Package: Rprobit (via r-universe)

November 7, 2026

Type Package

Title Estimation of Multinomial Probit Models

Version 1.0.0

Description Estimation of multinomial probit models based on panel data. It uses the full likelihood as well as the MaCML approach. The multinomial Gaussian CDF is approximated using the Solow-Joe method.

License GPL (>= 3)

Encoding UTF-8

Imports checkmate, ggplot2, graphics, oeli, R6, Rcpp, RprobitB, stats

LinkingTo Rcpp, RcppEigen

Depends R (>= 4.1.0)

Suggests katex, knitr, mlogit, rmarkdown (>= 2.12), testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Dietmar Bauer [aut, cre], Manuel Batram [aut], Sebastian Büscher [aut], Lennart Oelschläger [aut], Thanh Thao Bui [aut]

Maintainer Dietmar Bauer <dietmar.bauer@uni-bielefeld.de>

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

Date/Publication 2026-11-07 20:59:26 UTC

RemoteUrl https://github.com/cran/Rprobit

RemoteRef HEAD

RemoteSha 063898a3d3b66647f25d6841120844d8b5f15e68

Contents

calc_choice_probs . . . 2
choice_data . . . 3
fit . . . 8
initialize . . . 9
ll_cml . . . 10
mod . . . 11
pred_probit_approx . . . 18
Rprobit . . . 18
Rprobit-methods . . . 21
setup . . . 24
SJ . . . 27

Index **29**

`calc_choice_probs` *Calculate Approximated Probit Choice Probability*

Description

Calculate Approximated Probit Choice Probability

Usage

```
calc_choice_probs(X, y, b, Sigma, Omega)
```

Arguments

`X`	[`matrix`] Design matrix for one choice occasion, differenced with respect to the chosen alternative, see `choice_data$build_design(..., difference = TRUE)`.
`y`	[`integer(1)`] Index of chosen alternative.
`b`	[`numeric()`] Coefficient vector.
`Sigma`	[`matrix`] Error covariance matrix.
`Omega`	[`matrix`] Random-effect covariance matrix.

Value

A scalar log-probability for the chosen alternative.

choice_data *Choice Data*

Description

An object of the choice_data R6 class stores the data for choice modeling and can interpret data.frames in wide or long format.

Wide data contain one row per choice occasion. Alternative-varying covariates are supplied in columns with names of the form covariate_alternative, for example price_bus and price_car; alternative-constant covariates are supplied as ordinary columns. The argument column_choice names the column that contains the chosen alternative label. The argument column_decider_id optionally names the decider identifier column. If it is NULL, each row is treated as one decider with one choice occasion. The argument column_occasion_id optionally names a choice occasion identifier column. The argument column_alternative is not used for wide data.

Long data contain one row per alternative and choice occasion. Alternative-varying covariates are supplied as ordinary columns, one value per alternative row; alternative-constant covariates are supplied as columns that are constant within each choice set. The argument column_alternative is required and names the column that contains the alternative labels. The argument column_choice names the column that identifies the chosen row in each choice set, encoded by a logical indicator or a 0/1 numeric indicator. Choice sets are identified by column_decider_id and column_occasion_id when both are supplied, by consecutive complete alternative blocks within each decider when only column_decider_id is supplied, and by consecutive complete alternative blocks when neither is supplied.

For a given mod object,

- choice data can be simulated via the method $simulate(),
- choices can be redrawn via the method $redraw_choices(),
- design matrices are created via the method $build_design().

Active bindings

df (read only) [data.frame]
: Choice data.

format (read only) [character(1)]
: Data format, either "wide" or "long".

alt_names (read only) [character()]
: Alternative labels.

N (read only) [integer(1)]
: Number of deciders.

Tp (read only) [integer()]
: Number of choice occasions per decider.

theta_dgp (read only) [numeric() | NULL]
: Free parameter vector used for simulating the current data. NULL if the object does not contain simulated data.

`choice_counts` (read only) [`table`]
Absolute choice counts.

`choice_frequencies` (read only) [`table`]
Relative choice frequencies.

Methods

Public methods:

- `choice_data$new()`
- `choice_data$simulate()`
- `choice_data$redraw_choices()`
- `choice_data$build_design()`
- `choice_data$print()`
- `choice_data$plot()`
- `choice_data$clone()`

`choice_data$new()`: Create a new `choice_data` object.

Usage:

```
choice_data$new(
  df = data.frame(),
  format = "wide",
  column_decider_id = NULL,
  column_occasion_id = NULL,
  column_choice = "choice",
  column_alternative = NULL
)
```

Arguments:

`df` [`data.frame`]
Choice data.

`format` [`character(1)`]
Format of `df`, either `"wide"` or `"long"`.

`column_decider_id` [`character(1)` | `NULL`]
Decider identifier column in `df`.

`column_occasion_id` [`character(1)` | `NULL`]
Choice occasion identifier column in `df`.
If both `column_decider_id` and `column_occasion_id` are `NULL` and `format = "wide"`, every row is treated as one decider with one choice occasion.
If `column_occasion_id` is `NULL` and `format = "long"`, rows are interpreted as consecutive complete choice sets within each decider.

`column_choice` [`character(1)`]
Choice column in `df`.

`column_alternative` [`character(1)` | `NULL`]
Alternative column. Required for `format = "long"`.

Returns: A new `choice_data` object.

`choice_data$simulate()`: Simulate choice data. Covariates are drawn from a centered normal distribution.

Usage:

```
choice_data$simulate(
  mod,
  theta = stats::rnorm(mod$length_theta),
  N = 100L,
  Tp = 1L,
  sd = 1
)
```

Arguments:

`mod` [`mod`]
: Model object.

`theta` [`numeric()`]
: Free parameter vector.

`N` [`integer(1)`]
: Number of deciders.

`Tp` [`integer()`]
: Number of choice occasions per decider. Either scalar or length `N`.

`sd` [`numeric(1)`]
: Standard deviation of the centered normal distribution used to draw covariates.

Returns: The simulated choice `data.frame`.
It is also stored in the object together with `theta` in the fields `df` and `theta_dgp`.

`choice_data$redraw_choices()`: Redraw choices for existing wide-format covariates.
The `df` argument must be in wide format. It is recommended to use the output of `simulate()` and modify the covariates as desired.

Usage:

```
choice_data$redraw_choices(
  df,
  mod,
  theta = self$theta_dgp,
  column_decider_id = NULL,
  column_occasion_id = NULL,
  column_choice = "choice"
)
```

Arguments:

`df` [`data.frame`]
: Choice data.

`mod` [`mod`]
: Model object.

`theta` [`numeric()`]
: Free parameter vector.

`column_decider_id` [`character(1)|NULL`]
: Decider identifier column in `df`.

`column_occasion_id` [`character(1)` | `NULL`]
Choice occasion identifier column in `df`.
If both `column_decider_id` and `column_occasion_id` are `NULL` and `format = "wide"`, every row is treated as one decider with one choice occasion.
If `column_occasion_id` is `NULL` and `format = "long"`, rows are interpreted as consecutive complete choice sets within each decider.

`column_choice` [`character(1)`]
Choice column in `df`.

Returns: The choice `data.frame` with redrawn choices.
It is also stored in the object together with `theta` in the fields `df` and `theta_dgp`.

`choice_data$build_design()`: Build model design matrices.

Usage:

```
choice_data$build_design(mod, difference = FALSE, check_choice = TRUE)
```

Arguments:

`mod` [`mod`]
Model object.

`difference` [`logical(1)`]
If `TRUE`, subtract the row of the chosen alternative from each design matrix and drop that row.

`check_choice` [`logical(1)`]
If `TRUE`, warn about missing or invalid choices.

Returns: A nested `list`. Each decider element contains `X`, a list of design matrices, and `y`, the chosen alternative indices in `mod$alt_names` order.

`choice_data$print()`: Print a summary of the choice data.

Usage:

```
choice_data$print(
  what = c("header", "data_format", "dimensions", "alternatives", "covariates")
)
```

Arguments:

`what` [`character()`]
Parts to print, any subset of `"header"`, `"data_format"`, `"dimensions"`, `"alternatives"`, and `"covariates"`.

`choice_data$plot()`: Plot choice frequencies.

Usage:

```
choice_data$plot()
```

Returns: A `ggplot2::ggplot()` object.

`choice_data$clone()`: The objects of this class are cloneable with this method.

Usage:

```
choice_data$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### read choice data
if (requireNamespace("mlogit", quietly = TRUE)) {

  ### wide data
  data("Train", package = "mlogit")
  (train_data <- choice_data$new(
    df = Train,
    column_decider_id = "id",
    column_occasion_id = "choiceid"
  ))
  train_data$choice_frequencies

  train_mod <- mod$new(
    formula = choice ~ price + time | 0 | 0,
    alt = length(train_data$alt_names),
    alt_names = train_data$alt_names
  )
  train_design <- train_data$build_design(train_mod)
  train_design[[1]]

  ### long data
  data("ModeCanada", package = "mlogit")
  suppressWarnings(mode_data <- choice_data$new(
    df = subset(ModeCanada, noalt == 3),
    format = "long",
    column_decider_id = "case",
    column_alternative = "alt"
  ))
  mode_data$choice_counts

  mode_mod <- mod$new(
    formula = choice ~ cost + ivt | income | 0,
    alt = length(mode_data$alt_names),
    alt_names = mode_data$alt_names
  )
  mode_design <- mode_data$build_design(mode_mod, difference = TRUE)
  mode_design[[1]] # no information about 'air' available
}

### simulate choice data
sim_mod <- mod$new(
  formula = choice ~ price + time | income | 0,
  re = "time",
  alt = 3,
  alt_names = c("train", "bus", "car")
)
sim_data <- choice_data$new()
df <- sim_data$simulate(mod = sim_mod, N = 100, sd = 2)
head(df)
plot(sim_data)
```

`fit` *Fit an Rprobit Object Without Modifying It in Place*

Description

`fit()` is the functional counterpart to the mutable `object$fit()` method. It deep-clones `object`, fits the clone, and returns the fitted clone, see details.

Usage

```
fit(
  object,
  theta_init = NULL,
  cml_type = "probit",
  ...,
  calc_hess = FALSE,
  verbose = TRUE
)
```

Arguments

`object`	[`Rprobit`] An `Rprobit` object.
`theta_init`	[`numeric()` | `NULL`] Initial parameter vector. If `NULL`, an initialization procedure is used.
`cml_type`	[`character(1)`] Defines the type of composite marginal likelihood used. Possible values: • `"probit"` for the full probit likelihood, • `"FP"` for full pairwise composite marginal likelihood, • `"AP"` for adjacent pairwise composite marginal likelihood, • `"AC"` for cyclical adjacent pairwise composite marginal likelihood.
`...`	Additional arguments passed to `stats::nlm()`.
`calc_hess`	[`logical(1)`] If `TRUE`, calculate the analytical Hessian at the estimate. If `FALSE`, use the outer-product approximation returned by `ll_cml()`. The analytical Hessian can be considerably more expensive to compute.
`verbose`	[`logical(1)`] If `TRUE`, progress messages are shown.

Value

A fitted clone of `object`.

Fitting workflows

Calling `object$fit()` uses the mutable R6 method and stores the fitted quantities in `object` itself. Calling `fit(object)` first creates a deep clone of `object`, then calls `$fit()` on that clone and returns it. Use `fit(object)` when the original `Rprobit` object should remain unchanged. For more details on R6-specific modify-in-place logic, see https://r-statistics.co/R6-Classes-in-R.html.

`initialize` *Initialize nMMNP Model Parameters*

Description

This function computes initial values for all parameters of a normally mixed multinomial probit (nMMNP) model.

Usage

```
initialize(
  data,
  J,
  P,
  ref_alt_index = 1L,
  re_cols = integer(0),
  cluster_max = 10,
  cluster_min_weight = 0.1,
  cluster_value_tolerance = 0.001,
  R = 200,
  B = R%/%2,
  Q = 1,
  N_max = 1000,
  check_inputs = TRUE
)
```

Arguments

`data`	`[list]` A list, each element representing choice occasions by one decider, containing • `X`, a list of design matrices, • `y`, an integer vector, representing the chosen alternatives.
`J`	`[integer(1)]` The number of choice alternatives.
`P`	`[integer(3)]` The numbers of type-1, type-2, and type-3 coefficients.
`ref_alt_index`	`[integer(1)]` Position of the reference alternative for type-2 coefficients.

`re_cols` [integer]
The design matrix column positions connected to Gaussian random effects.

`cluster_max` [integer(1)]
The maximum number of clusters for localization in type-2 initialization.

`cluster_min_weight` [numeric(1)]
The minimum relative cluster size for localization in type-2 initialization.

`cluster_value_tolerance` [numeric(1)]
The minimum absolute cluster value for localization in type-2 initialization.

`R` [integer(1)]
The number of Gibbs iterations for covariance initialization.

`B` [integer(1)]
The length of the burn-in period for covariance initialization.

`Q` [integer(1)]
The thinning factor for covariance initialization.

`N_max` [integer(1)]
The maximum number of deciders used for covariance initialization.

`check_inputs` [logical(1)]
If `TRUE`, inputs are checked.

Value

A `numeric` vector containing

- the coefficient initial values `b`, scaled to have Euclidean norm one, which fixes the common utility scale,
- followed by the Cholesky-root parameters of the Gaussian random-effect covariance matrix `Omega` if any `re_cols` are set,
- and the Cholesky-root parameters of the differenced `(J - 1) x (J - 1)` error-term covariance matrix `Sigma` relative to alternative 1 for level identification.

The computation time in seconds is attached as attribute `sec`.

`ll_cml` *Calculate Approximated Composite Marginal or Full Probit Log-Likelihood*

Description

Calculate Approximated Composite Marginal or Full Probit Log-Likelihood

Usage

```
ll_cml(theta, design_diff, mod_list, cml_type, calc_hess = FALSE)
```

Arguments

`theta`	[`numeric()`] Parameter vector.
`design_diff`	[`list`] Differenced design list as returned by `choice_data$build_design(..., difference = TRUE)`.
`mod_list`	[`list`] Details of the model specification as returned by `mod$mod_list`.
`cml_type`	[`character(1)`] Defines the type of composite marginal likelihood used. Possible values: • `"probit"` for full probit, • `"FP"` for full pairwise, • `"AP"` for adjacent pairwise, • `"AC"` for cyclical adjacent pairwise.
`calc_hess`	[`logical(1)`] If `TRUE`, the returned value also has an analytical `"hessian"` attribute. The outer-product approximation in `"hessian_approx"` is returned regardless of this setting.

Value

A numeric vector containing the negative log-likelihood with `"gradient"`, `"hessian_approx"`, and `"Vhat"` attributes. If `calc_hess = TRUE`, it also has an analytical `"hessian"` attribute.

`mod` *Model Specification*

Description

An object of the `mod` R6 class defines the probit model structure and stores linear restrictions on the parameter blocks b, Ω, and Σ from the utility system

$$U_n = X_n \begin{bmatrix} b_f \\ \beta_{r,n} \end{bmatrix} + \epsilon_n, \quad \beta_{r,n} \sim N(b_r, \Omega), \quad b = \begin{bmatrix} b_f \\ b_r \end{bmatrix}, \quad \epsilon_n \sim N(0, \Sigma),$$

which are represented via a matrix H and a column vector f for each parameter block as follows:

$$b = H_b \theta_b + f_b,$$

$$O = \text{vech}(\text{chol}(\Omega)) = H_O \theta_O + f_O,$$

$$L = \text{vech}(\text{chol}(\Sigma)) = H_L \theta_L + f_L.$$

- A `mod` object can be created via

```
m <- mod$new(
  formula = choice ~ price | 1 | time,  # model formula
  re = c("price", "ASC"),               # random effects
  alt = 3,                              # number of alternatives
  alt_names = c("train", "bus", "car"), # alternative labels (optional)
  ref_alt = "train",                    # reference alternative
  corr_re = TRUE,                       # random effects are correlated
  error_struc = "diag"                  # diagonal error covariance matrix
)
```

with the inputs

- `formula`, a symbolic description of the choice model, see details,
- `re`, the covariates to be connected to random coefficients,
- `alt`, the number of choice alternatives,
- `alt_names`, labels for the choice alternatives,
- `ref_alt`, the reference alternative,
- and `corr_re` and `error_struc` can be used for common Ω and Σ structures, see details.

- The source of truth for the model structure is `formula`, `re`, `alt_names`, and `ref_alt`. These can be changed via their active bindings. The dependent quantities `alt`, `b_count`, `var_names`, `length_theta_b`, `length_theta_O`, `length_theta_L`, and `length_theta` are read-only and are updated internally, together with the dimensions of the H matrices and f vectors. The H matrices and f vectors can be accessed (for example, `m$f_L` for the vector f_L) and modified within the dimensions implied by `formula`, `re`, `alt_names`, and `ref_alt`.
- Warning messages can be suppressed by the active binding `m$suppress_warnings`. Its default is `FALSE`.
- The vector $\theta = \begin{bmatrix}\theta_b & \theta_O & \theta_L\end{bmatrix}^\top$ over which the model is fitted to data can be transformed to the parameters b, Ω, Σ via `m$build_parameter()`. The method `m$build_theta()` reverses this task.
- Restrictions on the model parameters are used to identify a model with respect to level and scale. By default, if `error_struc = "full"`, the first column and row of Σ is fixed to zero and the $(2,2)$-entry $\Sigma_{2,2}$ is fixed to one. The method `m$check_identifiability()` checks whether a specification seems identifiable.

Common covariance structures

The covariance matrices Ω and Σ are parameterized via the lower-triangular Cholesky factors O and L.

For the **random-effect covariance matrix** Ω, the argument `corr_re` controls whether correlations between random effects are included:

- If `corr_re = FALSE`, Ω is diagonal.
- If `corr_re = TRUE`, Ω is fully flexible.

For the **error covariance matrix** Σ, `error_struc` selects one of the following common structures:

- `"full"` creates a flexible covariance matrix. The entry `L[2,2]` of the Cholesky factor is fixed to one for scale identification, the first Cholesky column is fixed to zero, and all remaining Cholesky entries are free.

- `"diag"` creates a diagonal covariance matrix. The first Cholesky diagonal entry is fixed to one, and the remaining diagonal entries are free.
- `"ident"` creates a covariance matrix proportional to the identity. All Cholesky diagonal entries share one common free parameter.
- `"fixed"` fixes the covariance matrix to the identity, i.e. all Cholesky diagonal entries are fixed to one.

Specifying the model formula

The structure of `formula` is `choice ~ A | B | C`, i.e., a standard `formula` object but with (up to) 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`.

Active bindings

`formula [formula]`
: Model formula. Writable; changing it adapts dependent restrictions.

`re [character()]`
: Covariates with random coefficients. Writable; changing it adapts dependent restrictions.

`alt_names [character()]`
: Alternative labels, at least 2. Writable; changing them adapts dependent restrictions.

`ref_alt [character(1)]`
: Reference alternative. Writable; changing it adapts dependent restrictions.

`suppress_warnings [logical(1)]`
: Whether warnings are suppressed. Writable; defaults to `FALSE`.

`f_b [matrix | numeric]`
: Fixed offset vector for `b`. Writable within the current row dimension.

`H_b [matrix]`
: Transformation matrix for `b`. Writable within the current row dimension.

`f_O` [matrix | numeric]
Fixed offset vector for Omega. Writable within the current row dimension.

`H_O` [matrix]
Transformation matrix for Omega. Writable within the current row dimension.

`f_L` [matrix | numeric]
Fixed offset vector for Sigma. Writable within the current row dimension.

`H_L` [matrix]
Transformation matrix for Sigma. Writable within the current row dimension.

`alt` (read only) [integer(1)]
Number of alternatives.

`b_count` (read only) [integer(2)]
Number of fixed and random coefficients.

`var_names` (read only) [list()]
Names of fixed and random coefficients.

`length_theta_b` (read only) [integer(1)]
Number of free parameters for b.

`length_theta_O` (read only) [integer(1)]
Number of free parameters for Omega.

`length_theta_L` (read only) [integer(1)]
Number of free parameters for Sigma.

`length_theta` (read only) [integer(1)]
Total number of free model parameters.

`mod_list` (read only) [list()]
Model components in the list format expected by ll_cml:

- `length_theta_b`, `length_theta_O`, and `length_theta_L`, the number of free parameters in the b, Omega, and Sigma parameter blocks.
- `b_count`, the number of fixed and random coefficients.
- `alt`, the number of choice alternatives.
- `H_b` and `f_b`, the restriction matrix and offset vector for b.
- `H_O` and `f_O`, the restriction matrix and offset vector for `vech(chol(Omega))`.
- `H_L` and `f_L`, the restriction matrix and offset vector for `vech(chol(Sigma))`.

Methods

Public methods:

- `mod$new()`
- `mod$build_parameter()`
- `mod$build_theta()`
- `mod$make_theta_unique()`
- `mod$check_identifiability()`
- `mod$print()`
- `mod$clone()`

`mod$new()`: Create a new `mod` object.

Usage:

```
mod$new(
  formula,
  re = character(),
  alt = 2,
  alt_names = LETTERS[seq_len(alt)],
  ref_alt = alt_names[1],
  corr_re = FALSE,
  error_struc = "full",
  ...
)
```

Arguments:

`formula` [`formula`]
: A symbolic description of the choice model.

`re` [`character()`]
: Covariates with random coefficients.

`alt` [`integer(1)`]
: Number of choice alternatives.

`alt_names` [`character()`]
: Alternative labels. If supplied, its length must match `alt`.

`ref_alt` [`character(1)`]
: Reference alternative. Must match one of `alt_names`.

`corr_re` [`logical(1)`]
: Whether random effects are correlated.

`error_struc` [`character(1)`]
: Error structure, one of:
  - `"diag"`, diagonal matrix `Sigma`. The first Cholesky diagonal entry is fixed to one, and the remaining diagonal entries are free.
  - `"fixed"`, `Sigma` fixed to identity matrix.
  - `"ident"`, `Sigma` proportional to identity matrix. The first Cholesky diagonal entry shares one common free parameter with all remaining diagonal entries.
  - `"full"` (default), fully flexible matrix `Sigma`. The first Cholesky column is fixed to zero for level identification, `L[2,2]` is fixed to one for scale identification, and all remaining Cholesky entries are free.

`...` Optional restriction matrices or fixed offset vectors named `H_b`, `f_b`, `H_O`, `f_O`, `H_L`, and `f_L`. These must have the row dimensions implied by `formula`, `re`, `alt_names`, and `ref_alt`; all user-supplied dimnames are ignored and removed on assignment.

Returns: A new `mod` object.

`mod$build_parameter()`: Transform a free parameter vector to model parameters.

Usage:

```
mod$build_parameter(theta = stats::runif(self$length_theta))
```

Arguments:

`theta` [`numeric`]
A numeric vector containing the free model parameters in the order $\theta = \begin{bmatrix}\theta_b & \theta_O & \theta_L\end{bmatrix}^\top$.

Returns: A named list with the entries `b`, `O`, `Omega_chol`, `Omega`, `L`, `Sigma_chol`, and `Sigma`.

`mod$build_theta()`**:** Transform model parameters to a free parameter vector.
If the supplied parameters cannot be represented exactly under the current restrictions, the returned vector is a least-squares approximation.

Usage:

```
mod$build_theta(b, Omega = NULL, Sigma)
```

Arguments:

`b` [`numeric | matrix`]
Coefficient vector.

`Omega` [`matrix | NULL`]
Random-effect covariance matrix.
Can be `NULL` if there are no random effects.

`Sigma` [`matrix`]
Error-term covariance matrix.

Returns: A numeric vector `theta`.

`mod$make_theta_unique()`**:** Replace arbitrary Cholesky entry signs with a unique representation using positive diagonal entries.

Usage:

```
mod$make_theta_unique(theta)
```

Arguments:

`theta` [`numeric`]
A numeric vector containing the free model parameters in the order $\theta = \begin{bmatrix}\theta_b & \theta_O & \theta_L\end{bmatrix}^\top$.

Returns: A numeric vector representing the same model parameters as `theta`.

`mod$check_identifiability()`**:** Check whether the model specification seems identifiable.

Usage:

```
mod$check_identifiability(tolerance = 1e-06)
```

Arguments:

`tolerance` [`numeric(1)`]
A positive numeric value defining the check tolerance.

Returns: `TRUE` if the numerical rank check indicates that the model is identifiable, otherwise `FALSE`.

`mod$print()`**:** Print a summary of the model specification.

Usage:

```
mod$print(
  what = c("header", "formula", "re", "alt_names", "b_count", "length_theta",
    "utility_equations", "restrictions")
)
```

Arguments:

what [character()]
: Parts to print, any subset of "header", "formula", "re", "alt_names", "b_count", "length_theta", "utility_equations", and "restrictions".

Returns: The mod object, invisibly. Called for its console output side effect.

mod$clone(): The objects of this class are cloneable with this method.

Usage:

```
mod$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# define model structure:
m <- mod$new(
  formula = choice ~ price | 1 | time,
  re = c("price", "ASC"),
  alt = 3,
  alt_names = c("train", "bus", "car"),
  ref_alt = "train",
  corr_re = TRUE,
  error_struc = "diag"
)
print(m)

# transform parameter vector to b, Omega, and Sigma:
theta <- runif(m$length_theta)
parameter <- m$build_parameter(theta)
parameter$b
parameter$Omega
parameter$Sigma

# reverse this operation
theta2 <- m$build_theta(
  b = parameter$b,
  Omega = parameter$Omega,
  Sigma = parameter$Sigma
)
all.equal(theta, theta2)

# is the model specification identified?
m$check_identifiability()
m$H_L <- diag(6)[, 4:6]; m$f_L[4, ] <- 0 # removed scale identification
m$check_identifiability()
```

pred_probit_approx *Predict Probit Choice Probabilities*

Description

Predict Probit Choice Probabilities

Usage

```
pred_probit_approx(theta, design_single, mod_list)
```

Arguments

theta	[numeric()] Parameter vector.
design_single	[list] Design list (not differenced) for one decider, see choice_data$build_design(..., difference = FALSE).
mod_list	[list] Details of the model specification as returned by mod$mod_list.

Value

A matrix of predicted choice probabilities according to the model. The panel dimension is ignored.

Rprobit *Rprobit Object*

Description

An object of the Rprobit R6 class consists of a mod object (defines the model structure) and a choice_data object (interprets choice data).

- Estimation can be performed in place via the method Rprobit$fit() or without modifying the original object via fit().
- Methods can be used to analyze the model fit, see Rprobit-methods.

Fitting workflows

Calling object$fit() uses the mutable R6 method and stores the fitted quantities in object itself. Calling fit(object) first creates a deep clone of object, then calls $fit() on that clone and returns it. Use fit(object) when the original Rprobit object should remain unchanged. For more details on R6-specific modify-in-place logic, see https://r-statistics.co/R6-Classes-in-R.html.

Active bindings

`mod` (mutable) [`mod`]
Model specification object.

`choice_data` (mutable) [`choice_data`]
Choice data object.

`theta_init` (read only) [`numeric()` | `NULL`]
Initial parameter vector used for fitting.

`theta_fit` (read only) [`numeric()` | `NULL`]
Model estimate.

`ll_fit` (read only) [`numeric(1)` | `NULL`]
Log-likelihood value at the model estimate.

`optimization_details` (read only) [`list` | `NULL`]
Details of the numerical optimization. After fitting, this list contains all components returned by `stats::nlm()`, `ll_init` (the log-likelihood at `theta_init`), `init_type` (`"custom"`, `"auto"`, or `"fallback"`), `sec_fit` (the `stats::nlm()` runtime in seconds), and `sec_init` (the `.initialize_theta()` runtime in seconds).

`hessian_fit` (read only) [`matrix` | `NULL`]
Hessian of the negative fitting criterion at `theta_fit`. This is the analytical Hessian when fitting with `calc_hess = TRUE` and an outer-product approximation otherwise.

`Vhat_fit` (read only) [`matrix` | `NULL`]
Estimate of the variance of the score at `theta_fit`.

`theta_vcov` (read only) [`matrix` | `NULL`]
Covariance matrix of `theta_fit`.

Methods

Public methods:

- `Rprobit$new()`
- `Rprobit$fit()`
- `Rprobit$print()`
- `Rprobit$clone()`

`Rprobit$new()`: Create a new `Rprobit` object.

Usage:

```
Rprobit$new(mod, choice_data)
```

Arguments:

`mod` [`mod`]
Model specification object.

`choice_data` [`choice_data`]
Choice data object.

Returns: A new `Rprobit` object.

`Rprobit$fit()`: Fit the model in place.

Usage:

```
Rprobit$fit(
  theta_init = NULL,
  cml_type = "probit",
  ...,
  calc_hess = FALSE,
  verbose = TRUE
)
```

Arguments:

`theta_init` [`numeric()` | `NULL`]
: Initial parameter vector. If `NULL`, an initialization procedure is used.

`cml_type` [`character(1)`]
: Defines the type of composite marginal likelihood used. Possible values:
  - `"probit"` for the full probit likelihood,
  - `"FP"` for full pairwise composite marginal likelihood,
  - `"AP"` for adjacent pairwise composite marginal likelihood,
  - `"AC"` for cyclical adjacent pairwise composite marginal likelihood.

`...` Additional arguments passed to `stats::nlm()`.

`calc_hess` [`logical(1)`]
: If `TRUE`, calculate the analytical Hessian at the estimate. If `FALSE`, use the outer-product approximation returned by `ll_cml()`. The analytical Hessian can be considerably more expensive to compute.

`verbose` [`logical(1)`]
: If `TRUE`, progress messages are shown.

Returns: The `Rprobit` object, invisibly.
The following fields are updated:

- `theta_init` with the initial values,
- `theta_fit` with the estimates,
- `ll_fit` with the log-likelihood value at the estimate,
- `optimization_details` with the output of `stats::nlm()`, the log-likelihood value at the initial values, and initialization and optimization runtimes in seconds,
- `hessian_fit` with either the analytical or approximated Hessian and `Vhat_fit` as returned by `ll_cml()` at the estimate,
- `theta_vcov` with the covariance matrix of `theta_fit`.

`Rprobit$print()`**:** Print a summary of the `Rprobit` object.

Usage:

```
Rprobit$print()
```

`Rprobit$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Rprobit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Rprobit-methods *Rprobit Methods*

Description

Methods for fitted Rprobit objects.

Usage

```
## S3 method for class 'Rprobit'
AIC(object, ..., k = 2)

## S3 method for class 'Rprobit'
BIC(object, ...)

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

## S3 method for class 'Rprobit'
confint(object, parm, level = 0.95, ...)

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

## S3 method for class 'Rprobit'
plot(x, parm = NULL, cdf = FALSE, margin = 0.1, ...)

## S3 method for class 'Rprobit'
predict(
  object,
  newdata = NULL,
  type = c("probabilities", "choice", "frequencies", "confusion"),
  ...
)

## S3 method for class 'Rprobit'
residuals(object, ...)

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

## S3 method for class 'summary.Rprobit'
print(
  x,
  digits = max(3L, getOption("digits") - 3L),
  signif.stars = getOption("show.signif.stars"),
  ...
```

```
)

## S3 method for class 'Rprobit'
vcov(object, ...)
```

Arguments

object	[Rprobit] An Rprobit object.
...	[any] Further arguments passed to or from other methods.
k	[numeric(1)] Numeric penalty per parameter.
parm	[character() | numeric() | NULL] Coefficient names or positions selected by confint() or plot().
level	[numeric(1)] Confidence level.
x	[Rprobit | summary.Rprobit] An Rprobit object for plot() or a summary.Rprobit object for print().
cdf	[logical(1)] If TRUE, plot the cumulative distribution function instead of the probability density function.
margin	[numeric(1)] Non-negative factor by which the plotting range is extended.
newdata	[choice_data | NULL] Optional data for prediction. If NULL, the data stored in object are used.
type	[character(1)] Type of prediction to return: • "probabilities" for all choice probabilities, • "choice" for the probability of the observed choice, • "frequencies" for the frequencies of the predicted choices, • "confusion" for a confusion matrix of observed and predicted choices.
digits	[integer(1)] Minimum number of significant digits used for printing.
signif.stars	[logical(1)] If TRUE, significance stars are printed for the coefficient tests.

Details

AIC() computes $-2\ell + kp$, where ℓ is the fitted log-likelihood, p is the number of free model parameters, and k is the non-negative penalty applied to each free parameter. The default $k = 2$ gives the classical Akaike information criterion.

BIC() computes $-2\ell + \log(n)p$, where n is the total number of choice occasions, that is, sum(object$choice_data$Tp).

coef() returns the regression coefficients b. Coefficient names are derived from objectmodvar_names.

`confint()` constructs Wald confidence intervals for b.

`logLik()` returns the fitted log-likelihood value stored in `object$ll_fit`. The `"df"` attribute is the number of free model parameters, `object$mod$length_theta`, and the `"nobs"` attribute is the total number of choice occasions, `sum(object$choice_data$Tp)`.

`plot()` visualizes the fitted population distribution of one or two random effect coefficients selected by `parm`.

`predict()` computes marginal choice probabilities for each choice occasion.

`residuals()` computes response residuals $r_{ij} = 1(y_i = j) - \hat{p}_{ij}$ for every choice occasion i and alternative j, where $\hat{p}_{ij}$ is the fitted choice probability.

`summary()` summarizes the model results.

`vcov()` returns the covariance matrix of the regression coefficients b, matching the parameters returned by `coef()`.

Value

`AIC()` and `BIC()` return a numeric scalar.

`coef()` returns a named numeric vector of regression coefficients.

`confint()` returns a numeric matrix with one row per coefficient and two columns containing the lower and upper confidence limits.

`logLik()` returns the fitted log-likelihood as a numeric scalar of class `"logLik"` with `"df"` and `"nobs"` attributes.

`plot()` returns a `ggplot2::ggplot` object.

For `predict()`, the return value depends on `type`:

- `"probabilities"` returns a `data.frame` with one row per choice occasion. Its columns contain the predicted probabilities for the alternatives in `object$mod$alt_names` order.
- `"choice"` returns a numeric vector containing the predicted probability of the observed choice for each choice occasion.
- `"frequencies"` returns a `table` of predicted choices.
- `"confusion"` returns a two-dimensional `table` with observed choices in rows and predicted choices in columns.

`residuals()` returns a numeric matrix with one row per choice occasion and one column per alternative in `object$mod$alt_names` order.

`summary()` returns an object of class `"summary.Rprobit"`.

`vcov()` returns a named numeric covariance matrix with one row and column per regression coefficient returned by `coef()`.

setup *Setup for Estimation of Multinomial Probit Models*

Description

This function is usually the starting point for working with the {Rprobit} package. It helps to construct an Rprobit object that is used to fit a multinomial probit model to choice data.

- The probit model is specified via formula, see details.
- If no data is provided (df = NULL), data is simulated.
- If data is provided, setup() checks that design matrices can be built from df for the specified model.
- Estimation can be performed in place via the method Rprobit$fit() or without modifying the original object via fit().
- Methods can be used to analyze the model fit, see Rprobit-methods.

Usage

```
setup(
  formula,
  df = NULL,
  re = character(),
  format = "wide",
  column_decider_id = NULL,
  column_occasion_id = NULL,
  column_choice = NULL,
  column_alternative = NULL,
  alt_names = NULL,
  ref_alt = NULL,
  corr_re = FALSE,
  error_struc = "full",
  N = 100L,
  Tp = 1L,
  sd = 1
)
```

Arguments

formula	[formula] A symbolic description of the choice model, see details.
df	[data.frame | NULL] Empirical choice data. Can be NULL, in which case data is simulated.
re	[character()] Covariates with random coefficients.

`format` [character(1)]
Format of `df`, either `"wide"` or `"long"`.

`column_decider_id`
[character(1) | NULL]
Decider identifier column in `df`.

`column_occasion_id`
[character(1) | NULL]
Choice occasion identifier column in `df`.

`column_choice` [character(1) | NULL]
Choice column in `df`.
If `NULL`, the left-hand side variable of `formula` is used.

`column_alternative`
[character(1) | NULL]
Alternative column. Required for `format = "long"`.

`alt_names` [character() | NULL]
Alternative labels.
Must be supplied if `df = NULL`; otherwise derived from `df`.

`ref_alt` [character(1) | NULL]
Reference alternative.

`corr_re` [logical(1)]
Whether random effects are correlated, see details

`error_struc` [character(1)]
Error structure, see details. One of:

- `"diag"`, diagonal matrix `Sigma`. The first Cholesky diagonal entry is fixed to one, and the remaining diagonal entries are free.
- `"fixed"`, `Sigma` fixed to identity matrix.
- `"ident"`, `Sigma` proportional to identity matrix. The first Cholesky diagonal entry shares one common free parameter with all remaining diagonal entries.
- `"full"` (default), fully flexible matrix `Sigma`. The first Cholesky column is fixed to zero for level identification, `L[2,2]` is fixed to one for scale identification, and all remaining Cholesky entries are free.

`N` [integer(1)]
Number of deciders used when simulating data.

`Tp` [integer()]
Number of choice occasions per decider used when simulating data.

`sd` [numeric(1)]
Standard deviation for the centered normal distribution used when simulating covariates.

Value

An `Rprobit` object.

Specifying the model formula

The structure of `formula` is `choice ~ A | B | C`, i.e., a standard `formula` object but with (up to) 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`.

Common covariance structures

The covariance matrices Ω and Σ are parameterized via the lower-triangular Cholesky factors O and L.

For the **random-effect covariance matrix** Ω, the argument `corr_re` controls whether correlations between random effects are included:

- If `corr_re = FALSE`, Ω is diagonal.
- If `corr_re = TRUE`, Ω is fully flexible.

For the **error covariance matrix** Σ, `error_struc` selects one of the following common structures:

- `"full"` creates a flexible covariance matrix. The entry `L[2,2]` of the Cholesky factor is fixed to one for scale identification, the first Cholesky column is fixed to zero, and all remaining Cholesky entries are free.
- `"diag"` creates a diagonal covariance matrix. The first Cholesky diagonal entry is fixed to one, and the remaining diagonal entries are free.
- `"ident"` creates a covariance matrix proportional to the identity. All Cholesky diagonal entries share one common free parameter.
- `"fixed"` fixes the covariance matrix to the identity, i.e. all Cholesky diagonal entries are fixed to one.

Fitting workflows

Calling `object$fit()` uses the mutable R6 method and stores the fitted quantities in `object` itself. Calling `fit(object)` first creates a deep clone of `object`, then calls `$fit()` on that clone and returns it. Use `fit(object)` when the original `Rprobit` object should remain unchanged. For more details on R6-specific modify-in-place logic, see https://r-statistics.co/R6-Classes-in-R.html.

Examples

```
### empirical choice data
if (requireNamespace("mlogit", quietly = TRUE)) {
  data("Train", package = "mlogit")
  r_train <- setup(
    formula = choice ~ price + time | 0 | 0,
    df = Train,
    column_decider_id = "id",
    column_occasion_id = "choiceid"
  )
  r_train
}

### simulated choice data
r_sim <- setup(
  formula = choice ~ price + time | income | 0,
  alt_names = c("train", "bus", "car"),
  N = 20,
  Tp = 2
)
r_sim
```

SJ *Solow-Joe approximation*

Description

Computes the log CDF, gradient, and Hessian for a multivariate Gaussian distribution using the Solow-Joe approximation.

Usage

```
SJ(x, r)

SJ_grad(x, r)

SJ_hess(x, r)
```

Arguments

x	[numeric()] Evaluation point.
r	[matrix] Correlation matrix.

Value

`SJ(x, r)` Log-CDF value.

`SJ_grad(x, r)` Gradient vector wrt evaluation point and correlations. Last entry is the log-CDF value.

`SJ_hess(x, r)` Hessian matrix.

Functions

- `SJ_grad()`: Gradient computation
- `SJ_hess()`: Hessian computation

Examples

```
x <- c(0, 0)
r <- diag(2)
exp(SJ(x, r))
SJ_grad(x, r) # gradient + log-CDF value
SJ_hess(x, r) # Hessian matrix
```

Index

AIC.Rprobit *(*Rprobit-methods*)*, 21

BIC.Rprobit *(*Rprobit-methods*)*, 21

calc_choice_probs, 2
choice_data, 3, *18*
coef.Rprobit *(*Rprobit-methods*)*, 21
confint.Rprobit *(*Rprobit-methods*)*, 21

fit, 8
fit(), *18*, *24*
formula, *13*, *26*

ggplot2::ggplot, *23*
ggplot2::ggplot(), *6*

initialize, 9

ll_cml, 10, *14*
ll_cml(), *8*, *20*
logLik.Rprobit *(*Rprobit-methods*)*, 21

mod, *3*, 11, *18*

plot.Rprobit *(*Rprobit-methods*)*, 21
pred_probit_approx, 18
predict.Rprobit *(*Rprobit-methods*)*, 21
print.summary.Rprobit
        *(*Rprobit-methods*)*, 21

residuals.Rprobit *(*Rprobit-methods*)*, 21
Rprobit, 18, *24, 25*
Rprobit-methods, 21

setup, 24
SJ, 27
SJ_grad *(*SJ*)*, 27
SJ_hess *(*SJ*)*, 27
stats::nlm(), *8*, *19, 20*
summary.Rprobit *(*Rprobit-methods*)*, 21

vcov.Rprobit *(*Rprobit-methods*)*, 21