of dimension `P_r` times `P_r`, where `P_r` $\leq$ `P` is the number of random effects. Can be `NULL` in the case of `P_r = 0`.
`Sigma`	`[matrix(nrow = J, ncol = J) | numeric(1)]` Only relevant in the probit model. For unordered alternatives it is the covariance matrix of dimension `J` times `J` (number of alternatives) for the Gaussian error term $\epsilon = U - V$. In ordered models it reduces to a single variance term.
`gamma`	`[numeric(J - 1) | NULL]` Optional vector of strictly increasing threshold parameters for ordered models. The first element must equal zero for identification. Ignored for unordered alternatives.
`choice_effects`	`[choice_effects]` A `choice_effects` object describing the utility specification.
`fixed_parameters`	`[choice_parameters]` Optionally a `choice_parameters` object of parameters to keep fixed when sampling other parameters.
`choice_parameters`	`[choice_parameters]` A `choice_parameters` object.
`allow_missing`	`[logical(1)]` Should parameters be allowed to be missing (`TRUE`) or must all required elements be present (`FALSE`)?

Value

An object of class `choice_parameters`, which is a `list` with the elements:

`beta` The coefficient vector (if any).

`Omega` The covariance matrix of random effects (if any).

`Sigma` The error term covariance matrix (or variance in ordered models).

`gamma` Threshold parameters for ordered models (if any).

Sampling missing choice model parameters

Unspecified choice model parameters (if required) are drawn independently from the following distributions:

`beta` Drawn from a multivariate normal distribution with zero mean and a diagonal covariance matrix with value 10 on the diagonal.

`Omega` Drawn from an Inverse-Wishart distribution with degrees of freedom equal to `P_r` + 2 and scale matrix equal to the identity.

`Sigma` The first row and column are fixed to 0 for level normalization. The $(2, 2)$-value is fixed to 1 for scale normalization. The lower right block is drawn from an Inverse-Wishart distribution with degrees of freedom equal to `J` + 1 and scale matrix equal to the identity.

Parameter spaces

The `switch_parameter_space()` function transforms a `choice_parameters` object between the interpretation and optimization space.

- The interpretation space is a `list` of (not necessarily identified) parameters that can be interpreted.
- The optimization space is a `numeric` vector of identified parameters that can be optimized:
  - `beta` is not transformed
  - the first row and column of `Sigma` are fixed to 0 for level normalization and the second diagonal element is fixed to 1 for scale normalization
  - the covariance matrices (`Omega` and `Sigma`) are transformed to their vectorized Cholesky factor (diagonal fixed to be positive for uniqueness)

Examples

```
### generate choice parameters at random
J <- 3
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ A | B, error_term = "probit",
    random_effects = c("A" = "cn")
  ),
  choice_alternatives = choice_alternatives(J = J)
)
choice_parameters <- generate_choice_parameters(
  choice_effects = choice_effects,
  fixed_parameters = choice_parameters(
    Sigma = diag(c(0, rep(1, J - 1))) # scale and level normalization
  )
)
```

choice_preferences *Define choice preferences*

Description

The choice_preferences object defines the deciders' preferences in the choice model.

- choice_preferences() constructs a choice_preferences object.
- generate_choice_preferences() samples choice preferences at random.

Usage

```
choice_preferences(data_frame, column_decider = colnames(data_frame)[1])

generate_choice_preferences(
  choice_effects,
  choice_parameters = NULL,
  choice_identifiers = generate_choice_identifiers(N = 100)
)
```

Arguments

data_frame	[data.frame] Contains the deciders' preferences.
column_decider	[character(1) | NULL] The column name of data_frame with the decider identifiers. If NULL, decider identifiers are generated.
choice_effects	[choice_effects] A choice_effects object.
choice_parameters	[choice_parameters] A choice_parameters object.
choice_identifiers	[choice_identifiers] A choice_identifiers object.

Value

An object of class choice_preferences, which is a data.frame with the deciders' preferences. The column names are the names of the effects in the choice model. The first column contains the decider identifiers.

Examples

```
### generate choice preferences from choice parameters and effects
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | income | comfort,
    error_term = "probit",
    random_effects = c(
      "price" = "cn",
      "income" = "cn"
    )
  ),
  choice_alternatives = choice_alternatives(J = 3)
)

choice_parameters <- generate_choice_parameters(
  choice_effects = choice_effects
)

ids <- generate_choice_identifiers(N = 4)

(choice_preferences <- generate_choice_preferences(
  choice_parameters = choice_parameters,
  choice_effects = choice_effects,
  choice_identifiers = ids
))

### inspect decider-specific preference vectors
head(choice_preferences)
```

---

choice_probabilities *Define choice probabilities*

Description

The `choice_probabilities` object defines the choice probabilities.

- `compute_choice_probabilities()` calculates the choice probabilities based on the choice parameters and the choice data.

Usage

```
choice_probabilities(
  data_frame,
  choice_only = TRUE,
  column_decider = "deciderID",
  column_occasion = NULL,
  cross_section = FALSE,
  column_probabilities = if (choice_only) "choice_probability"
)
```

```
compute_choice_probabilities(
  choice_parameters,
  choice_data,
  choice_effects,
  choice_only = FALSE,
  input_checks = TRUE,
  ...
)
```

Arguments

`data_frame`	[`data.frame`] Contains the choice probabilities.
`choice_only`	[`logical(1)`] Only the probabilities for the chosen alternatives?
`column_decider`	[`character(1)`] The name of the identifier column for deciders.
`column_occasion`	[`character(1)` | `NULL`] The name of the identifier column for choice occasions (panel data). Can be `NULL` for the cross-sectional case.
`cross_section`	[`logical(1)`] Treat choice data as cross-sectional?
`column_probabilities`	[`character()`] The column name of the `data_frame` with the choice probabilities for all choice alternatives. If `choice_only = TRUE`, it is the name of a single column that contains the probabilities for the chosen alternatives.
`choice_parameters`	[`choice_parameters` | `list`] Either a `choice_parameters` object or a list in optimization space as returned by `switch_parameter_space`.
`choice_data`	[`choice_data`] A `choice_data` object providing responses and covariates.
`choice_effects`	[`choice_effects`] A `choice_effects` object defining the specification.
`input_checks`	[`logical(1)`] Should additional internal input checks be performed before computing the probabilities?
`...`	Passed to the underlying probability computation routine.

Value

A `choice_probabilities` S3 object (a data frame) that stores additional metadata in attributes such as `column_probabilities`, `choice_only`, and the identifier columns. These attributes are used by downstream helpers to reconstruct the original structure.

Examples

```
data(train_choice)
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | time,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(
    J = 2, alternatives = c("A", "B")
  )
)
choice_parameters <- generate_choice_parameters(choice_effects)
choice_data <- choice_data(
  data_frame = train_choice,
  format = "wide",
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = "occasionID"
)
compute_choice_probabilities(
  choice_parameters = choice_parameters,
  choice_data = choice_data,
  choice_effects = choice_effects,
  choice_only = TRUE
)
```

`choice_responses` *Define choice response*

Description

The `choice_responses` object defines the observed discrete responses. Additional response columns (for example ranked choice indicators) are preserved so they can be merged with covariates downstream.

- `generate_choice_responses()` simulates choices

Usage

```
choice_responses(
  data_frame,
  column_choice = "choice",
  column_decider = "deciderID",
  column_occasion = NULL,
  cross_section = FALSE
)

generate_choice_responses(
  choice_effects,
  choice_covariates = generate_choice_covariates(choice_effects = choice_effects),
  choice_parameters = generate_choice_parameters(choice_effects = choice_effects),
  choice_identifiers = extract_choice_identifiers(choice_covariates),
  choice_preferences = generate_choice_preferences(choice_parameters = choice_parameters,
    choice_effects = choice_effects, choice_identifiers = choice_identifiers),
  column_choice = "choice",
  choice_type = c("auto", "discrete", "ordered", "ranked")
)
```

Arguments

`data_frame`	`[data.frame]` Contains the choice responses.
`column_choice`	`[character(1)]` The column name of `data_frame` with the choice responses.
`column_decider`	`[character(1)]` The name of the identifier column for deciders.
`column_occasion`	`[character(1) | NULL]` The name of the identifier column for choice occasions (panel data). Can be `NULL` for the cross-sectional case.
`cross_section`	`[logical(1)]` Treat choice data as cross-sectional?
`choice_effects`	`[choice_effects]` A `choice_effects` object describing the model structure.
`choice_covariates`	`[choice_covariates]` Covariates used to construct utilities.
`choice_parameters`	`[choice_parameters]` Model parameters supplying the mean and covariance components.
`choice_identifiers`	`[choice_identifiers]` Identifiers describing the panel or cross-sectional structure.
`choice_preferences`	`[choice_preferences]` Preference draws to simulate the choices.
`choice_type`	`[character(1)]` The response type to simulate. Use `"auto"` (default) to derive the type from `choice_alternatives`, or explicitly request `"discrete"`, `"ordered"`, or `"ranked"` outcomes.

Value

A `data.frame`.

Examples

```
choice_effects <- choice_effects(
  choice_formula = choice_formula(
    formula = choice ~ price | time,
    error_term = "probit"
  ),
  choice_alternatives = choice_alternatives(J = 5)
)
generate_choice_responses(choice_effects = choice_effects)
```

`compute_P` *Number of model effects*

Description

These helper functions count the number of model effects:

- `compute_P()` returns the total number `P` of model effects.
- `compute_P_d()` returns the number `P_d` of non-random effects.
- `compute_P_r()` returns the number `P_r` of random effects.

Usage

```
compute_P(choice_effects)

compute_P_d(choice_effects)

compute_P_r(choice_effects)
```

Arguments

`choice_effects`	`[choice_effects]` A `choice_effects` object.

Value

An `integer`, the number of model effects.

`train_choice` *Stated Preferences for Train Traveling*

Description

Data set of 2929 stated choices by 235 Dutch individuals deciding between two hypothetical train trip options `"A"` and `"B"` based on the price, the travel time, the number of rail-to-rail transfers (changes), and the level of comfort.

The data were obtained in 1987 by Hague Consulting Group for the National Dutch Railways. Prices were recorded in Dutch guilder and in this data set transformed to Euro at an exchange rate of 2.20371 guilders = 1 Euro.

Usage

```
train_choice
```

Format

A `tibble` with 2929 rows and 11 columns:

deciderID [`integer`] The identifier for the decider.

occasionID [`integer`] The identifier for the choice occasion.

choice [`character`] The chosen alternative, either `"A"` or `"B"`.

price_A [`numeric`] The price for alternative `"A"` in Euro.

time_A [`numeric`] The travel time for alternative `"A"` in hours.

change_A [`integer`] The number of changes for alternative `"A"`.

comfort_A [`factor`] The comfort level for alternative `"A"`, where `0` is the best comfort and `2` the worst.

price_B [`numeric`] The price for alternative `"B"` in Euro.

time_B [`numeric`] The travel time for alternative `"B"` in hours.

change_B [`integer`] The number of changes for alternative `"B"`.

comfort_B [`factor`] The comfort level for alternative `"B"`, where `0` is the best comfort and `2` the worst.

References

Ben-Akiva M, Bolduc D, Bradley M (1993). “Estimation of travel choice models with randomly distributed values of time.” *Transportation Research Record*, **1413**.

`travel_mode_choice` *Revealed Preferences for Travel Mode Choice*

Description

Data set of revealed choices by 210 travelers between Sydney and Melbourne who report their choice between the four travel modes plane, train, bus, or car. The data were collected as part of a 1987 intercity mode choice study.

Usage

```
travel_mode_choice
```

Format

A `tibble` with 840 rows and 8 columns:

individual [`integer`] The identifier for the decider.

mode [`character`] The travel mode.

choice [`integer`] Whether the mode was chosen.

wait [`integer`] The terminal waiting time, 0 for car.

cost [`integer`] The travel cost in dollars.

travel [`integer`] The travel time in minutes.

income [`integer`] The household income in thousand dollars.

size [`integer`] The traveling group size.

References

Ben-Akiva M, Bolduc D, Bradley M (1993). “Estimation of travel choice models with randomly distributed values of time.” *Transportation Research Record*, **1413**.

Index