Package 'RprobitB'

August 25, 2025

Type Package

Title Bayesian Probit Choice Modeling

Version 1.2.0

Description Bayes estimation of probit choice models in cross-sectional and panel settings. The package can analyze binary, multivariate, ordered, and ranked choices, as well as heterogeneity of choice behavior among deciders. The main functionality includes model fitting via Gibbs sampling, tools for convergence diagnostic, choice data simulation, in-sample and out-of-sample choice prediction, and model selection using information criteria and Bayes factors. The latent class model extension facilitates preference-based decider classification, where the number of latent classes can be inferred via the Dirichlet process or a weight-based updating heuristic. This allows for flexible modeling of choice behavior without the need to impose structural constraints. For a reference on the method, see Oelschlaeger and Bauer (2021) <https://trid.trb.org/view/1759753>.

URL https://loelschlaeger.de/RprobitB/, https://github.com/loelschlaeger/RprobitB

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

License GPL-3

Encoding UTF-8

Imports checkmate, cli, crayon, doSNOW, foreach, ggplot2, graphics, gridExtra, MASS, mixtools, oeli (>= 0.7.5), parallel, plotROC, progress, Rcpp, Rdpack, rlang, stats, utils, viridis

LinkingTo oeli (>= 0.7.5), Rcpp, RcppArmadillo, testthat

Suggests knitr, mlogit, testthat (>= 3.0.0), xml2

Depends R (>= 4.1.0)

RoxygenNote 7.3.2

RdMacros Rdpack

VignetteBuilder knitr

LazyData true

LazyDataCompression xz

Config/testthat/edition 3

NeedsCompilation yes

Author Lennart Oelschläger [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5421-9313>),
Dietmar Bauer [ctb] (ORCID: <https://orcid.org/0000-0003-2920-7032>)

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

Repository CRAN

Date/Publication 2025-08-25 18:30:12 UTC

Contents

`as_cov_names` *Re-label alternative specific covariates*

Description

In {RprobitB}, alternative specific covariates must be named in the format "<covariate>_<alternative>". This helper function generates the format for a given `choice_data` set.

Usage

```
as_cov_names(choice_data, cov, alternatives)
```

Arguments

`choice_data`	[data.frame] Choice data in wide format, where each row represents one choice occasion.
`cov`	[character()] Names of alternative specific covariates in `choice_data`.
`alternatives`	[atomic()] The alternative names.

Value

The `choice_data` input with updated column names.

Examples

```
data("Electricity", package = "mlogit")
cov <- c("pf", "cl", "loc", "wk", "tod", "seas")
alternatives <- 1:4
colnames(Electricity)
Electricity <- as_cov_names(Electricity, cov, alternatives)
colnames(Electricity)
```

`check_form` *Check model formula*

Description

This function checks the input `form`.

Usage

```
check_form(form, re = NULL, ordered = FALSE)
```

Arguments

`form`	[`formula`] A model description with the structure `choice ~ A | B | C`, where • `choice` is the name of the dependent variable (the choices), • `A` are names of alternative and choice situation specific covariates with a coefficient that is constant across alternatives, • `B` are names of choice situation specific covariates with alternative specific coefficients, • and `C` are names of alternative and choice situation specific covariates with alternative specific coefficients. Multiple covariates (of one type) are separated by a + sign. By default, alternative specific constants (ASCs) are added to the model. They can be removed by adding `+0` in the second spot. In the ordered probit model (`ordered = TRUE`), the `formula` object has the simple structure `choice ~ A`. ASCs are not estimated.
`re`	[`character() | NULL`] Names of covariates with random effects. If `re = NULL` (the default), there are no random effects. To have random effects for the ASCs, include `"ASC"` in `re`.
`ordered`	[`logical(1)`] If `TRUE`, the choice set `alternatives` is assumed to be ordered from worst to best.

Value

A list that contains the following elements:

- The input `form`.
- The name `choice` of the dependent variable in `form`.
- The input `re`.
- A list `vars` of three character vectors of covariate names of the three covariate types.
- A boolean `ASC`, determining whether the model has ASCs.

See Also

`overview_effects()` for an overview of the model effects

`check_prior` *Check prior parameters*

Description

This function checks the compatibility of submitted parameters for the prior distributions and sets missing values to default values.

Usage

```
check_prior(
  P_f,
  P_r,
  J,
  ordered = FALSE,
  mu_alpha_0 = numeric(P_f),
  Sigma_alpha_0 = 10 * diag(P_f),
  delta = 1,
  mu_b_0 = numeric(P_r),
  Sigma_b_0 = 10 * diag(P_r),
  n_Omega_0 = P_r + 2,
  V_Omega_0 = diag(P_r),
  n_Sigma_0 = J + 1,
  V_Sigma_0 = diag(J - 1),
  mu_d_0 = numeric(J - 2),
  Sigma_d_0 = diag(J - 2)
)
```

Arguments

P_f	[integer(1)] The number of covariates connected to a fixed coefficient.
P_r	[integer(2)] The number of covariates connected to a random coefficient.
J	[integer(1)] The number >= 2 of choice alternatives.
ordered	[logical(1)] If TRUE, the choice set alternatives is assumed to be ordered from worst to best.
mu_alpha_0	[numeric(P_f)] The mean vector of the normal prior for alpha.
Sigma_alpha_0	[matrix(P_f, P_f)] The covariance matrix of the normal prior for alpha.
delta	[numeric(1)] The prior concentration for s.
mu_b_0	[numeric(P_r)] The mean vector of the normal prior for each b_c.
Sigma_b_0	[matrix(P_r, P_r)] The covariance matrix of the normal prior for each b_c.
n_Omega_0	[integer(1)] The degrees of freedom of the Inverse Wishart prior for each Omega_c.
V_Omega_0	[matrix(P_r, P_r)] The scale matrix of the Inverse Wishart prior for each Omega_c.
n_Sigma_0	[integer(1)] The degrees of freedom of the Inverse Wishart prior for Sigma.
V_Sigma_0	[matrix(J - 1, J - 1)] The scale matrix of the Inverse Wishart prior for Sigma.
mu_d_0	[numeric(J - 2)] The mean vector of the normal prior for d .
Sigma_d_0	[matrix(J - 2, J - 2)] The covariance matrix of the normal prior for d.

Details

A priori-distributions:

- $\alpha \sim N(\mu_{\alpha_0}, \Sigma_{\alpha_0})$
- $s \sim Dir(\delta)$
- $b_c \sim N(\mu_{b_0}, \Sigma_{b_0})$ for all c
- $\Omega_c \sim IW(n_{\Omega_0}, V_{\Omega_0})$ for all c
- $\Sigma \sim IW(n_{\Sigma_0}, V_{\Sigma_0})$
- $d \sim N(\mu_{d_0}, \Sigma_{d_0})$

Value

An object of class RprobitB_prior, which is a list containing all prior parameters.

Examples

```
check_prior(P_f = 1, P_r = 2, J = 3, ordered = TRUE)
```

---

choice_probabilities *Compute choice probabilities*

Description

This function returns the choice probabilities of an RprobitB_fit object.

Usage

```
choice_probabilities(x, data = NULL, par_set = mean)
```

Arguments

x	An object of class RprobitB_fit.
data	Either NULL or an object of class RprobitB_data. In the former case, choice probabilities are computed for the data that was used for model fitting. Alternatively, a new data set can be provided.
par_set	Specifying the parameter set for calculation and either • a function that computes a posterior point estimate (the default is mean()), • "true" to select the true parameter set, • an object of class RprobitB_parameter.

Value

A data frame of choice probabilities with choice situations in rows and alternatives in columns. The first two columns are the decider identifier "id" and the choice situation identifier "idc".

Examples

```
data <- simulate_choices(form = choice ~ covariate, N = 10, T = 10, J = 2)
x <- fit_model(data)
choice_probabilities(x)
```

classification *Preference-based classification of deciders*

Description

This function classifies the deciders based on their allocation to the components of the mixing distribution.

Usage

```
classification(x, add_true = FALSE)
```

Arguments

x	An object of class RprobitB_fit.
add_true	Set to TRUE to add true class memberships to output (if available).

Details

The relative frequencies of which each decider was allocated to the classes during the Gibbs sampling are displayed. Only the thinned samples after the burn-in period are considered.

Value

A data.frame.

The row names are the decider identifiers.

The first C columns contain the relative frequencies with which the deciders are allocated to classes. Next, the colu
contains the estimated class of the decider based on the highest allocation frequency.

If add_true = TRUE, the next column true contains the true class memberships.

See Also

update_z() for the updating function of the class allocation vector.

coef.RprobitB_fit *Extract model effects*

Description

This function extracts the estimated model effects.

Usage

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

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

## S3 method for class 'RprobitB_coef'
plot(x, sd = 1, het = FALSE, ...)
```

Arguments

object	An object of class RprobitB_fit.
...	Currently not used.
x	An object of class RprobitB_coef.
sd	The number of standard deviations to display.
het	Set to FALSE to show the standard deviation of the estimate. Set to TRUE to show the standard deviation of the mixing distribution.

Value

An object of class RprobitB_coef.

---

compute_p_si *Compute choice probabilities at posterior samples*

Description

This function computes the probability for each observed choice at the (normalized, burned and thinned) samples from the posterior.

These probabilities are required to compute the WAIC and the marginal model likelihood mml.

Usage

```
compute_p_si(x, ncores = parallel::detectCores() - 1, recompute = FALSE)
```

Arguments

x	An object of class RprobitB_fit.
ncores	[integer(1)] The number of cores for parallel computation. If set to 1, no parallel backend is used.
recompute	[logical(1)] Recompute the probabilities?

Value

The object x, including the object p_si, which is a matrix of probabilities, observations in rows and posterior samples in columns.

---

cov_mix *Extract estimated covariance matrix of mixing distribution*

Description

This convenience function returns the estimated covariance matrix of the mixing distribution.

Usage

```
cov_mix(x, cor = FALSE)
```

Arguments

x	An object of class RprobitB_fit.
cor	If TRUE, returns the correlation matrix instead.

Value

The estimated covariance matrix of the mixing distribution. In case of multiple classes, a list of matrices for each class.

---

create_lagged_cov *Create lagged choice covariates*

Description

This function creates lagged choice covariates from the data.frame choice_data, which is assumed to be sorted by the choice occasions.

Usage

```
create_lagged_cov(choice_data, column = character(), k = 1, id = "id")
```

Arguments

choice_data	[data.frame] Choice data in wide format, where each row represents one choice occasion.
column	[character()] Covariate names in choice_data.
k	[integer()] The number of lags (in units of observations), see the details.
id	[character(1)] The name of the column in choice_data that contains unique identifier for each decision maker.

Details

Say that choice_data contains the column column. Then, the function call

```
create_lagged_cov(choice_data, column, k, id)
```

returns the input choice_data which includes a new column named column.k. This column contains for each decider (based on id) and each choice occasion the covariate faced before k choice occasions. If this data point is not available, it is set to NA. In particular, the first k values of column.k will be NA (initial condition problem).

Value

The input choice_data with the additional columns named column.k for each element column and each number k containing the lagged covariates.

Examples

```
choice_data <- data.frame(id = rep(1:2, each = 3), cov = LETTERS[1:6])
create_lagged_cov(choice_data, column = "cov", k = 1:2)
```

---

d_to_gamma *Transform increments to thresholds*

Description

Transform increments to thresholds

Usage

```
d_to_gamma(d)
```

Arguments

d	[numeric(J - 2)] Threshold increments.

Value

The threshold vector of length J + 1.

Examples

```
d_to_gamma(c(0, 0, 0))
```

`fit_model` *Fit probit model to choice data*

Description

This function performs MCMC simulation for fitting different types of probit models (binary, multivariate, mixed, latent class, ordered, ranked) to discrete choice data.

Usage

```
fit_model(
  data,
  scale = "Sigma_1,1 := 1",
  R = 1000,
  B = R/2,
  Q = 1,
  print_progress = getOption("RprobitB_progress", default = TRUE),
  prior = NULL,
  latent_classes = NULL,
  fixed_parameter = list(),
  save_beta_draws = FALSE
)
```

Arguments

`data`	An object of class `RprobitB_data`.
`scale`	[`character(1)`] A character which determines the utility scale. It is of the form `<parameter> := <value>`, where `<parameter>` is either the name of a fixed effect or `Sigma_<j>,<j>` for the `<j>`th diagonal element of `Sigma`, and `<value>` is the value of the fixed parameter.
`R`	[`integer(1)`] The number of iterations of the Gibbs sampler.
`B`	[`integer(1)`] The length of the burn-in period.
`Q`	[`integer(1)`] The thinning factor for the Gibbs samples.
`print_progress`	[`logical(1)`] Print the Gibbs sampler progress?
`prior`	[`list`] A named list of parameters for the prior distributions. See the documentation of `check_prior` for details about which parameters can be specified.
`latent_classes`	[`list()` | `NULL`] Optionally parameters specifying the number of latent classes and their updating scheme. The values in brackets are the default.

- C (1): The fixed number (greater or equal 1) of (initial) classes.
- wb_update (FALSE): Set to TRUE for weight-based class updates.
- dp_update (FALSE): Set to TRUE for Dirichlet process class updates.
- Cmax (10): The maximum number of latent classes.

The following specifications are used for the weight-based updating scheme:

- buffer (50): The number of iterations to wait before the next update.
- epsmin (0.01): The threshold weight for removing a latent class.
- epsmax (0.7): The threshold weight for splitting a latent class.
- deltamin (0.1): The minimum mean distance before merging two classes.
- deltashift (0.5): The scale for shifting the class means after a split.

fixed_parameter
: [list]
A named list with fixed parameter values for alpha, C, s, b, Omega, Sigma, Sigma_full, beta, z, or d for the simulation.

See the vignette on model definition for definitions of these variables.

save_beta_draws
: [logical(1)]
Save draws for decider-specific coefficient vectors? Usually not recommended, as it requires a lot of storage space.

Value

An object of class RprobitB_fit.

See Also

- prepare_data() and simulate_choices() for building an RprobitB_data object
- update() for estimating nested models
- transform() for transforming a fitted model

Examples

```
set.seed(1)
form <- choice ~ var | 0
data <- simulate_choices(form = form, N = 100, T = 10, J = 3, re = "var")
model <- fit_model(data = data, R = 1000)
summary(model)
```

get_cov *Extract covariates of choice occasion*

Description

This convenience function returns the covariates and the choices of specific choice occasions.

Usage

```
get_cov(x, id, idc, idc_label)
```

Arguments

x	Either an object of class RprobitB_data or RprobitB_fit.
id	A numeric (vector), that specifies the decider(s).
idc	A numeric (vector), that specifies the choice occasion(s).
idc_label	The name of the column that contains the choice occasion identifier.

Value

A subset of the choice_data data frame specified in prepare_data().

gibbs_sampler *Gibbs sampler for probit models*

Description

Gibbs sampler for probit models

Usage

```
gibbs_sampler(
  sufficient_statistics,
  prior,
  latent_classes,
  fixed_parameter,
  R,
  B,
  print_progress,
  ordered,
  ranked,
  save_beta_draws = FALSE
)
```

Arguments

`sufficient_statistics`
[list]
The output of `sufficient_statistics`.

`prior` [list]
A named list of parameters for the prior distributions. See the documentation of `check_prior` for details about which parameters can be specified.

`latent_classes` [list() | NULL]
Optionally parameters specifying the number of latent classes and their updating scheme. The values in brackets are the default.

- `C` (`1`): The fixed number (greater or equal 1) of (initial) classes.
- `wb_update` (`FALSE`): Set to `TRUE` for weight-based class updates.
- `dp_update` (`FALSE`): Set to `TRUE` for Dirichlet process class updates.
- `Cmax` (`10`): The maximum number of latent classes.

The following specifications are used for the weight-based updating scheme:

- `buffer` (`50`): The number of iterations to wait before the next update.
- `epsmin` (`0.01`): The threshold weight for removing a latent class.
- `epsmax` (`0.7`): The threshold weight for splitting a latent class.
- `deltamin` (`0.1`): The minimum mean distance before merging two classes.
- `deltashift` (`0.5`): The scale for shifting the class means after a split.

`fixed_parameter`
[list]
A named list with fixed parameter values for `alpha`, `C`, `s`, `b`, `Omega`, `Sigma`, `Sigma_full`, `beta`, `z`, or `d` for the simulation.
See the vignette on model definition for definitions of these variables.

`R` [integer(1)]
The number of iterations of the Gibbs sampler.

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

`print_progress` [logical(1)]
Print the Gibbs sampler progress?

`ordered` [logical(1)]
If `TRUE`, the choice set `alternatives` is assumed to be ordered from worst to best.

`ranked` [logical(1)]
Are the alternatives ranked?

`save_beta_draws`
[logical(1)]
Save draws for decider-specific coefficient vectors? Usually not recommended, as it requires a lot of storage space.

Details

This function is not supposed to be called directly, but rather via `fit_model`.

Value

A list of Gibbs samples for

- Sigma,
- alpha (only if P_f > 0),
- s, z, b, Omega (only if P_r > 0),
- d (only if ordered = TRUE),

and a vector class_sequence of length R, where the r-th entry is the number of classes after iteration r.

---

ll_ordered *Compute ordered probit log-likelihood*

Description

Compute ordered probit log-likelihood

Usage

```
ll_ordered(d, y, sys, Tvec)
```

Arguments

d	[numeric(J - 2)] Threshold increments.
y	[matrix(nrow = N, ncol = max(Tvec))] Choices 1, . . . , J for each decider in each choice occasion.
sys	[matrix(nrow = N, ncol = max(Tvec))] Systematic utilties for each decider in each choice occasion.
Tvec	[integer(N)] Number of choice occasions per decider.

Value

The ordered probit log-likelihood value.

Examples

```
d <- c(0, 0, 0)
y <- matrix(c(1, 2, 1, NA), ncol = 2)
sys <- matrix(c(0, 0, 0, NA), ncol = 2)
Tvec <- c(2, 1)
ll_ordered(d = d, y = y, sys = sys, Tvec = Tvec)
```

mml *Approximate marginal model likelihood*

Description

This function approximates the model's marginal likelihood.

Usage

```
mml(x, S = 0, ncores = parallel::detectCores() - 1, recompute = FALSE)

## S3 method for class 'RprobitB_mml'
print(x, log = FALSE, ...)

## S3 method for class 'RprobitB_mml'
plot(x, log = FALSE, ...)
```

Arguments

x	An object of class RprobitB_fit.
S	The number of prior samples for the prior arithmetic mean estimate. Per default, S = 0. In this case, only the posterior samples are used for the approximation via the posterior harmonic mean estimator, see the details section.
ncores	Computation of the prior arithmetic mean estimate is parallelized, set the number of cores.
recompute	Set to TRUE to recompute the likelihood.
log	Return the logarithm of the marginal model likelihood?
...	Currently not used.

Details

The model's marginal likelihood $p(y \mid M)$ for a model M and data y is required for the computation of Bayes factors. In general, the term has no closed form and must be approximated numerically.

This function uses the posterior Gibbs samples to approximate the model's marginal likelihood via the posterior harmonic mean estimator. To check the convergence, call `plot(x$mml)`, where `x` is the output of this function. If the estimation does not seem to have converged, you can improve the approximation by combining the value with the prior arithmetic mean estimator. The final approximation of the model's marginal likelihood than is a weighted sum of the posterior harmonic mean estimate and the prior arithmetic mean estimate, where the weights are determined by the sample sizes.

Value

The object `x`, including the object `mml`, which is the model's approximated marginal likelihood value.

`model_selection` *Compare fitted models*

Description

This function returns a table with several criteria for model comparison.

Usage

```
model_selection(
  ...,
  criteria = c("npar", "LL", "AIC", "BIC"),
  add_form = FALSE
)

## S3 method for class 'RprobitB_model_selection'
print(x, digits = 2, ...)
```

Arguments

`...`	One or more objects of class `RprobitB_fit`.
`criteria`	[`character()`] One or more of the following: • `"npar"` for the number of model parameters (see `npar`), • `"LL"` for the log-likelihood value (see `logLik`), • `"AIC"` for the AIC value (see `AIC`), • `"BIC"` for the BIC value (see `BIC`), • `"WAIC"` for the WAIC value (also shows its standard error `sd(WAIC)` and the number `pWAIC` of effective model parameters, see `WAIC`), • `"MMLL"` for the marginal model log-likelihood, • `"BF"` for the Bayes factor, • `"pred_acc"` for the prediction accuracy (see `pred_acc`).
`add_form`	[`logical(1)`] Add the model formulas?
`x`	An object of class `RprobitB_model_selection`.
`digits`	[`integer(1)`] The number of digits.

Details

See the vignette on model selection for more details.

Value

A `data.frame`, criteria in columns, models in rows.

mode_approx *Gibbs sample mode*

Description

This function approximates the Gibbs sample mode.

Usage

```
mode_approx(samples)
```

Arguments

samples	[numeric()] Gibbs samples.

Value

The (approximated) mode.

Examples

```
samples <- oeli::rmixnorm(
  n = 1000, mean = matrix(c(-2, 2), ncol = 2),
  Sigma = matrix(c(1, 1), ncol = 2), proportions = c(0.7, 0.3)
)
hist(samples)
mean(samples) # expected: 0.7 * (-2) + 0.3 * 2 = -0.8
mode_approx(samples) # expected: -2
```

npar *Extract number of model parameters*

Description

This function extracts the number of model parameters of an RprobitB_fit object.

Usage

```
npar(object, ...)

## S3 method for class 'RprobitB_fit'
npar(object, ...)
```

Arguments

object	An object of class RprobitB_fit.
...	Optionally more objects of class RprobitB_fit.

Value

Either a numeric value (if just one object is provided) or a numeric vector.

---

overview_effects *Print effect overview*

Description

This function gives an overview of the effect names, whether the covariate is alternative-specific, whether the coefficient is alternative-specific, and whether it is a random effect.

Usage

```
overview_effects(
  form,
  re = NULL,
  alternatives,
  base = tail(alternatives, 1),
  ordered = FALSE
)
```

Arguments

form	[formula] A model description with the structure choice ~ A | B | C, where • choice is the name of the dependent variable (the choices), • A are names of alternative and choice situation specific covariates with a coefficient that is constant across alternatives, • B are names of choice situation specific covariates with alternative specific coefficients, • and C are names of alternative and choice situation specific covariates with alternative specific coefficients. Multiple covariates (of one type) are separated by a + sign. By default, alternative specific constants (ASCs) are added to the model. They can be removed by adding +0 in the second spot. In the ordered probit model (ordered = TRUE), the formula object has the simple structure choice ~ A. ASCs are not estimated.
re	[character() | NULL] Names of covariates with random effects. If re = NULL (the default), there are no random effects. To have random effects for the ASCs, include "ASC" in re.

`alternatives` [character()]
The names of the choice alternatives. If not specified, the choice set is defined by the observed choices.
If `ordered = TRUE`, `alternatives` is assumed to be specified with the alternatives ordered from worst to best.

`base` [character(1)]
The name of the base alternative for covariates that are not alternative specific (i.e. type 2 covariates and ASCs).
Ignored and set to `NULL` if the model has no alternative specific covariates (e.g. in the ordered probit model).
By default, `base` is the last element of `alternatives`.

`ordered` [logical(1)]
If `TRUE`, the choice set `alternatives` is assumed to be ordered from worst to best.

Value

A `data.frame`, each row is a effect, columns are the effect name `"effect"`, and booleans whether the covariate is alternative-specific `"as_value"`, whether the coefficient is alternative-specific `"as_coef"`, and whether it is a random effect `"random"`.

See Also

`check_form()` for checking the model formula specification.

Examples

```
overview_effects(
  form = choice ~ price + time + comfort + change | 1,
  re = c("price", "time"),
  alternatives = c("A", "B"),
  base = "A"
)
```

---

plot.RprobitB_fit *Visualize fitted probit model*

Description

This function is the plot method for an object of class `RprobitB_fit`.

Usage

```
## S3 method for class 'RprobitB_fit'
plot(x, type, ignore = NULL, ...)
```

Arguments

x	An object of class RprobitB_fit.
type	[character(1)] The type of plot, which can be one of: • "mixture" to visualize the mixing distribution, • "acf" for autocorrelation plots of the Gibbs samples, • "trace" for trace plots of the Gibbs samples, • "class_seq" to visualize the sequence of class numbers.
ignore	[character()] Covariate or parameter names that do not get visualized.
...	Currently not used.

Value

No return value. Draws a plot to the current device.

---

plot_class_allocation *Plot class allocation (for* P_r = 2 *only)*

Description

This function plots the allocation of decision-maker specific coefficient vectors beta given the allocation vector z, the class means b, and the class covariance matrices Omega.

Usage

```
plot_class_allocation(beta, z, b, Omega, ...)
```

Arguments

beta	[matrix(nrow = P_r, ncol = N)] The matrix of the decider-specific coefficient vectors.
z	[numeric(N)] The decider class allocations.
b	[matrix(nrow = P_r, ncol = C)] The matrix of class means as columns.
Omega	[matrix(nrow = P_r * P_r, ncol = C)] The matrix of vectorized class covariance matrices as columns.
...	Optional visualization parameters: • colors, a character vector of color specifications, • perc, a numeric between 0 and 1 to draw the perc percentile ellipsoids for the underlying Gaussian distributions (perc = 0.95 per default), • r, the current iteration number of the Gibbs sampler to be displayed in the legend, • sleep, the number of seconds to pause after plotting.

Details

Only applicable in the two-dimensional case, i.e. only if `P_r = 2`.

Value

No return value. Draws a plot to the current device.

---

plot_mixture_contour *Plot bivariate contour of mixing distributions*

Description

This function plots an estimated bivariate contour mixing distributions.

Usage

```
plot_mixture_contour(means, covs, weights, names)
```

Arguments

`means`	The class means.
`covs`	The class covariances.
`weights`	The class weights.
`names`	The covariate names.

Value

An object of class `ggplot`.

Examples

```
means <- list(c(0, 0), c(2, 2))
covs <- list(diag(2), 0.5 * diag(2))
weights <- c(0.7, 0.3)
names <- c("A", "B")
plot_mixture_contour(means, covs, weights, names)
```

plot_roc *Plot ROC curve*

Description

This function draws receiver operating characteristic (ROC) curves.

Usage

```
plot_roc(..., reference = NULL)
```

Arguments

...	One or more RprobitB_fit objects or data.frames of choice probability.
reference	The reference alternative.

Value

No return value. Draws a plot to the current device.

point_estimates *Compute point estimates*

Description

This function computes the point estimates of an RprobitB_fit. Per default, the mean of the Gibbs samples is used as a point estimate. However, any statistic that computes a single numeric value out of a vector of Gibbs samples can be specified for FUN.

Usage

```
point_estimates(x, FUN = mean)
```

Arguments

x	An object of class RprobitB_fit.
FUN	A function that computes a single numeric value out of a vector of numeric values.

Value

An object of class RprobitB_parameter.

Examples

```
data <- simulate_choices(form = choice ~ covariate, N = 10, T = 10, J = 2)
model <- fit_model(data)
point_estimates(model)
point_estimates(model, FUN = median)
```

---

predict.RprobitB_fit *Predict choices*

Description

This function predicts the discrete choice behaviour.

Usage

```
## S3 method for class 'RprobitB_fit'
predict(object, data = NULL, overview = TRUE, digits = 2, ...)
```

Arguments

`object`	An object of class `RprobitB_fit`.
`data`	Either • `NULL`, using the data in `object`, • an object of class `RprobitB_data`, for example the test part generated by `train_test`, • or a data frame of custom choice characteristics. It must have the same structure as `choice_data` used in `prepare_data`. Missing columns or `NA` values are set to 0.
`overview`	`[logical(1)]` Summarize the prediction in a confusion matrix?
`digits`	`[integer(1)]` The number of digits of the returned choice probabilities.
`...`	Currently not used.

Details

Predictions are made based on the maximum predicted probability for each choice alternative.

See the vignette on choice prediction for a demonstration on how to visualize the model's sensitivity and specificity by means of a receiver operating characteristic (ROC) curve.

Value

Either a table if `overview = TRUE` or a data frame otherwise.

Examples

```
set.seed(1)
data <- simulate_choices(form = choice ~ cov, N = 10, T = 10, J = 2)
data <- train_test(data, test_proportion = 0.5)
model <- fit_model(data$train)

predict(model)
predict(model, overview = FALSE)
predict(model, data = data$test)
predict(
  model,
  data = data.frame("cov_A" = c(1, 1, NA, NA), "cov_B" = c(1, NA, 1, NA)),
  overview = FALSE
)
```

---

pred_acc *Compute prediction accuracy*

Description

This function computes the prediction accuracy of an `RprobitB_fit` object. Prediction accuracy means the share of choices that are correctly predicted by the model, where prediction is based on the maximum choice probability.

Usage

```
pred_acc(x, ...)
```

Arguments

`x`	An object of class `RprobitB_fit`.
`...`	Optionally specify more `RprobitB_fit` objects.

Value

A numeric.

prepare_data *Prepare choice data for estimation*

Description

This function prepares choice data for estimation.

Usage

```
prepare_data(
  form,
  choice_data,
  re = NULL,
  alternatives = NULL,
  ordered = FALSE,
  ranked = FALSE,
  base = NULL,
  id = "id",
  idc = NULL,
  standardize = NULL,
  impute = "complete_cases"
)
```

Arguments

form	[formula] A model description with the structure choice ~ A | B | C, where • choice is the name of the dependent variable (the choices), • A are names of alternative and choice situation specific covariates with a coefficient that is constant across alternatives, • B are names of choice situation specific covariates with alternative specific coefficients, • and C are names of alternative and choice situation specific covariates with alternative specific coefficients. Multiple covariates (of one type) are separated by a + sign. By default, alternative specific constants (ASCs) are added to the model. They can be removed by adding +0 in the second spot. In the ordered probit model (ordered = TRUE), the formula object has the simple structure choice ~ A. ASCs are not estimated.
choice_data	[data.frame] Choice data in wide format, where each row represents one choice occasion.
re	[character() | NULL] Names of covariates with random effects. If re = NULL (the default), there are no random effects. To have random effects for the ASCs, include "ASC" in re.

`alternatives` [character()]
The names of the choice alternatives. If not specified, the choice set is defined by the observed choices.
If `ordered = TRUE`, `alternatives` is assumed to be specified with the alternatives ordered from worst to best.

`ordered` [logical(1)]
If `TRUE`, the choice set `alternatives` is assumed to be ordered from worst to best.

`ranked` [logical(1)]
Are the alternatives ranked?

`base` [character(1)]
The name of the base alternative for covariates that are not alternative specific (i.e. type 2 covariates and ASCs).
Ignored and set to `NULL` if the model has no alternative specific covariates (e.g. in the ordered probit model).
By default, `base` is the last element of `alternatives`.

`id` [character(1)]
The name of the column in `choice_data` that contains unique identifier for each decision maker.

`idc` [character(1)]
The name of the column in `choice_data` that contains unique identifier for each choice situation of each decision maker. By default, these identifier are generated by the order of appearance.

`standardize` [character() | "all"]
Names of covariates that get standardized.
Covariates of type 1 or 3 have to be addressed by <covariate>_<alternative>.
If `standardize = "all"`, all covariates get standardized.

`impute` A character that specifies how to handle missing covariate entries in `choice_data`, one of:

- `"complete_cases"`, removes all rows containing missing covariate entries (the default),
- `"zero"`, replaces missing covariate entries by zero (only for numeric columns),
- `"mean"`, imputes missing covariate entries by the mean (only for numeric columns).

Details

Requirements for the `data.frame` `choice_data`:

- It **must** contain a column named `id` which contains unique identifier for each decision maker.
- It **can** contain a column named `idc` which contains unique identifier for each choice situation of each decision maker. If this information is missing, these identifier are generated automatically by the appearance of the choices in the data set.
- It **can** contain a column named `choice` with the observed choices, where `choice` must match the name of the dependent variable in `form`. Such a column is required for model fitting but not for prediction.

- It **must** contain a numeric column named *p_j* for each alternative specific covariate *p* in `form` and each choice alternative *j* in `alternatives`.
- It **must** contain a numeric column named *q* for each covariate *q* in `form` that is constant across alternatives.

In the ordered case (`ordered = TRUE`), the column `choice` must contain the full ranking of the alternatives in each choice occasion as a character, where the alternatives are separated by commas, see the examples.

See the vignette on choice data for more details.

Value

An object of class `RprobitB_data`.

See Also

- `check_form()` for checking the model formula
- `overview_effects()` for an overview of the model effects
- `create_lagged_cov()` for creating lagged covariates
- `as_cov_names()` for re-labeling alternative-specific covariates
- `simulate_choices()` for simulating choice data
- `train_test()` for splitting choice data into a train and test subset

Examples

```
data <- prepare_data(
  form = choice ~ price + time + comfort + change | 0,
  choice_data = train_choice,
  re = c("price", "time"),
  id = "deciderID",
  idc = "occasionID",
  standardize = c("price", "time")
)

### ranked case
choice_data <- data.frame(
  "id" = 1:3, "choice" = c("A,B,C", "A,C,B", "B,C,A"), "cov" = 1
)
data <- prepare_data(
  form = choice ~ 0 | cov + 0,
  choice_data = choice_data,
  ranked = TRUE
)
```

`RprobitB_parameter` *Define probit model parameter*

Description

This function creates an object of class `RprobitB_parameter`, which contains the parameters of a probit model.

If `sample = TRUE`, missing parameters are sampled. All parameters are checked against the values of `P_f`, `P_r`, `J`, and `N`.

Note that parameters are automatically ordered with respect to a non-ascending `s` for class identifiability.

Usage

```
RprobitB_parameter(
  P_f,
  P_r,
  J,
  N,
  C = 1,
  ordered = FALSE,
  alpha = NULL,
  s = NULL,
  b = NULL,
  Omega = NULL,
  Sigma = NULL,
  Sigma_full = NULL,
  beta = NULL,
  z = NULL,
  d = NULL,
  sample = TRUE
)

## S3 method for class 'RprobitB_parameter'
print(x, ..., digits = 4)
```

Arguments

`P_f`	`[integer(1)]` The number of covariates connected to a fixed coefficient.
`P_r`	`[integer(2)]` The number of covariates connected to a random coefficient.
`J`	`[integer(1)]` The number >= 2 of choice alternatives.
`N`	`[integer(1)]` The number of decision makers.

`C` [integer(1)]
The number (greater or equal 1) of latent classes of decision makers.

`ordered` [logical(1)]
If TRUE, the choice set `alternatives` is assumed to be ordered from worst to best.

`alpha` [numeric(P_f)]
The fixed coefficient vector.

`s` [numeric(C)]
The vector of class weights.

`b` [matrix(nrow = P_r, ncol = C)]
The matrix of class means as columns.

`Omega` [matrix(nrow = P_r * P_r, ncol = C)]
The matrix of vectorized class covariance matrices as columns.

`Sigma` [matrix(nrow = J - 1, ncol = J - 1) | numeric(1)]
The differenced (wrt. alternative `J`) error covariance matrix.
In case of `ordered` = TRUE, the single error variance.

`Sigma_full` [matrix(nrow = J, ncol = J)]
The error covariance matrix.
Ignored if `Sigma` is specified or `ordered` = TRUE.
Internally, `Sigma_full` gets differenced wrt. alternative `J`.

`beta` [matrix(nrow = P_r, ncol = N)]
The matrix of the decider-specific coefficient vectors.

`z` [numeric(N)]
The decider class allocations.

`d` [numeric(J - 2)]
The logarithmic increases of the utility thresholds in the ordered probit case (`ordered` = TRUE).

`sample` [logical(1)]
Sample missing parameters?

`x` An `RprobitB_parameter` object.

`...` [character()]
Names of parameters to be printed. If not specified, all parameters are printed.

`digits` [integer(1)]
The number of decimal places.

Value

An object of class `RprobitB_parameter`, which is a named list with the model parameters.

Examples

```
RprobitB_parameter(P_f = 1, P_r = 2, J = 3, N = 10, C = 2)
```

R_hat *Compute Gelman-Rubin statistic*

Description

This function computes the Gelman-Rubin statistic R_hat.

Usage

```
R_hat(samples, parts = 2)
```

Arguments

samples	[numeric() | matrix] Samples from a Markov chain. If it is a matrix, each column gives the samples for a separate chain.
parts	[integer(1)] The number of parts to divide each chain into sub-chains.

Details

NA values in samples are ignored. The degenerate case is indicated by NA. The Gelman-Rubin statistic is bounded by 1 from below. Values close to 1 indicate reasonable convergence.

Value

The Gelman-Rubin statistic.

Examples

```
no_chains <- 2
length_chains <- 1e3
samples <- matrix(NA_real_, length_chains, no_chains)
samples[1, ] <- 1
Gamma <- matrix(c(0.8, 0.1, 0.2, 0.9), 2, 2)
for (c in 1:no_chains) {
  for (t in 2:length_chains) {
    samples[t, c] <- sample(1:2, 1, prob = Gamma[samples[t - 1, c], ])
  }
}
R_hat(samples)
```

sample_allocation *Sample allocation*

Description

Sample allocation

Usage

```
sample_allocation(prob)
```

Arguments

prob	[numeric(C)] The vector of class probabilities.

Value

An integer `1:C`.

Examples

```
sample_allocation(c(0.5, 0.3, 0.2))
```

simulate_choices *Simulate choice data*

Description

This function simulates choice data from a probit model.

Usage

```
simulate_choices(
  form,
  N,
  T = 1,
  J,
  re = NULL,
  alternatives = NULL,
  ordered = FALSE,
  ranked = FALSE,
  base = NULL,
  covariates = NULL,
  true_parameter = list()
)
```

Arguments

form	[formula] A model description with the structure choice ~ A | B | C, where • choice is the name of the dependent variable (the choices), • A are names of alternative and choice situation specific covariates with a coefficient that is constant across alternatives, • B are names of choice situation specific covariates with alternative specific coefficients, • and C are names of alternative and choice situation specific covariates with alternative specific coefficients. Multiple covariates (of one type) are separated by a + sign. By default, alternative specific constants (ASCs) are added to the model. They can be removed by adding +0 in the second spot. In the ordered probit model (ordered = TRUE), the formula object has the simple structure choice ~ A. ASCs are not estimated.
N	[integer(1)] The number of decision makers.
T	[integer(1) | integer(N)] The number of choice occasions or a vector of decider-specific choice occasions of length N.
J	[integer(1)] The number >= 2 of choice alternatives.
re	[character() | NULL] Names of covariates with random effects. If re = NULL (the default), there are no random effects. To have random effects for the ASCs, include "ASC" in re.
alternatives	[character()] The names of the choice alternatives. If not specified, the choice set is defined by the observed choices. If ordered = TRUE, alternatives is assumed to be specified with the alternatives ordered from worst to best.
ordered	[logical(1)] If TRUE, the choice set alternatives is assumed to be ordered from worst to best.
ranked	[logical(1)] Are the alternatives ranked?
base	[character(1)] The name of the base alternative for covariates that are not alternative specific (i.e. type 2 covariates and ASCs). Ignored and set to NULL if the model has no alternative specific covariates (e.g. in the ordered probit model). By default, base is the last element of alternatives.
covariates	[list] A named list of covariate values. Each element must be a vector of length equal

to the number of choice occasions and named according to a covariate. Covariates for which no values are supplied are drawn from a standard normal distribution.

`true_parameter` [list]
A named list with true parameter values for `alpha`, `C`, `s`, `b`, `Omega`, `Sigma`, `Sigma_full`, `beta`, `z`, `d` for the simulation.

See the vignette on model definition for definitions of these variables.

Details

See the vignette on choice data for more details.

Value

An object of class `RprobitB_data`.

See Also

- `check_form()` for checking the model formula
- `overview_effects()` for an overview of the model effects
- `create_lagged_cov()` for creating lagged covariates
- `as_cov_names()` for re-labelling alternative-specific covariates
- `prepare_data()` for preparing empirical choice data
- `train_test()` for splitting choice data into a train and test subset

Examples

```
### simulate data from a binary probit model with two latent classes
data <- simulate_choices(
  form = choice ~ cost | income | time,
  N = 100,
  T = 10,
  J = 2,
  re = c("cost", "time"),
  alternatives = c("car", "bus"),
  true_parameter = list(
    "alpha" = c(-1, 1),
    "b" = matrix(c(-1, -1, -0.5, -1.5, 0, -1), ncol = 2),
    "C" = 2
  )
)

### simulate data from an ordered probit model
data <- simulate_choices(
  form = opinion ~ age + gender,
  N = 10,
  T = 1:10,
  J = 5,
  alternatives = c("very bad", "bad", "indifferent", "good", "very good"),
```

```
  ordered = TRUE,
  covariates = list(
    "gender" = rep(sample(c(0, 1), 10, replace = TRUE), times = 1:10)
  )
)

### simulate data from a ranked probit model
data <- simulate_choices(
  form = product ~ price,
  N = 10,
  T = 1:10,
  J = 3,
  alternatives = c("A", "B", "C"),
  ranked = TRUE
)
```

`train_choice` *Stated Preferences for Train Traveling*

Description

Data set of 2929 stated choices by 235 Dutch individuals deciding between two virtual 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 `data.frame` with 2929 rows and 11 columns:

deciderID `integer` identifier for the decider

occasionID `integer` identifier for the choice occasion

choice `character` for the chosen alternative (either `"A"` or `"B"`)

price_A `numeric` price for alternative `"A"` in Euro

time_A `numeric` travel time for alternative `"A"` in hours

change_A `integer` number of changes for alternative `"A"`

comfort_A `integer` comfort level (in decreasing comfort order) for alternative `"A"`

price_B `numeric` price for alternative `"B"` in Euro

time_B `numeric` travel time for alternative `"B"` in hours

change_B `integer` number of changes for alternative `"B"`

comfort_B `integer` comfort level (in decreasing comfort order) for alternative `"B"`

References

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

Meijer E, Rouwendal J (2006). "Measuring welfare effects in models with random coefficients." *Journal of Applied Econometrics*, **21**(2).

`train_test` *Split choice data into train and test subset*

Description

This function splits choice data into a train and a test part.

Usage

```
train_test(
  x,
  test_proportion = NULL,
  test_number = NULL,
  by = "N",
  random = FALSE
)
```

Arguments

`x`	An object of class `RprobitB_data`.
`test_proportion`	[`numeric(1)`] The proportion of the test subset.
`test_number`	[`integer(1)`] The number of observations in the test subset.
`by`	[`character(1)`] Either `"N"` (split by deciders) and `"T"` (split by occasions).
`random`	[`logical(1)`] Build subsets randomly?

Value

A list with two objects of class `RprobitB_data`, named `"train"` and `"test"`.

Examples

```
### simulate choices for demonstration
x <- simulate_choices(form = choice ~ covariate, N = 10, T = 10, J = 2)

### 70% of deciders in the train subsample,
### 30% of deciders in the test subsample
train_test(x, test_proportion = 0.3, by = "N")

### 2 randomly chosen choice occasions per decider in the test subsample,
### the rest in the train subsample
train_test(x, test_number = 2, by = "T", random = TRUE)
```

---

transform.RprobitB_fit

Transform fitted probit model

Description

Given an object of class `RprobitB_fit`, this function can:

- change the length `B` of the burn-in period,
- change the the thinning factor `Q` of the Gibbs samples,
- change the utility `scale`.

Usage

```
## S3 method for class 'RprobitB_fit'
transform(
  `_data`,
  B = NULL,
  Q = NULL,
  scale = NULL,
  check_preference_flip = TRUE,
  ...
)
```

Arguments

`_data`	An object of class `RprobitB_fit`.
`B`	`[integer(1)]` The length of the burn-in period.
`Q`	`[integer(1)]` The thinning factor for the Gibbs samples.

scale [character(1)]
A character which determines the utility scale. It is of the form <parameter> := <value>, where <parameter> is either the name of a fixed effect or Sigma_<j>,<j> for the <j>th diagonal element of Sigma, and <value> is the value of the fixed parameter.

check_preference_flip
Set to TRUE to check for flip in preferences after new scale.

... Currently not used.

Details

See the vignette "Model fitting" for more details: vignette("v03_model_fitting", package = "RprobitB").

Value

The transformed RprobitB_fit object.

---

update.RprobitB_fit *Update and re-fit probit model*

Description

This function estimates a nested probit model based on a given RprobitB_fit object.

Usage

```
## S3 method for class 'RprobitB_fit'
update(
  object,
  form,
  re,
  alternatives,
  id,
  idc,
  standardize,
  impute,
  scale,
  R,
  B,
  Q,
  print_progress,
  prior,
  latent_classes,
  ...
)
```

Arguments

object	An object of class RprobitB_fit.
form	[formula] A model description with the structure choice ~ A | B | C, where • choice is the name of the dependent variable (the choices), • A are names of alternative and choice situation specific covariates with a coefficient that is constant across alternatives, • B are names of choice situation specific covariates with alternative specific coefficients, • and C are names of alternative and choice situation specific covariates with alternative specific coefficients. Multiple covariates (of one type) are separated by a + sign. By default, alternative specific constants (ASCs) are added to the model. They can be removed by adding +0 in the second spot. In the ordered probit model (ordered = TRUE), the formula object has the simple structure choice ~ A. ASCs are not estimated.
re	[character() | NULL] Names of covariates with random effects. If re = NULL (the default), there are no random effects. To have random effects for the ASCs, include "ASC" in re.
alternatives	[character()] The names of the choice alternatives. If not specified, the choice set is defined by the observed choices. If ordered = TRUE, alternatives is assumed to be specified with the alternatives ordered from worst to best.
id	[character(1)] The name of the column in choice_data that contains unique identifier for each decision maker.
idc	[character(1)] The name of the column in choice_data that contains unique identifier for each choice situation of each decision maker. By default, these identifier are generated by the order of appearance.
standardize	[character() | "all"] Names of covariates that get standardized. Covariates of type 1 or 3 have to be addressed by <covariate>_<alternative>. If standardize = "all", all covariates get standardized.
impute	A character that specifies how to handle missing covariate entries in choice_data, one of: • "complete_cases", removes all rows containing missing covariate entries (the default), • "zero", replaces missing covariate entries by zero (only for numeric columns), • "mean", imputes missing covariate entries by the mean (only for numeric columns).

`scale` [character(1)]
A character which determines the utility scale. It is of the form <parameter> := <value>, where <parameter> is either the name of a fixed effect or Sigma_<j>,<j> for the <j>th diagonal element of Sigma, and <value> is the value of the fixed parameter.

`R` [integer(1)]
The number of iterations of the Gibbs sampler.

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

`Q` [integer(1)]
The thinning factor for the Gibbs samples.

`print_progress` [logical(1)]
Print the Gibbs sampler progress?

`prior` [list]
A named list of parameters for the prior distributions. See the documentation of check_prior for details about which parameters can be specified.

`latent_classes` [list() | NULL]
Optionally parameters specifying the number of latent classes and their updating scheme. The values in brackets are the default.

- C (1): The fixed number (greater or equal 1) of (initial) classes.
- wb_update (FALSE): Set to TRUE for weight-based class updates.
- dp_update (FALSE): Set to TRUE for Dirichlet process class updates.
- Cmax (10): The maximum number of latent classes.

The following specifications are used for the weight-based updating scheme:

- buffer (50): The number of iterations to wait before the next update.
- epsmin (0.01): The threshold weight for removing a latent class.
- epsmax (0.7): The threshold weight for splitting a latent class.
- deltamin (0.1): The minimum mean distance before merging two classes.
- deltashift (0.5): The scale for shifting the class means after a split.

`...` Currently not used.

Details

All parameters (except for object) are optional and if not specified retrieved from the specification for object.

Value

An object of class RprobitB_fit.

update_b *Update class means*

Description

Update class means

Usage

```
update_b(beta, Omega, z, m, Sigma_b_0_inv, mu_b_0)
```

Arguments

beta	[matrix(nrow = P_r, ncol = N)] The matrix of the decider-specific coefficient vectors.
Omega	[matrix(nrow = P_r * P_r, ncol = C)] The matrix of vectorized class covariance matrices as columns.
z	[numeric(N)] The decider class allocations.
m	[numeric(C)] The vector of current class frequencies.
Sigma_b_0_inv	[matrix(P_r, P_r)] The prior precision of the class mean.
mu_b_0	[numeric(P_r)] The mean vector of the normal prior for each b_c.

Value

A matrix of updated means for each class in columns.

Examples

```
N <- 100
b <- cbind(c(0, 0), c(1, 1))
Omega <- matrix(c(1, 0.3, 0.3, 0.5, 1, -0.3, -0.3, 0.8), ncol = 2)
z <- c(rep(1, N / 2), rep(2, N / 2))
m <- as.numeric(table(z))
beta <- sapply(
  z, function(z) oeli::rmvnorm(n = 1, b[, z], matrix(Omega[, z], 2, 2))
)
update_b(
  beta = beta, Omega = Omega, z = z, m = m,
  Sigma_b_0_inv = diag(2), mu_b_0 = c(0, 0)
)
```

update_b_c *Update mean of a single class*

Description

Update mean of a single class

Usage

```
update_b_c(bar_b_c, Omega_c, m_c, Sigma_b_0_inv, mu_b_0)
```

Arguments

bar_b_c	[numeric(P_r)] The average observation of this class.
Omega_c	[matrix(P_r, P_r)] The class covariance matrix.
m_c	[integer(1)] The number of observations in this class.
Sigma_b_0_inv	[matrix(P_r, P_r)] The prior precision of the class mean.
mu_b_0	[numeric(P_r)] The mean vector of the normal prior for each b_c.

Value

An update for b_c.

Examples

```
update_b_c(
  bar_b_c = c(0, 0), Omega_c = diag(2), m_c = 10,
  Sigma_b_0_inv = diag(2), mu_b_0 = c(0, 0)
)
```

update_classes_dp *Dirichlet process class updates*

Description

Dirichlet process class updates

Usage

```
update_classes_dp(
  beta,
  z,
  b,
  Omega,
  delta,
  mu_b_0,
  Sigma_b_0,
  n_Omega_0,
  V_Omega_0,
  identify_classes = FALSE,
  Cmax = 10L
)
```

Arguments

beta	[matrix(nrow = P_r, ncol = N)] The matrix of the decider-specific coefficient vectors.
z	[numeric(N)] The decider class allocations.
b	[matrix(nrow = P_r, ncol = C)] The matrix of class means as columns.
Omega	[matrix(nrow = P_r * P_r, ncol = C)] The matrix of vectorized class covariance matrices as columns.
delta	[numeric(1)] The prior concentration for s.
mu_b_0	[numeric(P_r)] The mean vector of the normal prior for each b_c.
Sigma_b_0	[matrix(P_r, P_r)] The covariance matrix of the normal prior for each b_c.
n_Omega_0	[integer(1)] The degrees of freedom of the Inverse Wishart prior for each Omega_c.
V_Omega_0	[matrix(P_r, P_r)] The scale matrix of the Inverse Wishart prior for each Omega_c.
identify_classes	[logical(1)] Identify classes by decreasing class weights?
Cmax	[integer(1)] The maximum number of classes, used to allocate space.

Value

A list of updated values for z, b, Omega, and C.

Examples

```
set.seed(1)
z <- c(rep(1, 10),rep(2, 10))
b <- matrix(c(5, 5, 5, -5), ncol = 2)
Omega <- matrix(c(1, 0.3, 0.3, 0.5, 1, -0.3, -0.3, 0.8), ncol = 2)
beta <- sapply(
  z, function(z) oeli::rmvnorm(n = 1, b[, z], matrix(Omega[, z], 2, 2))
)
beta[, 1] <- c(-10, 10)
update_classes_dp(
  beta = beta, z = z, b = b, Omega = Omega,
  delta = 1, mu_b_0 = numeric(2), Sigma_b_0 = diag(2),
  n_Omega_0 = 4, V_Omega_0 = diag(2)
)
```

---

update_classes_wb *Weight-based class updates*

Description

Weight-based class updates

Usage

```
update_classes_wb(
  s,
  b,
  Omega,
  epsmin = 0.01,
  epsmax = 0.7,
  deltamin = 0.1,
  deltashift = 0.5,
  identify_classes = FALSE,
  Cmax = 10L
)
```

Arguments

s	[numeric(C)] The vector of class weights.
b	[matrix(nrow = P_r, ncol = C)] The matrix of class means as columns.
Omega	[matrix(nrow = P_r * P_r, ncol = C)] The matrix of vectorized class covariance matrices as columns.
epsmin	[numeric(1)] The threshold weight for removing a class.

epsmax [numeric(1)]
The threshold weight for splitting a class.

deltamin [numeric(1)]
The threshold difference in class means for joining two classes.

deltashift [numeric(1)]
The scale for shifting the class means after a split.

identify_classes
[logical(1)]
Identify classes by decreasing class weights?

Cmax [integer(1)]
The maximum number of classes, used to allocate space.

Details

The following updating rules apply:

- Class c is removed if $s_c < \epsilon_{min}$.
- Class c is split into two classes, if $s_c > \epsilon_{max}$.
- Two classes c_1 and c_2 are merged to one class, if $||b_{c_1} - b_{c_2}|| < \delta_{min}$.

Value

A list of updated values for s, b, and Omega and the indicator update_type which signals the type of class update:

- 0: no update
- 1: removed class
- 2: split class
- 3: merged classes

Examples

```
s <- c(0.7, 0.3)
b <- matrix(c(1, 1, 1, -1), ncol = 2)
Omega <- matrix(c(0.5, 0.3, 0.3, 0.5, 1, -0.1, -0.1, 0.8), ncol = 2)

### no update
update_classes_wb(s = s, b = b, Omega = Omega)

### remove class 2
update_classes_wb(s = s, b = b, Omega = Omega, epsmin = 0.31)

### split class 1
update_classes_wb(s = s, b = b, Omega = Omega, epsmax = 0.69)

### merge classes 1 and 2
update_classes_wb(s = s, b = b, Omega = Omega, deltamin = 3)
```

update_coefficient *Update coefficient vector*

Description

Update coefficient vector

Usage

```
update_coefficient(mu_beta_0, Sigma_beta_0_inv, XSigX, XSigU)
```

Arguments

mu_beta_0	[numeric(P)] The prior mean for the coefficient vector,
Sigma_beta_0_inv	[matrix(P, P)] The prior precision for the coefficient vector.
XSigX	[matrix(P, P)] The matrix $\sum_{n=1}^{N} X_n' \Sigma^{-1} X_n$.
XSigU	[numeric(P)] The vector $\sum_{n=1}^{N} X_n' \Sigma^{-1} U_n$.

Value

An update for the coefficient vector.

Examples

```
beta_true <- matrix(c(-1, 1), ncol = 1)
Sigma <- matrix(c(1, 0.5, 0.2, 0.5, 1, 0.2, 0.2, 0.2, 2), ncol = 3)
N <- 100
X <- replicate(N, matrix(rnorm(6), ncol = 2), simplify = FALSE)
eps <- replicate(
  N, oeli::rmvnorm(n = 1, mean = c(0, 0, 0), Sigma = Sigma),
  simplify = FALSE
)
U <- mapply(
  function(X, eps) X %*% beta_true + eps, X, eps, SIMPLIFY = FALSE
)
mu_beta_0 <- c(0, 0)
Sigma_beta_0_inv <- diag(2)
XSigX <- Reduce(
  `+`, lapply(X, function(X) t(X) %*% solve(Sigma) %*% X)
)
XSigU <- Reduce(
  `+`, mapply(function(X, U) t(X) %*% solve(Sigma) %*% U, X, U,
  SIMPLIFY = FALSE)
```

```
)
R <- 10
beta_draws <- replicate(
  R, update_coefficient(mu_beta_0, Sigma_beta_0_inv, XSigX, XSigU),
  simplify = TRUE
)
rowMeans(beta_draws)
```

`update_d` *Update utility threshold increments*

Description

Update utility threshold increments

Usage

```
update_d(d, y, sys, ll, mu_d_0, Sigma_d_0, Tvec, step_scale = 0.1)
```

Arguments

`d`	[`numeric(J - 2)`] Threshold increments.
`y`	[`matrix(nrow = N, ncol = max(Tvec))`] Choices `1, . . . , J` for each decider in each choice occasion.
`sys`	[`matrix(nrow = N, ncol = max(Tvec))`] Systematic utilties for each decider in each choice occasion.
`ll`	[`numeric(1)`] Current log-likelihood value.
`mu_d_0`	[`numeric(J - 2)`] The mean vector of the normal prior for `d` .
`Sigma_d_0`	[`matrix(J - 2, J - 2)`] The covariance matrix of the normal prior for `d`.
`Tvec`	[`integer(N)`] Number of choice occasions per decider.
`step_scale`	[`numeric(1)`] Scaling the variance for the Gaussian proposal.

Value

An update for `d`.

Examples

```
set.seed(1)
N <- 1000
d_true <- rnorm(2)
gamma <- c(-Inf, 0, cumsum(exp(d_true)), Inf)
X <- matrix(rnorm(N * 2L), ncol = 2L)
beta <- c(0.8, -0.5)
mu <- matrix(as.vector(X %*% beta), ncol = 1L)
U <- rnorm(N, mean = mu[, 1], sd = 1)
yvec <- as.integer(cut(U, breaks = gamma, labels = FALSE))
y <- matrix(yvec, ncol = 1L)
Tvec <- rep(1, N)
mu_d_0 <- c(0, 0)
Sigma_d_0 <- diag(2) * 5
d <- rnorm(2)
ll <- -Inf
R <- 1000
for (iter in seq_len(R)) {
  ans <- update_d(
    d = d, y = y, sys = mu, ll = ll, mu_d_0 = mu_d_0, Sigma_d_0 = Sigma_d_0,
    Tvec = Tvec
  )
  d  <- ans$d
  ll <- ans$ll
}
cbind("true" = d_true, "sample" = d) |> round(2)
```

`update_m` *Update class sizes*

Description

Update class sizes

Usage

```
update_m(C, z, non_zero = FALSE)
```

Arguments

`C`	[`integer(1)`] The number (greater or equal 1) of latent classes of decision makers.
`z`	[`numeric(N)`] The decider class allocations.
`non_zero`	[`logical(1)`] Enforce strictly positive values in `m` (for numerical stability)?

Value

An update for m.

Examples

```
update_m(C = 4, z = c(1, 1, 1, 2, 2, 3))
```

---

update_Omega *Update class covariances*

Description

Update class covariances

Usage

```
update_Omega(beta, b, z, m, n_Omega_0, V_Omega_0)
```

Arguments

beta	[matrix(nrow = P_r, ncol = N)] The matrix of the decider-specific coefficient vectors.
b	[matrix(nrow = P_r, ncol = C)] The matrix of class means as columns.
z	[numeric(N)] The decider class allocations.
m	[numeric(C)] The vector of current class frequencies.
n_Omega_0	[integer(1)] The degrees of freedom of the Inverse Wishart prior for each Omega_c.
V_Omega_0	[matrix(P_r, P_r)] The scale matrix of the Inverse Wishart prior for each Omega_c.

Value

A matrix of updated covariance matrices for each class in columns.

Examples

```
N <- 100
b <- cbind(c(0, 0), c(1, 1))
Omega <- matrix(c(1, 0.3, 0.3, 0.5, 1, -0.3, -0.3, 0.8), ncol = 2)
z <- c(rep(1, N / 2), rep(2, N / 2))
m <- as.numeric(table(z))
beta <- sapply(
```

```
  z, function(z) oeli::rmvnorm(n = 1, b[, z], matrix(Omega[, z], 2, 2))
)
update_Omega(
  beta = beta, b = b, z = z, m = m,
  n_Omega_0 = 4, V_Omega_0 = diag(2)
)
```

`update_Omega_c` *Update covariance of a single class*

Description

Update covariance of a single class

Usage

```
update_Omega_c(S_c, m_c, n_Omega_0, V_Omega_0)
```

Arguments

`S_c`	[`matrix(P_r, P_r)`] The scatter matrix of this class.
`m_c`	[`integer(1)`] The number of observations in this class.
`n_Omega_0`	[`integer(1)`] The degrees of freedom of the Inverse Wishart prior for each `Omega_c`.
`V_Omega_0`	[`matrix(P_r, P_r)`] The scale matrix of the Inverse Wishart prior for each `Omega_c`.

Value

An update for `Omega_c`.

Examples

```
update_Omega_c(S_c = diag(2), m_c = 10, n_Omega_0 = 4, V_Omega_0 = diag(2))
```

update_s *Update class weight vector*

Description

Update class weight vector

Usage

```
update_s(delta, m)
```

Arguments

delta	[numeric(1)] The prior concentration for s.
m	[numeric(C)] The vector of current class frequencies.

Value

An update for s.

Examples

```
update_s(delta = 1, m = 4:1)
```

update_Sigma *Update error covariance matrix*

Description

Update error covariance matrix

Usage

```
update_Sigma(n_Sigma_0, V_Sigma_0, N, S)
```

Arguments

n_Sigma_0	[integer(1)] The degrees of freedom of the Inverse Wishart prior for Sigma.
V_Sigma_0	[matrix(J - 1, J - 1)] The scale matrix of the Inverse Wishart prior for Sigma.
N	[integer(1)] The sample size.
S	[matrix(J - 1, J - 1)] The sum over the outer products of the residuals $(\epsilon_n)_{n=1,\dots,N}$.

Value

An update for Sigma.

Examples

```
(Sigma_true <- matrix(c(1, 0.5, 0.2, 0.5, 1, 0.2, 0.2, 0.2, 2), ncol = 3))
beta <- matrix(c(-1, 1), ncol = 1)
N <- 100
X <- replicate(N, matrix(rnorm(6), ncol = 2), simplify = FALSE)
eps <- replicate(
  N, oeli::rmvnorm(n = 1, mean = c(0, 0, 0), Sigma = Sigma_true),
  simplify = FALSE
)
U <- mapply(function(X, eps) X %*% beta + eps, X, eps, SIMPLIFY = FALSE)
n_Sigma_0 <- 4
V_Sigma_0 <- diag(3)
outer_prod <- function(X, U) (U - X %*% beta) %*% t(U - X %*% beta)
S <- Reduce(`+`, mapply(
  function(X, U) (U - X %*% beta) %*% t(U - X %*% beta), X, U,
  SIMPLIFY = FALSE
))
Sigma_draws <- replicate(100, update_Sigma(n_Sigma_0, V_Sigma_0, N, S))
apply(Sigma_draws, 1:2, mean)
```

---

update_U *Update utility vector*

Description

Update utility vector

Usage

```
update_U(U, y, sys, Sigma_inv)
```

Arguments

U	[numeric(J - 1)] The current utility vector.
y	[integer(1)] The index of the chosen alternative, from 1 to J.
sys	[numeric(J - 1)] The systematic utility.
Sigma_inv	[matrix(J - 1, J - 1)] The inverted error covariance matrix.

Value

An update for (a single) U.

Examples

```
U <- sys <- c(0, 0, 0)
Sigma_inv <- diag(3)
lapply(1:4, function(y) update_U(U, y, sys, Sigma_inv))
```

---

update_U_ranked *Update ranked utility vector*

Description

Update ranked utility vector

Usage

```
update_U_ranked(U, sys, Sigma_inv)
```

Arguments

U	[numeric(J - 1)] The current utility vector.
sys	[numeric(J - 1)] The systematic utility.
Sigma_inv	[matrix(J - 1, J - 1)] The inverted error covariance matrix.

Value

An update for (a single) ranked U.

Examples

```
U <- sys <- c(0, 0)
Sigma_inv <- diag(2)
update_U_ranked(U, sys, Sigma_inv)
```

update_z *Update class allocation vector*

Description

Update class allocation vector

Usage

```
update_z(s, beta, b, Omega)
```

Arguments

s	[numeric(C)] The vector of class weights.
beta	[matrix(nrow = P_r, ncol = N)] The matrix of the decider-specific coefficient vectors.
b	[matrix(nrow = P_r, ncol = C)] The matrix of class means as columns.
Omega	[matrix(nrow = P_r * P_r, ncol = C)] The matrix of vectorized class covariance matrices as columns.

Value

An update for z.

Examples

```
update_z(
  s = c(0.6, 0.4), beta = matrix(c(-2, 0, 2), ncol = 3),
  b = cbind(0, 1), Omega = cbind(1, 1)
)
```

Index